Conception et sécurisation d'unités arithmétiques hautes performances pour courbes elliptiques

Julien Francq

Numéro d'identification :

Académie de Montpellier

Université Montpellier II

— **Sciences et Techniques du Languedoc** —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité	: **Informatique**
Formation Doctorale	: **Informatique**
École Doctorale	: **Information, Structures, Systèmes**

Conception et sécurisation d'unités arithmétiques hautes performances pour courbes elliptiques

par

Julien FRANCQ

Soutenue le 16 Décembre 2009 devant le Jury composé de :

M. Jean-Luc Danger, Directeur d'études, TélécomParisTech, ENST, ... Rapporteur
M. Guy Gogniat, Professeur, Université de Bretagne Sud, Lab-STICC, ... Rapporteur
M. Viktor Fischer, Professeur, Université de Saint-Étienne, ... Président du Jury
M. Pierre-Yvan Liardet, *Senior Research Expert*, ST Microelectronics, ... Examinateur
M. Lionel Torres, Professeur, Université Montpellier 2, LIRMM, ... Examinateur
M. Jean-Claude Bajard, Professeur, Univ. Pierre et Marie Curie Paris 6, LIP6, ... Directeur de Thèse
M. Jean-Baptiste Rigaud, Maître Assistant, ENSMSE, CMP-GC, ... Co-directeur de Thèse
M. Arnaud Tisserand, Chargé de Recherche CNRS, IRISA, ... Co-directeur de Thèse

Sommaire

Liste des algorithmes

Liste des tableaux

Liste des figures

Introduction

La cryptographie est l'étude des techniques théoriques et pratiques pouvant permettre de sécuriser des échanges de données entre parties communicantes, et/ou d'effectuer des opérations de signature, d'authentification, d'identification, de certification, etc. L'objectif de la cryptographie est de permettre de protéger les parties communicantes de l'action d'éventuelles personnes malveillantes, également appelées « attaquants ». Son histoire est à la fois longue et fascinante. Ses origines remontent à l'édification de la civilisation égyptienne il y a 4000 ans environ [Kah67]. Plus récemment, les deux guerres mondiales ainsi que la guerre froide ont provoqué l'intensification de la recherche dans ce domaine. Ainsi, jusqu'à la fin des années 1950, la cryptographie était principalement destinée à protéger les secrets et les stratégies des différents pays. Elle était donc principalement réservée aux militaires, aux services diplomatiques ou aux gouvernements.

Puis vinrent les années 1960 avec la prolifération des ordinateurs et des systèmes de communications au sens large. C'est pendant cette période qu'est apparue la nécessité de protéger des informations concernant des utilisateurs privés, et non plus seulement institutionnels. C'est dans ce contexte qu'un employé d'IBM nommé Feistel imagina dans le début des années 1970 un système cryptographique (ou « cryptosystème ») permettant le chiffrement de données : le DES (*Data Encryption Standard*) [FIP93]. C'est un cryptosystème dit « symétrique », c'est-à-dire que les parties communicantes disposent du même secret, appelé « clé », pour chiffrer et déchiffrer des textes. La cryptographie symétrique a plusieurs avantages. Tout d'abord, elle est très efficace : avec un DES, il suffit de quelques centaines de nanosecondes pour chiffrer 64 bits de données, et l'on peut atteindre des débits de l'ordre du Gigabit par seconde. De plus, la taille des clés est relativement petite : un DES utilisant une clé de 56 bits peut chiffrer 64 bits de données.

Toutefois, l'utilisation de ces systèmes entraîne des contraintes majeures concernant la distribution et la gestion des clés [MVO96]. Tout d'abord, les clés doivent être distribuées uniquement aux parties communicantes concernées, et non à des attaquants. De plus, ces derniers ne doivent pas être capables de récupérer les clés durant leur distribution. Surtout, dans un réseau connectant un grand nombre de parties communicantes, il y a plusieurs paires de clés à gérer. Du coup, pour une gestion de clés efficace, un tiers digne de confiance est souvent nécessaire. Enfin, pour chaque nouvelle session d'échange de données, il est souvent conseillé de changer de clé, ce qui peut être assez contraignant suivant les applications.

L'un des développements les plus spectaculaires dans l'histoire de la cryptographie s'est produit en 1976 lorsque Diffie et Hellman ont publié « *New Directions in Cryptography* » [DH76]. Cet article a introduit le concept révolutionnaire de cryptographie « asymétrique » : chaque partie communicante sélectionne une paire de clés, l'une restant privée c'est-à-dire secrète et l'autre étant publique c'est-à-dire connue de tous. Le fait qu'une clé appartenant à une partie communicante soit révélée ne pose pas de problème de sécurité, car il est mathématiquement difficile de retrouver sa clé privée connaissant sa clé publique. Même si les auteurs n'ont présenté à l'époque aucune réalisation concrète d'un cryptosystème à clé publique, cela a attisé la curiosité du monde de la cryptographie.

En 1978, Rivest, Shamir et Adleman ont présenté la première réalisation concrète d'un cryptosystème asymétrique, appelé RSA [RSA78]. Le RSA est basé sur un problème mathématique difficile : la factorisation de grands nombres premiers. Pendant les années 80 et 90, la recherche de méthodes efficaces dédiées à la factorisation a connu des avancées importantes, à tel point que la taille des clés publique et privée doit augmenter régulièrement afin que l'utilisation du RSA reste sûre. Il y a une dizaine d'années, utiliser une clé de 1024 bits conférait un niveau de sécurité suffisant ; de nos jours, il est plutôt conseillé d'utiliser des clés de 2048, voire 4096 bits ! Il est évident qu'implanter un chemin de données de 4096 bits pose des problèmes aux concepteurs de circuits sécurisés, car il leur faut redoubler d'ingéniosité afin de garantir des performances acceptables en terme de surface de circuit utilisé et de temps de calcul pour leur cryptosystème.

En 1985, Miller [Mil86] et Koblitz [Kob87] ont posé indépendamment les bases de la cryptographie basée sur les courbes elliptiques (*Elliptic Curve Cryptography*, ECC). La sécurité de ces systèmes repose sur le problème du logarithme discret sur courbes elliptiques. L'algorithme connu comme étant le plus efficace pour résoudre un tel problème est à temps exponentiel, contrairement au RSA pour lequel il existe des algorithmes à temps sous-exponentiel. Ainsi, à niveau de sécurité équivalent, ECC requiert des clés beaucoup plus petites que le RSA. Il est couramment admis par exemple qu'utiliser un RSA avec une longueur de clé de 1024 bits apporte la même sécurité qu'un ECC avec une longueur de clé de 160 bits [SEC00a]. Or, implanter des chemins de données plus petits confère beaucoup d'avantages : les calculs sont plus rapides, la consommation électrique globale est diminuée, et l'espace mémoire est réduit. Ainsi, les systèmes basés sur les courbes elliptiques sont très prometteurs lorsqu'ils sont implantés dans des environnements à fortes contraintes industrielles (les cartes à puces, les téléphones mobiles, les assistants personnels, etc.). Ils sont donc appelés à remplacer progressivement le RSA [Van04]. La publication de courbes elliptiques standardisées par l'institut américain des standards et de la technologie (*National Institute of Standards and Technology*, NIST) en 2000 contribue à accélérer ce processus [FIP00].

Jusqu'au milieu des années 90, les concepteurs de circuits sécurisés focalisaient principalement leurs recherches sur l'amélioration des performances de leurs cryptosystèmes, ainsi que sur leur robustesse mathématique. Cependant, en 1996, Kocher présenta une nouvelle forme d'attaque dite « par observation » [Koc96]. Un attaquant n'a plus besoin de résoudre le problème mathématique sous-jacent au cryptosystème : il lui suffit simplement d'observer les caractéristiques physiques de ce système durant le traitement de la clé afin de récupérer celle-ci. Parallèlement, en 1997, Boneh, DeMillo et Lipton proposèrent un autre type d'attaque dite « par perturbation » [BDL97]. Contrairement à l'attaque passive proposée par Kocher, celle-ci est active : elle consiste à modifier l'environnement du cryptosystème de telle sorte que le fonctionnement de celui-ci s'en trouve altéré et qu'il apparaisse des fautes. Si ces dernières sont correctement injectées en temps et en espace, elles peuvent être exploitées afin de récupérer la clé.

Ainsi, ces dernières années, une nouvelle problématique est apparue pour les concep-

teurs de circuits sécurisés : non seulement il faut que leurs cryptosystèmes soient performants, mais il faut en plus les protéger contre ces nouvelles attaques en insérant des parades, ou « contre-mesures ». Cependant, l'implantation de ces contre-mesures entraîne souvent un surcoût en terme de surface et/ou de temps. Il est ainsi souvent difficile en pratique d'obéir au double impératif de performance et de sécurité du fait du surcoût parfois élevé des contre-mesures implantées.

L'objectif de cette thèse est de concevoir une unité arithmétique (UA) à hautes performances pour courbes elliptiques, protégée à la fois contre les attaques par observation et par perturbation. Cette thèse peut être découpée en plusieurs sous-objectifs.

Le premier sous-objectif de cette thèse est de concevoir une UA à hautes performances pour courbes elliptiques. Le deuxième sous-objectif est de sécuriser cette UA contre les attaques par observation en utilisant les différentes contre-mesures existantes dans l'état de l'art. Le troisième sous-objectif est de protéger l'UA obtenue après validation du deuxième sous-objectif contre les attaques par perturbation. Pour ce faire, nous proposons l'implantation d'une contre-mesure prometteuse, appelée « préservation de la parité ».

La préservation de la parité est un concept qui a été préalablement proposé dans le cadre de l'amélioration de la fiabilité des circuits [Par06]. Nous proposons ici d'appliquer ce concept à l'amélioration de la sécurité d'un cryptosystème vis-à-vis des attaques par perturbation. Préserver la parité d'un cryptosystème consiste à le concevoir en faisant en sorte que la parité de ses entrées (ou des entrées d'une de ses fonctions intermédiaires) est égale à la parité de ses sorties (ou des sorties d'une de ses fonctions intermédiaires).

Le présent document se compose de deux parties.

La première partie décrit différents états de l'art. Le chapitre 1 est consacré à l'arithmétique des ordinateurs. Ce chapitre montre que selon le mode de représentation des nombres choisi lors de l'implantation d'une UA, les opérations arithmétiques élémentaires (addition, multiplication...) sur ces nombres se font à un coût plus ou moins acceptable. Les calculs cryptographiques sur les courbes elliptiques se déroulant sur des corps finis, ce chapitre détaille également différents algorithmes permettant d'effectuer des calculs modulaires efficacement. Le chapitre 2 décrit l'arithmétique sous-jacente aux courbes elliptiques. Il y est notamment expliqué comment on peut accélérer les calculs cryptographiques sur ces dernières. Le chapitre 3 détaille les différentes attaques mises en évidence sur les courbes elliptiques, qu'elles soient par observation ou par perturbation. Ceci permet de comprendre contre quel danger potentiel un concepteur de circuit sécurisé doit protéger le circuit qu'il implante. Le chapitre 4 expose différentes contre-mesures à ces attaques. Certaines contre-mesures sont implantables dans n'importe quel cryptosystème, d'autres ont été spécialement conçues pour le cas particulier des courbes elliptiques. Ce chapitre tend à faire une étude comparative de l'impact de ces contre-mesures sur les performances d'un cryptosystème basé sur les courbes elliptiques.

La seconde partie contient les chapitres 5 et 6. Le chapitre 5 décrit l'architecture de l'UA proposée, avec et sans les contre-mesures contre les attaques par observation décrites dans le chapitre 4. Ce chapitre est important car il décrit l'UA qui sera notre cas d'étude

pour l'implantation de la préservation de la parité. Enfin, le chapitre 6, expose le concept de préservation de parité, ainsi que son application concrète sur notre UA sécurisée contre les attaques par observation.

Première partie

État de l'art

Chapitre 1

Arithmétique des ordinateurs

Sommaire

L'arithmétique des ordinateurs est une discipline qui étudie les systèmes de représentation des nombres utiles au calcul, ainsi que les algorithmes permettant d'effectuer les opérations arithmétiques et d'évaluer les principales fonctions mathématiques que sont par exemple l'addition (ou la soustraction), la multiplication, la division, etc., mais également les fonctions trigonomètriques (sinus, cosinus, etc.) ou encore les fonctions exponentielle et logarithmique [EL04].

Ce chapitre détaille différents systèmes de représentation des nombres, avec leurs avantages et leurs inconvénients respectifs lorsqu'ils sont utilisés pour concevoir des opérateurs arithmétiques. Les calculs cryptographiques se déroulant souvent sur des corps finis, ce chapitre détaille également différents algorithmes permettant d'effectuer des calculs modulaires efficacement. Ainsi, cette étude préalable permettra de concevoir une UA performante, en termes de fréquence de fonctionnement et de surface occupée. Cette UA sera décrite dans le chapitre 5.

Ce chapitre permet également de montrer que les algorithmes utilisés en arithmétique des ordinateurs sont souvent plus complexes qu'on ne croit, et qu'ils sont en général très différents de ceux que l'on utilise pour faire des calculs « à la main » [EL04].

1.1 Représentation des nombres

Cette section présente quelques méthodes couramment utilisées afin de représenter des nombres (positifs ou négatifs). Nous travaillons uniquement avec des entiers. Les représentations des nombres peuvent être classées en deux catégories : les systèmes binaires et redondants.

1.1.1 Systèmes de numération classiques

Dans un premier temps, il est listé quelques systèmes de numération binaires couramment utilisés. Puis, quelques inconvénients inhérents à l'utilisation de ces représentations seront évoqués.

1.1.1.1 Vue d'ensemble

Pour coder des nombres entiers positifs, un codage est très souvent utilisé : la numération simple de position[1] [Mul89]. En revanche, pour coder des nombres entiers potentiellement négatifs, plusieurs solutions sont offertes aux concepteurs de circuits. On peut par exemple utiliser la représentation « signe et magnitude[2] ». On peut également utiliser des représentations biaisées, ou même des représentations à complément, qui sont des cas particuliers des représentations biaisées. Les plus connues sont celle dites à (la base) un et deux.

Numération simple de position. Un nombre X peut être codé à l'aide de n chiffres dans n'importe quelle base β supérieure ou égale à 2, tels que :

$$X \;=\; (x_{n-1}\cdots x_1 x_0) \;=\; \sum_{i=0}^{n-1} x_i \beta^i,$$

avec $x_i \in \{0, 1, \cdots, \beta - 1\}$ et où $X \in [0, \beta^n - 1]$. Ainsi, pour le cas particulier de la base 2, on obtient :

$$X \;=\; (x_{n-1}\cdots x_1 x_0) \;=\; \sum_{i=0}^{n-1} x_i 2^i,$$

avec $x_i \in \{0, 1\}$ et où $X \in [0, 2^n - 1]$.

Exemple 1.1. $X = 11 = 2^3 + 2^1 + 2^0$. Ainsi, X peut être représenté en numération simple de position avec $\beta = 2$ par (1011), avec $n = 4$ et $X \in [0, 2^4 - 1]$.

[1] Il y a près de 4000 ans, les Babyloniens utilisèrent historiquement le premier système de numération simple de position en base 60.

[2] La magnitude est également appelée « valeur absolue ».

Signe et magnitude. Représenter un nombre relatif X en notation « signe et magnitude » consiste à utiliser un bit de signe noté s_a. Par convention, si $s_a = 0$ (respectivement si $s_a = 1$) alors X est positif (négatif). Ainsi :

$$X \;=\; (s_a x_{n-2} \cdots x_1 x_0) \;=\; (-1)^{s_a} . \sum_{i=0}^{n-2} x_i 2^i,$$

avec $x_i \in \{0, 1\}$ et où $X \in [-(2^{n-1} - 1), (2^{n-1} - 1)]$. L'opposé de X s'obtient en inversant son chiffre de signe :

$$-X \;=\; (\overline{s_a} x_{n-2} \cdots x_1 x_0).$$

Cette représentation comporte deux inconvénients majeurs. Tout d'abord, la comparaison de deux nombres en utilisant cette représentation est difficile du fait de la double représentation du nombre $X = 0$ (par exemple, si $n = 4$, $X = 0$ a deux représentants qui sont (0000) et (1000)). Ensuite, dans ce codage, les algorithmes mis en jeu pour l'addition et la soustraction sont complexes.

Représentations biaisées. Le principe de cette représentation réside dans l'addition d'un biais R à tout nombre relatif X_{math}, de sorte que $X_{\text{math}} + R$ soit toujours positif. X_{math} est ainsi codé par le nombre X tel que $X = X_{\text{math}} + R$. Afin que les domaines de variation des nombres positifs et négatifs soient approximativement les mêmes, le biais est souvent défini par $R = 2^{n-1} - 1$. Ainsi, le domaine de variation des nombres dans cette représentation est $[-2^{n-1} + 1, 2^{n-1}]$[3]. Le chiffre de poids fort indique le signe du nombre. Contrairement à la représentation « signe et magnitude », si le chiffre de poids fort de X est égal à 0 (respectivement à 1) alors le nombre X_{math} est négatif (positif).

Même si l'addition et la soustraction dans cette représentation sont relativement faciles à implanter, cela est plus problématique pour la multiplication et la division.

Représentations à complément. La méthode du complément est basée sur une **représentation biaisée appliquée uniquement aux nombres négatifs**. Si l'on veut coder des nombres relatifs dans $[-P, +Q]$, et afin de garantir l'absence de recouvrement entre valeurs positives et négatives, il est nécessaire et suffisant que le biais R obéisse à la relation :

$$R \geq P + Q + 1. \tag{1.1}$$

[3]Parfois, le biais est défini par $R = 2^{n-1}$. Dans ce cas, le domaine de variation des nombres dans cette représentation est $[-2^{n-1}, 2^{n-1} - 1]$.

Exemple 1.2. Si l'on veut coder des nombres dans $[-8, +7]$, le biais R devra donc être supérieur ou égal à $8+7+1 = 16$. Considérons le cas où $R = 16$. On devra donc additionner 16 à tous les nombres relatifs inclus dans $[-8, -1]$ tandis que ceux inclus dans $[0, 7]$ resteront inchangés dans cette représentation. Par exemple, -8 deviendra $-8+16 = 8$, et 4 restera 4. Il n'y aura pas de recouvrement entre valeurs positives et négatives car l'ensemble des nombres négatifs $[-8, -1]$ deviendra $[8, 15]$, l'ensemble des nombres positifs $[0, 7]$ restera inchangé et : $[0, 7] \cap [8, 15] = \emptyset$. En revanche, si le biais R est égal à 15, l'ensemble des nombres négatifs $[-8, -1]$ deviendra $[7, 14]$, l'ensemble des positifs $[0, 7]$ restera inchangé et $[0, 7] \cap [7, 14] = 7$: il y aura donc recouvrement entre valeurs positives et négatives.

Cette représentation s'avère particulièrement intéressante pour la réalisation d'opérateurs arithmétiques. Par exemple, il peut être démontré que la somme de deux opérandes appartenant à l'intervalle $[-P, +Q]$ s'effectue en additionnant leurs valeurs non signées modulo R. Il faut noter que si R est correctement choisi, cela peut conduire à des réductions modulaires efficaces[4].

Complément à un. Cette approche, utilisant un biais $R = 2^n - 1$, permet le codage de nombres relatifs appartenant à l'intervalle symétrique $[-2^{n-1}+1, 2^{n-1}-1]$. X est codé dans cette représentation par :

$$X = -x_{n-1}.(2^{n-1} - 1) + \sum_{i=0}^{n-2} x_i 2^i.$$

Il peut être démontré que le complément à un de X peut être obtenu par inversion de tous les bits du nombre considéré initialement représenté en numération simple de position [EL04]. Le chiffre de poids fort d'un nombre strictement positif (respectivement négatif) est égal à 0 (à 1).

Cette représentation comporte deux inconvénients majeurs. Tout d'abord, la détection de la valeur zéro se révèle problématique. En effet, il existe un « zéro positif » lorsque tous les bits du nombre sont égaux à 0 et un « zéro négatif » lorsque tous les bits sont égaux à 1. De plus, concevoir un circuit effectuant l'addition de deux nombres codés en complément à 1 n'est pas chose aisée.

Complément à deux (ou complément vrai). La méthode du complément à deux utilise un biais R égal à 2^n et autorise le codage de nombres entiers relatifs appartenant

[4]Par exemple, nous verrons en section 1.2.2 que les nombres premiers dits « de Mersenne » conduisent à des réductions modulaires efficaces.

à $[-2^{n-1}, 2^{n-1} - 1]$. X est codé dans cette représentation par :

$$X \ = \ -x_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i.$$

Il peut être démontré que le complément à deux de X peut être obtenu en ajoutant 1 au complément à un du nombre considéré [EL04].

Exemple 1.3. Le tableau 1.1 résume des exemples de représentations de nombres avec $n = 4$. Lorsque un nombre ne peut pas être représenté dans un système donné, celui-ci est remplacé par un tiret.

Nombre	**Signe/Magnitude**	**Biaisée ($R = 7$)**	**Comp. à 1**	**Comp. à 2**
-8	–	–	–	(1000)
-7	(1111)	(0000)	(1000)	(1001)
-6	(1110)	(0001)	(1001)	(1010)
-5	(1101)	(0010)	(1010)	(1011)
-4	(1100)	(0011)	(1011)	(1100)
-3	(1011)	(0100)	(1100)	(1101)
-2	(1010)	(0101)	(1101)	(1110)
-1	(1001)	(0110)	(1110)	(1111)
0	(0000) / (1000)	(0111)	(0000) / (1111)	(0000)
1	(0001)	(1000)	(0001)	(0001)
2	(0010)	(1001)	(0010)	(0010)
3	(0011)	(1010)	(0011)	(0011)
4	(0100)	(1011)	(0100)	(0100)
5	(0101)	(1100)	(0101)	(0101)
6	(0110)	(1101)	(0110)	(0110)
7	(0111)	(1110)	(0111)	(0111)
8	–	(1111)	–	–

TAB. 1.1 – Exemples de représentations de nombres.

1.1.1.2 Inconvénients communs à toutes ces représentations binaires

En plus des contraintes inhérentes à l'utilisation de certaines représentations présentées précédemment, au moins deux inconvénients majeurs communs à toutes celles-ci tendent à diminuer leur attrait.

Temps de calcul. La figure 1.1 représente la cellule de base pour l'addition, qui est la cellule d'addition complète (*Full-Adder*, FA). Elle est connue également sous le nom de compteur (3,2) [EL04] : elle compte le nombre de 1 présents sur ses 3 entrées a, b et c et donne le résultat en numération simple de position sur 2 bits de sortie r et s.

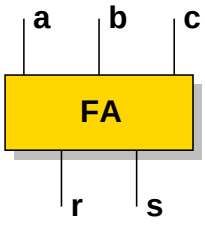

FIG. 1.1 – Cellule d'addition complète ou compteur (3,2).

Elle obéit à l'équation arithmétique $2r + s = a + b + c$ et aux équations logiques :

$$\begin{cases} s = a \oplus b \oplus c \\ r = ab + ac + bc \end{cases}$$

La figure 1.2 représente l'additionneur sur n chiffres le plus simple, également appelé additionneur à propagation séquentielle de retenue. Il permet l'addition de deux nombres $A = (a_{n-1} \cdots a_1 a_0)$ et $B = (b_{n-1} \cdots b_1 b_0)$ du chiffre de poids faible au chiffre de poids fort[5]. Il est composé de n cellules FA connectées en série, leurs entrées nommées c (cf. figure 1.2) servant à récupérer la retenue r générée par la cellule FA précédente.

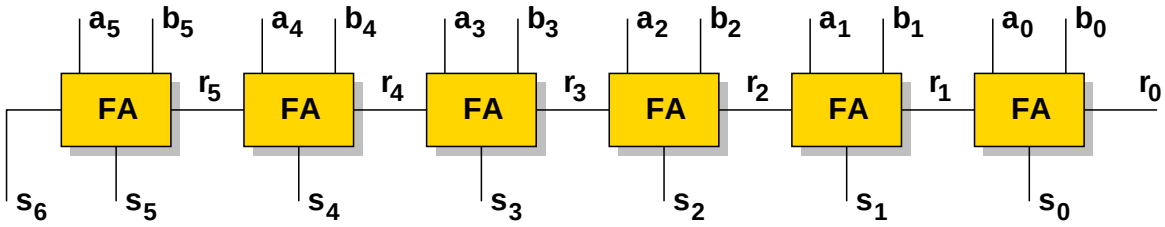

FIG. 1.2 – Additionneur à propagation séquentielle de retenue sur cinq chiffres.

Le principal défaut de cette structure est que la propagation de retenue implique théoriquement un temps de calcul proportionnel à la taille des opérandes. Il existe d'autres architectures améliorant sensiblement ce délai, comme les additionneurs dits « à selection de retenue » [Sla60], « à retenue bondissante » [MM61], « à retenues anticipées » [WS56] ou « préfixes » [LF80], mais le temps nécessaire au calcul d'une addition reste élevé (de l'ordre de $\log(n)$).

[5] C'est d'ailleurs le sens dans lequel on effectue cette opération « à la main ».

Ordre de traitement des chiffres différent selon les opérations. Pour illustrer le second inconvénient, considérons que nous voulons implanter un opérateur arithmétique calculant $\frac{A+B}{C}$, où A, B et C sont sur n chiffres. L'addition $S = A + B$ s'effectuera du chiffre de poids faible au chiffre de poids fort. Or, la division $\frac{S}{C}$ s'effectuera du chiffre de poids fort au chiffre de poids faible, ce qui implique qu'elle doit attendre la fin du calcul de S afin de commencer. Cela engendrera ainsi une latence supplémentaire au système, ce qui n'est pas souhaitable.

Ainsi, parce que certaines représentations détaillées précédemment entraînent théoriquement un temps de calcul proportionnel à la taille des opérandes et suivant les cas des latences supplémentaires dus à une succesion d'opérations, leur utilisation n'est pas souhaitée dans le cadre de la conception d'UA performantes. Choisir d'implanter une représentation redondante peut être une solution à ces problèmes.

1.1.2 Représentations redondantes pour une addition rapide

Comme expliqué dans la section précédente, l'utilisation de systèmes de numération binaires pose des problèmes en terme de temps de calcul et d'ordre de traitement des opérandes lorsque plusieurs opérations se succèdent. Ces deux problèmes suggèrent l'implantation d'additionneurs **à faible délai** et dont **les sorties sont calculées en même temps**. Cette section montre que l'utilisation de **représentations redondantes** permet d'atteindre ces deux objectifs. Cette classe de notations autorise plusieurs représentations possibles pour certains nombres, ce qui peut permettre d'accélérer le temps de calcul de l'additions car cette dernière peut s'effectuer à temps constant. Cette propriété est d'autant plus intéressante lorsqu'il est calculé plusieurs additions successives, comme par exemple lors du calcul d'une multiplication (cf. section 1.1.3), cette dernière opération pouvant être assimilée à une série d'additions et de décalages.

Utiliser des représentations redondantes en vue d'accélérer les calculs n'est pas un concept récent[6]. En 1840, Cauchy utilisait déjà en base 10 tous les chiffres allant de -5 à 5 afin d'accélérer le calcul des multiplications effectuées à la main et de corriger les erreurs se produisant dans ce calcul [Cau40a] [Cau40b].

1.1.2.1 Conditions d'Avizienis et addition à temps constant

Avizienis suggéra en 1961 l'utilisation de systèmes de numération à chiffres signés [Avi61]. Le principe consiste à coder les nombres en base β à l'aide de chiffres appartenant à l'ensemble symétrique $D_\rho = \{\overline{\rho}, \cdots, \rho\}$, où $\overline{\rho} = -\rho$, $\rho \leq \beta - 1$ et $2\rho + 1 \geq \beta$. Avizienis a démontré que si $2\rho + 1 = \beta$, chaque nombre possède une écriture unique.

[6]Le boulier chinois, dont les premières illustrations connues datent d'un livre daté de 1175, est un exemple de système redondant. Il est assez frappant de constater que dans de nombreuses échoppes étatiques chinoises, la caisse enregistreuse n'a pas encore remplacé le boulier.

Lorsque $2\rho + 1 > \beta$, le système de numération devient redondant.

Exemple 1.4. Le tableau 1.2 donne le codage de chaque nombre de l'ensemble $D = \{0, 1, \cdots, 9\}$ en base $\beta = 5$ à l'aide de chiffres appartenant à $D_\rho = \{\overline{\rho}, \cdots, \rho\}$, avec différents ρ. Nous pouvons constater par exemple que si $D_1 = \{\overline{1}, \cdots, 1\}$, la condition $2\rho + 1 \geq \beta$ n'est pas satisfaite et tous les nombres contenant les chiffres 2, 3, 7, 8 et 9 n'admettent aucune représentation. En revanche, si $D_2 = \{\overline{2}, \cdots, 2\}$, chaque nombre possède une écriture unique car la condition $2\rho+1 = \beta$ est vérifiée. Enfin, avec $D_3 = \{\overline{3}, \cdots, 3\}$, certains nombres possèdent maintenant plusieurs représentations : par exemple, le nombre 9 peut s'écrire 014 ($0\times5^2+1\times5^1+4\times5^0$), $02\overline{1}$ ou $1\overline{3}\overline{1}$.

$\boldsymbol{D = \{0, 1, \cdots, 9\}}$	$\boldsymbol{D_1 = \{\overline{1}, \cdots, 1\}}$	$\boldsymbol{D_2 = \{\overline{2}, \cdots, 2\}}$	$\boldsymbol{D_3 = \{\overline{3}, \cdots, 3\}}$
0	000	000	000
1	001	001	001
2	–	002	002, $01\overline{3}$
3	–	$01\overline{2}$	003, $01\overline{2}$
4	$01\overline{1}$	$01\overline{1}$	$01\overline{1}$
5	010	010	010
6	011	011	011
7	–	012	012, $02\overline{3}$, $1\overline{3}\overline{3}$
8	–	$02\overline{2}$	013, $02\overline{2}$, $1\overline{3}\overline{2}$
9	–	$02\overline{1}$	014, $02\overline{1}$, $1\overline{3}\overline{1}$

TAB. 1.2 – Codages de chaque nombre de l'ensemble $D = \{0, 1, \cdots, 9\}$ en base $\beta = 5$ à l'aide de chiffres appartenant à $D_\rho = \{\overline{\rho}, \cdots, \rho\}$, avec différents ρ.

Le principal intérêt des systèmes de représentation redondants est qu'il existe dans ces systèmes des algorithmes permettant d'effectuer des additions de façon totalement parallèle, c'est-à-dire sans propagation de retenues : ce sont des algorithmes d'addition dits « à temps constant ». Pour ce faire, d'après Avizienis [Avi61], les chiffres des opérandes doivent appartenir à un ensemble $D_{\rho'} = \{\overline{\rho'}, \cdots, \rho'\} \subset D_\rho$ tel que $2\rho' \geq \beta+1$ et $\rho' \leq \beta-1$.

Il faut noter que la notation d'Avizienis n'est valable que pour des valeurs de β supérieures ou égales à trois. Pour le cas particulier $\beta = 2$, deux représentations redondantes n'utilisant pas la notation d'Avizienis mais ayant néanmoins des propriétés intéressantes ont été mises en évidence : la représentation à retenues conservées et son homologue à retenues conservées signées.

1.1.2.2 Addition à retenues conservées signées (*Borrow-Save*)

La base $\beta = 2$ est bien adaptée aux circuits numériques. Guyot *et al.* ont proposé une représentation redondante en base deux à chiffres signés [GHM89] : la représentation à retenues conservées signées (*Borrow-Save*, BS). Dans cette représentation, $X = (x_{n-1} \cdots x_1 x_0)_{\mathrm{BS}}$ où chaque chiffre $x_i \in D_1 = \{\overline{1}, 0, 1\}$ est codé sur 2 bits x_i^+ et x_i^- tels que $x_i = x_i^+ - x_i^-$ (tableau 1.3). Ainsi, $X = \sum_{i=0}^{n-1} x_i 2^i = \sum_{i=0}^{n-1} (x_i^+ - x_i^-) 2^i$.

Chiffre	**Représentation BS (x_i^+, x_i^-)**
$\overline{1}$	(0,1)
0	(0,0) ou (1,1)
1	(1,0)

TAB. 1.3 – Représentation à retenues conservées signées des chiffres de $D_1 = \{\overline{1}, 0, 1\}$.

Cette représentation ne satisfait pas les hypothèses requises par Avizienis pour effectuer une addition en temps constant (cf. section 1.1.2.1), en particulier $2\rho' \not\geq \beta+1$. Guyot *et al.* ont toutefois réussi à proposer un algorithme d'addition à temps constant (Algorithme 1) destiné au codage BS : l'addition à retenues conservées signées (*Borrow-Save Addition*, BSA).

Algorithme 1 Addition à retenues conservées signées

Entrées : $A = (a_{n-1} \cdots a_1 a_0)_{\mathrm{BS}}$, $B = (b_{n-1} \cdots b_1 b_0)_{\mathrm{BS}}$.
Sorties : $S = \mathrm{BSA}[(A^+, A^-), (B^+, B^-)] = (s_n \cdots s_1 s_0)_{\mathrm{BS}}$.

1. $c_0^+ \leftarrow 0, \quad s_0^- \leftarrow 0$
2. **pour** $i = 0$ à $n-1$ **faire** ▷ *boucle parallèle*
3. $\quad 2c_{i+1}^+ - c_i^- \xleftarrow{PPM} a_i^+ + b_i^+ - a_i^-$
4. **fin pour**
5. **pour** $i = 0$ à $n-1$ **faire** ▷ *boucle parallèle*
6. $\quad 2s_{i+1}^- - s_i^+ \xleftarrow{PPM} b_i^- + c_i^- - c_i^+$
7. **fin pour**
8. $s_l^+ \leftarrow c_l^+$
9. **retourner** S

Il faut noter que les n passages dans chacune des boucles peuvent s'exécuter en parallèle. Les opérations dans la première boucle sont indépendantes les unes des autres, elles peuvent se faire en parallèle. Une fois cette boucle terminée, la seconde boucle se fait dans les mêmes conditions.

Le circuit découlant de l'algorithme 1 est représenté dans la figure 1.3. Cet additionneur utilise deux lignes de cellules dites « Plus, Plus, Moins » (PPM), et son temps de

calcul ne dépend pas de la taille des opérandes n : la somme est obtenue dans un délai de deux cellules PPM. La cellule PPM obéit à l'équation arithmétique $2r^+ - s^- = a^+ + b^+ - c^-$ et aux équations logiques :

$$\begin{cases} s = a^+ \oplus b^+ \oplus c^- \\ r = a^+ b^+ + a^+ \overline{c^-} + b^+ \overline{c^-} \end{cases}$$

Ainsi, si on compare les équations logiques de la cellule PPM à celles relatives à la cellule FA, on peut constater qu'elles sont très proches : la cellule PPM comporte seulement un inverseur supplémentaire (voir Figure 1.4).

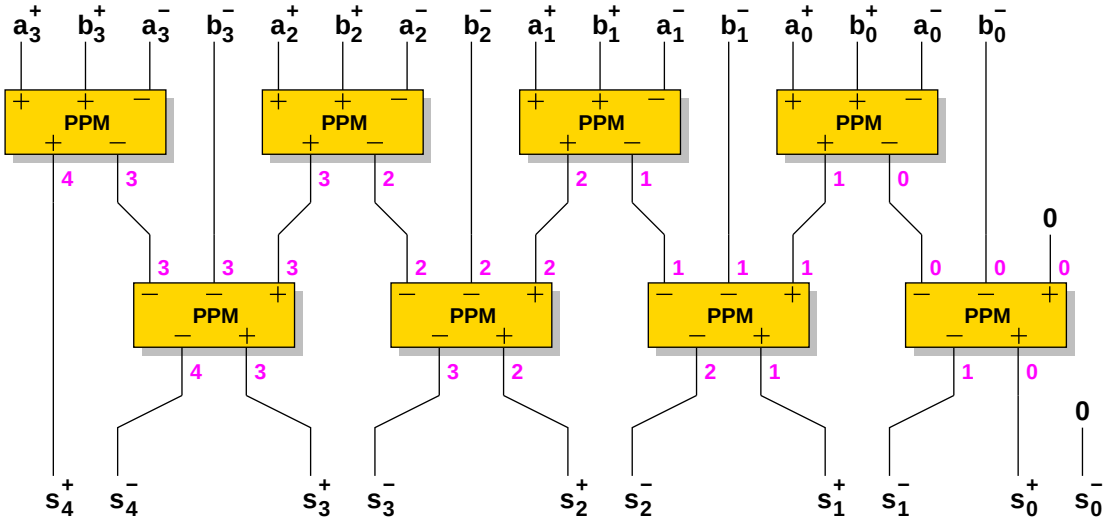

FIG. 1.3 – Addition à retenues conservées signées sur quatre chiffres.

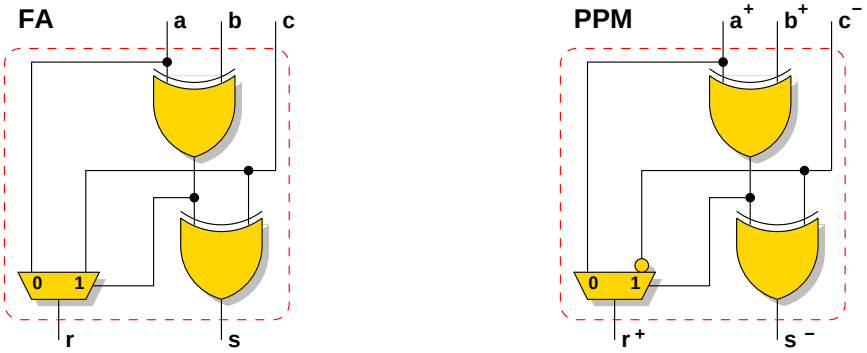

FIG. 1.4 – Comparaison de la cellule d'addition complète et de la cellule PPM.

1.1.2.3 Addition de deux nombres à retenues conservées (*Carry-Save*) avec résultat dans la même représentation

Dans la représentation à retenues conservées (*Carry-Save*, CS), on représente le nombre X en base 2 avec les chiffres $x_i \in \{0,1,2\}$ sur deux bits tels que $x_i = x_{i,c} + x_{i,s}$ où $x_{i,c}, x_{i,s} \in \{0,1\}$ (cf. tableau 1.4).

Chiffre	Représentation CS $(x_{i,c}, x_{i,s})$
0	(0,0)
1	(1,0) ou (0,1)
2	(1,1)

TAB. 1.4 – Représentation à retenues conservées des chiffres de $\{0,1,2\}$.

La somme s'exprime dès lors par : $X = \sum_{i=0}^{n-1} x_i 2^i = \sum_{i=0}^{n-1} (x_{i,c} + x_{i,s}) 2^i$. Elle peut être obtenue dans le délai de deux cellules FA à l'aide de l'architecture représentée en Figure 1.5.

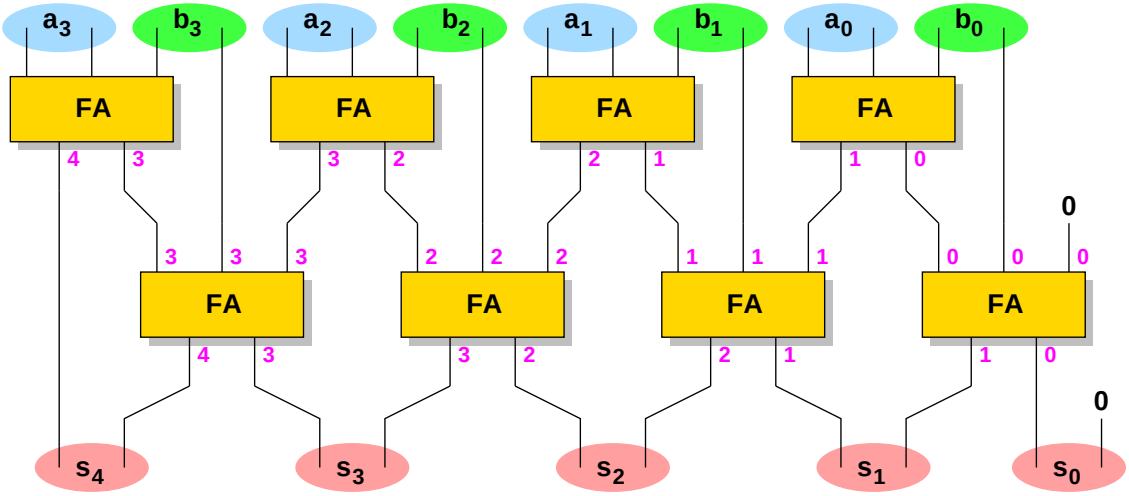

FIG. 1.5 – Addition de deux nombres à retenues conservées avec résultat dans la même représentation.

1.1.2.4 Addition de plusieurs nombres non-redondants avec résultat à retenues conservées

Soient A, B et C trois nombres en représentation non-redondante. On peut obtenir leur somme S en représentation CS avec l'opérateur représenté en figure 1.6.

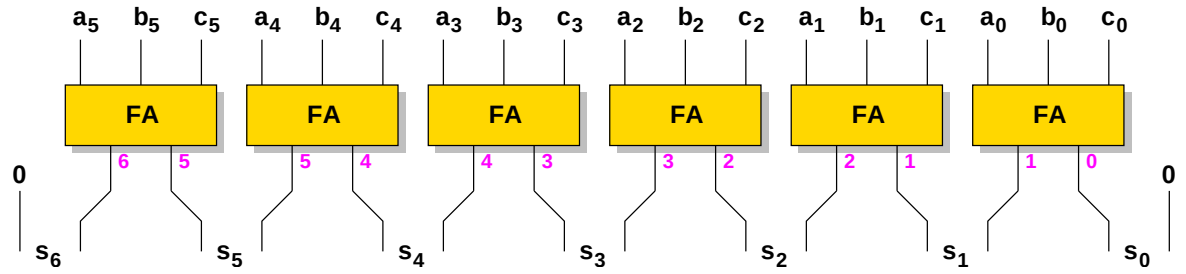

FIG. 1.6 – Addition de trois nombres codés en numération simple de position avec résultat à retenues conservées.

Exemple 1.5. L'addition de $A = 40 = (101000)$, $B = 25 = (011001)$ et $C = 20 = (010100)$ codés en numération simple de position donnera $s_6 = (0,0)$, $s_5 = (1,1)$, $s_4 = (0,1)$, $s_3 = (0,0)$, $s_2 = (1,0)$, $s_1 = (0,0)$, $s_0 = (1,0)$, soit $S = (0210101)_{\text{CS}}$.

Cette approche peut être généralisée : si l'on veut additionner au plus $n(h)$ entrées non-redondantes, on pourra utiliser un arbre constitué de h lignes de cellules FA. $n(h)$ obéit ainsi à la relation de récurrence : $n(h) = \lfloor 3n(h-1)/2 \rfloor$ avec $n(0) = 2$.

Cet opérateur est particulier car ses sorties (respectivement ses entrées) sont en notation (non-)redondante. Ainsi, il est effectué un **changement de représentation**.

1.1.2.5 Inconvénients de ces représentations redondantes

Ces représentations redondantes ont beaucoup d'avantages, mais leur utilisation comporte cependant quelques inconvénients.

1. La conversion pour passer d'une représentation non-redondante à une représentation redondante est « gratuite », mais le surcoût est plus important dans le sens inverse. Par exemple, pour convertir un nombre X initialement codé en numération simple de position en représentation BS (c'est-à-dire sous la forme $X = X^+ - X^-$), il suffit d'effectuer $X^+ = X$ et $X^- = 0$. En revanche, dans le sens inverse, il faudra effectuer la soustraction $X^+ - X^-$ pour retrouver la valeur de X en non-redondant. Par conséquent, utiliser des représentations redondantes de type BS ou CS n'a un sens que si l'on **effectue plusieurs additions successivement**.
2. **La comparaison de deux nombres codés en représentation redondante est une opération plus complexe à effectuer que dans le cas où ceux-ci sont codés en représentation non-redondante**. En effet, pour comparer deux nombres A et B, une méthode classique consiste à effectuer la soustraction $(A - B)$, puis à trouver le signe du résultat de cette soustraction. Or, même si la soustraction peut être calculée rapidement en représentation BS en calculant $\text{BSA}[(A^+, A^-), (B^-, B^+)]$ à l'aide de l'algorithme 1, la recherche du signe de $(A-B)$ est une opération difficile à implanter matériellement, cette dernière consistant à

chercher le chiffre de poids le plus fort non-nul de $(A - B)$.

3. La redondance entraîne un surcoût en terme de mémoire utilisée, car **il faut deux fois plus de mémoire pour stocker un nombre redondant BS ou CS**.

Cependant, il faut être convaincu des performances intrinsèques des représentations redondantes. Notamment, le « produit surface/temps », critère souvent utilisé pour évaluer les performances des circuits, est très avantageux pour les additionneurs redondants : vu que leur délai est constant et que leur surface est proportionnelle à n, le produit surface/temps de ces additionneurs est donc proportionnel à n. Ce produit est beaucoup plus faible que ceux relatifs à tous les autres types d'additionneurs (dits « à selection de retenue », « à retenue bondissante », « à retenues anticipées » ou « préfixes »). Par exemple, le produit surface/temps de l'additionneur à retenue bondissante est proportionnel à $n\sqrt{n}$.

Ainsi, malgré leurs défauts, les représentations redondantes BS et CS doivent être considérées par les concepteurs de circuits comme une alternative performante face aux représentations non-redondantes.

Les représentations redondantes permettent donc d'accélérer les additions et dans certains cas, comme nous allons le voir dans la section 1.1.3, elles peuvent également permettre d'accélérer les multiplications.

1.1.3 Représentations redondantes pour une multiplication rapide

La multiplication de deux nombres A et B (A et B étant respectivement le « multiplicande » et le « multiplicateur ») est une opération arithmétique coûteuse du fait du calcul de nombreux produits partiels (PPs) et de la manipulation ultérieure de ces derniers pour calculer le produit $S = A \times B$.

Une optimisation possible consiste à réécrire le multiplicateur dans une grande base exploitant une notation redondante afin de réduire le nombre de PPs à calculer : c'est l'objectif du recodage de Booth.

1.1.3.1 Recodage de Booth

Booth proposa en 1951 un recodage destiné à diminuer le nombre de PPs à calculer lors d'une multiplication [Boo51]. Son but est d'augmenter le nombre de zéros (ou de diminuer le nombre de non-zéros) dans le multiplicateur en le recodant en base 2 avec l'ensemble de chiffres signés $D_1 = \{\overline{1}, 0, 1\}$. Il utilise l'identité :

$$2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + \cdots + 2^i. \tag{1.2}$$

Le tableau 1.5 résume le recodage de Booth d'un multiplicateur $B = (b_n b_{n-1} \cdots b_1 b_0 b_{-1})_2$, avec $b_n = 0$ et $b_{-1} = 0$.

Chaque chiffre $\tilde{b_i}$ du multiplicateur converti $\tilde{B}$ est déterminé par : $\tilde{b_i} = b_{i-1} - b_i$.

b_i	b_{i-1}	$\tilde{b_i}$	Signification	Opération
0	0	0	chaîne de 0	$+0$
0	1	1	début chaîne de 1	$+B$
1	0	$\overline{1}$	fin chaîne de 1	$-B$
1	1	0	milieu chaîne de 1	$+0$

TAB. 1.5 – Table de vérité pour le recodage de Booth.

Exemple 1.6. Le nombre $B = 60 = (111100)_2$ se recode grâce au tableau 1.5 en $\tilde{B} = 1000\overline{1}00$. Ainsi, la multiplication dans la base $\beta = 2$ d'un nombre A par le multiplicateur recodé $\tilde{B}$ va donner :

$$\begin{aligned} A \times \tilde{B} &= A \times (2^6 - 2^2) \\ &= A \times 2^6 - A \times 2^2 \end{aligned}$$

Ainsi, seulement deux PPs non-nuls interviendront lors de la multiplication.

Cette méthode permet de transformer les chaînes de 1 par une écriture avec plus de zéros. Malheureusement, cette approche comporte deux inconvénients. Tout d'abord, la chaîne recodée peut ne pas avoir un nombre maximal de zéros. Ensuite, dans certains cas, on peut faire apparaître plus de 1 dans la chaîne recodée que dans la chaîne initiale !

1.1.3.2 Recodage de Booth modifié

Reitwiesner propose un recodage de Booth modifié [Rei60]. Le but est toujours de recoder en base 2 le multiplicateur avec l'ensemble de chiffres signés $\{\overline{1}, 0, 1\}$. En plus d'utiliser l'identité 1.2, on utilise également la relation $-2^{i+1} + 2^i = 2^i.(-2 + 1) = -2^i$. Ainsi, après recodage du multiplicateur, il n'y a plus de non-zéros adjacents. De plus, comparée à toutes les représentations signées, celle-ci donne le plus petit nombre de non-zéros dans le multiplicateur. Enfin, elle peut être implantée efficacement sous la forme d'une table (*Look-Up Table*, LUT).

1.1.3.3 Recodages en plus grandes bases

Le principe exploité pour le développement du recodage de Booth en base 2 peut également s'appliquer à de plus grandes bases. On peut par exemple recoder le multiplicateur en base 4 avec l'ensemble de chiffres $\{-2, -1, 0, 1, 2\}$: ce recodage est appelé « Booth 2 ». L'idée sous-jacente est de ne pas recoder les 1 isolés mais seulement les chaînes de 1 du multiplicateur. Ce recodage traite trois bits du multiplicateur à la fois. Plus précisément, il est calculé

$$b_i + b_{i-1} - 2b_{i+1} = (\tilde{b_i}\tilde{b_{i-1}})_2.$$

b_i	b_{i-1}	b_{i-2}	$\tilde{b_i}$	$\tilde{b_{i-1}}$	**Signification**	**Opération**
0	0	0	0	0	chaîne de 0	$+0$
0	0	1	0	1	fin chaîne de 1	$+B$
0	1	0	0	1	1 isolé	$+B$
0	1	1	1	0	fin chaîne de 1	$+2B$
1	0	0	$\overline{1}$	0	début chaîne de 1	$-2B$
1	0	1	0	$\overline{1}$	0 isolé	$-B$
1	1	0	0	$\overline{1}$	début chaîne de 1	$-B$
1	1	1	0	0	milieu chaîne de 1	$+0$

TAB. 1.6 – Table de vérité pour le recodage Booth 2.

Le recodage Booth 2 d'un multiplicateur $B = (b_{n-1} \cdots b_1 b_0 b_{-1})_2$ (avec $b_{-1} = 0$) est résumé dans le tableau 1.6. L'avantage de cette démarche est que le calcul du produit de deux nombres de n bits peut maintenant s'effectuer en $\lfloor n/2 \rfloor + 1$ additions/décalages au plus.

On peut généraliser ce raisonnement à de plus grandes bases : on peut recoder le multiplicateur en base 2^k avec l'ensemble de chiffres $\{-2^{k-1}, -2^{k-1}+1, \cdots, 0, \cdots, 2^{k-1}-1, 2^{k-1}\}$: c'est le recodage « Booth k ». Au maximum $\lfloor n/k \rfloor + 1$ additions/décalages seront nécessaires au calcul de la multiplication.

1.1.3.4 Discussion sur le réel gain du recodage de Booth

D'un point de vue théorique, l'utilisation du recodage de Booth amène deux avantages. Tout d'abord, le nombre de PPs à calculer diminue. Ensuite, le nombre d'additions de PPs à effectuer pour l'obtention du produit diminue également.

D'un point de vue pratique, la multiplication de Booth entraîne l'implantation matérielle de trois dispositifs. Il faut tout d'abord **des étages générant les différents PPs**. Il faut ensuite **un arbre de réduction** (par exemple constitué de cellules FA, cf. section 1.1.1.2) pour calculer la somme des PPs en CS (cf. section 1.1.2.3). Enfin, **un additionneur rapide** est nécessaire pour calculer l'addition finale des deux composantes CS de la somme des PPs.

Un recodage Booth 2 voit généralement le surcoût engendré par l'implantation de l'étage de génération des PPs diminuer le gain apporté par la diminution du nombre de PPs à calculer. Les recodages Booth 3 et/ou utilisant des bases supérieures sont rarement utilisés en pratique car les étages de génération des PPs deviennent beaucoup trop complexes : ils sont constitués de nombreuses portes, dont certaines avec 3 ou 4 entrées.

1.1.4 Bilan

Il a pu être constaté dans cette section que l'utilisation des représentations non-redondantes comporte quelques inconvénients, notamment le temps nécessaire au calcul de l'addition, opération fondamentale en arithmétique, qui est de l'ordre de $\log_2(n)$. Ainsi, parce qu'une UA pour courbes elliptiques doit manipuler des opérandes codés sur plusieurs centaines de chiffres (cf. chapitre 2), il n'est pas souhaitable d'implanter un chemin de données utilisant une représentation binaire.

En revanche, l'utilisation de représentations redondantes telles que le BS et CS permet une addition rapide et à temps constant. Elles doivent donc être considérées comme des alternatives crédibles aux représentations non-redondantes dans le cadre de l'implantation d'UA hautes performances. Il faut maintenant s'assurer que les représentations BS et CS sont bien adaptées aux calculs dans les corps finis : ce sera l'un des objets de la section 1.2 (plus particulièrement la section 1.2.1).

Enfin, il est également possible de calculer des multiplications rapidement en réécrivant le multiplicateur dans une grande base en utilisant une représentation redondante : c'est le principe du recodage de Booth. Théoriquement, l'utilisation du recodage de Booth amène un avantage intéressant : il permet de diminuer le nombre de PPs à calculer. Cependant, lorsqu'on implémente matériellement cette méthode de multiplication, on s'aperçoit que l'étage de génération des PPs peut être complexe, ce qui peut limiter son utilisation.

1.2 Les corps finis

Dans la section 1.1, il a été montré que le choix du mode de représentation des nombres traité par un cryptosystème est crucial en vue d'obtenir de bonnes performances. En particulier, l'utilisation de représentations redondantes peut permettre de diminuer le temps de calcul de l'addition et de la multiplication.

La plupart des cryptosystèmes effectuent des calculs sur une structure algébrique appelée « **corps fini premier** » et notée $\mathbb{F}_p$, où p est un nombre premier. Le corps fini premier $\mathbb{F}_p$ comprend l'ensemble des entiers

$$\{0, 1, 2, \cdots, p-1\}$$

sur lequel les opérations arithmétiques (additions/soustractions, multiplications et inversions) sont calculées modulo p.

Exemple 1.7. Les éléments de $\mathbb{F}_7$ sont $\{0, 1, 2, 3, 4, 5, 6\}$. Donnons quelques exemples d'opérations sur $\mathbb{F}_7$:
- $6 + 4 = 3$,
- $3 \times 5 = 1$,
- $2^{-1} = 4$.

Dans cette section, il sera décrit deux façons de calculer efficacement des opérations arithmétiques (l'addition, la multiplication, l'inversion) dans un corps fini : bien choisir

ce dernier, et/ou utiliser des algorithmes efficaces.

1.2.1 Opérations arithmétiques modulaires

Cette section détaille des méthodes efficaces pour calculer l'addition, la multiplication et l'inversion modulo p.

1.2.1.1 Addition modulaire

Afin de calculer le résultat d'une addition modulaire, trois méthodes ont été mises en évidence : une méthode « classique », une méthode proposée par Omura [Omu90] et une autre proposée par Takagi *et al.* [TY92] utilisant la représentation BS présentée en section 1.1.2.2.

Méthode classique d'addition modulaire. L'algorithme 2 décrit une méthode classique d'addition modulaire.

Algorithme 2 Méthode classique d'addition modulaire

Entrées : p, $A = (a_{n-1} \cdots a_1 a_0) < p$, $B = (b_{n-1} \cdots b_1 b_0) < p$.
Sorties : $S = A + B \pmod p$.

1. $S \leftarrow A + B$
2. $S' \leftarrow S - p$
3. **si** $S' < 0$ **alors**
4. **retourner** S
5. **sinon**
6. **retourner** S'
7. **finsi**

Vérification.

- Si $S' = A + B - p < 0$, alors $A + B < p$, donc il faut retourner S.
- Si $S' = A + B - p \geq 0$, alors $A + B \geq p$, donc il faut retourner S'.

Addition modulaire d'Omura. Omura optimise l'addition modulaire en évitant d'avoir à faire une comparaison et en calculant uniquement des additions [Omu90]. Il propose l'algorithme 3 qui permet d'accélérer les calculs en remplaçant la comparaison $S' = A+B-p < 0$ effectuée dans l'algorithme 2 par une simple analyse de la retenue sortante de $S" = A + B + m$, où $m = 2^n - p$ est un facteur de correction précalculé.

Vérification.

- Si $A + B < p$, alors $A + B + m < p + m = 2^n$, et $P = A + B < 2^n - m = p$.
- Si $A+B \geq p$, alors $A+B+m \geq p+m = 2^n$, et $P = A+B+m-2^n = A+B-p < p$.

Algorithme 3 Addition modulaire d'Omura

Entrées : p, $A = (a_{n-1} \cdots a_1 a_0) < p$, $B = (b_{n-1} \cdots b_1 b_0) < p$, $m = 2^n - p$ (précalculé).
Sorties : $S = A + B \pmod p$.

1. $S \leftarrow A + B$
2. $S" \leftarrow S + m$
3. **si** $S" \geq 2^n$ **alors**
4. **retourner** $S \leftarrow S" \pmod{2^n}$
5. **sinon**
6. **retourner** S
7. **finsi**

Exemple 1.8. Soit $p = 39, n = 6$ (donc $m = 2^6 - 39 = 25$), $A = 23 = (010111)_2$, et $B = 26 = (011010)_2$.
À la fin de l'étape 1, on a : $S = A + B = 49 = (110001)_2$.
À la fin de l'étape 2, on a : $S" = S + m = 74 = (\mathbf{1}001010)_2$.
La condition de l'étape 3 est bien vérifiée, il faut donc réduire $S"$ modulo 2^n : $S = 10 = (001010)_2$.

Il est à noter que le test de l'étape 3 de l'algorithme 3 ne coûte rien : il correspond à l'observation de la retenue sortante de $S"$.

Algorithme d'addition modulaire de Takagi *et al.*. La représentation BS (cf. section 1.1.2.2) permet d'effectuer efficacement l'addition modulaire de deux opérandes A et B [TY92]. Le calcul de $A + B \pmod p$ peut être décomposé en deux étapes.

La première étape consiste à effectuer l'addition des opérandes $A = (a_n a_{n-1} \cdots a_0)_{BS}$ et $B = (b_n b_{n-1} \cdots b_0)_{BS}$, avec $-p < A < p$, $-p < B < p$ et $2^{n-1} \leq p < 2^n$. Le résultat $T = (t_{n+1} t_n \cdots t_0)_{BS}$ satisfait la relation $T \equiv A + B \pmod p$ et $-2p < T < 2p$.

La seconde étape est une correction dans le but d'obtenir un nombre $S = (s_n s_{n-1} \cdots s_0)_{BS}$ qui satisfait la relation $S \equiv T \pmod p$, et $-p < S < p$.

Tout d'abord, la valeur $tv = 4t_{n+1} + 2t_n + t_{n-1}$ qui utilise les trois chiffres de poids fort de T doit être calculée. Ensuite, selon la valeur de tv, trois cas se présentent :

- si $tv < 0$, alors on a $-2p < T < 0$, donc il faut calculer $S = T + p$,
- si $tv > 0$, alors on a $0 < T < 2p$, donc il faut calculer $S = T - p$,
- si $tv = 0$, alors on a $-2^{n-1} < T < 2^{n-1}$, soit encore $-p < T < p$, donc il faut calculer $S = T + 0$.

Pour comprendre le dernier cas de figure, il faut utiliser les relations $2^{n-1} \leq p < 2^n$ et $-2^{n-1} < T < 2^{n-1}$ afin de construire l'inégalité

$$-2^n < -p \leq -2^{n-1} < T < 2^{n-1} \leq p < 2^n. \tag{1.3}$$

Ainsi, on peut tirer de l'inégalité 1.3 que $-p < T < p$. Par conséquent, il n'est pas

nécessaire d'additionner $\pm p$ à T pour obtenir $-p < S < p$. L'algorithme 4 détaille toute cette procédure.

Algorithme 4 Algorithme d'addition modulaire de Takagi *et al.*

Entrées : $2^{n-1} \leq p < 2^n$, $-p < A = (a_n a_{n-1} \cdots a_0)_{BS} < p$, $-p < B = (b_n b_{n-1} \cdots b_0)_{BS} < p$.
Sorties : $S = A + B \pmod{p}$, avec $S = (s_n s_{n-1} \cdots s_0)_{BS}$.

1. $(T^+, T^-) \leftarrow \text{BSA}[(A^+, A^-), (B^+, B^-)]$
2. **si** $tv = 4t_{n+1} + 2t_n + t_{n-1} < 0$ **alors**
3. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (p, 0)]$
4. **sinon si** $tv > 0$ **alors**
5. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (0, p)]$
6. **sinon si** $tv = 0$ **alors**
7. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (0, 0)]$
8. **finsi**
9. **retourner** S

Étude comparative. Concernant la méthode classique d'addition modulaire (cf. Algorithme 2), un constat s'impose : elle est bien adaptée si le signe de la variable intermédiaire S' peut être rapidement détecté. Deux conséquences peuvent être tirées de ce constat. La première est que cette méthode n'est donc pas adaptée si la variable intermédiaire S' est exprimé en représentation redondante, car il est difficile de connaître son signe. La seconde conséquence est *a priori* que S' doit être exprimé en représentation binaire, par exemple en complément à 2 (cf. section 1.1.1.1). Malheureusement, utiliser cette représentation implique un temps nécessaire pour calculer l'addition de l'ordre de $\log_2(n)$ (cf. section 1.1.2.5).

L'addition modulaire d'Oruma (cf. Algorithme 3) effectue seulement deux additions de n bits. Malheureusement, cet algorithme conduit à l'implantation d'une comparaison, opération qui interdit l'utilisation de représentations redondantes, et qui, par voie de conséquence, conduit à une architecture ayant un temps de calcul élevé.

L'avantage principal de l'algorithme d'addition modulaire de Takagi (cf. Algorithme 4) est sa très grande rapidité : en effet, cet algorithme effectue deux BSAs, ce qui conduit à un délai de 4 cellules PPM (cf. section 1.1.2.2).

Ainsi, au vu de cet état de l'art et dans le cadre de la conception de notre UA, l'algorithme de Takagi sera donc choisi pour calculer des additions modulaires. Il faudra cependant veiller à ce que notre UA exécute tous ses calculs en représentation redondante BS car si l'entrée dans cette représentation ne coûte rien, la sortie correspond à une soustraction classique. Utiliser l'algorithme de Takagi n'a donc de l'intérêt que lorsque de nombreuses additions modulaires sont calculées successivement (cf. section 1.1.2.5).

1.2.1.2 Multiplication modulaire

La multiplication modulaire est une opération très utilisée dans le cadre de la cryptographie à clé publique, notamment pour le RSA [RSA78] et les courbes elliptiques (cf. chapitre 2). La multiplication modulaire consiste à calculer $S = A.B \pmod{p}$, A et B étant deux entiers strictement inférieurs à p.

Il existe deux méthodes de multiplication modulaire : soit les opérations de multiplication et de réduction sont séparées, soit ces deux opérations sont entremêlées (les méthodes de ce type sont dites « à réduction intégrée »).

Multiplication et réduction calculées séparément. Le calcul de la multiplication modulaire $S = A.B \pmod{p}$ séparant les phases de multiplication et de réduction utilise la procédure suivante :

- il est d'abord calculé la multiplication $S' = A.B$, avec $S' = (s'_{2n-1} \cdots s'_0)_\beta$,
- il est finalement effectué la réduction $S = S' \pmod{p}$, avec $S = (s_{n-1} \cdots s_0)_\beta$.

Dans les sections suivantes, il est détaillé l'état de l'art des méthodes de multiplication et de réduction modulaire.

Multiplication. Une méthode de multiplication a déjà été décrite en section 1.1.3 : celle utilisant le recodage de Booth.

Une autre méthode existe : elle consiste à diviser la multiplication initiale en plusieurs multiplications traitant des opérandes de taille inférieure. Le premier à avoir mis en œuvre cette méthode en 1960 fût Karatsuba [KO63], un des étudiants de Kolomogorov, ce dernier ayant au milieu des années 1950 conjecturé que la borne inférieure de la complexité du calcul de la multiplication était de l'ordre de $\Omega(n^2)$, et ce, quelque soit la méthode utilisée. Karatsuba, de son côté, a découvert une méthode de multiplication récursive ayant une complexité en $O(n^{log_2 3}) \approx O(n^{1,585})$.

Sa méthode peut être divisée en trois phases :

- considérer les deux opérandes A et B et le produit S' comme des polynômes : $A(x) = \alpha_1 x + \alpha_0$, $B(x) = \beta_1 x + \beta_0$, $S'(x) = A(x) \times B(x) = \varphi_2 x^2 + \varphi_1 x + \varphi_0$,
- évaluer S' en un nombre suffisant de points (un de ces points pouvant être l'infini),
- interpoler pour retrouver φ_0, φ_1 et φ_2.

D'autres méthodes de multiplication basées sur l'utilisation d'interpolations, comme celle appelée méthode de « Toom-Cook » [Too63] ou celle utilisant des transformées rapides de Fourier [SS71] sont asymptotiquement plus efficaces, mais le gain apporté par ces méthodes n'est vraiment significatif que pour de très grandes valeurs de n, valeurs qui ne correspondent pas à celles utilisées dans le cadre de l'ECC[7].

Réduction. La réduction modulaire est une opération cruciale en arithmétique modulaire. Elle consiste à évaluer le reste de la division entière S'/p, où $S' = (s'_{2n-1} \cdots s'_0)_\beta =$

[7] Ce constat s'appuie sur les tests exécutés par Granlund sur la bibliothèque de fonctions mathématiques spécialisée dans le calcul des grands nombres GMP (GNU Multiprecision Package) : http ://gmplib.org/.

$A.B$, A et B étant deux entiers strictement inférieurs à p. Soit $0 \leq S' = A.B < p^2$, alors il existe $q = \lfloor S'/p \rfloor \geq 0$ ($\lfloor x \rfloor$ étant la partie entière inférieure de x) et $0 \leq r < p$ tels que

$$S' = qp + r.$$

La valeur r, appelée le « reste », est notée $r = S' \pmod{p}$. Ainsi, « mod » est un opérateur qui calcule l'unique entier $0 \leq r < p$; la relation de congruence étant notée $S' \equiv r \pmod{p}$.

Afin que la réduction modulaire soit une opération efficace, il faut éviter d'effectuer des divisions. C'est le cas des algorithmes de réduction de Barrett [Bar87] et de Montgomery [Mon85] présentés dans cette section.

Réduction de Barrett. Le but de l'algorithme de réduction de Barrett (Algorithme 5) est de calculer le reste $r = S' - qp$ grâce à une approximation du quotient $\tilde{q}$, en utilisant le fait que le quotient

$$q = \lfloor S'/p \rfloor = \left\lfloor \frac{\frac{S'}{\beta^{n-1}} \frac{\beta^{2n}}{p}}{\beta^{n+1}} \right\rfloor$$

peut être évalué par la quantité

$$\tilde{q} = \left\lfloor \left\lfloor \frac{S'}{\beta^{n-1}} \right\rfloor \frac{\mu}{\beta^{n+1}} \right\rfloor,$$

avec $\mu = \left\lfloor \frac{\beta^{2n}}{p} \right\rfloor$.

Le calcul de $\tilde{q}$ est effectué à l'étape 1 de l'algorithme 5. Une question peut être posée : $\tilde{q}$ est-elle une « bonne » approximation de q ? À ce propos, Barrett fait remarquer que $\tilde{q}$ obéit à la relation

$$q - 2 \leq \tilde{q} \leq q.$$

Surtout, il peut être montré que $\tilde{q} = q$ dans 90% des cas, tandis que $\tilde{q} = q - 2$ dans 1% des cas : $\tilde{q}$ est donc une bonne approximation de q.

Il faut également noter que les divisions par β^{n-1} et β^{n+1} nécessaires au calcul de $\tilde{q}$ sont de simples décalages à droite dans la base β : elles sont donc calculables facilement.

La réduction de Barrett permet donc de calculer $r = S' \pmod{p}$ sans effectuer de division, sauf dans le calcul préliminaire de la quantité $\mu = \left\lfloor \frac{\beta^{2n}}{p} \right\rfloor$. Vu que la valeur de μ doit être recalculée à chaque fois que le modulo p change, la réduction de Barrett (Algorithme 5) n'est intéressante que lorsque plusieurs réductions modulo p, avec p constant, doivent être calculées successivement.

À la fin de l'étape 2, $\tilde{r} \equiv S' - \tilde{q}p \pmod{\beta^{n+1}}$, avec $|\tilde{r}| < \beta^{n+1}$, et à la fin de l'étape 5, on obtient $0 \leq \tilde{r} < \beta^{n+1}$.

Algorithme 5 Réduction de Barrett

Entrées : $p = (p_{n-1} \cdots p_0)_\beta$, $S' = (s'_{2n-1} \cdots s'_0)_\beta < p^2$, $\mu = \left\lfloor \frac{\beta^{2n}}{p} \right\rfloor$ (précalculé).

Sorties : $r = (r_{n-1} \cdots r_0)_\beta$ tel que $S' \equiv r \pmod{p}$.

1. $\tilde{q} \leftarrow \left\lfloor \left\lfloor \frac{S'}{\beta^{n-1}} \right\rfloor \frac{\mu}{\beta^{n+1}} \right\rfloor$
2. $\tilde{r} \leftarrow S' \pmod{\beta^{n+1}} - (\tilde{q}p) \pmod{\beta^{n+1}}$
3. **si** $\tilde{r} < 0$ **alors**
4. $\quad \tilde{r} \leftarrow \tilde{r} + \beta^{n+1}$
5. **finsi**
6. **tantque** $\tilde{r} \geq p$ **faire**
7. $\quad \tilde{r} \leftarrow \tilde{r} - p$
8. **fin tantque**
9. **retourner** $\tilde{r}$

Enfin, dans le cas où $\beta > 3$, il peut être démontré que la boucle « tant que » de l'étape 6 ne sera jamais effectuée plus de 2 fois.

Réduction de Montgomery. En 1985, grâce à Montgomery, on franchit un cap dans la réduction modulaire [Mon85]. Pour calculer $r = S' \pmod{p}$, il utilise la propriété suivante : soient deux entiers p et m tels que leur Plus Grand Commun Diviseur (PGCD) est égal à 1. Soit un entier précalculé

$$p' = -p^{-1} \pmod{m}, \tag{1.4}$$

et S' l'entier à réduire (initialement, $0 \leq S' < pm$). Si

$$q = S'p' \pmod{m}, \tag{1.5}$$

alors $(S' + qp)/m$ est un entier, et $(S' + qp)/m \equiv S'm^{-1} \pmod{p}$.

Vérification. Sachant que $S' + qp \equiv S' \pmod{p}$, nous obtenons $(S' + qp)m^{-1} \equiv S'm^{-1} \pmod{p}$. Pour constater que $(S' + qp)m^{-1}$ est un entier, utilisons successivement la relation 1.5 et la relation 1.4 pour obtenir $q = S'p' + km$ et $pp' = -1 + lm$, k et l étant des entiers. Il s'en suit que :

$$\begin{aligned} (S' + qp)/m &= [S' + (S'p' + km)p]/m \\ &= [S' + S'pp' + kmp]/m \\ &= [S' + S'(-1 + lm) + kmp]/m \\ &= [lmS' + kmp]/m \\ &= lS' + kp. \end{aligned}$$

Par conséquent, $(S' + qp)m^{-1}$ est bien un entier.

La réduction de Montgomery est détaillée dans l'algorithme 6. À l'étape 1, on calcule la valeur de q à l'aide de notre entier précalculé p', puis à l'étape 2, on calcule $r = (S'+qp)/m$.

Or, r doit être réduit modulo p, car $S' < pm$ (c'est une condition initiale), $q < m$ (cf. relation 1.5), et :

$$(S' + qp)/m < [pm + pm]/m = 2p$$

Par conséquent, $((S' + qp)/m) = S'm^{-1} \pmod{p}$, ou $((S' + qp)/m) = S'm^{-1} \pmod{p} + p$. Une soustraction conditionnelle doit donc être implantée à l'étape 4 de l'algorithme 6 afin d'obtenir exactement $r = S'm^{-1} \pmod{p}$.

Si tous les entiers sont représentés dans la base β, et si le paramètre m est choisi comme étant une puissance de β, alors $S'm^{-1} \pmod{p}$ peut être calculé à l'aide de deux multiplications ($S'p'$ et qp), et la division présente à l'étape 2 va se résumer à de simples décalages à droite dans la base β.

Algorithme 6 Réduction de Montgomery

Entrées : $p = (p_{n-1} \cdots p_0)_\beta$, $S' = (s'_{2n-1} \cdots s'_0)_\beta < p^2$, avec $\text{pgcd}(p,\ m) = 1$, $p' = (-p^{-1}) \pmod{m}$ (précalculé).
Sorties : $r = S'm^{-1} \pmod{p}$.

1. $q \leftarrow S'p' \pmod{m}$
2. $r \leftarrow (S' + qp)/m$
3. **si** $r \geqslant p$ **alors**
4. $\quad r \leftarrow r - p$
5. **finsi**
6. **retourner** r

Étude comparative. Dans le cadre de la conception matérielle de ces deux méthodes, l'algorithme de Montgomery (Algorithme 6) paraît plus intéressant que l'algorithme de Barrett (Algorithme 5). Tout d'abord, l'algorithme de Barrett amène au calcul de deux comparaisons, contre une seule pour l'algorithme de Montgomery. Ensuite, si le modulo p change entre deux applications, l'algorithme de Barrett amène plus de contraintes matérielles. En effet, il faudra mettre en œuvre une division pour la méthode de Barrett (pour calculer μ), ce qui constitue une opération supplémentaire à implanter, car elle n'est pas présente dans l'algorithme initial. En revanche, concernant l'algorithme de Montgomery, il faudra calculer une inversion modulaire (pour calculer p'), opération qui peut être implantée à l'aide, par exemple, d'une série de multiplications modulaires (cf. section 1.2.1.3). Or, la multiplication modulaire est déjà implantée pour calculer q (étape 1 de l'algorithme 6). Enfin et surtout, l'implantation de la fonction « valeur entière inférieure » (notée $\lfloor\rfloor$) présente dans le calcul de l'algorithme de Barrett est difficile à implanter matériellement.

Cependant, il faut noter que l'algorithme de Montgomery est certes plus rapide mais il nécessite l'utilisation d'une représentation particulière. Il paraît donc judicieux de ne conseiller l'implantation de l'algorithme de Barrett que dans le cadre d'une simple réduction et de proposer l'algorithme de Montgomery, plus rapide, dans le cadre de calculs plus nombreux. Une étude plus détaillée sur ces deux algorithmes est présentée dans [BGV94].

Multiplication à réduction intégrée. Dans le cas où la multiplication modulaire est calculée en croisant les opérations de multiplication et de réduction, trois méthodes

peuvent être utilisées : la multiplication par additions-décalages, la multiplication de Montgomery, ou la combinaison de plusieurs méthodes de réduction.

Utilisation de la multiplication par additions-décalages. Afin de croiser les opérations de multiplication et de réduction, la méthode dite de la « multiplication par additions-décalages » peut être utilisée afin de pouvoir intercaler des opérations de réduction modulaire. Le produit de deux nombres A et B peut en effet s'écrire :

$$\begin{aligned} A.B &\equiv A.\sum_{i=0}^{n-1} b_i\beta^i \pmod p \\ &\equiv \sum_{i=0}^{n-1}(A.b_i)\beta^i \pmod p \\ &\equiv ((\cdots(A.b_{n-1} \pmod p)\,\beta + \cdots + A.b_1 \pmod p)\,\beta + A.b_0) \pmod p \end{aligned}$$

Cette formulation mène à l'algorithme 7 : à chaque pas i, le résultat précédent est décalé (étape 3), une première réduction modulaire est effectuée (étape 4), le PP associé à la $i^{\text{ème}}$ puissance de deux qui est $A.b_i$ est ajouté (étape 5), puis une seconde réduction modulaire est effectuée (étape 6).

Algorithme 7 Multiplication (par additions-décalages) et réduction croisées

Entrées : $p = (p_{n-1}\cdots p_1p_0)_\beta$, $A = (a_{n-1}\cdots a_1a_0)_\beta$, $B = (b_{n-1}\cdots b_1b_0)_\beta$.
Sorties : $S = A.B \pmod p$.

1. $S \leftarrow 0$
2. **pour** $i = n-1$ à 0 **faire**
3. $\quad S \leftarrow \beta S$
4. $\quad S \leftarrow S \pmod p$ (Réduction modulaire)
5. $\quad S \leftarrow S + A.b_i$
6. $\quad S \leftarrow S \pmod p$ (Réduction modulaire)
7. **fin pour**
8. **retourner** S

Pour effectuer les réductions modulaires situées aux étapes 4 et 6 de l'algorithme 7, plusieurs choix sont possibles. On peut par exemple utiliser la méthode d'Omura (cf. Algorithme 3), celle de Takagi (cf. Algorithme 4) ou celle de Barrett (cf. Algorithme 5).

On peut également remarquer comme Blakley [Bla83] que, pour $0 \leq j \leq n-1$:

$$S = 2S + A.b_j \leq 2(p-1) + (p-1) = 3p - 3.$$

Ainsi, deux soustractions par p au maximum par itération seront nécessaires afin de réduire S dans l'intervalle $[0, p[$

Multiplication modulaire de Montgomery. La multiplication modulaire de Montgomery (cf. Algorithme 8) évite de calculer une division en exécutant une réduction à

chaque itération de l'algorithme, à l'aide d'un décalage vers la droite dans la base β [Mon85].

Utilisant le même principe que la réduction de Montgomery (cf. Algorithme 6), dans laquelle il est calculé une quantité q rendant la variable $(S' + qp)$ divisible par β, la quantité m_i est calculé (étape 3 de l'Algorithme 8) à chaque itération de telle sorte que la valeur $(S + a_i.B + m_i.p)$ peut être divisée par β.

Il peut être montré qu'à la fin de l'étape 5 de l'algorithme 8, $S \in [0, 2p[$: une soustraction au maximum est alors nécessaire afin de réduire S dans l'intervalle $[0, p[$.

À la fin de l'algorithme 8, il est donc obtenu la valeur $S = \dfrac{AB + mp}{\beta^n}$ soit encore $S \equiv A.B.\beta^{-n} \pmod{p}$.

Algorithme 8 Multiplication modulaire de Montgomery avec soustraction finale

Entrées : $p = (p_{n-1} \cdots p_1 p_0)_\beta$, $\text{pgcd}(p, \beta) = 1$, $A = (a_{n-1} \cdots a_1 a_0)_\beta$, $B = (b_{n-1} \cdots b_1 b_0)_\beta$, $p' = (-p^{-1}) \pmod{m}$ (précalculé).
Sorties : $S = \text{MMM1}(A, B, p) = A.B.\beta^{-n} \pmod{p}$.

1. $S \leftarrow 0$
2. **pour** $i = 0$ à $n - 1$ **faire**
3. $\quad m_i \leftarrow ((s_0 + a_i.b_0)\, p') \pmod{\beta}$
4. $\quad S \leftarrow (S + a_i.B + m_i.p)\,/\beta$
5. **fin pour**
6. **si** $S \geqslant p$ **alors**
7. $\quad S \leftarrow S - p$
8. **finsi**
9. **retourner** S

Combinaison de plusieurs méthodes de réduction. Kaihara *et al.* proposent une nouvelle méthode pour calculer la multiplication modulaire [KT05]. Ils proposent l'implantation de l'opérateur arithmétique noté $\circledast$, défini par :

$$S = A \circledast B = A.B.\beta^{-\alpha n} \pmod{p},$$

α étant un nombre rationnel (avec $0 < \alpha < 1$), et αn un entier.

Cette nouvelle représentation permet d'éclater le multiplicateur en deux parties, ces deux parties étant ensuite traitées séparément : la partie supérieure (respectivement inférieure) du multiplicateur est traitée par une méthode classique de multiplication modulaire (la multiplication modulaire de Montgomery). Concrètement, cette méthode utilise le fait qu'une méthode classique de multiplication modulaire (respectivement la multiplication modulaire de Montgomery) traite les bits du multiplicateur de la gauche vers la droite (de la droite vers la gauche) : ces deux routines peuvent donc bien fonctionner en parallèle.

Le paramètre α est une variable d'ajustement permettant de répartir les bits du multiplicateur sur les deux multiplieurs modulaires afin de tirer profit au maximum de leurs

performances intrinsèques. Si le temps de calcul des deux multiplieurs est équivalent, on choisira $\alpha n = \lceil n/2 \rceil$ (où $\lceil x \rceil$ représente la valeur entière supérieure de x), ce qui répartira équitablement les bits du multiplicateur sur les deux multiplieurs. En revanche, si le multiplieur de Montgomery implanté est beaucoup plus rapide que l'autre, on choisira plutôt $\alpha n > \lceil n/2 \rceil$.

Concrètement, cette méthode peut être potentiellement deux fois plus rapide que la multiplication modulaire de Montgomery (cf. Algorithme 8).

Étude comparative. Cette étude comparative de l'ensemble des algorithmes à réduction intégrée recoupe quelque peu celle effectuée initialement pour les algorithmes de multiplication modulaire séparant les étapes de multiplication et de réduction.

Concernant les méthodes utilisant la multiplication par additions-décalages pour intercaler les réductions modulaires (cf. Algorithme 7), on peut critiquer les approches d'Omura (cf. Algorithme 3) et de Blakley [Bla83], car ces deux méthodes conduisent à des comparaisons, opérations qui rendent difficile l'utilisation de représentations redondantes : cela conduira donc à des implantations matérielles lentes. En revanche, la méthode de Takagi *et al.* (cf. Algorithme 4), parce qu'elle utilise intrinsèquement la représentation BS, peut être implantée : elle conduira au calcul de $3n$ BSAs, ce qui en fait une architecture rapide.

L'algorithme de Barrett et celui de Montgomery sont en concurrence directe : ce sont tous deux des algorithmes de réduction fonctionnant pour tous les moduli avec précalculs. Cependant, comme évoqué dans l'étude comparative précédente, et dans le cadre de la conception matérielle de ces deux méthodes, l'algorithme de Montgomery paraît plus intéressant que l'algorithme de Barrett lorsque de nombreuses réductions modulaires sont calculées succesivement.

Enfin, la méthode proposée par Kaihara *et al.* [KT05] semble *a priori* intéressante, mais elle conduit à l'utilisation d'une méthode classique de réduction modulaire : il doit donc être estimé la valeur du quotient $q = \lfloor S/p \rfloor$ au fur et à mesure du calcul, ce qui conduit soit au calcul d'une division, opération difficile à réaliser en pratique, soit à l'utilisation de méthodes de type Barrett (cf. Algorithme 5) difficiles à implanter matériellement.

Réduction séparée de la multiplication ou réduction intégrée : que choisir ? Les algorithmes de réduction qui commencent par faire la multiplication $S' = A.B$ puis qui réduisent le résultat de cette multiplication et les algorithmes qui réduisent leurs calculs au fur et à mesure de la multiplication peuvent maintenant être comparés.

L'intérêt des premiers est que la multiplication, laissée libre, peut être optimisée par des algorithmes de multiplication basés sur le principe d'interpolation [KO63] [Too63] [SS71]. Malheureusement, le gain apporté par ces méthodes ne devient vraiment significatif que pour des grandes valeurs de n, valeurs qui ne correspondent pas à celles utilisées pour l'ECC. Il faut également noter que l'implantation de la multiplication modulaire sous la forme d'une multiplication suivie d'une réduction n'est pas vraiment adaptée, car la réduction modulaire s'effectue alors sur le résultat S' qui s'écrit sur $2n$ bits (si les

opérandes A et B s'écrivent sur n bits). Du coup, il peut être considéré que le gain hypothétiquement amené par l'utilisation de méthodes de multiplication basés sur le principe d'interpolation [KO63] [Too63] [SS71] est compensé par le surcoût engendré par le calcul de la réduction modulaire.

C'est pour cela que **les algorithmes à réduction intégrée** en général (et la multiplication modulaire de Montgomery en particulier, cf. Algorithme 8) sont considérés comme **plus adaptés aux calculs modulaires**, notamment car la réduction modulaire s'effectue à moindre coût.

1.2.1.3 Inversion modulaire

Un cryptosystème dédié à l'ECC a besoin de pouvoir calculer une inversion modulaire rapidement (cf. chapitre 2). Le calcul de $X^{-1} \pmod p$ correspond à trouver l'entier Y tel que $XY = 1 \pmod p$. Trois méthodes sont possibles : la première est basée sur le petit théorème de Fermat, la seconde propose de calculer le PGCD de deux nombres, la troisième utilise la multiplication de Montgomery (cf. Algorithme 8).

Petit théorème de Fermat. Au dix-septième siècle, Fermat propose le « petit » théorème suivant : soit $0 < X < p$ avec p premier alors

$$X^{p-1} \equiv 1 \pmod p. \tag{1.6}$$

Ainsi, en multipliant l'équation 1.6 par $X^{-1} \pmod p$, nous obtenons finalement :

$$X^{-1} \equiv X^{p-2} \pmod p.$$

Le calcul de l'inverse sur un corps fini premier correspond donc à une exponentiation modulaire.

Algorithmes utilisant des calculs de PGCD. Une autre méthode pour calculer une inversion modulaire utilise le calcul de PGCD [Ste67] : elle consiste à calculer $X^{-1} \pmod p$ en trouvant un entier A, tel que :

$$AX + pY' = 1. \tag{1.7}$$

En effet, la relation 1.7 conduit à :

$$\begin{array}{rcl} AX & \equiv & 1 \pmod p \\ X^{-1} & \equiv & A \pmod p \end{array}$$

Les principales opérations utilisées dans le calcul du PGCD de deux nombres sont :

1. l'addition,
2. la soustraction,
3. la division par 2 d'un nombre pair,

4. le calcul de la valeur absolue $|A' - B'|$ de deux variables impaires A' et B' présentes dans l'algorithme.

Inversion de Montgomery. Si la multiplication modulaire de Montgomery est déjà implantée dans une UA, des algorithmes d'inversion théoriquement plus efficaces utilisant cette opération existent [Kal95] [SK00].

Les principales opérations utilisées dans le cadre de l'inversion de Montgomery sont :

1. le calcul du PGCD de deux nombres,
2. la multiplication de Montgomery (cf. Algorithme 8),
3. la comparaison.

Étude comparative. Dans le cadre de l'implantation matérielle de notre UA pour l'ECC, l'étude comparative des trois méthodes d'inversion modulaire décrites précédemment comporte plusieurs critères, notamment les contraintes matérielles liées à leur implantation et la possibilité ou non d'utiliser des représentations redondantes (cf. section 1.1.2) afin de pouvoir accélérer des calculs intermédiaires.

Le petit théorème de Fermat permet de calculer une inversion modulaire à l'aide d'une exponentiation modulaire, et donc à l'aide d'une série de multiplications modulaires : c'est donc une méthode d'inversion facile à implanter.

Concernant les algorithmes d'inversion utilisant le calcul du PGCD de deux nombres, les opérations numérotées 1. et 2. peuvent être calculées rapidement si des représentations redondantes sont utilisées. L'opération 3. peut être également calculée rapidement à l'aide d'un simple décalage vers la droite. En revanche, le calcul de la valeur absolue $|A' - B'|$ est moins trivial. En effet, il est nécessaire dans ce cas de connaître rapidement le signe de $A' - B'$, ce qui n'est pas possible si des représentations redondantes sont utilisées. Cela conduira à l'utilisation de représentations binaires, donc à des implantations lentes pour le calcul des additions et des soustractions (cf. section 1.1.2.5), et par voie de conséquence à un temps d'exécution important pour le calcul global du PGCD. De plus, l'implantation de tels algorithmes conduisent à un ordonnancement complexe à mettre en œuvre des différentes opérations nécessaires.

Enfin, les mêmes constats peuvent être dressés concernant l'inversion de Montgomery, notamment ceux portant sur les difficultés inhérentes à son implantation matérielle.

1.2.2 Faciliter les calculs dans les corps finis

Historiquement, dans le but d'accélérer les réductions modulaires (cf. section 1.2.1.2), les mathématiciens ont concentré leurs recherches sur la découverte de classes de nombres permettant d'effectuer les calculs modulaires plus facilement : c'est par exemple le cas des nombres dits « de Mersenne », et cette section commencera d'ailleurs par les présenter. Nous verrons que la réduction modulaire à l'aide de ces nombres se calcule de manière très efficace. Mais nous montrons également le principal défaut de cette classe de nombre qui réside dans sa trop faible densité. La solution trouvée à ce problème de densité est la classe

des nombres dits « pseudo-Mersennes » [Cra92]. Cette classe est plus dense que celles des nombres de Mersenne originaux, mais elle a pour contrepartie d'avoir une arithmétique moins rapide que celle des nombres de Mersenne. Pour concilier les impératifs de densité et de rapidité, un compromis a été trouvé à travers la classe des nombres de Mersenne dits « généralisés » [Sol99] [Wu00]. D'autres généralisations existent encore [CH04].

1.2.2.1 Nombres de Mersenne

En 1644, Mersenne a découvert des propriétés intéressantes concernant les entiers premiers de la forme $p = 2^n - 1$. En effet, en posant : $S' = s'_1 2^n + s'_0$, et en utilisant la relation $2^n \equiv 1 \pmod{p}$, S' peut être réduit grâce à $S' \equiv s'_1 + s'_0 \pmod{p}$. Ainsi, la réduction modulo p se réduit à l'addition (modulo p) de la partie haute s'_1 et de la partie basse s'_0 du nombre que l'on désire réduire !

Exemple 1.9. Soit $S' = 273$, et $p = 31 = 2^5 - 1$. Nous avons donc $2^5 \equiv 1 \pmod{31}$. Vu que $S' = 8 \times 2^5 + 17$, alors : $S' \equiv 8 + 17 = 25 \pmod{31}$.

Cette tactique sera ensuite reprise par d'autres chercheurs : essayer de trouver des nombres aux caractéristiques spéciales permettant de calculer une réduction modulaire en effectuant le plus petit nombre d'opérations possible traitant elles-mêmes des opérandes les plus petites possibles.

Malheureusement, les nombres premiers de Mersenne ont un défaut majeur : il n'en existe que très peu pour les tailles cryptographiques usuelles[8]. En effet, les indices n inférieurs à 1000 qui conduisent à de tels nombres sont 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521 et 607. Ainsi, aucun premier de Mersenne ne se situe dans l'intervalle $[2^{160}, 2^{256}]$, intervalle qui est d'une importance capitale pour la cryptographie à base de courbes elliptiques (cf. chapitre 2 pour obtenir plus de détails sur les courbes elliptiques).

1.2.2.2 Pseudo-Mersennes

C'est pour cela que la recherche s'est ensuite concentrée sur d'autres classes d'entiers dont la forme et les propriétés arithmétiques sont proches de celles de Mersenne. C'est ainsi que Crandall [Cra92] proposa des « pseudo-Mersennes » de la forme $p = 2^n - c$, avec $|c| \leq 2^{n/2} - 1$. Ainsi, en utilisant la relation $2^n \equiv c \pmod{p}$, et en décomposant le nombre à réduire avec la même méthode que précédemment, on peut montrer que cette réduction se ramène au maximum au calcul de trois multiplications par $|c|$ et un petit nombre d'additions/soustractions modulaires.

La densité de pseudo-Mersennes est beaucoup plus importante que celle des nombres de Mersenne originaux. Cependant, il faut noter que Crandall a déposé un brevet en 1992 concernant cette classe de nombres, ce qui peut limiter l'utilisation des pseudo-Mersennes

[8] Depuis 1996, des équipes de contributeurs *via* Internet peuvent relever le défi de trouver le prochain plus grand nombre de Mersenne premier. Ainsi, il a été trouvé dans le cadre du GIMPS (Great Internet Mersenne Prime Search) l'entier $2^{43112609} - 1$ qui est le premier nombre de Mersenne contenant plus de 10 millions de chiffres décimaux. Il peut être trouvé plus d'informations sur le site http ://www.mersenne.org.

dans le contexte industriel. Ce brevet a certainement conduit à relancer la recherche de d'autres formes de nombres premiers adaptés à l'arithmétique modulaire. En particulier, la communauté s'est attelée à rechercher des nombres premiers ayant une forme spécifique permettant d'appliquer les principes élémentaires de Mersenne.

1.2.2.3 Nombres de Mersenne généralisés

C'est ce que proposa Solinas [Sol99] : il introduisit la famille des nombres de Mersenne généralisés de la forme $p = f(t)$, où f est un polynôme unitaire à coefficients entiers et t une puissance de 2.

Exemple 1.10. Considérons un nombre de Mersenne généralisé de la forme $p = 2^{3n} - 2^n - 1$, par exemple $p = 2^{192} - 2^{64} - 1$: ce dernier peut être défini par le polynôme $f(t) = t^3 - t - 1$ évalué en $t = 2^{64}$. Ainsi, les entiers modulo p sont considérés comme des entiers de longueur $3n$ $(= 192)$ bits, et chaque entier à réduire $S' < p^2$ est considéré comme une expression de longueur $6n$ $(= 384)$ bits :

$$S' = \sum_{j=0}^{5} s'_j 2^{jn} = s'_5 2^{320} + s'_4 2^{256} + s'_3 2^{192} + s'_2 2^{128} + s'_1 2^{64} + s'_0, \text{ avec } 0 \leq s'_j < 2^{64}.$$

Pour résumer, il peut être prouvé que $r = S' \pmod{(2^{192} - 2^{64} - 1)}$ peut être obtenu en additionnant les quatre entiers de 192 bits $(s'_5 s'_5 s'_5)_{2^{64}}$, $(s'_4 s'_4 0)_{2^{64}}$, $(s'_2 s'_1 s'_0)_{2^{64}}$, et $(0 s'_3 s'_3)_{2^{64}}$, et en effectuant des soustractions successives par p jusqu'à ce que r soit inférieur strictement à p [Sol99].

Wu, quant à lui, propose d'utiliser deux autres familles de nombres de Mersenne généralisés [Wu00]. La première famille s'écrit sous la forme $p = 2^n - 2^m - 1$, avec $0 < m < \dfrac{n+1}{2}$. L'opération de réduction modulaire $S' \pmod p$ où $S' < p^2$ peut alors être effectuée de la manière suivante :

$$r = S' \pmod p \equiv s'_1 + s'_2 + s'_4 + 2^m (s'_3 + s'_4), \text{ avec par définition}$$

$$\begin{cases} S' = s'_1 + s'_2 2^n, 0 \leq s'_1 \leq 2^n - 1 \\ s'_2 = s'_3 + s'_4 2^{n-m}, 0 \leq s'_3 \leq 2^{n-m} - 1. \end{cases}$$

La deuxième famille s'écrit sous la forme $p = 2^n - 2^m - 2^{m_1} - 1$, avec $0 < m_1 < m < \dfrac{n+1}{2}$. L'opération de réduction modulaire $S' \pmod p$ où $S' < p^2$ peut alors être effectuée de la manière suivante :

$$r = S' \pmod p \equiv s'_1 + s'_2 + s'_4 + s'_6 + 2^m (s'_3 + s'_4 + s'_6) + 2^{m_1} (s'_3 + s'_4 + s'_6 + s'_5 2^{n-m}),$$

$$\text{avec par définition} \begin{cases} S' = s'_1 + s'_2 2^n, 0 \leq s'_1 \leq 2^n - 1 \\ s'_2 = s'_3 + s'_4 2^{n-m}, 0 \leq s'_3 \leq 2^{n-m} - 1 \\ s'_4 = s'_5 + s'_6 2^{m-m_1}, 0 \leq s'_5 \leq 2^{m-m_1} - 1. \end{cases}$$

Il faut noter que les relations mises en évidence par Wu (comme celles de Solinas) sont des congruences, et non des égalités strictes : il faudra donc effectuer des soustractions successives par p jusqu'à ce que r soit inférieur strictement à p.

1.2.2.4 Nombres de Mersenne « encore plus » généralisés

L'approche consistant à choisir un nombre de Mersenne généralisé pour accélérer la phase de réduction modulaire est très efficace. Elle conduit toutefois à un manque de flexibilité. En effet, lorsqu'on implante matériellement une méthode de réduction relative à un modulo ayant la forme d'un Mersenne, cette architecture n'est plus valable si ce modulo change entre deux applications.

Chung *et al.* remédient à cette situation en étendant l'idée de Solinas : leur méthode autorise n'importe quelle valeur pour t dans l'expression $p = f(t)$ [CH04]. Ainsi, leur approche permet d'engendrer, grâce à des valeurs de t différentes, plusieurs nombres premiers de même taille avec le même polynôme de départ : la même arithmétique polynomiale peut donc être réutilisée par plusieurs implantations matérielles.

1.3 Conclusion de chapitre

Il a pu être constaté tout au long de ce chapitre que l'arithmétique des ordinateurs, qui est en fait l'art d'implanter des opérateurs arithmétiques performants, est une discipline très riche et en constante évolution.

Ce chapitre a permis également de montrer que les algorithmes utilisés en arithmétique des ordinateurs sont souvent plus complexes et très différents de ceux qu'on appelle « les algorithmes classiques (ou naïfs) ».

Dans le but d'offrir de bonnes performances à une UA pour l'ECC, plusieurs leçons peuvent être tirées de ce chapitre.

Premièrement, il a été montré que le choix de la représentation des nombres traités par une UA est crucial pour pouvoir obtenir de bonnes performances, notamment un temps de calcul de l'addition de deux opérandes acceptable. L'addition est une opération fondamentale en arithmétique car beaucoup d'autres en découlent (par exemple, la multiplication et la division peuvent être considérées comme une succession d'additions et de décalages) : il faut donc l'implanter avec soin.

Cette étude a montré que l'utilisation de représentations binaires conduit à des temps de calcul prohibitifs pour l'addition de deux opérandes. *A contrario*, l'implantation dans une UA de **représentations redondantes** permettra de calculer l'addition à temps constant, et ce, avec un délai très faible : c'est ce type de représentation que nous choisirons d'implanter. Il existe également des représentations redondantes permettant de calculer les multiplications rapidement (par exemple, Booth 2), mais elles ne conviennent pas pour notre application.

Un deuxième constat s'impose : effectuer efficacement des opérations arithmétiques sur un corps fini premier n'est pas chose aisée, et là encore, utiliser des algorithmes dits « classiques » ne permet pas d'obtenir des performances acceptables.

Les opérations nécessaires à l'ECC sont l'addition, la multiplication, et l'inversion modulaire.

Pour calculer **l'addition modulaire** efficacement, nous choisirons **l'algorithme de Takagi** qui utilise intrinsèquement la représentation BS, représentation permettant une addition rapide.

Concernant la **multiplication modulaire**, et compte tenu des tailles de moduli recommandés dans le cadre de l'ECC, nous recommandons l'utilisation **d'algorithmes à réduction intégrée**. Parmi ceux-ci, notre préférence revient à la **multiplication de Montgomery**, qui est celle comportant le moins d'obstacles à son implantation matérielle.

Enfin, pour calculer **l'inversion modulaire**, le **petit théorème de Fermat**, de par sa simplicité d'implantation (c'est une suite de multiplications modulaires), nous paraît être l'alternative idéale.

Enfin, il a été montré que l'on peut **accélérer les calculs dans les corps finis** en choisissant par exemple un système modulaire adapté utilisant un **nombre premier de Mersenne**. Cette dernière astuce est d'ailleurs préconisée par les organismes standardisant des courbes elliptiques (NIST et Certicom© notamment). Cependant, dans notre cas d'étude, il ne sera pas pris en compte ces différentes améliorations.

Chapitre 2

Les courbes elliptiques

Sommaire

Un des objectifs de cette thèse étant de concevoir une UA pour courbes elliptiques, ce chapitre détaille quelques aspects mathématiques de ces dernières en vue de leur implantation. Il est à noter que certaines démonstrations mathématiques ne seront pas détaillées, le but de ce chapitre étant de décrire uniquement les éléments indispensables à la compréhension des chapitres suivants. Le lecteur voulant obtenir plus de détails sur certaines notions développées dans ce chapitre pourra consulter [HMV04].

Historiquement, les courbes elliptiques sont apparus dans la résolution de nombreux problèmes bien avant la naissance de la cryptographie à clé publique. De nombreux problèmes de gravitation ou d'électromagnétisme ont par exemple été résolus par des calculs

d'intégrales elliptiques ; on en retrouve en particulier dans des travaux d'Euler et de Gauss datant des 18ème et 19ème siècles. Mais les courbes elliptiques interviennent aussi dans la résolution de nombreux problèmes en théorie des nombres.

L'utilisation cryptographique des courbes elliptiques définie sur un corps fini a quant à elle été proposée indépendemment par Miller [Mil86] et Koblitz [Kob87] dans les années 80. Même si, pour des raisons purement économiques, elles tardent à s'imposer dans le monde de l'industrie et du commerce, les avantages procurés par les courbes elliptiques ont rapidement convaincu la communauté universitaire.

Une courbe elliptique peut être définie de manière très générale comme l'ensemble des solutions d'une équation à deux variables. Une courbe elliptique est tout d'abord un objet géométrique : c'est une courbe non-singulière obéissant à l'équation $y^2 = f(x)$, avec f ayant un degré égal à 3 ou 4. C'est ce genre de courbes définies sur un corps fini que nous utilisons en cryptographie. Nous verrons également dans ce chapitre qu'une courbe elliptique est aussi un objet algébrique.

2.1 Introduction aux courbes elliptiques

Dans cette partie, il est présenté quelques propriétés importantes des courbes elliptiques, notamment l'équation algébrique à laquelle obéissent ces dernières, leur forme géométrique, ainsi que le nombre de points qu'elles contiennent. Il est également décrit comment utiliser des courbes elliptiques pour faire de la cryptographie.

2.1.1 Équation de Weierstrass simplifiée

Une courbe elliptique E sur $\mathbb{R}$ est l'ensemble des points $(x, y) \in E$, tels que :

$$E : y^2 = x^3 + ax + b \tag{2.1}$$

où $a, b \in \mathbb{R}$ et où le déterminant de E noté Δ est non-nul, avec

$$\Delta = -16(4a^3 + 27b^2).$$

La condition $\Delta \neq 0$ assure qu'il n'existe pas de points sur la courbe admettant deux tangentes ou plus. Cette courbe est munie d'un point à l'infini noté ∞.

2.1.2 Loi de groupe

L'ensemble des points d'une courbe elliptique E sur $\mathbb{R}$ muni de l'opération « addition de points » (notée $+$) forme un groupe abélien pour lequel ∞ est l'élément neutre, et la loi de groupe est donnée par la règle de la « sécante-tangente ».

La figure 2.1 illustre cette dernière sur la courbe elliptique $y^2 = x^3 - 2{,}5x + 3$ définie sur $\mathbb{R}$. Si $P_1 \neq P_2$, le point $P_3 = P_1 + P_2$ est le symétrique par rapport à l'axe des abscisses du point d'intersection entre la courbe et la droite (P_1, P_2) ; on calcule $[2]P_3$ de la même manière, en considérant la tangente à la courbe au point P_3. Les droites de pente infinie

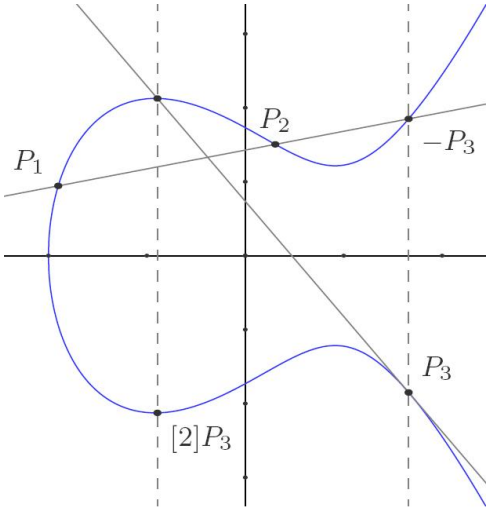

FIG. 2.1 – Illustration de la loi de groupe sur la courbe elliptique $y^2 = x^3 - 2{,}5x + 3$ définie sur $\mathbb{R}$ [Imb08].

passent toutes par le point ∞ ; l'opposé du point de coordonnées (x, y) est donc le point $(x, -y)$: le calcul de l'opposé d'un point sur une courbe elliptique est donc quasiment gratuit.

Les formules algébriques pour cette loi de groupe dérivent de cette description géométrique. Considérons deux points $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$. L'addition des points P_1 et P_2 (notée $P_1 + P_2$) donne le point $P_3 = (x_3, y_3)$, où :

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = (x_1 - x_3)\lambda - y_1 \end{cases} \quad \text{avec } \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, \text{ si } x_1 \neq x_2 \text{ [Addition]} \\ \dfrac{3x_1^2 + a}{2y_1}, \text{ si } x_1 = x_2 \text{ [Doublement]} \end{cases} \tag{2.2}$$

On peut vérifier que $(E, +)$ est bien un groupe abélien. Notamment, il peut être montré que :

1. $+$ est associative : $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$,
2. il existe un point à l'infini $\infty \in E$ avec $P_1 + \infty = \infty + P_1 = P_1$,
3. pour tout élément $P_1 \in E$, $-P_1 \in E$ tel que $P_1 + -P_1 = -P_1 + P_1 = \infty$,
4. c'est un groupe commutatif $P_1 + P_2 = P_2 + P_1$.

La démonstration mathématique de ces différents points est détaillée notamment dans le rapport technique de Joye [Joy95].

Les courbes elliptiques définies sur un corps fini (par exemple $\mathbb{F}_p$, avec $p \neq 2, 3$) sont à l'origine des applications cryptographiques. Les formules d'addition et de doublement (cf. relation 2.2) s'appliquent de la même manière aux points à coordonnées dans $\mathbb{F}_p$. Dans

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

2.1.3 Nombre de points d'une courbe elliptique définie sur un corps fini

La communauté des mathématiciens s'est notamment intéressée à compter le nombre de points d'une courbe elliptique définie sur un corps fini (noté $\#E(\mathbb{F}_p)$). C'est Hasse qui en 1922 démontra le résultat suivant sur ce nombre également appelé « ordre » du groupe des points d'une courbe elliptique sur un corps fini [HMV04] :

$$|\#E(\mathbb{F}_p) - p - 1| \leq 2\sqrt{p}. \tag{2.3}$$

Compter le nombre de points d'une courbe elliptique fait partie des problématiques importantes en cryptographie ; c'est une étape indispensable dans la recherche de courbes cryptographiquement sûres (voir section 2.3.1).

2.1.4 Utiliser les courbes elliptiques pour faire de la cryptographie

Soit $\mathbb{G}$, un sous-groupe cyclique de $E(\mathbb{F}_p)$ d'ordre premier n généré par le point de base P (noté $n = \mathrm{ord}_E(P)$), soit

$$\mathbb{G} = \langle P \rangle = \{\infty, P, [2]P, \cdots, [n-1]P\} \subseteq E(\mathbb{F}_p), \text{ avec } [n]P = \infty.$$

L'opération à effectuer afin de concevoir un cryptosystème basé sur les courbes elliptiques est la multiplication scalaire de ce point P par k (qui est une clé durant les protocoles). Elle est définie comme suit :

$$E(\mathbb{F}_p) \times \mathbb{Z} \to E(\mathbb{F}_p), (P, k) \mapsto Q = [k]P = \underbrace{P + P + \cdots + P}_{k \text{ fois}}.$$

Cette opération est dite « à sens unique » : connaissant l'entier k et $P \in \mathbb{G}$, il est facile de calculer $Q = [k]P \in \mathbb{G}$, mais connaissant P et $[k]P$, il est difficile de retrouver k.

La sécurité de ce cryptosystème repose donc sur le problème du logarithme discret sur courbes elliptiques (*Elliptic Curve Discrete Logarithm Problem*, ECDLP).

Des attaques mathématiques ont été mises en évidence sur l'ECDLP : elles seront détaillées dans la section 2.3.1.

2.2 Arithmétique des courbes elliptiques efficace

L'ECC conduit à effectuer un grand nombre de calculs sur des nombres de 200 à 500 bits. Vu la longueur de ces nombres, il faut donc implanter des algorithmes optimisés

permettant d'effectuer ces calculs le plus rapidement possible.

Il a déjà été présenté quelques idées pour accélérer les calculs au niveau du corps fini (cf. section 1.2) : les calculs au niveau de la courbe peuvent également être accélérés.

Cette section présente quelques pistes connues pour accélérer les calculs sur les courbes elliptiques : combiner des opérations sur les points, choisir une représentation des points adéquate, implanter un algorithme de multiplication scalaire efficace. Une autre solution peut également consister à choisir des courbes aux propriétés intrinsèques intéressantes.

Lorsque cela est possible, ces différentes méthodes peuvent également être combinées afin d'obtenir de meilleurs résultats.

2.2.1 Succession d'opérations de points

Si l'on calcule naïvement $[2]Q + P$ à l'aide de la relation 2.2, cela conduit au calcul de deux inversions modulaires (notées I) et de sept multiplications modulaires (notées M), car un doublement coûte $(I + 4M)$ tandis qu'une addition de points coûte $(I + 3M)$.

Eisenträger *et al.* [ELM03] proposent une méthode efficace pour accélérer un doublement suivi d'une addition ($[2]Q+P$), qui sont deux opérations de points successives. Leur méthode consiste à calculer $(Q + P) + Q$ à la place de $[2]Q + P$, en omettant de calculer la coordonnée y du point $(Q + P)$ (cf. relation 2.2).

Ciet *et al.* ont par la suite obtenu de meilleurs résultats [CJLM06]. Comme dans [ELM03], ils calculent $(Q + P) + Q$ à la place de $[2]Q + P$, en omettant de calculer les coordonnées y **et** x du point $(Q + P)$.

Les coûts relatifs des deux méthodes pour différentes successions d'opérations de points sont répertoriés dans le tableau 2.1. À cette occasion, on peut voir apparaître deux nouvelles opérations de points : le triplement d'un point P ($[3]P = [2]P + P$) noté T, et la soustraction d'un point P ($[2]Q - P = [2]Q + (-P)$). Il faut rappeler que le calcul du point $-P$ est quasiment gratuit.

Opération	[ELM03]	[CJLM06]	[CJLM06] plus intéressant que [ELM03] quand
$[2]P \pm Q$	$2I + 5M$	$1I + 11M$	$I > 6M$
$[3]P$	$2I + 5M$	$1I + 11M$	$I > 6M$
$[3]P \pm Q$	$3I + 7M$	$2I + 12M$	$I > 5M$

Tab. 2.1 – Coût de quelques successions d'opérations de points.

La section 2.2.1 a montré qu'en combinant des opérations de points, il est possible d'économiser des opérations modulaires. Cependant, il est toujours nécessaire de calculer des inversions modulaires, opérations souvent coûteuses à calculer en termes de temps de calcul (cf. section 1.2.1.3). La section suivante détaille une solution simple pour éviter de calculer des inversions modulaires : effectuer un choix pertinent du système de coordonnées sur lequel vont se dérouler les calculs.

2.2.2 Choix pertinent du système de coordonnées

Afin d'éviter de calculer des inversions modulaires, cette section décrit deux alternatives aux coordonnées affines efficaces : utiliser les coordonnées dites « projectives » et « mixtes ».

2.2.2.1 Utilisation des coordonnées projectives

L'utilisation des coordonnées projectives permet d'éviter de calculer des inversions modulaires, en considérant qu'un point P de la courbe représenté par un triplet $(X : Y : Z)$ en coordonnées projectives correspond au point représenté en coordonnées affines $(X/Z^c, Y/Z^d)$, 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 (cf. relation 2.1) ainsi que dans les formules d'addition et de doublement (cf. relation 2.2) : il n'y a donc plus d'inversions modulaires à calculer pour effectuer l'addition et le doublement de point.

Exemple 2.1. Considérons le cas où $c = 2$ et $d = 3$. Le point représenté en coordonnées projectives $(X : Y : Z)$, avec $Z \neq 0$, correspond au point représenté en coordonnées affines $(X/Z^2, Y/Z^3)$. La forme projective de l'équation de Weierstrass (cf. relation 2.1) définie sur $\mathbb{F}_p$ s'écrit :

$$\begin{array}{rcl} (Y/Z^3)^2 & = & (X/Z^2)^3 + a(X/Z^2) + b \\ Y^2/Z^6 & = & X^3/Z^6 + aX/Z^2 + b \\ Y^2 & = & X^3 + aXZ^4 + bZ^6. \end{array}$$

Le point à l'infini ∞ correspond à $(1 : 1 : 0)$, tandis que l'opposé du point $(X : Y : Z)$ s'écrit $(X : -Y : Z)$.

Pour doubler le point représenté en coordonnées projectives $P_1 = (X_1 : Y_1 : Z_1)$, il faut injecter dans la formule du doublement du point (cf. relation 2.2) $x = X_1/Z_1^2$ et $y = Y_1/Z_1^3$. Ainsi, nous obtenons :

$$\begin{array}{rcl} X_3' & = & \left(\dfrac{3\dfrac{X_1^2}{Z_1^4} + a}{2\dfrac{Y_1}{Z_1^3}} \right)^2 - 2\dfrac{X_1}{Z_1^2} \\ & = & \dfrac{(3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2}{4Y_1^2Z_1^2} \end{array} \tag{2.4}$$

et

$$\begin{aligned} Y_3' &= \left(\frac{3\dfrac{X_1^2}{Z_1^4}+a}{2\dfrac{Y_1}{Z_1^3}}\right)\left(\frac{X_1}{Z_1^2}-X_3'\right)-\frac{Y_1}{Z_1^3} \\ &= \frac{3X_1^2+aZ_1^4}{2Y_1Z_1}\left(\frac{X_1}{Z_1^2}-X_3'\right)-\frac{Y_1}{Z_1^3} \end{aligned} \tag{2.5}$$

Afin d'éliminer les dénominateurs dans les expressions 2.4 et 2.5, il faut utiliser $X_3 = X_3'Z_3^2$ et $Y_3 = Y_3'Z_3^3$, où $Z_3 = 2Y_1Z_1$. Nous obtenons finalement les formules permettant de calculer $[2]P = (X_3 : Y_3 : Z_3)$ en utilisant ce système de coordonnées projectives :

$$\begin{cases} X_3 = (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 = (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 = 2Y_1Z_1. \end{cases} \tag{2.6}$$

En utilisant les variables intermédiaires A, B, C et D, les coordonnées X_3, Y_3 et Z_3 peuvent être calculées à l'aide de $10M$:

$$\begin{array}{lllllllll} A & \leftarrow & Y_1^2, & B & \leftarrow & 4X_1A, & C & \leftarrow & 8A^2, \\ D & \leftarrow & 3X_1^2 + aZ_1^4, & & & & & & \\ X_3 & \leftarrow & D^2 - 2B, & Y_3 & \leftarrow & D(B - X_3) - C, & Z_3 & \leftarrow & 2Y_1Z_1. \end{array}$$

Plusieurs systèmes de coordonnées projectives ont été proposés dans $\mathbb{F}_p$:

- les coordonnées projectives standard (notées $\mathcal{P}$) [HMV04],
- les coordonnées projectives Jacobiennes ($\mathcal{J}$) [HMV04],
- les coordonnées Jacobiennes modifiées ($\mathcal{J}^m$) [HMV04],
- les coordonnées de Chudnovsky *et al.* ($\mathcal{J}^c$) [CC87].

Le tableau 2.2 résume les propriétés de ces différents systèmes. Il faut noter que dans tous les cas l'opposé du point $(X : Y : Z)$ s'écrit $(X : -Y : Z)$.

Coor.	$\boldsymbol{P}$ **en coor. proj.**	$(\boldsymbol{x}, \boldsymbol{y}) =$	**Équation de courbe**	$\boldsymbol{\infty}$
$\mathcal{P}$	$(X : Y : Z)$	$(X/Z, Y/Z)$	$Y^2Z = X^3 + aXZ^2 + bZ^3$	$(0 : 1 : 0)$
$\mathcal{J}$	$(X : Y : Z)$	$(X/Z^2, Y/Z^3)$	$Y^2 = X^3 + aXZ^4 + bZ^6$	$(1 : 1 : 0)$
$\mathcal{J}^c$	$(X : Y : Z : Z^2 : Z^3)$	$(X/Z^2, Y/Z^3)$	$Y^2 = X^3 + aXZ^4 + bZ^6$	$(1 : 1 : 0)$
$\mathcal{J}^m$	$(X : Y : Z : aZ^4)$	$(X/Z^2, Y/Z^3)$	$Y^2 = X^3 + aXZ^4 + bZ^6$	$(1 : 1 : 0)$

TAB. 2.2 – Tableau résumant les propriétés des différents systèmes de coordonnées

Ce qui va déterminer le choix du système de coordonnées est le nombre d'opérations modulaires à effectuer pour calculer le doublement et l'addition de points. Le tableau

2.3 compare le coût du doublement et de l'addition pour chaque système de coordonnées projectives. Un constat s'impose : il n'existe pas de système de coordonnées fournissant

Système de coordonnées	**Coût d'un doublement**	**Coût d'une addition**
$\mathcal{A}$	$I + 4M$	$I + 3M$
$\mathcal{P}$	$12M$	$\mathbf{14M}$
$\mathcal{J}$	$10M$	$16M$
$\mathcal{J}^m$	$\mathbf{8M}$	$19M$
$\mathcal{J}^c$	$11M$	$\mathbf{14M}$

TAB. 2.3 – Coût du doublement et de l'addition pour chaque système de coordonnées projectives.

à la fois un doublement et une addition rapide. Par exemple, $\mathcal{J}^m$ assure le meilleur doublement (dès lors que $I > 4M$) mais l'addition est relativement coûteuse par rapport à $\mathcal{J}^c$.

2.2.2.2 Utilisation des coordonnées mixtes

Une solution à ce problème proposée initialement par Cohen *et al.* [CMO98] est de changer de système de coordonnées au cours de l'algorithme afin d'utiliser systématiquement le système de coordonnées le plus approprié suivant l'opération en cours d'exécution. Le choix se fait alors selon deux critères : l'efficacité vis-à-vis de l'opération concernée et le coût du passage d'un système de coordonnées au suivant.

Afin d'exprimer simplement ces deux notions, nous adopterons la notation suivante : soient $\mathcal{C}^1$, $\mathcal{C}^2$, et $\mathcal{C}^3$ trois systèmes de coordonnées, on notera $\mathcal{C}^1 + \mathcal{C}^2 \rightarrow \mathcal{C}^3$ l'opération consistant à calculer la somme de deux points donnés respectivement en coordonnées $\mathcal{C}^1$ et $\mathcal{C}^2$ et à retourner le résultat en coordonnées $\mathcal{C}^3$. Par exemple, $\mathcal{J} + \mathcal{J}^c \rightarrow \mathcal{J}^m$ signifie que l'on additionne un point en coordonnées Jacobiennes avec un point en coordonnées de Chudnovsky et que le résultat est donné en coordonnées Jacobiennes modifiées. Pour le doublement, on utilisera la notation similaire suivante : $[2]\mathcal{C}^1 \rightarrow \mathcal{C}^2$. Les coûts du doublement (respectivement de l'addition) de point(s) en utilisant toutes les combinaisons possibles de coordonnées mixtes sont listés dans le tableau 2.4 (2.5).

Comme nous le verrons dans la section 2.2.3, les algorithmes de multiplication scalaire consistent, pour la plupart, en une série de doublements entrecoupée d'additions. Sachant cela, afin d'optimiser l'utilisation de coordonnées mixtes, il est proposé dans [CMO98] la procédure suivante.

1. On commence par effectuer une série de doublements avec le même système de coordonnées. Vu que $\mathcal{J}^m$ offre les meilleures performances (cf. tableau 2.3), on effectuera donc $[2]\mathcal{J}^m \rightarrow \mathcal{J}^m$.
2. On place le résultat du dernier doublement dans un système de coordonnées $\mathcal{C}$ adapté afin de préparer la prochaine addition. Il sera donc effectué $[2]\mathcal{J}^m \rightarrow \mathcal{C}$.

Opération	Coût du doublement
$[2]\mathcal{J}^m \rightarrow \mathcal{J}^c$	$9M$
$[2]\mathcal{A} \rightarrow \mathcal{J}^c$	$8M$
$[2]\mathcal{J}^m \rightarrow \mathcal{J}$	$7M$
$[2]\mathcal{A} \rightarrow \mathcal{J}^m$	$7M$
$[2]\mathcal{A} \rightarrow \mathcal{J}$	$\mathbf{6M}$

TAB. 2.4 – Coût du doublement en utilisant les coordonnées mixtes.

Opération	Coût de l'addition
$\mathcal{J}^m + \mathcal{J}^c \rightarrow \mathcal{J}^m$	$17M$
$\mathcal{J} + \mathcal{J}^c \rightarrow \mathcal{J}^m$	$17M$
$\mathcal{J}^c + \mathcal{J}^c \rightarrow \mathcal{J}^m$	$15M$
$\mathcal{J}^c + \mathcal{J} \rightarrow \mathcal{J}$	$14M$
$\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}^m$	$14M$
$\mathcal{J}^m + \mathcal{A} \rightarrow \mathcal{J}^m$	$14M$
$\mathcal{J}^c + \mathcal{J}^c \rightarrow \mathcal{J}$	$12M$
$\mathcal{J}^c + \mathcal{A} \rightarrow \mathcal{J}^m$	$12M$
$\mathcal{J}^c + \mathcal{A} \rightarrow \mathcal{J}^c$	$11M$
$\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$	$11M$
$\mathcal{J}^m + \mathcal{A} \rightarrow \mathcal{J}$	$11M$
$\mathcal{A} + \mathcal{A} \rightarrow \mathcal{J}^m$	$9M$
$\mathcal{A} + \mathcal{A} \rightarrow \mathcal{J}^c$	$\mathbf{8M}$

TAB. 2.5 – Coût de l'addition de deux points en utilisant les coordonnées mixtes.

3. On calcule l'addition de ce point exprimé dans $\mathcal{C}$ avec un autre point, qui est une constante de l'algorithme de multiplication scalaire généralement exprimé dans $\mathcal{A}$. Le résultat est donné en coordonnées $\mathcal{J}^m$ pour préparer une nouvelle série de doublements. On effectuera donc $\mathcal{C} + \mathcal{A} \rightarrow \mathcal{J}^m$.
4. On retourne à l'étape 1.

Reste à déterminer quel système de coordonnées $\mathcal{C}$ est le plus adapté afin d'optimiser ce processus. Dans [CMO98], il est indiqué que suivant le *ratio* I/M, c'est soit $\mathcal{A}$, soit $\mathcal{J}^c$ qui doit être choisi. Par exemple, dans le cadre de la multiplication scalaire $Q = [k]P$ (k étant de longueur 192 bits), $\mathcal{J}^c$ doit être choisi si $I/M \geq 33{,}9$.

L'utilisation de coordonnées projectives ou mixtes permet donc d'éviter le calcul d'inversions modulaires lorsque il est exécuté des opérations de points (doublement, addition ou succession de ces deux opérations) : celles-ci peuvent donc être calculées plus rapidement. Ainsi, lorsque plusieurs opérations de points sont effectuées successivement (comme

c'est le cas dans le cadre de l'ECC), le gain obtenu est substantiel.

Pour obtenir une arithmétique des courbes elliptiques efficace, il peut donc être utilisé des combinaisons d'opérations de points (cf. section 2.2.2), et il peut être choisi un système de coordonnées adéquat (cf. section 2.2.1). Un autre facteur d'amélioration de cette efficacité peut également venir de la manière dont est implanté la multiplication scalaire, qui est l'opération fondamentale à effectuer dans le cadre de l'ECC (cf. section 2.1.4). La section suivante décrit quelques méthodes pour pouvoir effectuer $Q = [k]P$ (où P est un point de la courbe elliptique, et k la clé) rapidement.

2.2.3 Algorithmes de multiplication scalaire efficaces

Il sera tout d'abord présenté dans cette section un algorithme de base pour calculer $Q = [k]P$ appelé algorithme du « doublement-et-addition » puis des raffinements de celui-ci.

2.2.3.1 Algorithme du « doublement-et-addition »

Afin d'effectuer la multiplication scalaire, l'algorithme du « doublement-et-addition » peut être utilisé (cf. Algorithme 9). Cette méthode élémentaire pour le calcul de $Q = [k]P = \underbrace{P + P + \cdots + P}_{k \text{ fois}}$ consiste à parcourir les bits de k en partant des poids forts : on effectue ainsi un doublement pour chaque bit de k, suivi d'une addition pour chaque bit non nul.

Algorithme 9 Algorithme du « doublement-et-addition »

Entrées : P, $k = (k_{\ell-1}, \cdots, k_0)_2$.
Sorties : $Q = [k]P$.

1. $Q \leftarrow \infty$
2. **pour** $i = \ell - 1$ à 0 **faire**
3. $\quad Q \leftarrow [2]Q$
4. $\quad$ **si** $k_i = 1$ **alors**
5. $\quad\quad Q \leftarrow Q + P$
6. $\quad$ **finsi**
7. **fin pour**
8. **retourner** Q

Si k est représenté sur ℓ bits, l'algorithme 9 implique en moyenne le calcul de $\boldsymbol{\ell D}$ **et de** $\boldsymbol{(\ell/2)A}$ [HMV04].

Il existe d'autres algorithmes de multiplication scalaire plus efficaces, diminuant le nombre global d'opérations à effectuer : la section 2.2.3.2 décrit une méthode permettant de diminuer le nombre d'additions de points à calculer, tandis que la méthode décrite dans la section 2.2.3.3 propose de calculer des triplements de points en lieu et place de doublements et d'additions de points.

2.2.3.2 Algorithme utilisant une forme non-adjacente du scalaire traité par fenêtres

En remarquant que l'opposé d'un point sur une courbe elliptique est calculable facilement, une première optimisation naturelle consiste à considérer une représentation signée du scalaire k appelée « forme non-adjacente[9] » (*Non-Adjacent Form*, NAF). Cette représentation a déjà été présentée en section 1.1.3.2 dans sa version utilisant l'ensemble de chiffres $D_1 = \{\overline{1}, 0, 1\}$. Il faut noter que cette représentation est relativement facile à calculer (que ce soit poids faibles [Rei60] ou poids forts [JY00] en tête).

Ainsi, dans le cadre du calcul de la multiplication scalaire, lorsqu'un des chiffres de cette nouvelle représentation de k (notée $\mathrm{NAF}_2(k)$) vaut $\overline{1}$, on calcule une soustraction de point. Ce NAF utilisant l'ensemble de chiffres D_1 permet de porter le nombre de non-zéros du scalaire, et donc le nombre moyen d'additions/soustractions de points, à $\boldsymbol{\ell/3}$.

Ce principe peut être généralisé : k peut également être réécrit en utilisant un ensemble de chiffres $D_{2^w} = \{\overline{2^{w-1}}, \cdots, 2^{w-1}\}$, ce qui revient à le découper en fenêtres de longueur fixe w. On peut ainsi définir le $\mathrm{NAF}_w(k)$ suivant la formulation :

$$\mathrm{NAF}_w(k) = \sum_{i=0}^{\ell} k_i 2^i, \text{avec } |k_i| < 2^{w-1}.$$

Exemple 2.2. Si $k = 763 = (1011111011)_2$, alors il peut être calculé $\mathrm{NAF}_2(k)$ grâce au tableau 1.6. L'algorithme permettant de calculer $\mathrm{NAF}_3(k)$ et $\mathrm{NAF}_4(k)$ peut être trouvé dans [HMV04] :

$$\begin{array}{rcl} \mathrm{NAF}_2(k) & = & (1\ 0\ \overline{1}\ 0\ 0\ 0\ 0\ 0\ \overline{1}\ 0\ \overline{1}) \\ \mathrm{NAF}_3(k) & = & (\quad\ 3\ 0\ 0\ 0\ 0\ \overline{1}\ 0\ 0\ 3) \\ \mathrm{NAF}_4(k) & = & (\quad\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \overline{5}). \end{array}$$

Sous cette forme, le nombre moyen d'additions/soustractions de points nécessaires pour une multiplication scalaire est ramené à $\boldsymbol{\ell/(w+1)}$. Il faut cependant noter que cette méthode induit le précalcul des points $[j]P$ pour $j = 1, 3, \cdots, 2^{w-1} - 1$. Plus précisément, le coût de ces précalculs est de $\boldsymbol{(1D + (2^{w-2} - 1)A)}$, mais malgré cela, l'utilisation du $\mathrm{NAF}_w(k)$ dans le cadre de l'algorithme du « doublement-et-addition » (cf. Algorithme 10) reste plus intéressante que l'utilisation directe de l'algorithme 9.

Enfin, une dernière généralisation de cette méthode existe : on utilise toujours $\mathrm{NAF}_w(k)$, mais cette fois-ci il est traité par fenêtres de longueur variable (ou « glissante ») ayant un nombre maximum de chiffres. Ces fenêtres commencent et terminent par un non-zéro.

[9] L'expression « forme non-adjacente » est plutôt utilisée dans la communauté cryptographique. Dans la communauté arithmétique des ordinateurs, on utilise plutôt l'expression « représentation à chiffres signés canonique » (*Canonic Sign Digit*, CSD).

Algorithme 10 Algorithme du « doublement-et-addition » utilisant une forme non-adjacente du scalaire traité par fenêtres de longueur fixe

Entrées : P, $\mathrm{NAF}_w(k)$, les points $[j]P$ avec $j = 1, 3, \cdots, 2^{w-1} - 1$ (précalculs).
Sorties : $Q = [k]P$.

1. $Q \leftarrow \infty$
2. **pour** $i = \ell - 1$ à 0 **faire**
3. $\quad Q \leftarrow [2]Q$
4. $\quad$ **si** $k_i \neq 0$ **alors**
5. $\quad\quad Q \leftarrow Q + [k_i]P$
6. $\quad$ **finsi**
7. **fin pour**
8. **retourner** Q

Exemple 2.3. Soit $\mathrm{NAF}_2(k) = (10\bar{1}00000\bar{1}0\bar{1})$.
Traitons ce $\mathrm{NAF}_2(k)$ à l'aide de fenêtres glissantes ayant une longueur maximale de 3 chiffres, ces fenêtres commençant et terminant par un non-zéro. On obtient donc :
$\mathrm{NAF}_2(k) = (\underline{10\bar{1}}00000\underline{\bar{1}0\bar{1}})$.
Ainsi, si les valeurs $[3]P$ et $[5]P$ sont précalculées, alors la multiplication scalaire peut se dérouler de la manière suivante :
$[3]P \rightarrow [6]P \rightarrow [12]P \rightarrow [24]P \rightarrow [48]P \rightarrow [96]P \rightarrow [192]P \rightarrow [384]P \rightarrow [768]P \rightarrow [763]P$.
Ce faisant, on a exécuté 9 opérations de points, contre 13 si on s'était contenté d'exécuter l'algorithme 10.

Grâce à cette méthode, le nombre moyen d'additions/soustractions de points nécessaires pour une multiplication scalaire est encore diminué : il devient égal à $\boldsymbol{\ell A/(w + \nu(w))}$, avec $\boldsymbol{\nu(w) = 4/3 - (-1)^w/(3 \times 2^{w-2})}$. Le coût induit par les précalculs est de $\boldsymbol{(1D + ((2^w - (-1)^w)/3 - 1)A)}$.

2.2.3.3 Système de numération à double base

Dans les sections précédentes, le scalaire était représenté en base 2. Dimitrov *et al.* [DJM98] proposent de le représenter dans un autre système de numération redondant appelé système de numération à double base (*Double-Base Number System*, DBNS). Dans ce système, tout nombre entier strictement positif est représenté comme une somme de puissances combinées de 2 et 3 :

$$k = \sum_{i=1}^{\ell'} k_i 2^{a_i} 3^{b_i}, \text{avec } k_i \in \{-1, 1\} \text{ et } a_i, b_i \geq 0. \tag{2.7}$$

Il a été montré dans [DJM98] que chaque entier positif k peut être représenté sous la forme d'une somme ou d'une différence de seulement $O(\log(k)/\log(\log(k)))$ termes au

maximum.

Cependant, l'utilisation directe de cette méthode pour le calcul d'une multiplication scalaire n'est pas pleinement satisfaisante car elle peut demander jusqu'à $\sum_i b_i T$, $\sum_i a_i D$, et $\ell' A$ additions.

Pour réduire de manière significative ce coût, Dimitrov *et al.* [DIM05] ont introduit des expansions de DBNS, qui permettent d'évaluer $[k]P$ en réutilisant tous les calculs intermédiaires. Pour ce faire, on garde la représentation de k définie dans la relation 2.7, avec la contrainte supplémentaire que les exposants forment deux suites décroissantes : $a_1 \geq a_2 \geq \cdots \geq a_{\ell'} \geq 0$ et $b_1 \geq b_2 \geq \cdots \geq b_{\ell'} \geq 0$.

Cette formulation permet ainsi de ne calculer lors d'une multiplication scalaire que seulement $a_1 = \max_i a_i\ D$, $b_1 = \max_i b_i\ T$ et $(\ell' - 1)A$.

Enfin, cette approche a été généralisée par Doche *et al.* [DI06] : un espace de chiffres un peu plus large et quelques précalculs sont ainsi autorisés.

Un bilan sur le coût de l'utilisation des différents algorithmes de multiplication scalaire présentés dans cette section sera effectué en section 2.2.5.

Il a donc été montré dans cette section comment accélérer l'opération fondamentale dans le cadre de l'ECC qu'est la multiplication scalaire. Afin d'accélérer les calculs sur les courbes elliptiques, on peut également choisir des courbes aux propriétés intrinsèques intéressantes.

2.2.4 Courbes aux propriétés intéressantes

Dans cette section, il est décrit pourquoi il peut être intéressant d'effectuer des calculs sur des courbes isogènes, de Montgomery ou d'Edwards.

2.2.4.1 Courbes isogènes

Il peut être calculé efficacement des multiplications scalaires à l'aide d'isogénies de courbes.

Deux courbes E_1 et E_2 définies sur $\mathbb{F}_p$ sont isogènes sur ce corps fini si il existe une application $\varphi : E_1 \to E_2$ telle que $\varphi(\infty_{E_1}) = \infty_{E_2}$ (∞_{E_i} représente l'élément neutre de E_i).

Une isogénie est un homomorphisme de groupe de $E_1(\mathbb{F}_p)$ vers $E_2(\mathbb{F}_p)$, c'est-à-dire que pour deux points P et $Q \in E_1(\mathbb{F}_p)$:

$$\varphi(P + Q) = \varphi(P) + \varphi(Q).$$

Une propriété importante est que pour chaque isogénie φ, il existe une unique isogénie $\hat{\varphi} : E_2 \to E_1$ appelée « isogénie duale » telle que

$$\hat{\varphi} \circ \varphi = [m], \tag{2.8}$$

où m est le degré de l'isogénie φ.

De plus, deux courbes elliptiques E_1 et E_2 définies sur $\mathbb{F}_p$ sont isogènes sur ce corps fini si et seulement si $\#E_1(\mathbb{F}_p) = \#E_2(\mathbb{F}_p)$.

Les isogénies peuvent être utilisées pour calculer une multiplication scalaire en utilisant la relation 2.8 :

$$\hat{\varphi} \circ \varphi(P) = [m]P.$$

Doche *et al.* [DIK06] ont trouvé que la famille de courbes elliptiques définie sur $\mathbb{F}_p$ par

$$y^2 = x^3 + ux^2 + 16ux \text{ (respectivement } y^2 = x^3 + 3u(x+1)^2)$$

notée DIK2 (DIK3) accepte des isogénies de degré 2 (3), et permet donc un doublement (triplement) rapide.

Ainsi, lorsqu'une variante des coordonnées $\mathcal{J}$ est utilisée pour représenter le point P que l'on veut doubler (tripler) ($P = (X : Y : Z : Z^2)$, où $x = X/Z^2$ et $y = Y/Z^3$) sur cette courbe, calculer $[2]P = (X_3 : Y_3 : Z_3 : Z_3^2)$ ($[3]P = (X_3 : Y_3 : Z_3 : Z_3^2)$) coûte $7M$ ($14M$).

Le résultat obtenu concernant le triplement de points est à mettre en relation avec ceux relatifs au DBNS (cf. section 2.2.3.3) : si on combine un algorithme de multiplication scalaire impliquant un faible nombre de triplements à calculer, avec une courbe sur laquelle le triplement peut être calculée efficacement, il peut être espéré que l'arithmétique sur celle-ci soit efficace. À ce propos, un bilan sera effectué en section 2.2.5.

2.2.4.2 Courbes de Montgomery

Les courbes de Montgomery [Mon87] définies sur $\mathbb{F}_p$ par

$$by^2 = x^3 + ax^2 + x \tag{2.9}$$

utilisent un algorithme de multiplication scalaire adapté, appelé échelle de Montgomery (cf. Algorithme 11), qui permet de ne calculer que la coordonnée x de $[k]P$ sous certaines conditions.

À première vue, utiliser l'échelle de Montgomery apparaît moins intéressant qu'utiliser l'algorithme du « doublement-et-addition » (cf. Algorithme 9) : en effet, l'algorithme 11 conduit en moyenne au calcul de $\ell D + \ell A$ contre $\ell D + (\ell/2)A$ pour l'algorithme 9. Cependant, l'échelle de Montgomery peut se révéler plus efficace [JY02] en observant notamment que

1. la valeur $R_1 - R_0$ reste constante pendant l'exécution de l'algorithme ($R_1 - R_0 = P$),
2. à chaque itération de l'algorithme 11, les opérations d'addition et de doublement de points sont indépendantes.

Utilisant la première remarque, il a été montré dans [Mon87] que la coordonnée x du point $Q = [k]P$ peut être calculée à l'aide de la coordonnée x de R_0, la coordonnée x de R_1, et des coordonnées x et y de P. Puisque les calculs peuvent être effectués avec les coordonnées x seulement, un certain nombre de multiplications modulaires peuvent être

Algorithme 11 Échelle de Montgomery

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow P$; $R_1 \leftarrow [2]P$
2. **pour** $i = \ell - 2$ à 0 **faire**
3. **si** $k_i = 0$ **alors**
4. $R_1 \leftarrow R_0 + R_1$; $R_0 \leftarrow [2]R_0$
5. **sinon si** $k_i = 1$ **alors**
6. $R_0 \leftarrow R_0 + R_1$; $R_1 \leftarrow [2]R_1$
7. **finsi**
8. **fin pour**
9. **retourner** R_0

économisées. De plus, on économise également de la mémoire car les coordonnées y n'ont pas besoin d'être stockées. Si cela est nécessaire, à la fin du calcul, la coordonnée y de Q peut être retrouvée à la fin du calcul à l'aide du point P, de la coordonnnée x de Q et de la coordonnée x du point $Q + P$.

Concernant la deuxième remarque, pour montrer comment l'échelle de Montgomery peut être parallélisée, il faut simplifier l'algorithme 11 en utilisant k_i et son inverse $\neg k_i$ comme des adresses de registre. Les deux étapes 4 et 6 de l'algorithme 11 peuvent ainsi être réécrits sous la forme d'une seule étape :

$$R_{\neg k_i} \leftarrow R_0 + R_1; R_{k_i} \leftarrow [2]R_{k_i}.$$

L'échelle de Montgomery peut donc être réécrite sous forme parallélisée (cf. Algorithme 12).

Algorithme 12 Échelle de Montgomery sous forme parallélisée

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow P$; $R_1 \leftarrow [2]P$
2. **pour** $i = \ell - 1$ à 0 **faire**
3. $R_{\neg k_i} \leftarrow R_0 + R_1$ (Processeur 1) | $R_{k_i} \leftarrow [2]R_{k_i}$ (Processeur 2)
4. **fin pour**
5. **retourner** R_0

Sous cette forme, l'échelle de Montgomery a donc un temps d'exécution dépendant de l'opération de point la plus longue. Pour les courbes obéissant à la relation 2.9, l'addition de point revient à $6M$, et le doublement à $5M$. Le temps de calcul de l'algorithme 12 pour ces courbes est donc égal à $(1D + 6\ell M)$, contre $(\ell D + \ell A) = (5\ell M + 6\ell M) = 11\ell M$ dans le cadre de l'utilisation de l'algorithme 11.

Brier *et al.* [BJ02] ont par la suite généralisé l'idée originale de Montgomery aux

courbes elliptiques de Weierstrass (cf. relation 2.1). Malheureusement, leurs formules d'addition et de doublement de points conduisent à un nombre plus important de calculs à effectuer. En effet, l'addition de points sur ces courbes coûte $11M$, tandis que le doublement conduit au calcul de $9M$. Le temps de calcul de l'algorithme 12 pour ces courbes devient donc égal à $(1D + 11\ell M)$.

Il faut noter que l'échelle de Montgomery est un moyen de protection possible pour les cryptosystèmes basés sur l'ECC contre les attaques dites « par observation » (cf. section 3.1 pour une définition des attaques par observation).

2.2.4.3 Courbes d'Edwards

La plupart des courbes elliptiques standardisées [SEC00b] [FIP00] obéit à l'équation de Weierstrass (cf. relation 2.1). Cependant, il existe des courbes elliptiques possédant une loi de groupe plus efficace.

En 2007, Edwards [Edw07] proposa des courbes elliptiques définies sur $\mathbb{F}_p$ par l'équation :

$$x^2 + y^2 = a^2(1 + x^2y^2), \text{avec } a^5 \neq a. \tag{2.10}$$

Bernstein *et al.* [BL07a] ont introduit leurs dérivées plus adaptées à l'ECC. Un paramètre d est présent dans la nouvelle équation afin de couvrir plus de courbes sur $\mathbb{F}_p$:

$$x^2 + y^2 = c^2(1 + dx^2y^2), \text{avec } cd(1 - dc^4) \neq 0. \tag{2.11}$$

Contrairement à la loi de groupe des courbes obéissant à l'équation de Weierstrass, le point à l'infini est un point affine ($\infty = (0, c)$). L'opposé du point $P = (x, y)$ est $-P = (-x, y)$.

L'addition de deux points $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$ donne un troisième point $P_3 = (x_3, y_3)$, avec :

$$\begin{cases} x_3 = \dfrac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)} \\ y_3 = \dfrac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)}. \end{cases}$$

Une propriété intéressante de cette loi de groupe est que, contrairement aux courbes obéissant à l'équation de Weierstrass (cf. relation 2.1), les formules d'addition de point peuvent être réutilisées afin de calculer le doublement d'un point $P_1 = (x_1, y_1)$. On obtient ainsi le point $P'_3 = (x'_3, y'_3)$, avec :

$$\begin{cases} x'_3 = \dfrac{x_1y_1 + y_1x_1}{c(1 + dx_1x_1y_1y_1)} \\ y'_3 = \dfrac{y_1y_1 - x_1x_1}{c(1 - dx_1x_1y_1y_1)}. \end{cases}$$

Lorsque les mêmes formules peuvent être utilisées pour calculer l'addition et le doublement de points, ces formules sont dites « unifiées ». C'est un moyen de protection possible pour les cryptosystèmes basés sur l'ECC contre les attaques dites « par observation » (cf.

section 3.1 pour une définition des attaques par observation). D'autres courbes possédant des formules unifiées ont été préalablement mises en évidence (cf. section 4.1.1.2).

En utilisant des coordonnées projectives $\mathcal{P}$, et en prenant $c = 1$, le coût du doublement (ou de l'addition) de points sur ces courbes est de $12M$.

Si la priorité du concepteur de circuit sécurisé se porte sur la recherche de performances élevées plutôt que sur la protection du circuit contre les attaques dites « par observation », il faut noter que le doublement sur les courbes d'Edwards peut être optimisé. En effet, en utilisant la relation 2.11 et à l'aide de quelques manipulations algébriques, le doublement du point $P_1 = (x_1, y_1)$ peut s'écrire si $c = 1$:

$$\begin{cases} x'_3 = \dfrac{(x_1 + y_1)^2}{x_1^2 + y_1^2} - 1 \\ y'_3 = \dfrac{y_1^2 - x_1^2}{2 - x_1^2 - y_1^2}. \end{cases}$$

Sous cette forme, le coût du doublement est ainsi ramené à $7M$: on économise ainsi le calcul de $5M$. En revanche, on perd un attrait important de ces courbes qui est le caractère unifié des formules d'addition et de doublement.

Bernstein *et al.* [BL07b] ont par la suite proposé d'effectuer les opérations de points dans un autre système de coordonnées dit « inversé » (noté $\mathcal{E}$), dans lequel un point P de la courbe représenté par un triplet $(X : Y : Z)$ dans ce système correspond au point affine $(Z/X, Z/Y)$. Le coût du doublement passe de $7M$ à $8M$, tandis que celui relatif à l'addition de point passe de $12M$ à $11M$.

2.2.5 Bilan

Le tableau 2.6 résume le coût de l'addition, du doublement et du triplement sur les courbes présentées précédemment[10].

À cet effet, il peut être constaté que l'arithmétique des courbes d'Edwards sur $\mathcal{P}$ est particulièrement efficace, puisque le doublement et le triplement sur ces courbes est très rapide.

Connaissant le coût des opérations de points sur les courbes de l'état de l'art, il peut être estimé celui relatif au calcul d'un algorithme de multiplication scalaire donné (cf. section 2.2.3). Pour illustrer ce propos, il est détaillé dans le tableau 2.7 le coût d'une multiplication scalaire $Q = [k]P$ sur ces courbes en utilisant $\mathrm{NAF}_2(k)$ traité par fenêtres de longueur fixe (cf. Algorithme 10).

Dans cet exemple précis, ce sont les courbes d'Edwards sur $\mathcal{P}$ qui seront les plus efficaces.

[10]Tous les résultats du tableau 2.6 peuvent être retrouvés sur le site http ://www.hyperelliptic.org/EFD/. Lorsque le coût d'une opération de points n'est pas caractérisé dans l'état de l'art, celui-ci est remplacé par un tiret dans le tableau. Les multiplications par des constantes « petites » (par u pour les courbes DIK2 et DIK3, par d pour les courbes d'Edwards) ont un coût égal à $1M$ dans cette estimation.

Type de courbe	**Coord.**	**Coût de l'A**	**Coût du D**	**Coût du T**
DIK2	$\mathcal{J}$	18M	**7M**	–
DIK3	$\mathcal{J}$	18M	11M	14M
Edwards ($c = 1$)	$\mathcal{P}$	12M	**7M**	**13M**
Edwards ($c = 1$)	$\mathcal{E}$	**11M**	8M	14M
Weierstrass	$\mathcal{J}$	16M	10M	16M
Weierstrass ($a = -3$)	$\mathcal{J}$	16M	8M	14M
Weierstrass	$\mathcal{P}$	14M	12M	–
Weierstrass ($a = -3$)	$\mathcal{P}$	14M	10M	–

TAB. 2.6 – Coût de l'addition, du doublement et du triplement de point sur des courbes de l'état de l'art.

Type de courbe	**Coord.**	**Coût du calcul de$[k]P$**
DIK2	$\mathcal{J}$	$13\ell M$
DIK3	$\mathcal{J}$	$17\ell M$
Edwards ($c = 1$)	$\mathcal{P}$	$\boldsymbol{11\ell M}$
Edwards ($c = 1$)	$\mathcal{E}$	$11{,}66\ell M$
Weierstrass	$\mathcal{J}$	$15{,}33\ell M$
Weierstrass ($a = -3$)	$\mathcal{J}$	$13{,}33\ell M$
Weierstrass	$\mathcal{P}$	$16{,}66\ell M$
Weierstrass ($a = -3$)	$\mathcal{P}$	$14{,}66\ell M$

TAB. 2.7 – Coût du calcul d'une multiplication scalaire $[k]P$ en utilisant $\mathrm{NAF}_2(k)$ traité par fenêtres de longueur fixe.

Il convient de noter que l'utilisation du DBNS (cf. section 2.2.3.3) conduit à l'implantation de $[k]P$ la plus rapide (dans le cadre des courbes génériques sur $\mathbb{F}_p$ exprimées en coordonnées $\mathcal{J}$) pourvu que des formules de triplement efficaces sont utilisées [DIM05]. La version décrite dans [DI06] est, à ce jour, l'algorithme le plus rapide pour calculer $[k]P$ sur une courbe générique définie sur $\mathbb{F}_p$.

2.3 Justification du choix des paramètres des courbes standardisées

Deux organismes sont connus pour breveter des courbes elliptiques aux propriétés intéressantes : le NIST [FIP00] et Certicom© [SEC00b]. Ils proposent tous les deux d'utiliser des courbes obéissant à l'équation de Weierstrass qui utilisent les paramètres a, b et p (cf. relation 2.1).

Si les concepteurs de circuits sécurisés veulent implanter des courbes elliptiques de Weierstrass, ils doivent donc choisir ces trois paramètres en conciliant deux impératifs : résister aux attaques mathématiques et obtenir une arithmétique de courbe efficace.

2.3.1 Résister aux attaques mathématiques

Comme décrit dans la section 2.1.4, la sécurité de l'ECC repose sur l'ECDLP.

La méthode la plus efficace pour résoudre l'ECDLP, appelée attaque « Pollard's ρ » [Pol78], nécessite de l'ordre de $\sqrt{n}$ opérations, où $n = \mathrm{ord}_E(P) = \#E/h$ (le cofacteur h doit absolument être petit ; idéalement $h = 1$)[11]. Par conséquent, savoir évaluer $\#E$ pour un choix de paramètres (a, b, p) donné, et donc compter le nombre de points d'une courbe elliptique est une étape indispensable dans la recherche de courbes cryptographiquement sûres. À cette fin, des algorithmes de comptage de points générique peuvent être utilisés [Sch85] [Sch95].

Il faut noter qu'en plus de compter le nombre de points d'une courbe, d'autres précautions doivent être prises pour le choix des paramètres (notamment p) afin d'éviter certaines attaques spécifiques [HMV04]. Le tableau 2.8 résume les attaques connues sur l'ECDLP.

Attaques connues sur ECDLP	**Contraintes de sécurité**
Recherche exhaustive	n suffisamment grand ($n \geq 2^{80}$)
Pohlig-Hellman et Pollard's ρ [OW99]	$\#\ E(\mathbb{F}_p) = hn$ avec $h \in \{1, 2, 3, 4\}$ et n premier ($n > 2^{160}$)
« Anomalous » [HMV04]	$\#\ E(\mathbb{F}_p) \neq p$
Attaque Menezes-Okamoto-Vanstone [HMV04]	$n \nmid (p^k - 1),\ \forall\ 1 \leq k \leq 20$

Tab. 2.8 – Résumé des attaques connues sur l'ECDLP, et leurs conséquences sur le choix des paramètres de courbes.

Les courbes standardisées [FIP00] [SEC00b] respectent évidemment les contraintes de sécurité dicteés dans le tableau 2.8. Si un concepteur de circuit sécurisé veut utiliser d'autres courbes que celles qui sont standardisées, il doit préalablement s'assurer que les paramètres des courbes qu'il choisit respecte ces contraintes.

[11] Certicom propose depuis 1997 un challenge : l'industriel publie les paramètres d'une courbe elliptique qu'il faut cryptanalyser mathématiquement. Ce challenge a été créé afin que les industriels du secteur soient moins réticents à utiliser les courbes elliptiques, en constatant notamment que l'ECDLP est un problème difficile à résoudre. À titre d'illustration, le challenge Certicom ECCp-109 consistant à résoudre l'ECDLP sur une courbe définie sur un corps premier de longueur 109 bits a été relevé en 2002 par une équipe de contributeurs menée par Monico. Cela a pris 549 jours de calcul avec plus de 10000 stations de travail exécutant en parallèle grâce à Internet la version parallélisée de l'algorithme Pollard's ρ présentée dans [OW99].

2.3.2 Obtenir une arithmétique de courbe efficace

Il a été vu dans la section 2.3.1 que les paramètres a, b et p doivent être choisis tous les trois correctement afin de pouvoir résister aux attaques mathématiques. Dans cette section, il est également montré que le choix individuel des paramètres a et p peut conduire à des arithmétiques de courbe efficaces.

2.3.2.1 Bien choisir le paramètre a

Les courbes standardisées [FIP00] [SEC00b] proposent d'utiliser des courbes elliptiques obéissant à l'équation de Weierstrass avec $a = -3$. Ce choix est justifié par le fait que le calcul du doublement en coordonnées $\mathcal{J}$ et $\mathcal{J}^c$ se trouve délesté d'une multiplication modulaire.

Choisir une courbe avec $a = -3$ peut *a priori* apparaître comme une restriction. Cette section montre également que ce n'est pas le cas.

Le calcul du doublement en coordonnées $\mathcal{J}$ (cf. exemple 2.1.) et en coordonnées $\mathcal{J}^c$ induit une multiplication par la constante a. Or, si cette constante est choisie astucieusement, le doublement peut s'effectuer plus efficacement, en économisant des multiplications modulaires.

En particulier, si $a = -3$, alors l'expression $3X_1^2 + aZ_1^4$ (cf. exemple 2.1., relation 2.6), qui conduit initialement au calcul de $3M$, peut être calculée en seulement $2M$:

$$3X_1^2 - 3Z_1^4 = 3(X_1 - Z_1^2)(X_1 + Z_1^2).$$

Ainsi, en utilisant cette nouvelle formulation, le coût du doublement en coordonnées $\mathcal{J}$ (respectivement en coordonnées $\mathcal{J}^c$) est ramené à $8M$ ($9M$).

Choisir $a = -3$ permet donc d'obtenir une arithmétique des courbes efficace. Reste à savoir si ce choix se fait au détriment d'une perte de généralité. Brier *et al.* [BJ03b] ont montré que pour la plupart des courbes choisies aléatoirement $E_1 : y^2 = x^3 + ax + b$, il existe une isogénie φ de petit degré (cf. 2.2.4.1 pour la définition d'une isogénie), telle que $\varphi : E_1 \to E_2$, avec $E_2 : y^2 = x^3 - 3x + b'$. La conséquence directe de cette observation est que si un concepteur de circuit sécurisé choisit d'implanter l'arithmétique d'une courbe obéissant à l'équation E_2, elle pourra être utilisée pour implanter celle relative à une courbe d'équation E_1.

Toutes ces raisons justifient donc l'utilisation de courbes de Weierstrass, avec $a = -3$.

2.3.2.2 Bien choisir le paramètre p

Dans la section 1.2.2, il a été montré que la réduction modulaire modulo un nombre premier de Mersenne (généralisé) est très efficace.

C'est ces nombres aux propriétés particulières que les organismes standardisant des courbes elliptiques proposent d'utiliser [FIP00] [SEC00b] : $p_{192} = 2^{192} - 2^{64} - 1$, $p_{224} = 2^{224} - 2^{96} + 1$, $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, $p_{521} = 2^{521} - 1$.

2.3.3 Synthèse

Afin d'obtenir des courbes à l'arithmétique performante (cf. section 2.3.2), on peut utiliser des courbes de Weierstrass, avec p étant un premier de Mersenne, et $a = -3$. De plus, afin de résister aux attaques mathématiques, il faut que ces courbes munies d'un paramètre b adéquat résistent aux attaques détaillées dans le tableau 2.8.

Les courbes standardisées obéissent à ce raisonnement [FIP00] [SEC00b].

Exemple 2.4. La courbe P-192 standardisée par le NIST d'équation $E : y^2 = x^3 + ax + b$ définie sur $\mathbb{F}_p$ a comme paramètres $a = -3$ et $p = 2^{192} - 2^{64} - 1$ pour les raisons évoquées dans les sections 2.3.2.1 et 2.3.2.2. Pour résister aux attaques mathématiques décrites dans le tableau 2.8, il est choisi :
$b = (64210519\ E59C80E7\ 0FA7E9AB\ 72243049\ FEB8DEEC\ C146B9B1)_{16}$.

2.4 Protocoles

Un protocole cryptographique est un algorithme défini par une séquence d'étapes spécifiant précisément les actions nécessaires pour plusieurs parties communicantes afin d'atteindre un des objectifs cryptographique suivant :

1. la confidentialité,
2. l'intégrité des données,
3. l'authentification de l'origine des données,
4. l'authentification des parties communicantes,
5. la non-répudiation.

En plus de ces cinq objectifs principaux, on peut en citer d'autres qui sont complémentaires, comme par exemple la signature numérique. Cette dernière est très importante notamment pour l'authentification (de l'origine des données ou des parties communicantes), ainsi que pour la non-répudiation. La signature numérique consiste à relier une partie communicante à de l'information qu'elle transmet.

Pour illustrer ce qu'est un protocole cryptographique, il est présenté dans cette section l'algorithme de signature numérique sur courbes elliptiques (*Elliptic Curve Digital Signature Algorithm*, ECDSA) [HMV04], qui est le protocole de signature utilisant les courbes elliptiques le plus largement standardisé.

Un protocole de signature nécessite deux clés, l'une privée d, l'autre publique Q. La clé privée d permettra à la première partie communicante de produire une signature, la

clé publique Q permettra à une seconde partie communicante de vérifier cette signature. La génération de ces clés est décrite dans l'algorithme 13.

Algorithme 13 Génération des clés publique et privée pour l'ECDSA

Entrées : un point P d'une courbe elliptique E, avec $n = \mathrm{ord}_E(P)$.
Sorties : une clé publique Q et une clé privée d.

1. Choisir aléatoirement $d \in [1, n-1]$
2. $Q \leftarrow [d]P$
3. **retourner** (Q, d)

Si une première partie communicante veut produire la signature ECDSA (r, s) d'un message m avec sa clé privée d, elle utilisera l'algorithme 14. Cet algorithme utilise le calcul d'une multiplication scalaire d'un point de base P, avec $n = \mathrm{ord}_E(P)$ (étape 2). De plus, cet algorithme de signature a également besoin d'une fonction de hâchage H (étape 7).

Algorithme 14 Signature ECDSA

Entrées : un message m et une clé privée d.
Sorties : une signature (r, s) du message m.

1. Choisir aléatoirement $k \in [1, n-1]$
2. $[k]P \leftarrow (x_1, y_1)$
3. $r \leftarrow x_1 \pmod{n}$
4. **si** $r = 0$ **alors**
5. Aller à l'étape 1
6. **finsi**
7. $e \leftarrow H(m)$
8. $s \leftarrow k^{-1}(e + dr) \pmod{n}$
9. **si** $s = 0$ **alors**
10. Aller à l'étape 1
11. **finsi**
12. **retourner** (r, s)

Si une seconde partie communicante veut vérifier la signature ECDSA (r, s) du message m produite par la première partie communicante, elle pourra le faire munie de la clé publique Q en utilisant l'algorithme 15.

Preuve que la signature est bien vérifiée. À la fin de l'étape 8 de l'algorithme 14, on a :

$$s \equiv k^{-1}(e + dr) \pmod{n}. \tag{2.12}$$

La relation 2.12 peut également s'écrire $k \equiv s^{-1}(e + dr) \pmod{n}$, soit encore :

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d \pmod{n}$$

Algorithme 15 Vérification ECDSA

Entrées : un message m, une clé publique Q et une signature (r, s).
Sorties : « Acceptation » ou « Rejet ».

1. **si** $r, s \notin [1, n-1]$ **alors**
2. « Rejet »
3. **finsi**
4. $e \leftarrow H(m)$
5. $w \leftarrow s^{-1} \pmod{n}$
6. $u_1 \leftarrow ew \pmod{n}, u_2 \leftarrow rw \pmod{n}$
7. $X = (x_1, y_1) \leftarrow [u_1]P + [u_2]Q$
8. **si** $X = \infty$ **alors**
9. « Rejet »
10. **finsi**
11. $v \leftarrow x_1 \pmod{n}$
12. **si** $v = r$ **alors**
13. « Acceptation »
14. **sinon**
15. « Rejet »
16. **finsi**

Ainsi, on calcule à l'étape 7 de l'algorithme 15 :

$$X = [u_1]P + [u_2]Q = ([u_1] + [u_2 d])P = [k]P$$

Donc, il faut bien vérifier à l'étape 12 de l'algorithme 15 que $v = r$.

Concevoir un protocole cryptographique sécurisé ou amener la preuve de la sécurité d'un protocole est un exercice difficile. On peut raisonnablement penser que la sécurité d'un protocole repose sur celle des routines qu'il contient : dans le cadre de l'ECDSA, le choix de la courbe elliptique et de la fonction de hâchage doit être pertinent.

Cependant, dans le cadre plus large de la conception d'un protocole cryptographique, ce n'est pas suffisant : il faut aussi vérifier que des conditions particulières sur les variables mises en jeu dans ce protocole ne remette pas en cause sa sécurité globale.

Par exemple, dans l'algorithme 14, le paramètre k mis en jeu dans la multiplication scalaire de l'étape 2 doit être changé à chaque signature. Cette condition trouve sa justification dans ce qui suit.

Si un attaquant utilise le même k pour générer deux signatures ECDSA (r, s_1) et (r, s_2) de deux messages m_1 et m_2, si $s_1 \not\equiv s_2 \bmod n$, il peut trouver k grâce à :

$$k \equiv (s_1 - s_2)^{-1}(e_1 - e_2) \bmod n. \tag{2.13}$$

Or, d'après la relation 2.12, l'attaquant peut retrouver la clé privée d grâce à

$$d = r^{-1}(ks - e) \bmod n. \tag{2.14}$$

De la même manière, une vérification est placée à l'étape 4 de l'algorithme 14 car si un attaquant induit le cas $r = 0$, sa signature peut être acceptée.

D'autres protocoles utilisant des courbes elliptiques sont couramment utilisés :

- pour le chiffrement de données en utilisant le système de chiffrement intégré pour courbes elliptiques (*Elliptic Curve Integrated Encryption System*, ECIES) [HMV04],
- pour l'échange de clés en utilisant le processus de Diffie-Hellman pour courbes elliptiques (*Elliptic Curve Diffie-Hellman*, ECDH) [HMV04],
- pour la génération de clés en utilisant le protocole pour courbes elliptiques de Menezes-Qu-Vanstone (*Elliptic Curve Menezes-Qu-Vanstone*, ECMQV) [HMV04].

2.5 Conclusion de chapitre

Il a pu être constaté dans ce chapitre que les courbes elliptiques sont des objets géomètriques et algébriques intéressants pour faire de la cryptographie.

À ce propos, l'opération à effectuer afin de concevoir un cryptosystème basé sur les courbes elliptiques est la multiplication scalaire : la sécurité de ce cryptosystème est donc basé sur l'ECDLP, problème mathématique difficile à résoudre si les paramètres de la courbe sont correctement choisis.

L'arithmétique des courbes elliptiques est également très riche.

Il a été notamment montré que le choix pertinent d'un système de coordonnées (projectives ou mixtes) permet d'accélérer le calcul sur ces courbes.

De même, vu que les algorithmes de multiplication scalaire consistent, pour la plupart, en une série de doublements entrecoupée d'additions, une autre source d'optimisations réside potentiellement dans le calcul de la quantité $(Q + P) + Q$ au lieu de $[2]Q + P$, en économisant des calculs de variables intermédiaires.

Afin de calculer le résultat d'une multiplication scalaire, des algorithmes bien plus efficaces que celui basique du « doublement-et-addition » existent : ceux notamment basés sur le recodage NAF_w ou le DBNS du scalaire sont très performants.

Les courbes standardisées obéissent à l'équation de Weierstrass (cf. relation 2.1). Le choix des trois paramètres de courbe a, b et p est justifiable facilement : pour des raisons d'efficacité, $a = -3$ (il faut noter que le choix de ce paramètre n'est pas restrictif) et p est un nombre premier de Mersenne. Le paramètre b, quant à lui, est choisi pour résister aux attaques mathématiques sur l'ECDLP.

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. Il faut noter que ces deux types de courbes possèdent une arithmétique qui ont une résistance intrinsèque contre les attaques dites

« par observation » (cf. section 3.1 pour une définition des attaques par observation).

Enfin, il a pu être constaté que bâtir un protocole cryptographique basé par exemple sur les courbes elliptiques est un art difficile.

Chapitre 3

Attaques

Sommaire

Les conditions pour qu'un cryptosystème embarqué basé sur l'ECC soit sûr d'un point de vue mathématique sont bien connues par les concepteurs de circuits sécurisés (cf. section 2.3.1). Cependant, des implantations matérielles de ces cryptosystèmes peuvent faire l'objet de manipulations frauduleuses, appelées communément attaques.

Dans le cas général, il a été mis en évidence des attaques :

– par observation,
– par perturbation.

Ces deux types d'attaques peuvent être également suivant les cas :

– non-invasives,
– semi-invasives,
– totalement invasives.

Ce chapitre détaille dans le cas général, puis dans le cas particulier de l'ECC, comment ces attaques peuvent permettre de retrouver la clé secrète k. Il montre ainsi que ces attaques sont de véritables menaces : les concepteurs de circuits sécurisés doivent donc les prendre au sérieux.

3.1 Attaques par observation

L'observation de certaines caractéristiques d'un cryptosystème (comme sa consommation électrique par exemple) durant la manipulation de la clé secrète peut permettre de récupérer cette dernière. Ces caractéristiques sont appelées « canaux cachés ». Le premier document officiel relatant une utilisation d'un canal caché date de 1956. Wright [Wri87] raconte que le MI5, le service de renseignement britannique pour lequel il a travaillé, a concentré ses efforts pour « casser » une machine servant à chiffrer des textes clairs à l'ambassade égyptienne de Londres. Cette machine à rotors était de type « Hagelin » (cf. figure 3.1).

FIG. 3.1 – Machine à rotors de type « Hagelin » (source : http ://home.ca.inter.net/ hagelin/crypto.html).

Casser la machine d'un point de vue mathématique excédait les capacités de calcul du MI5. Aussi, Wright suggéra de placer un micro près de cette machine afin d'espionner les « clics » produits par celle-ci. Wright découvrit ainsi que ces clics lui permettait de déterminer la position relative de quelques rotors de la machine. Cette information

supplémentaire permit au MI5 de diminuer drastiquement le nombre de calculs à effectuer pour casser cette machine, et il fût ainsi capable d'espionner les communications de cette ambassade pendant des années.

Ainsi, un concept nouveau en cryptanalyse était né : un cryptosystème (ici, la machine à rotors de type « Hagelin ») qui émet un canal caché (ici, des clics) dépendant de la clé secrète peut permettre à des attaquants (ici, le MI5) de la récupérer.

3.1.1 Les canaux cachés

Il a été montré que les circuits implantant des algorithmes cryptographiques peuvent émettre des canaux cachés de différentes natures. Les principaux canaux cachés mis en évidence sont :

- la consommation électrique par Kocher *et al.* [KJJ99],
- le rayonnement électromagnétique par Gandolfi *et al.* [GMO01] et Quisquater *et al.* [QS01],
- le temps de calcul par Kocher [Koc96].

Il n'est pas exclu que d'autres canaux cachés soient mis en exergue dans un proche avenir, comme le son émis par un cryptosystème[12] par exemple.

3.1.1.1 La consommation électrique

Kocher *et al.* [KJJ99] ont montré que la consommation électrique d'un cryptosystème est un canal caché.

La plupart des cryptosystèmes implantés matériellement utilisent la logique complémentaire pour semiconducteurs métal-oxyde (*Complementary Metal Oxide Semiconductor*, CMOS), qui est la technologie semiconducteur dominante notamment dans la fabrication de microprocesseurs, de mémoires et de circuits intégrés à applications spécifiques (*Application Specific Integrated Circuits*, ASIC) [Fan06]. Le composant le plus simple dans cette logique est l'inverseur (ou porte NOT, cf. Figure 3.2).

L'inverseur est constitué de deux transistors, l'un de type P, l'autre de type N, qui peuvent être assimilés à des interrupteurs contrôlés en tension (cf. Figure 3.2, à gauche). Un signal haute (respectivement basse) tension est interprété comme un « 1 » (« 0 ») logique.

Si la tension A est à l'état bas, c'est le transistor de type P qui conduit (c'est-à-dire qu'il se comporte comme un interrupteur fermé) tandis que le transistor de type N ne conduit pas ; dans ce cas, il y a un court-circuit qui se crée entre la tension d'alimentation et la sortie, par conséquent Y est à l'état haut (cf. Figure 3.2, au milieu). Inversement, si A est à l'état haut, alors le transistor de type P ne conduit pas, tandis que le transistor de

[12] Il est connu que les anciens processeurs des ordinateurs, suivant qu'ils calculent ou non, émettent un son d'une amplitude plus ou moins élevée. Une augmentation de la consommation du processeur fait vibrer la puce plus fort aux harmoniques de l'horloge. Le site http ://people.csail.mit.edu/tromer/acoustic/ montre que le son produit par un processeur d'ordinateur peut être analysé afin de trouver les différentes opérations exécutées par celui-ci. Il reste maintenant à montrer que cette attaque dite « par analyse du son émis » peut être appliquée sur des systèmes embarqués, comme les cartes à puce par exemple.

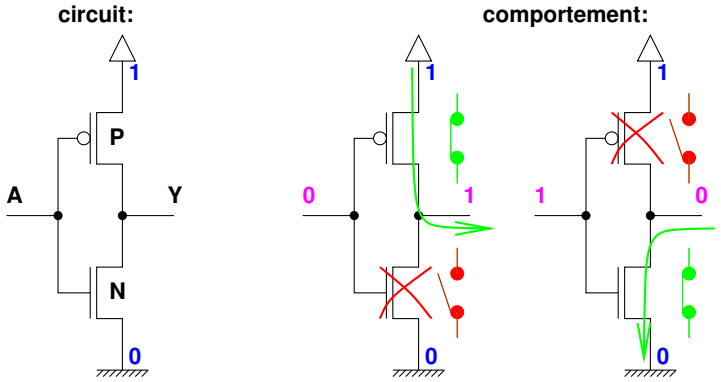

FIG. 3.2 – Le schéma de l'inverseur (à gauche), et son comportement (à droite).

type N conduit ; dans ce cas, il y a un court-circuit qui se crée entre la masse et la sortie, et donc Y est à l'état bas (cf. Figure 3.2, à droite). Les portes logiques plus complexes (par exemple OR, AND, etc.) ont un comportement global similaire.

Ainsi, pendant un cycle d'horloge, et dans un circuit CMOS constitué de nombreuses portes logiques, une grande quantité de transistors commutent de la sorte, et le nombre total de ces commutations dépend de l'opération exécutée et des valeurs utilisées dans le cadre de cette opération. Par conséquent, il peut être attendu que la puissance consommée par le circuit change continuellement lorsque le circuit exécute une suite complexe d'opérations dépendant de la clé secrète. Ainsi, en examinant des courbes de consommation ayant éventuellement subi un traitement statistique, un attaquant peut espérer trouver de l'information sur la clé secrète.

La consommation électrique d'un circuit est proportionnelle à l'intensité du courant qui y circule. L'intensité du courant peut quant à elle être mesurée à l'aide d'une faible résistance placée en série avec la tension d'alimentation et en utilisant un oscilloscope numérique pour mesurer la différence de tension aux bornes de cette résistance. Un ordinateur est également requis pour l'acquisition des courbes de consommation. Une extension de circuit peut également être nécessaire pour communiquer avec celui-ci (cf. Figure 3.3).

L'ensemble de la chaîne d'acquisition doit être la meilleure possible afin de limiter le bruit inhérent aux mesures effectuées.

Pour les attaques nécessitant un très grand nombre d'acquisitions de courbes (telles que les attaques différentielles, cf. section 3.1.3), un serveur est souvent utilisé pour recevoir, enregistrer et faire subir un procédé de traitement de signal aux courbes envoyées par l'ordinateur principal. C'est lui également qui s'occupe du traitement statistique des courbes, et donc de l'attaque proprement dite.

Ainsi, ce type d'attaque est tout-à-fait réalisable de par sa relative simplicité et de par son coût : c'est donc une menace potentiellement très importante qui pèse sur les cryptosystèmes.

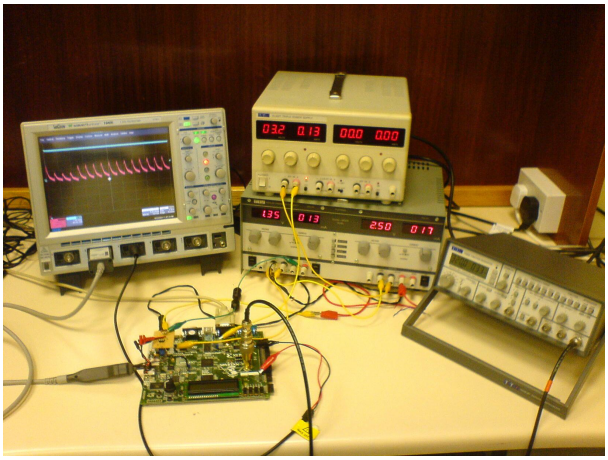

FIG. 3.3 – Dispositif pour une attaque par analyse de la consommation électrique.

Enfin, il faut noter que les attaques utilisant la mesure de la consommation électrique (*Power Analysis*, PA) sont non-invasives, c'est-à-dire qu'elles ne nécessitent pas de modifier ou de détruire tout ou partie du cryptosystème.

3.1.1.2 Le rayonnement électromagnétique

Les cercles militaires exploitent depuis longtemps les phénomènes électromagnétiques afin de récupérer des informations secrètes. Cette information a été rendue publique seulement au début du vingt-et-unième siècle grâce à des documents écrits par l'agence de sécurité nationale américaine (*National Security Agency*, NSA) qui ont été déclassés[13]. Des chercheurs n'ont pas attendu que la NSA divulgue au grand public cette information pour mener des recherches sur ce phénomène. Notamment, Van Eck dans les années 80 [Eck85] et Kuhn *et al.* dans les années 90 [KA98] ont montré tour à tour que l'on peut reconstituer l'image inscrite sur un écran vidéo en captant le rayonnement électromagnétique produit par celui-ci.

C'est la circulation du courant à travers un circuit CMOS qui cause son rayonnement ElectroMagnétique (EM). Un attaquant peut ainsi espérer récupérer des informations sur les opérations exécutées par le cryptosystème (dépendantes de la clé privée) ainsi que sur les variables mises en jeu en analysant ces signaux EM.

Le rayonnement EM émis par un cryptosystème peut être mesuré **globalement** ou **localement**. Lorsqu'un attaquant veut effectuer une mesure **globale**, il peut entourer l'ensemble du cryptosystème par une sonde en forme de boucle (cf. Figure 3.4) : il effectue donc des mesures dites « en champ proche ». Il peut également effectuer des mesures dites « en champ lointain » à plusieurs mètres du cryptosystème [AARR02]. Dans ce cas, des

[13]Certains de ces documents peuvent être trouvés sur le site : http ://cryptome.info/0001/nacsim-5000.htm.

antennes pourront être utilisées. Lors d'une mesure **locale**, une sonde est placée à l'endroit précis où l'on veut mesurer le rayonnement (cf. Figure 3.4).

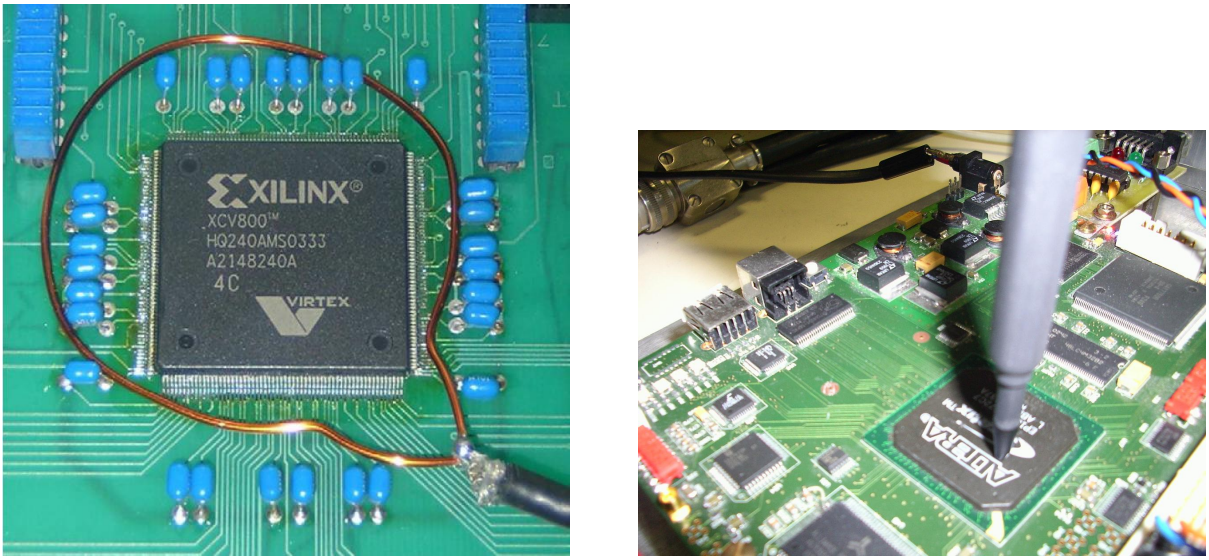

FIG. 3.4 – Dispositif pour une attaque globale (à gauche, [MOPV07]) et locale (à droite, source : http ://www.Secure-IC.com/) par analyse du rayonnement électromagnétique.

Il est à noter que si un attaquant veut obtenir une topologie très précise du rayonnement EM émis par le cryptosystème, le dispositif de mesures **locales** est préféré au dispositif de mesures **globales**.

Il existe deux classes principales de rayonnement EM : il y a d'une part un rayonnement uniquement résultat de la circulation du courant dans le circuit, et qui peut donc s'exprimer relativement facilement à l'aide des relations fondamentales de l'électromagnétisme (notamment les équations de Maxwell, les lois d'Ampère et de Faraday), et d'autre part des émanations additionnelles (effets de couplage) principalement causées par la miniaturisation et la complexité des circuits CMOS modernes. L'attaquant peut toutefois faire l'hypothèse que le rayonnement EM d'un cryptosystème obéit à un modèle relativement simple : il peut considérer qu'une sonde placée à une distance très proche du circuit (notée d) mesure **un champ électromagnétique proportionnel au courant circulant dans le circuit** (en fait, ce champ mesuré depend également de la dérivée du courant par rapport au temps). Ce champ est également inversement proportionnel à d : il faut donc que la sonde soit très proche du circuit afin que le rayonnement EM mesuré soit le plus grand possible.

Bien qu'une sonde EM détecte des signaux provenant de différentes parties du cryptosystème, ceux-ci peuvent ensuite être séparés et analysés individuellement [AARR02]. Ce qui n'est pas le cas des attaques PA, dans lesquelles la puissance électrique mesurée est l'agrégation des puissances électriques consommées par toutes les unités actives du circuit. Par conséquent, les attaques par analyse du rayonnement électromagnétique (*ElectroMagnetic Analysis*, EMA) peuvent potentiellement révéler plus d'informations que les

attaques PA, et constituent par conséquent une menace plus importante. Par exemple, des accès mémoire, des comparaisons d'octets qui n'étaient pas visibles sur un relevé de consommation électrique peuvent être discernés lors d'une attaque EMA [AARR02].

Les attaques EMA, comme les attaques PA (cf. section 3.1.1.1), peuvent être montées à l'aide d'un équipement relativement peu coûteux [GMO01] [QS01]. Il faut tout d'abord du matériel servant à la mesure du rayonnement EM : sondes en forme de boucle ou de solénoïde (fil enroulé en forme d'hélice) en cuivre (ou en or pour des sondes plus fines) de quelques millimètres de longueur [GMO01]. Il faut ensuite des composants servant à faire du traitement analogique des courbes obtenues : un oscilloscope, un analyseur de spectre, des amplificateurs. Dans le cas où l'attaquant effectue une mesure locale du rayonnement EM (cf. Figure 3.4), un système permettant d'effectuer des mouvements précis est requis. Enfin, un ordinateur principal est utilisé pour acquérir les différents relevés EM, et un serveur pourra être utilisé pour traiter ces courbes lors d'attaques plus complexes (telles que les attaques différentielles, cf. section 3.1.3).

Une attaque EMA est donc tout-à-fait réalisable de par la simplicité du concept de base sur laquelle elle repose. Cependant, si un attaquant veut améliorer l'efficacité d'une telle attaque, il devra investir dans une sonde et un amplificateur de qualité, et donc, relativement coûteux. Afin de diminuer le bruit présent dans les mesures, un attaquant pourra également acquérir des dispositifs supplémentaires, comme par exemple des filtres, ou encore un réseau de stabilisation d'impédance de ligne (*Line Impedance Stabilization Network*, LISN). Suivant ses moyens, il pourra également se munir de démodulateurs d'amplitude et/ou de fréquence afin d'affiner encore un peu plus les mesures [AARR02].

Enfin, il faut observer que l'attaque EMA est par défaut une attaque non-invasive.

3.1.1.3 Le temps de calcul

Il arrive que des codes embarqués dans des cryptosystèmes contiennent des branchements conditionnels dépendant de la clé secrète du type « **si**$\{k_i = 1\}$, **alors** {Procédure A}, **sinon**{Procédure B} », où les deux procédures A et B ont un temps d'exécution différent.

De plus, les opérations arithmétiques mises en jeu dans ces procédures peuvent être implantées de telle sorte que leur temps d'exécution dépend de la valeur des opérandes mises en jeu. Par exemple, dans l'algorithme de Montgomery (cf. Algorithme 8), il peut être montré qu'à la fin de l'étape 5, le résultat S appartient à l'intervalle $[0, 2p[$: une soustraction au maximum est alors nécessaire afin de réduire S dans l'intervalle $[0; p[$.

Par conséquent, sous ces conditions, un attaquant qui est capable de mesurer très précisément le temps qu'un circuit prend pour exécuter des opérations cryptographiques (par exemple la génération d'une signature) peut analyser les mesures obtenues afin d'en déduire des informations sur la clé secrète. L'approche de cette attaque dite « par analyse du temps de calcul » (*Timing Attack*, TA) a tout d'abord été mise en évidence sur des implantations logicielles du RSA par Kocher [Koc96] puis mise en pratique sur carte à puce par Dhem *et al.* [DKL$^+$00].

Quelques conditions doivent être remplies pour que cette attaque non-invasive soit

vraiment efficace. Tout d'abord, elle nécessite pour l'attaquant d'avoir une certaine connaissance de l'implantation matérielle et logicielle du cryptosystème. Ensuite, l'attaquant doit pouvoir choisir les messages d'entrée que le cryptosystème va signer. De plus, il faut noter que ce type d'attaque nécessite un grand nombre de mesures : il faut donc que l'attaquant trouve le moyen d'automatiser son attaque. Enfin, les mesures de temps de calcul doivent être très précises : la marge d'erreur sur ces mesures ne doit pas excéder quelques cycles d'horloge.

Il faut remarquer qu'aucune attaque par analyse des temps de calcul n'est possible sur les cryptosystèmes ECC si toutes les opérations s'effectuant à tous les niveaux de hiérarchie (opérations modulaires, opérations de points, multiplication scalaire, protocole cryptographique) s'exécutent à temps constant, et ce, quelque soit la valeur de la clé.

3.1.1.4 Messages d'erreur

Les attaques dites « par analyse des messages d'erreur » visent généralement les cryptosystèmes implantant une procédure de déchiffrement ou de validation de signature. Par exemple, en utilisant la procédure de validation de l'ECDSA (cf. Algorithme 15), une partie communicante peut valider la signature d'un homologue (calculée à l'aide de l'algorithme 14). Dans cet exemple précis, le cryptosystème renvoie lors de la procédure de validation un message, selon qu'il accepte la signature ou qu'il la rejette. Ainsi, si un attaquant connaît la raison d'un rejet, il peut retrouver de l'information sur la clé secrète en envoyant au cryptosystème des messages correctement choisis.

Cette attaque non-invasive a été décrite pour la première fois par Bleichenbacher sur le standard de chiffrement RSA PKCS# 1 [Ble98].

Il faut noter que seules quelques implantations particulières de courbes elliptiques sont susceptibles d'être attaquées de la sorte (cf. section 4.1.1.2).

3.1.1.5 Combiner des canaux cachés

Chaque canal caché pris individuellement peut amener le cas échéant de l'information sur la clé secrète. Il est également possible pour un attaquant d'en combiner plusieurs afin d'améliorer potentiellement l'efficacité de son attaque.

Par exemple, le temps de calcul et la consommation électrique sont connus pour être efficaces lorsqu'ils sont associés : des mesures de consommation électrique du cryptosystème peuvent être utilisées afin de récupérer des mesures très précises de temps de calcul d'opérations intermédiaires (si celles-ci sont clairement visibles sur les mesures de consommation électrique). Ces mesures de consommation électrique peuvent donc être utilisées dans le cadre de l'amélioration d'une TA, puisque des mesures de temps globales sont remplacées par des mesures locales.

Walter *et al.* [WT01] et Schindler [Sch02] ont les premiers exploité cette combinaison pour attaquer un cryptosystème calculant le RSA à l'aide de l'algorithme de Montgomery (cf. Algorithme 8).

Une fois défini les différents canaux cachés dans le cas général, il est montré dans

la prochaine section comment ils peuvent être utilisés pour attaquer spécifiquement un cryptosystème ECC.

3.1.2 Les attaques simples

Une attaque par analyse des canaux cachés peut être simple ou différentielle. Le premier type d'attaque est potentiellement le plus dangereux, car il permet théoriquement de déterminer la clé d'un cryptosystème à l'aide de l'enregistrement d'un seul motif du canal caché mesuré.

3.1.2.1 Description

La consommation électrique ou le rayonnemment électromagnétique d'un cryptosystème est différent suivant l'opération exécutée et les opérandes manipulées. Par exemple, une multiplication modulaire nécessite plus de cycles d'horloge qu'une addition, et cette différence pourra être visible sur un relevé de canal caché. Vu que les variantes d'implantations sont limitées et connues pour un algorithme donné, un attaquant peut retrouver la structure de l'implantation d'un algorithme cryptographique.

Les implantations des algorithmes de multiplication scalaire sont particulièrement vulnérables face aux attaques par observation dites « simples » car les formules utilisées pour l'addition et le doublement sont différentes (plus précisément la formule calculant le paramètre λ, cf. relation 2.2), et le chemin d'exécution des algorithmes de multiplication scalaire présentés en section 2.2.3 est déterminé par les bits de clé secrète. Par conséquent, dans ces conditions, le cryptosystème engendre des traces de canaux cachés qui peuvent être facilement distinguables. La figure 3.5 représente un relevé de la consommation électrique d'un cryptosystème ECC, effectuant une multiplication scalaire avec l'algorithme du « doublement-et-addition » (cf. Algorithme 9).

L'attaquant peut discerner les traces de consommation engendrées par les opérations de doublement et d'addition de points et ainsi retrouver la clé, car le motif {doublement + addition} (respectivement le motif {doublement} seul) correspond à un bit de clé traité égal à 1 (0). Les bits de clé ainsi trouvés sont précisés sur la figure 3.5.

On peut également se placer dans le cadre plus large de l'utilisation de la multiplication scalaire dans un protocole. Par exemple, si le scalaire k est trouvé à l'aide d'une attaque simple par analyse de canaux cachés (*Simple Side-Channel Analysis*, SSCA), il peut être retrouvé facilement la clé privée d utilisée lors d'une signature ECDSA grâce à la relation 2.14. Les attaques SSCA sont donc des menaces très sérieuses pesant sur les cryptosystèmes ECC auxquelles les concepteurs de circuits sécurisés doivent impérativement se prémunir (cf. section 4.1.1).

Les attaques SSCA peuvent également s'appliquer sur des implantations d'algorithmes de multiplication scalaire plus sophistiquées, comme par exemple ceux utilisant le $\mathrm{NAF}_w(k)$ (cf. Algorithme 10). Si les relevés du canal caché choisi révèlent le motif {doublement} (respectivement {doublement + addition}), un attaquant peut récupérer les chiffres du $\mathrm{NAF}_w(k)$ qui (ne) sont (pas) égaux à 0. Cela mène donc à de l'information substantielle

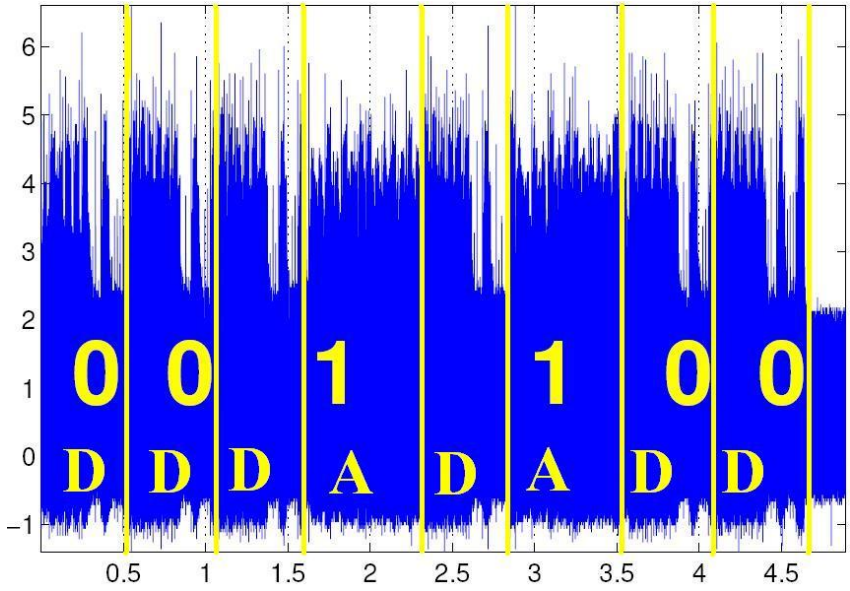

FIG. 3.5 – Un relevé de consommation électrique de l'algorithme du « doublement-et-addition » [MOPV07].

sur k, information qui peut suffire à retrouver entièrement la clé privée d utilisée lors d'une signature ECDSA [NS03].

Si un attaquant connaît l'implantation de l'algorithme cryptographique, cela lui facilite la tâche pour monter une attaque SSCA. En revanche, si il a peu d'informations, il devra s'en remettre à son expérience afin de mener à bien son attaque.

3.1.2.2 Quand les attaques « simples » sont plus compliquées

Afin qu'une attaque SSCA soit facile à réaliser pour un attaquant, chaque motif de canal caché qu'il reconnaît doit correspondre à un bit de clé secrète. Malheureusement pour lui, dans certains algorithmes de multiplication scalaire, le lien entre les opérations exécutées et les bits de clé n'est pas évident. C'est par exemple le cas de l'algorithme proposé par Morain *et al.* [MO90].

Pour pratiquer une attaque SSCA sur ce type d'algorithme de multiplication scalaire, Oswald [Osw02] propose de l'assimiler à un modèle de Markov [GS92]. Un modèle de Markov ressemble à un graphe d'états dans lequel chaque transition d'un état à un autre est conditionnée par la valeur d'une variable (ici, la valeur du bit de clé courant) : le prochain état est dépendant seulement de l'état courant, et indépendant de la manière dont a été produit l'état courant. On dit qu'un modèle de Markov est « sans mémoire ».

L'attaque proposée par Oswald peut être découpée en quatre étapes distinctes.

1. Des **précalculs** doivent d'abord être effectués. Notamment, le modèle de Markov de l'algorithme doit être trouvé, ce qui revient à tracer son graphe d'état. Ensuite, il

doit être calculé la matrice des transitions T qui regroupe les probabilités de passer d'un état à un autre. Ainsi, en considérant que le graphe obtenu contient n états différents S_0, S_1, $\cdots$, S_{n-1}, alors il doit être calculé :

$$T = \begin{pmatrix} \mathrm{P}(s_{i+1} = S_0 | s_i = S_0) & \mathrm{P}(s_{i+1} = S_0 | s_i = S_1) & \cdots & \mathrm{P}(s_{i+1} = S_0 | s_i = S_{n-1}) \\ \mathrm{P}(s_{i+1} = S_1 | s_i = S_0) & \mathrm{P}(s_{i+1} = S_1 | s_i = S_1) & \cdots & \mathrm{P}(s_{i+1} = S_1 | s_i = S_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{P}(s_{i+1} = S_{n-1} | s_i = S_0) & \mathrm{P}(s_{i+1} = S_{n-1} | s_i = S_1) & \cdots & \mathrm{P}(s_{i+1} = S_{n-1} | s_i = S_{n-1}) \end{pmatrix}$$

où $\mathrm{P}(s_{i+1} = S_r | s_i = S_s)$ représente la probabilité conditionnelle d'aller de l'état S_s sachant que l'on part de l'état S_r $(0 \leq r, s \leq n-1)$. L'attaquant doit également calculer le vecteur propre $\pi = \begin{pmatrix} \pi_0 & \pi_1 & \cdots & \pi_{n-1} \end{pmatrix}$ qui obéit aux relations :

$$\begin{aligned} \sum_{i=0}^{n-1} \pi_i &= 1, \\ \pi T &= \pi. \end{aligned}$$

Enfin, avec l'aide des matrices T et π, l'attaquant doit calculer les probabilités conditionnelles

$$\mathrm{P}(Y = y | X = x)$$

pour un grand nombre de combinaisons possibles de X et de Y, X représentant une séquence d'opérations de points (D, A, ou autres) discernable dans le relevé du canal caché choisi, et Y une séquence de bits de la clé.

2. L'algorithme de multiplication scalaire doit ensuite être **attaqué à l'aide d'une attaque SSCA**. À l'issu de cette phase, l'attaquant doit être capable de distinguer les différentes opérations de points effectuées.
3. L'attaquant doit dans une troisième phase **analyser le résultat de l'attaque SSCA**. Il doit découper la séquence d'opérations obtenue dans la deuxième phase en une série de sous-séquences de longueur l, ces dernières devant être choisies de telle sorte qu'elles ont pu être produites par une séquence de bits de la clé. L'attaquant doit s'assurer que le découpage qu'il a effectué est bien valide : le nombre total de bits qui peut être obtenu grâce à ce partitionnement ne doit pas dépasser le nombre de bits de k.
4. Enfin, l'attaquant doit **tester toutes les séquences possibles de bits de la clé**, dans l'ordre décroissant de leurs probabilités conditionnelles.

Ainsi, en suivant cette procédure, et en reprenant l'exemple de l'algorithme de multiplication scalaire décrit dans [MO90], il peut être montré qu'il faut tester en moyenne 2^{18} hypothèses de clé (pour $\ell = 192$ et $l = 16$) ce qui est réalisable en termes de capacité de calcul. Par conséquent, même si un algorithme de multiplication scalaire dans lequel le lien entre les opérations exécutées et les bits de clé n'est pas évident est utilisé, il peut

être possible de retrouver la clé *via* une SSCA seulement et la méthode proposée par Oswald.

3.1.2.3 Attaque dite « du doublement »

Fouque *et al.* [FV03] ont proposé une variante d'attaque SSCA appelée attaque dite « du doublement », dans laquelle les relevés de canal caché du calcul de $[k]P$ et de $[k]([2]P)$ sont utilisés.

Les auteurs utilisent le fait que le résultat du doublement à l'étape j $(0 \leq j \leq \ell - 1)$ de l'algorithme du « doublement-et-addition » (étape 3 de l'algorithme 9) noté $Q_j(P)$ peut s'écrire également :

$$\begin{aligned} Q_j(P) &= \left[\sum_{i=0}^{j} k_{\ell-i} 2^{j-i}\right] P \\ &= \left[\sum_{i=0}^{j-1} k_{\ell-i} 2^{j-1-i}\right] ([2]P) + [k_{\ell-j}]P \\ &= [Q_{j-1}]\,([2]P) + [k_{\ell-j}]P. \end{aligned}$$

Ainsi, $Q_j(P) = [Q_{j-1}]\,([2]P)$ si $k_{\ell-j} = 0$. Plus concrètement pour l'attaquant, si la portion du relevé du canal caché correspondant au calcul de Q_j est identique à celle relative au calcul de $[Q_{j-1}]\,([2]P)$, alors $k_{\ell-j} = 0$, sinon $k_{\ell-j} = 1$.

Le tableau 3.1 représente un exemple d'application de cette attaque, avec $k = (1001110)_2$ et donc $\ell = 6$.

$\boldsymbol{j}$	0	1	2	3	4	5	6
$\boldsymbol{k_{\ell-j}}$	1	**0**	**0**	1	1	1	**0**
$\boldsymbol{Q_j}$	P	$\boldsymbol{[2]P}$	$\boldsymbol{[4]P}$	$[9]P$	$[19]P$	$[39]P$	$\boldsymbol{[78]P}$
$\boldsymbol{Q_{j-1}}$	$\boldsymbol{[2]P}$	$\boldsymbol{[4]P}$	$[8]P$	$[18]P$	$[38]P$	$\boldsymbol{[78]P}$	$[156]P$

TAB. 3.1 – Exemple d'application de l'attaque dite « du doublement ».

Il est montré dans le tableau 3.1 qu'en décalant le second relevé (correspondant au calcul de Q_{j-1}) d'une étape j vers la droite, et en la comparant avec la première courbe, l'attaquant peut récupérer l'intégralité de la clé.

Cette attaque est très efficace, puisqu'elle nécessite seulement deux attaques SSCA. Il faut noter que cette attaque peut également s'appliquer sur d'autres implantations d'algorithmes de multiplication scalaire, comme par exemple ceux utilisant le $\mathrm{NAF}_w(k)$ (cf. Algorithme 10).

Une attaque similaire a été proposé par Yen *et al.* [YLMH05]. Leur attaque utilise un point P_0 de la courbe E tel que $\mathrm{ord}_E(P_0) = 2$ (cela signifie notamment que $[2]P_0 = \infty$, cf. section 2.1.4). Si il est calculé $[k]P_0$ à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9), alors le point calculé à la fin de chaque itération j est égal à ∞ si

$k_j = 0$ ou à P_0 si $k_j = 1$. Si un attaquant arrive à distinguer les traces correspondant aux calculs des points résultats ∞ et P_0, l'attaquant peut retrouver tous les bits de la clé.

3.1.3 Les attaques différentielles

Il a été présenté dans la section 3.1.2 les attaques par observation simples. Lorsqu'une attaque SSCA n'est pas réalisable à cause par exemple d'un bruit trop important sur les mesures effectuées, une attaque par observation plus sophistiquée peut être utilisée, appelée attaque différentielle par analyse de canaux cachés (*Differential Side-Channel Analysis*, DSCA).

Ce type d'attaque nécessite un grand nombre de mesures du canal caché observé pour déterminer la clé, contrairement aux attaques simples pour lesquelles un seul enregistrement du motif du canal caché observé (ou deux dans la cadre d'une attaque dite « du doublement », cf. section 3.1.2.3) peut suffire. Ces nombreux enregistrements du canal caché choisi doivent ensuite être traités à l'aide d'outils statistiques plus ou moins évolués.

Une attaque SSCA exploite la relation entre les **opérations exécutées** et le canal caché mesuré (cf. section 3.1.2). Une attaque DSCA exploite la relation entre les **données traitées** et le canal caché mesuré.

3.1.3.1 Le phénomène à l'origine de l'attaque différentielle

Afin d'expliquer comment marche une attaque DSCA, il faut utiliser le modèle de consommation électrique d'un circuit CMOS, dont le composant de base est l'inverseur (cf. Figure 3.2).

Le point important à souligner est que les transitions de la tension d'entrée A ($0 \rightarrow 0$) et ($1 \rightarrow 1$) n'entraînent pas de changement de la consommation électrique de l'inverseur, ce qui n'est pas le cas des transitions ($0 \rightarrow 1$), ($1 \rightarrow 0$). En particulier, la transition pour A ($1 \rightarrow 0$) entraîne une plus grande consommation d'énergie que la transition ($0 \rightarrow 1$). Cette différence de comportement s'explique par le rôle joué par la capacité de sortie de l'inverseur, cette dernière étant connectée entre la sortie Y et la masse (la valeur de cette capacité dépend notamment des caractéristiques physiques des interconnections mises en jeu). Ainsi, la transition pour A ($1 \rightarrow 0$) conduit à la charge de cette capacité par le truchement de la tension d'alimentation, tandis que la transition pour A ($0 \rightarrow 1$) conduit à la décharge de cette capacité vers la masse.

Ce phénomène, qui peut être généralisé aux autres portes logiques plus complexes (par exemple les portes logiques OR, AND, etc.), peut donc être résumé ainsi : **la transition pour la sortie d'un inverseur ($0 \rightarrow 1$) met en jeu une consommation électrique plus importante que la transition ($1 \rightarrow 0$)**. Ce constat peut être étendu au rayonnement EM émis par le circuit, dans la mesure où celui-ci (suivant certaines conditions) peut être proportionnel au courant circulant dans le circuit.

3.1.3.2 Le protocole expérimental d'une attaque différentielle

Une attaque DSCA utilise des statistiques pour amplifier et révéler les différences de comportement évoquées dans la section 3.1.3.1 qui sont difficiles de constater avec une attaque SSCA seule. Ces différences une fois révélées peuvent permettre à l'attaquant de retrouver la clé.

Cette attaque a été initialement proposée par Kocher *et al.* [KJJ99]. Supposons qu'un attaquant essaie de trouver le bit de clé k_j et qu'il connaisse déjà les bits $k_{\ell-1}, k_{\ell-2}, \cdots, k_{j+1}$. Faisant l'hypothèse que l'algorithme du « doublement-et-addition » soit attaqué (cf. Algorithme 9), un attaquant peut monter une attaque DSCA en suivant le protocole expérimental suivant. Considérons qu'il fasse l'hypothèse que le prochain bit de clé est $k_j = 1$.

1. Dans une première phase, l'attaquant effectue des **précalculs**. Il commence par choisir t points $P_0, P_1, \cdots, P_{t-1}$ pour pouvoir calculer

$$Q_r = \left[\sum_{i=j}^{\ell-1} k_i 2^{i-j}\right] P_r, \text{ pour } 0 \leq r \leq t-1.$$

   Il prépare ensuite deux groupes $\mathcal{S}_0$ et $\mathcal{S}_1$ à l'aide de la valeur prédictible d'un seul bit attaqué de Q_r (noté q_0), tel que :

$$\mathcal{S}_0 = \{P_r \mid q_0 = 0\} \text{ et } \mathcal{S}_1 = \{P_r \mid q_0 = 1\}.$$

2. Dans une deuxième phase, l'attaquant effectue des **relevés du canal caché choisi**. Il collecte t mesures du canal caché observé $\mathcal{C}(i)$ générées par le calcul de Q_r. Les courbes sont ensuite réparties dans les deux groupes $\mathcal{S}_0$ et $\mathcal{S}_1$ en fonction de la valeur de q_0.

3. Enfin, l'attaquant réalise une **analyse statistique**. Les courbes sont moyennées dans chaque groupe, et la différence de ces moyennes appelée « courbe différentielle », notée $\Delta(q_0)$ et s'écrivant

$$\Delta(q_0) = \langle \mathcal{C}(r) \rangle_{\substack{0 \leq r \leq t-1 \\ P_r \in \mathcal{S}_1}} - \langle \mathcal{C}(r) \rangle_{\substack{0 \leq r \leq t-1 \\ P_r \in \mathcal{S}_0}},$$

   est tracée. Si l'hypothèse $k_j = 1$ est correcte, les deux courbes moyennées auront des différences notables ($\Delta(q_0) \not\approx 0$) : des pics apparaîtront donc sur la courbe différentielle dans la région où est calculée Q_r (cf. Figure 3.6). En revanche, si l'hypothèse $k_j = 1$ est incorrecte, $\Delta(q_0) \approx 0$: la courbe différentielle est plate (cf. Figure 3.6).

L'attaquant peut réitérer cette procédure pour retrouver les autres bits de clé k_{j-1}, $\cdots$, k_0.

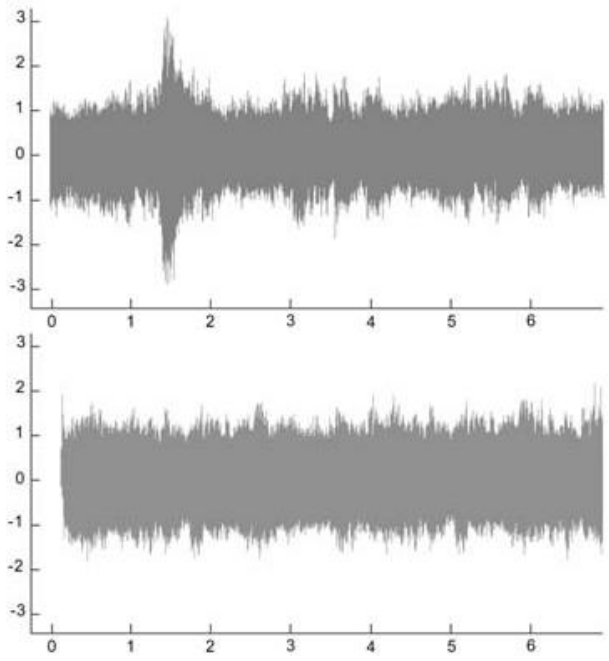

FIG. 3.6 – Courbes différentielles pour une bonne (en haut) et une mauvaise (en bas) hypothèse sur la valeur de Q_r potentiellement calculée par l'algorithme du « doublement-et-addition » [CF06].

Exemple 3.1. Supposons qu'un attaquant ait réalisé des mesures d'un canal caché des multiplications scalaires $[k]P_0, [k]P_1, \cdots, [k]P_{t-1}$ selon l'algorithme du « doublement-et-addition » (cf. Algorithme 9). Il connaît les points $P_1, P_2, \cdots, P_{t-1}$ et souhaite retrouver k.
Vu qu'à l'étape 1, $Q = \infty$, l'opération de doublement à l'étape 3 est triviale : ainsi, elle pourra être distinguée d'un doublement non-trivial grâce à une attaque SSCA. L'attaquant peut donc facilement déterminer le premier bit de k égal à 1 (en partant de la gauche).
Supposons que $k_{\ell-1} = 1$. À la fin de l'étape 6 pour $i = \ell - 1$, on a donc $Q = P$.
À la fin de l'étape 6 pour $i = \ell - 2$, il est calculé $Q = [3]P$ si et seulement si $k_{\ell-2} = 1$.
Ainsi, l'attaquant peut monter une attaque DSCA en se concentrant sur le moment où est calculé potentiellement $Q_r = [3]P_r$. Il divisera les traces de consommation en deux groupes suivant la valeur prédictible d'un bit de $[3]P_r$.
Si la courbe différentielle contient des pics, $Q = [3]P$ a bien été calculé. L'attaquant peut donc conclure que $k_{\ell-2} = 1$; sinon $k_{\ell-2} = 0$. Une fois que $k_{\ell-2}$ a été déterminé, l'attaquant peut reproduire ce schéma d'attaque pour récupérer $k_{\ell-3}$, en utilisant le fait que la valeur $[7]P$ n'est calculée que si et seulement si $k_{\ell-3} = 1$.

3.1.3.3 L'explication de l'apparition (ou non) d'un pic

Il reste à expliquer pourquoi un pic apparaît pour la bonne hypothèse de clé lors d'une attaque DSCA (cf. Figure 3.6). Pour cela, il faut se souvenir que la consommation électrique (respectivement le rayonnement EM émis) d'une (par une) porte CMOS est différent(e) suivant sa valeur en sortie (cf. section 3.1.3.1).

À ce propos, il peut être utilisé un indice de consommation normalisé : il peut être considéré que la transition à la sortie d'une porte logique $(0 \rightarrow 1)$ met en jeu 1 unité de consommation électrique, et que la transition $(1 \rightarrow 0)$ utilise $(1-\delta)$ unité, avec $0 < \delta < 1$. Les transitions $(0 \rightarrow 0)$ et $(1 \rightarrow 1)$ mettent en jeu quant à elles 0 unité. Après avoir défini cet indice de consommation, il peut être défini un premier modèle de consommation basé sur la **distance de Hamming** d'une variable intermédiaire : dans ce modèle, la consommation est reliée aux bits de cette variable qui changent.

Exemple 3.2. Considérons une variable intermédiaire passant de la valeur $(\mathbf{11}\mathbf{0}0)_2$ à $(\mathbf{01}\mathbf{1}0)_2$.
Dans ce cas, il peut être constaté les transitions $(1 \rightarrow 0)$, $(1 \rightarrow 1)$, $(0 \rightarrow 1)$ et $(0 \rightarrow 0)$.
Selon l'indice de consommation normalisé défini auparavant, ces transitions entraînent une consommation électrique globale égal à : $(1-\delta) + 0 + 1 + 0 = (2-\delta)$ unités.

Dans ce qui suit, pour faciliter l'explication de l'apparition d'un pic pour la bonne hypothèse de clé, un indice de consommation simplifié sera utilisé : un « 1 » (respectivement un « 0 ») en sortie d'une porte logique met en jeu 1 (0) unité de consommation électrique. Dans ces conditions, il peut être défini un modèle de consommation simplifié basé sur le **poids de Hamming** d'une variable intermédiaire : dans ce modèle, la consommation électrique de cette dernière est proportionnelle au nombre de bits égaux à « 1 » qu'elle contient.

Exemple 3.3. Considérons une variable intermédiaire ayant comme valeur $(1100)_2$. Ainsi, son poids de Hamming est égal à 2, et cette variable entraîne donc une consommation électrique égale à 2 unités.

Le tableau 3.2 représente ce que pourrait être le contenu des ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ à la fin de l'étape 1 du protocole expérimental d'une attaque DSCA (cf. section 3.1.3.2), si l'hypothèse sur le bit de clé k_j est exacte. Dans ce cas, la prédiction sur la valeur de q_0 (indiqué en gras) est exacte. Si Q_r est exprimé sur n bits, il peut être notamment constaté que le poids de Hamming moyen dans l'ensemble $\mathcal{S}_1$ est de $1 + (n-1)/2$, tandis que le poids de Hamming moyen dans l'ensemble $\mathcal{S}_0$ est de $0 + (n-1)/2$. Ainsi, si l'on fait la différence des moyennes des courbes, et en prenant le poids de Hamming comme modèle de consommation, on obtient $\Delta(q_0) = (1+(n-1)/2) - (0+(n-1)/2) = 1$ et donc un pic apparaît. On a donc réussi **à extraire la contribution d'un bit dans le canal caché mesuré.**

En revanche, si l'hypothèse sur k_j est mauvaise, la prédiction sur la valeur de q_0

$\mathcal{S}_1$	$\mathcal{S}_0$
$(11001\cdots\mathbf{1}01)_2$	$(01001\cdots\mathbf{0}11)_2$
$(10110\cdots\mathbf{1}00)_2$	$(01101\cdots\mathbf{0}10)_2$
$\vdots$	$\vdots$
$(01011\cdots\mathbf{1}10)_2$	$(11001\cdots\mathbf{0}00)_2$

TAB. 3.2 – Contenu possible des ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ lorsque l'hypothèse sur le bit de clé est correcte.

(indiqué en gras) est inexacte. Ainsi, le poids de Hamming moyen dans les ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ est $n/2$ (cf. tableau 3.3), ce qui fait que $\Delta(q_0) = n/2 - n/2 = 0$: aucun pic n'apparaît.

$\mathcal{S}_1$	$\mathcal{S}_0$
$(01010\cdots\mathbf{1}01)_2$	$(10101\cdots\mathbf{0}11)_2$
$(11100\cdots\mathbf{0}00)_2$	$(01111\cdots\mathbf{1}10)_2$
$\vdots$	$\vdots$
$(00111\cdots\mathbf{1}10)_2$	$(10000\cdots\mathbf{0}00)_2$

TAB. 3.3 – Contenu possible des ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ lorsque l'hypothèse sur le bit de clé est inexacte.

Pour mener à bien une attaque DSCA, un attaquant doit connaître les différents points P_r ainsi que l'implantation matérielle de l'algorithme cryptographique, cette dernière pouvant être retrouvée à l'aide d'une SSCA. Il faut noter que pour retrouver un bit de clé, et dans le but de filtrer le bruit inhérent aux mesures effectuées, cette attaque nécessite parfois plusieurs centaines ou milliers de relevés de canal caché.

3.1.3.4 Amélioration des attaques différentielles

Le protocole expérimental d'une attaque DSCA détaillé dans la section 3.1.3.2 se base sur la valeur **d'un seul bit** d'une variable intermédiaire d'un algorithme de multiplication scalaire dépendante de la clé (noté q_0). Cette attaque appelée « DSCA mono-bit » bien que relativement simple à monter, comporte cependant quelques inconvénients. Par exemple, à l'issu de l'analyse statistique, des pics significatifs peuvent apparaître même lorsqu'une mauvaise hypothèse de clé est formulée (ces derniers sont appelés « pics fantômes »). De même, des pics dits « secondaires » d'amplitude non-négligeable peuvent être présents, et ce, quelque soit l'hypothèse de clé faite par l'attaquant. Ainsi, pour différentier les « bons » pics (c'est-à-dire ceux qui donnent réellement des informations sur

la clé) des « mauvais », il est nécessaire d'augmenter le nombre de courbes, ce qui peut être rédhibitoire pour un attaquant.

Partant de ces constats, il a été mis en évidence que les attaques DSCA se basant sur la valeur **de plusieurs bits** (notés $q_{d-1} \cdots q_1 q_0$) d'une variable intermédiaire d'un algorithme de multiplication scalaire donnent de meilleurs résultats. Ces attaques sont appelées attaques « DSCA multi-bits ».

Dans cet esprit, Bevan *et al.* [BK03] proposent une nouvelle méthode statistique. L'attaquant réalise dans une première phase une attaque DSCA mono-bit pour chaque bit q_0, $q_1, \cdots, q_{d-1}$ (il obtient donc respectivement les courbes différentielles $\delta(q_0), \delta(q_1), \cdots, \delta(q_{d-1})$), pour ensuite pouvoir calculer dans une seconde phase $\sum_{i=0}^{d-1} \delta(q_i)$. C'est ce nouveau signal qui sera analysé : des pics d'amplitude amplifiée (et donc beaucoup plus significatifs que d'éventuels pics fantômes ou secondaires) apparaîtront lorsqu'une bonne hypothèse de clé sera émise. Le *et al.* [LNVCC07] généralisent cette méthode, en considérant que les bits q_0, q_1, $\cdots$, q_{d-1} n'ont pas la même contribution au canal caché mesuré. D'où leur idée d'attibuer un poids $\boldsymbol{\alpha_i}$ à chaque courbe différentielle $\delta(q_i)$, et de calculer $\sum_{i=0}^{d-1} \boldsymbol{\alpha_i}\ \delta(q_i)$.

Un autre type d'attaque DSCA multi-bits consiste à étendre la notion de groupe dans le cadre d'une attaque par analyse de canaux cachés dite « par partitionnement » (*Partitioning Side-Channel Analysis*, PSCA). Le *et al.* [LCC+06] proposent de préparer d groupes $\mathcal{S}_0, \mathcal{S}_1, \cdots, \mathcal{S}_{d-1}$ (au lieu de deux dans le cadre d'une DSCA décrite dans la section 3.1.3.2) dans lesquelles les mesures du canal caché observé $\mathcal{C}(i)$ seront réparties à l'aide de la valeur prédictible du poids de Hamming de $(q_{d-1} \cdots q_1 q_0)$. En moyennant dans chaque groupe les courbes, en effectuant une somme pondérée de ces courbes moyennées, et en observant la courbe obtenue, il peut être constaté que les pics fantômes ou secondaires sont vraiment atténués par rapport aux pics réellement informatifs.

Une autre attaque DSCA multi-bits dite « par analyse de corrélation » (*Correlation Side-Channel Analysis*, CSCA) a été mise évidence par Brier *et al.* [BCO04]. Dans le cadre d'une attaque CSCA, un attaquant utilise un modèle hypothétique pour prédire le comportement du canal caché du cryptosystème. La qualité de ce modèle dépend des capacités de l'attaquant. Ces prédictions sont comparées au canal caché mesuré (réel) à l'aide du calcul d'un coefficient de corrélation appelé coefficient de Pearson, dont sa valeur dépend de l'espérance mathématique et de la variance des mesures du canal caché hypothétique et réel. Si ce coefficient a une valeur proche de 1 ou -1, cela veut dire que la corrélation entre le comportement réel et hypothétique est élevée, et donc que l'hypothèse de clé est correcte.

Les attaques DSCA mono ou multi-bits, PSCA et CSCA utilisent généralement un modèle de consommation basé sur le poids de Hamming ou sur la distance de Hamming d'une variable intermédiaire d'un algorithme de multiplication scalaire dépendante de la clé. Cependant, en pratique, ces modèles ne reflètent pas tout à fait le comportement réel d'un canal caché. Or, si le comportement du canal caché est mal modélisé, l'attaquant

aura du mal à retrouver la clé. De plus, en général, ces attaques nécessitent un très grand nombre de mesures de canal caché pour retrouver la clé à cause du bruit présent sur ces mesures. Cette condition peut parfois être rédhibitoire pour un attaquant.

C'est ainsi qu'en parallèle de ces attaques se sont développées celles utilisant des « patrons » (*Template Attacks*). L'idée sous-jacente est d'utiliser un circuit-test identique (ou très proche) de celui qui est attaqué, afin de construire une base de données mémorisant le comportement du canal caché mesuré. Ce type d'attaque fut d'abord introduit par Fahn *et al.* [FP99] puis développée par Chari *et al.* [CRR02]. Les attaques par patrons sont constituées de deux étapes. La première consiste à établir le profil du cryptosystème, en mémorisant pour un grand nombre de messages et d'hypothèses de clé les moyennes et les covariances du canal caché mesuré sur le circuit-test. Cette phase permet aussi de recueillir des détails concernant l'implantation de l'algorithme cryptographique. Dans la seconde étape, l'attaquant tente de retrouver la clé en analysant des mesures de canal caché provenant du cryptosystème attaqué. L'intérêt principal de cette attaque réside dans cette phase d'analyse qui mettra en jeu beaucoup moins de signaux que dans le cadre de toutes les autres attaques différentielles décrites précédemment.

Il est à noter que dans le cadre de ce type d'attaque, l'attaquant n'essaie pas de réduire le bruit (contrairement aux attaques précédentes) mais le modélise afin de pouvoir extraire l'information utile présente dans les mesures.

Pour résumer et conclure cette section, il peut être noté que monter une attaque différentielle conventionnelle ou améliorée est possible si un attaquant peut faire une hypothèse sur une petite partie de la clé pour prédire le comportement du canal caché mesuré grâce à son modèle.

3.1.4 Bilan

Comparer **quantitativement** la performance des différentes attaques par observation n'est pas une tâche facile, car elles ne sont pas montées dans les mêmes conditions. Par exemple, les attaques de type DSCA, PSCA et CSCA ne nécessitent pas de circuit-test identique (ou très proche) du circuit attaqué, contrairement aux attaques par patrons.

Leurs performances relatives sont basiquement évaluées par le nombre de signaux nécessaires pour retrouver la clé. Ce nombre dépend de la qualité des signaux, notamment de la quantité de bruit présent sur les mesures. Dans [LCC08], Le *et al.* comparent les performances de différentes attaques différentielles par analyse du rayonnement EM sur un ASIC exécutant un DES. Leur attaque DSCA mono-bit nécessite 2000 signaux pour retrouver la clé entière, contre 300 signaux pour une attaque DSCA menée sur 4 bits, et 10 pour une attaque par patrons. Cependant, pour obtenir de telles performances, les attaques par patrons nécessitent une base de données caractérisant le plus fidèlement possible le rayonnement EM ainsi que le bruit du cryptosystème. Une telle base de données ne peut être construite que si le circuit-test est identique ou très proche de celui qui est attaqué. Si ce n'est pas le cas, l'efficacité de cette attaque est drastiquement diminuée.

Dans le cas général, si un attaquant veut attaquer un cryptosystème en analysant le comportement d'un (ou de plusieurs) de ses canaux cachés, il peut adopter la procédure suivante. Il peut tout d'abord tenter de monter une SSCA (ou deux dans le cas d'une attaque dite « du doublement »). Si il n'arrive pas à récupérer la clé (soit parce qu'il y a trop de bruits sur les mesures qu'il enregistre, soit parce que le concepteur de circuits sécurisés a introduit des contre-mesures à ces attaques), il devra donc monter une attaque DSCA ou une amélioration de celle-ci (PSCA ou CSCA).

Pour l'attaquant, le choix d'une attaque différentielle basique ou améliorée dépend de son contexte. Si il dispose d'un circuit-test identique (ou très proche) de celui qui est attaqué, alors son choix se portera sur une attaque par patrons car elle sera plus efficace. Si il ne dispose pas d'un circuit de référence, il n'aura pas d'autre choix que de monter une attaque DSCA, PSCA ou CSCA.

Si l'on compare **qualitativement** ces trois derniers types d'attaques, il peut être noté que le signal issu de l'analyse statistique des attaques DSCA et PSCA a un niveau de bruit plus bas que celui relatif aux attaques CSCA. C'est pourquoi il peut dans un premier temps analyser les signaux avec une DSCA (et/ou une PSCA) pour trouver des moments d'apparition de pics (significatifs, fantômes ou secondaires), puis dans un second temps appliquer une CSCA à ces moments précis pour discerner les « bons » pics des « mauvais », et ainsi pour pouvoir retrouver la clé plus facilement. Ces attaques peuvent par ailleurs être combinées avec des outils de traitement du signal pour réduire le bruit des mesures.

Du point de vue du concepteur de circuits sécurisés, il faudra évidemment se prémunir des attaques simples (cf. section 4.1.1) et différentielles (cf. section 4.1.2) par observation.

3.2 Attaques par perturbation

La section précédente a présenté les attaques dites « par observation », dans lesquelles l'attaquant se contente d'observer certaines caractéristiques d'un cryptosystème en fonctionnement afin de récupérer la clé. Ces attaques sont principalement non-invasives.

Les attaques par perturbation consistent quant à elles à porter atteinte au fonctionnement du cryptosystème.

3.2.1 Description

Les effets des fautes et des perturbations sur les composants électroniques ont été observés depuis les années 70. Il a été constaté notamment que des éléments radioactifs naturellement présents dans le conditionnement des circuits provoquaient des fautes [BECN+06]. Des fautes ont également été obtenues **involontairement** lorsque des circuits étaient utilisés dans des contextes particuliers, comme par exemple le domaine aérospatial [BECN+06]. D'autres phénomènes involontaires peuvent être à l'origine de fautes : des défaillances matérielles ou logicielles, du bruit extérieur, mais également des erreurs

de conception.

Dans les années 90, il a été observé que des injections **volontaires** de fautes pendant l'exécution d'un algorithme cryptographique pouvaient conduire à la révélation d'informations parfois suffisantes pour retrouver la clé. La première attaque dite « par injection de fautes » a été publiée par Boneh *et al.* en 1997 sur le RSA utilisant le théorème chinois des restes (*Chinese Remainder Theorem*, CRT), également appelé RSA-CRT, qui est un cryptosystème à clé publique [BDL97]. Cette attaque par injection de fautes est dite « différentielle » (*Differential Fault Attacks*, DFA), car elle nécessite plusieurs exécutions de l'algorithme cryptographique (normales ou fautées) pour récupérer la clé. Quelques mois plus tard, Biham *et al.* [BS97] ont proposé une DFA sur le DES, cryptosystème à clé privée.

Aujourd'hui, de nombreuses attaques sont publiées sur de très nombreux cryptosystèmes, et sont donc une source de vulnérabilité très importante pour ces derniers[14].

3.2.1.1 Méthodes de perturbation

Une attaque par injection de fautes consiste à modifier l'environnement du cryptosystème de telle sorte que le fonctionnement de celui-ci s'en trouve altéré et qu'il apparaisse des fautes. Pour injecter des fautes, l'attaquant peut :

- **faire varier la température du cryptosystème** [BECN+06]. La notice d'utilisation des circuits définissent une gamme de température dans laquelle ceux-ci fonctionnent correctement. Le but d'un attaquant peut donc être de faire varier la température du cryptosystème dans le but de le sortir de cette gamme.
  Les variations en température d'un cryptosystème peuvent entraîner deux conséquences : la modification aléatoire des cellules de mémoires volatiles et l'arrêt des opérations de lecture dans les mémoires non-volatiles en fonctionnement ;
- **émettre de puissantes variations de champ EM au voisinage du circuit**, à l'aide de sondes chargées par exemple [QS02]. Cette méthode d'injection de fautes ne requiert pas la décapsulation du cryptosystème ;
- **injecter des rayons X ou des ions lourds** [BECN+06]. Grâce à cette méthode, il n'est pas nécessaire de décapsuler le cryptosystème ;
- **injecter des pics de courant ou de tension** (*Spike Attack*, SA) ou **modifier la fréquence d'horloge du circuit** (*Glitch Attack*, GA) [AK96]. Ces variations peuvent provoquer suivant les cas une mauvaise exécution d'une opération (voir même son éviction), et/ou une mauvaise lecture des données ou des opérandes.
  L'équipement nécessaire pour ce type d'attaque est un générateur de signal. Cependant, pour des attaques plus précises, il peut être nécessaire d'utiliser un générateur très précis accompagné d'un dispositif sophistiqué de synchronisation (cf. Figure 3.7), dans le but de perturber seulement des opérations choisies de l'algorithme cryptographique ;

[14] Pour illustrer ces propos, il peut être pris l'exemple du cryptosystème à clé privée CLEFIA proposé par Sony en 2007 (http ://www.sony.net/Products/cryptography/clefia/) : il a fallu moins d'un an après sa mise sur le marché pour que Chen *et al.* attaquent ce cryptosystème à l'aide d'une DFA [CWF07].

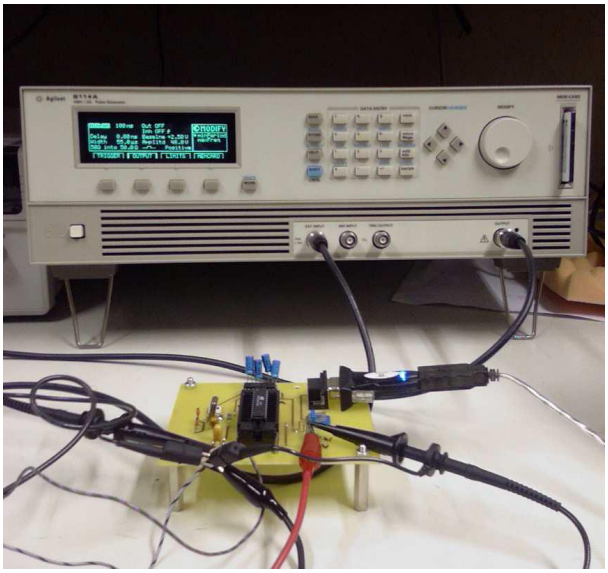

FIG. 3.7 – Dispositif pour une GA [KQ07].

– **illuminer le cryptosystème à l'aide de radiations lumineuses** [SA02] : cela conduit à l'induction de courants additifs dû à des effets photoélectriques, et cela peut être suffisant pour perturber le comportement électrique du cryptosystème.
La lumière blanche (par exemple issue d'un flash d'un appareil photo) est un moyen peu onéreux pour injecter des fautes. Le laser produit des effets similaires que la lumière multichromatique, mais il permet de viser précisément la partie du circuit qu'un attaquant veut fauter. Envisager l'utilisation d'un laser conduit cependant à deux contraintes majeures. Tout d'abord, le coût financier du dispositif peut être prohibitif pour certains attaquants. Ensuite, manipuler un tel dispositif nécessite de l'expérience, d'une part parce que manipuler un laser peut être dangereux, mais aussi parce que si un attaquant dose mal la quantité d'énergie qu'il injecte au cryptosystème, il peut détruire ce dernier partiellement ou complètement et le rendre ainsi inutilisable. Lorsque ce procédé d'injection est combiné avec un microscope (cf. Figure 3.8), la localisation **spatiale** des fautes peut être très précise, tandis que combiné avec des mesures de consommation, la localisation **temporelle** des fautes s'en trouve améliorée.
Il faut noter que, contrairement aux attaques GA qui sont non-invasives, les attaques par radiations lumineuses sont semi-invasives, car le cryptosystème doit être ouvert pour permettre à la lumière injectée d'atteindre la surface active du cryptosystème (métal).

Giraud [GT04] et Bar-El *et al.* [BECN+06] notamment ont produit des états de l'art très complets dans le domaine de la réalisation pratique des attaques par injection de fautes.

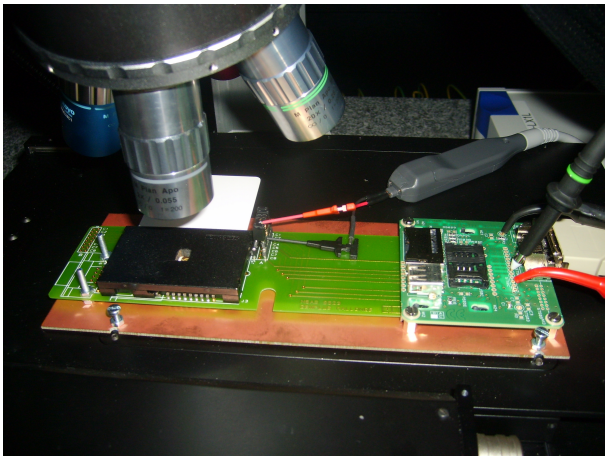

FIG. 3.8 – Dispositif d'attaque par laser (source : CMP-GC, laboratoire SAS).

3.2.1.2 Types de perturbation

Il a été mis en évidence deux types principaux de perturbation : les perturbations **permanentes** et **transitoires**.

Dans le cadre des perturbations **permanentes**, ce sont des fautes destructives : la valeur d'une cellule mémoire est définitivement changée, et cela concerne soit les données (contenues soit dans des mémoires volatiles ou non-volatiles), soit le code implanté (contenu dans une mémoire non-volatile).

Dans le cadre de perturbations **transitoires**, ce sont des fautes provisoires. Le circuit est donc amené à retrouver son comportement original, et est donc encore utilisable après sa réinitialisation ou quand le phénomène responsable de la faute cesse. Dans ce modèle de faute, l'exécution du code implanté ou une opération peut être perturbée : soit une opération différente de celle qui est prévue est exécutée, soit ce sont les opérandes qui sont différentes. Il faut noter que la plupart des attaques par injection de fautes utilisent des perturbations transitoires.

3.2.1.3 Modèles de perturbation

Plusieurs modèles de perturbation sont réalisables dans le cadre d'une attaque par injection de fautes. Ces modèles dépendent directement du dispositif qu'un attaquant utilise pour injecter des fautes. Afin de caractériser les différents dispositifs d'injection de fautes décrits dans la littérature, cinq propriétés sont couramment utilisées :

- **le degré d'invasivité**. Selon le dispositif d'injection de fautes choisi, il sera (respectivement il ne sera pas) nécessaire de décapsuler le cryptosystème : dans ce cas, l'attaque est semi- (non-) invasive ;
- **le contrôle de la localisation temporelle de la faute**. Là encore, l'attaquant a soit aucun contrôle, soit un contrôle partiel, soit un contrôle précis ;

– **le contrôle de la localisation spatiale de la faute**. Selon le dispositif d'injection de fautes choisi, l'attaquant a soit aucun contrôle, soit un contrôle partiel, soit un contrôle total sur l'instant de début et la durée de la faute injectée ;
– **le nombre de bits fautés**. Ce paramètre est très important dans les attaques par injection de fautes, car certaines nécessitent de fauter un nombre restreint de bits afin de récupérer la clé. L'attaquant peut soit fauter un seul bit, soit un nombre réglable de bits, soit un nombre aléatoire de bits ;
– **le modèle de faute injecté**. Il a été mis en évidence trois modèles de faute différents dans la littérature : le collage d'un bit à une valeur précise, l'inversion de la valeur d'un bit, et un modèle de faute aléatoire.

Une fois ces différentes propriétés définies, plusieurs remarques peuvent être édictées concernant les dispositifs d'injection de fautes présentés auparavant.

Tout d'abord, les perturbations par rayonnement EM sont assez destructives car elles consistent essentiellement en des décharges électrostatiques : le circuit fauté peut être ainsi rendu inutilisable.

Ensuite, les attaques SA ne permettent pas d'injecter des fautes spatialement précises et ces dernières ont une valeur aléatoire : elles sont donc rarement utilisées en pratique.

Il peut être noté également que parce que les attaques GA affectent la fréquence de fonctionnement du cryptosystème, peu de bits peuvent être potentiellement affectés, mais ceux-ci sont non-choisis et dépendants de l'implantation matérielle du cryptosystème, comme l'a montré par exemple Faurax *et al.* sur l'AES (*Advanced Encryption Standard*) [FIP01], successeur du DES [FF07].

Enfin, dans le cadre de l'utilisation d'un laser, un attaquant a la capacité de régler les dimensions du rayon qu'il injecte, et donc il peut potentiellement sélectionner le nombre de bits qu'il faute, mais ces bits sont le plus souvent contigus, c'est-à-dire voisins géographiquement [SA02]. Vu que l'énergie d'un rayon laser est également réglable, un attaquant peut injecter des fautes dans un cryptosystème sans pour autant le détruire, et le rendre ainsi inutilisable.

Il a été décrit dans cette section les aspects théoriques et pratiques des attaques par injection de fautes. Comme notre but est de protéger un cryptosystème ECC contre les attaques en fautes, il convient également de faire un état de l'art sur ce point précis : c'est tout l'objet des sections suivantes.

Il a été mis en évidence deux procédures d'attaques DFA sur courbes elliptiques : l'attaquant peut retrouver la clé soit en forçant le calcul de la multiplication scalaire (cf. section 2.1.4) sur une autre courbe ayant des propriétés particulières [BMM00] [CJ05], soit en maintenant le calcul de la multiplication scalaire sur la courbe initiale [BOS06].

3.2.2 Forcer le calcul de la multiplication scalaire sur une autre courbe (cryptographiquement plus faible)

Il a été montré dans la section 2.1.4 que la multiplication scalaire $Q = [k]P$ est l'opération fondamentale dans le cadre de l'ECC, et que la sécurité de ce cryptosystème

repose donc sur l'ECDLP. Si les paramètres de la courbe elliptique sur laquelle est calculée la multiplication scalaire obéissent à certaines conditions (cf. tableau 2.8), alors elle est considérée comme « cryptographiquement forte », c'est-à-dire que l'ECDLP est difficile à résoudre avec les méthodes connues. Notamment, si un concepteur de circuits sécurisés choisit le paramètre $n = \mathrm{ord}_E(P)$ suffisamment grand, il se prémunit d'une recherche exhaustive ou des attaques du type Pohlig-Hellman ou Pollard's ρ (cf. tableau 2.8). Le but d'un attaquant peut donc être d'exécuter la multiplication scalaire sur une courbe elliptique reposant sur un groupe cyclique d'ordre inférieur à n. Ainsi, sur cette nouvelle courbe « cryptographiquement plus faible », un attaquant peut espérer résoudre l'ECDLP plus facilement.

Il existe deux méthodes pour forcer le calcul de la multiplication scalaire sur une autre courbe cryptographiquement plus faible : soit en perturbant les deux coordonnées du point de base P (x et y), soit en perturbant une seule de ces coordonnées ou un des paramètres de courbe a ou p.

Avant de détailler ces attaques, il convient d'introduire la notation suivante : une courbe elliptique obéissant à l'équation de Weierstrass (cf. équation 2.1) sera notée dans ce qui suit

$$E(a, b) = E(a, y^2 - x^3 - ax).$$

3.2.2.1 Perturber les deux coordonnées du point de base

Biehl *et al.* ont proposé les premiers une attaque DFA sur les courbes elliptiques [BMM00]. Si un point $\hat{P} = (\hat{x}, \hat{y}) \in \mathbb{F}_p \times \mathbb{F}_p$ ($\hat{x}$ et $\hat{y}$ représentant les valeurs fautées par l'attaquant, c'est-à-dire $\hat{x} \neq x$ et $\hat{y} \neq y$) avec $\hat{P} \notin E$ est pris comme entrée de la multiplication scalaire, alors il se déroulera le calcul

$$\hat{Q} = [k]\hat{P}$$

sur la courbe

$$\hat{E}(a, \hat{b}) = \hat{E}(a, \hat{y}^2 - \hat{x}^3 - a\hat{x}).$$

Sachant cela, un attaquant peut choisir un point $\hat{P}_0 = (\hat{x}_0, \hat{y}_0) \in \hat{E}_0$ tel que avec $\hat{b_0} = \hat{y_0}^2 - \hat{x_0}^3 - a\hat{x}_0$, $\mathrm{ord}_{\hat{E}_0}(\hat{P}_0) = n_0$ est petit. Du coup, il sera calculé :

$$\hat{Q}_0 = [k]\hat{P}_0.$$

Parce que $\hat{P}_0$ et $\hat{Q}_0$ sont connus, et n_0 est suffisamment petit, toutes les conditions sont alors réunies pour que l'attaquant puisse utiliser les méthodes connues permettant le calcul d'un logarithme discret (Pohlig-Hellman, Pollard's ρ), et ainsi retrouver $k \pmod{n_0}$. Si l'attaquant réitère cette procédure avec d'autres $\hat{P}_j$ (avec $0 \leq j \leq d-1$, $\mathrm{ord}_{\hat{E}_j}(\hat{P}_j) = n_j$) et en utilisant le CRT, il peut finalement retrouver $k \pmod{(n_0 n_1 \cdots n_{d-1})}$. Il faut noter que cette attaque par perturbation n'est pas applicable sur certains protocoles cryptographiques impliquant le changement du scalaire k à chaque nouvelle exécution (comme par exemple l'ECDSA, cf. Algorithmes 14 et 15). En revanche, cette attaque peut être

montée sur des multiplications scalaires impliquées dans des protocoles de chiffrement, comme par exemple l'ECIES [HMV04].

Notons également que cette attaque requiert une injection très précise en termes de localisation

- **spatiale** : l'attaquant doit forcer le calcul de la multiplication scalaire sur une courbe « cryptographiquement plus faible », et donc il ne doit fauter que quelques bits de x et de y (choisis et pas forcément contigus) ;
- **temporelle** : l'attaquant doit fauter le point P au début de la multiplication scalaire.

Or, pour un attaquant, le simple fait de devoir fauter des bits pas forcément contigus rend cette attaque difficile à mettre en oeuvre.

Biehl *et al.* [BMM00] ont également proposé deux autres attaques DFA plus sophistiquées.

La première attaque consiste à fauter un seul bit (aléatoire) au début de la multiplication scalaire. Ainsi, le seul dispositif d'injection de fautes obéissant à ces contraintes est le laser, car seule son utilisation permet de régler le nombre de bits que l'on veut fauter (ici, un seul). La conséquence directe de cette observation est que si l'attaquant n'a pas à sa disposition un tel dispositif (à cause de son coût financier par exemple) ou si il ne dispose pas de procédé de décapsulation de cryptosystèmes (car c'est un procédé semi-invasif), alors cette attaque lui sera très difficile à mettre en oeuvre.

La seconde attaque DFA proposée permet de retrouver la clé, et ce même lorsque la localisation temporelle de la faute injectée pendant le calcul de la multiplication scalaire est inconnue. En revanche, l'attaquant doit toujours fauter un registre précis du cryptosystème, ce qui restreint le nombre de dispositifs d'injection de fautes adéquats pour monter cette attaque à un seul : le laser.

Enfin, il faut noter que les attaques de Biehl *et al.* peuvent également s'appliquer sur des implantations d'algorithmes de multiplication scalaire sophistiquées, comme par exemple ceux utilisant le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3).

3.2.2.2 Perturber une des coordonnées du point de base ou un paramètre de courbe

Il a été montré dans la section précédente qu'une première façon de forcer le calcul de la multiplication scalaire sur une courbe cryptographiquement plus faible est de perturber les deux coordonnées x et y du point de base P. Une seconde façon est de perturber soit une des coordonnées de P (x ou y), soit les paramètres de courbe a ou p.

Ces attaques mises en évidence par Ciet *et al.* [CJ05] reprennent le principe des attaques de Biehl *et al.* [BMM00] : en fautant une des coordonnées de P ou un paramètre de courbe, il est attendu que le calcul de la multiplication scalaire se déroule sur une courbe cryptographiquement plus faible, sur laquelle le calcul du logarithme discret (dans le but

de retrouver k) est possible avec les méthodes connues.

Le tableau 3.4 résume les attaques DFA proposées par Ciet *et al.*. Les valeurs inconnues

	$\boldsymbol{x_1 \longrightarrow \hat{x_1}}$	$\boldsymbol{p \longrightarrow \hat{p}}$	$\boldsymbol{a \longrightarrow \hat{a}}$
$\boldsymbol{P \longrightarrow}$	$\hat{P} = (\hat{x_1}, y_1)$	$\hat{P} = (\hat{x_1}, \hat{y_1})$	inchangé
$\boldsymbol{\hat{Q} =}$	$[k]\hat{P}$	$[k]\hat{P}$	$[k]P$
$\boldsymbol{(\hat{x_Q}, \hat{y_Q}) =}$	$[k](\hat{x_1}, y_1)$	$[k](\hat{x_1}, \hat{y_1})$	$[k](x_1, y_1)$
$\boldsymbol{\hat{Q} \in}$	$\hat{E}(a, \hat{b})$	$\hat{E}(a, \hat{b})$	$\hat{E}(\hat{a}, \hat{b})$
Inconnues	$\hat{x_1}$	$\hat{p}$	$\hat{a}$, $\hat{b}$
Équations	$\hat{x}_1^3 + a\hat{x_1} + \hat{b} - y_1^2 = 0 \pmod p$ ♠	$\hat{p} \mid (b_Q - b_1)$ ♦	$\begin{cases} y_1^2 = x_1^3 + \hat{a}x_1 + \hat{b} \\ \hat{y}_Q^2 = \hat{x}_Q^3 + \hat{a}\hat{x_Q} + \hat{b} \end{cases}$

TAB. 3.4 – Résumé des attaques DFA proposées par Ciet *et al.*.

nécessaires pour retrouver la clé (cinquième ligne du tableau) peuvent être retrouvées à l'aide d'équations ou d'un système d'équations listées dans la sixième ligne du tableau (♠ : $\hat{b} = \hat{y}_Q^2 - \hat{x}_Q^3 - a\hat{x_Q} \pmod p$, ♦ : $b_Q = \hat{y}_Q^2 - \hat{x}_Q^3 - a\hat{x_Q} \pmod p$, $b_1 = y_1^2 - x_1^3 - ax_1 \pmod p$).

Dès que l'attaquant connaît (respectivement retrouve) le point de base P ($\hat{P}$), le point résultat fauté $\hat{Q}$, la courbe fautée cryptographiquement plus faible $\hat{E}$, et si $\mathrm{ord}_{\hat{E}}(P) = n$ ($\mathrm{ord}_{\hat{E}}(\hat{P}) = n$) est suffisamment petit, toutes les conditions sont alors réunies pour que l'attaquant puisse utiliser les méthodes connues permettant le calcul d'un logarithme discret (Pohlig-Hellman, Pollard's ρ), et ainsi retrouver $k \pmod n$.

Il faut noter que les auteurs proposent deux scénarios différents d'attaques. En effet, ces dernières sont valables si l'attaquant injecte une faute :

- **permanente** sur les paramètres de courbe ou sur une des coordonnées de P,
- **transitoire** pendant le transfert de ces derniers d'une mémoire non-volatile à la mémoire de travail du cryptosystème.

Par rapport aux attaques décrites dans [BMM00], celles mises en évidence par Ciet *et al.* mettent en jeu un modèle de faute facile à réaliser pour un attaquant : il lui suffit d'injecter des fautes à valeurs aléatoires (au lieu de fautes à valeurs choisies).

3.2.3 Attaquer en conservant la courbe initiale

Les attaques proposées dans [BMM00] [CJ05] consistent à calculer la multiplication scalaire $Q = [k]P$ sur une courbe elliptique $\hat{E}$ cryptographiquement plus faible que la courbe E sur laquelle devait se dérouler normalement les calculs. *A contrario*, un autre type d'attaque DFA sur les courbes elliptiques mis en évidence par Blömer *et al.* [BOS06] propose de rester sur E.

Les auteurs proposent d'attaquer l'implantation de l'algorithme de multiplication scalaire. Par exemple, si ce dernier est l'algorithme du « doublement-et-addition » (cf. Algorithme 9), alors leur attaque appelée « attaque en fautes par changement de signe » (*Sign-Change Fault Attack*, SCFA) consiste à calculer à l'étape 5 de l'algorithme 9

$$\textbf{si } (k_i = 1) \textbf{ alors } Q \leftarrow \textbf{-}Q + P$$

au lieu de

$$\textbf{si } (k_i = 1) \textbf{ alors } Q \leftarrow Q + P.$$

Plus précisément, si $Q = (x_Q, y_Q)$, alors pour retrouver le bit de clé k_i, il faut changer le signe de y_Q (modulo p) avant d'effectuer l'addition des points Q et P. Ce changement de signe conduit à la fin de la multiplication scalaire à la valeur $\hat{Q}$, qui peut également s'écrire :

$$\hat{Q} = -Q + \left[2 \sum_{j=0}^{i} k_j 2^j\right] P.$$

Pour que cette attaque puisse être menée à bien, l'attaquant doit d'abord fauter les bits de clé de poids faible. Ainsi, une fois retrouvé le bit de clé k_0 grâce à

$$\hat{Q} = -Q + \left[2(k_0 2^0)\right] P,$$

l'attaquant doit renouveler l'expérience pour le bit de clé k_1, en utilisant

$$\hat{Q} = -Q + \left[2(k_0 2^0 + k_1 2^1)\right] P,$$

et ainsi de suite jusqu'au bit de clé $k_{\ell-1}$.

Il faut noter que le succès de cette attaque est très dépendant de l'implantation matérielle du cryptosystème, et en particulier de la représentation des données traitées. Par exemple, si Q est représenté en notation « signe et magnitude » (cf. section 1.1), alors il suffira à l'attaquant d'inverser le chiffre de poids fort codant le signe de y_Q pour obtenir $-y_Q$ (modulo p). Dans ce cas, le seul dispositif d'injection de fautes adapté est le laser. En revanche, si une autre représentation des données est implantée (biaisée, à complément, etc.), l'attaquant aura plus de difficultés à obtenir $-y_Q$ (modulo p), car il lui faudra fauter un certain nombre de bits (dépendant de la valeur de y_Q et de la représentation des données implantée) pas forcément contigus. Dans ce cas, la SCFA est techniquement plus difficile à réaliser.

Il faut noter que cette attaque par perturbation n'est pas applicable sur certains protocoles cryptographiques impliquant le changement du scalaire k à chaque nouvelle exécution (comme par exemple l'ECDSA, cf. Algorithmes 14 et 15). En revanche, cette attaque peut être montée sur des multiplications scalaires impliquées dans des protocoles de chiffrement, comme par exemple l'ECIES [HMV04]. Enfin, le fait d'utiliser le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3) dans le cadre du calcul de la multiplication scalaire ne contrarie pas le bon déroulement de cette attaque.

3.2.4 Autres attaques DFA initialement mises en évidence sur d'autres cryptosystèmes que l'ECC

Cette section regroupe deux types d'attaques DFA initialement mises en évidence sur d'autres cryptosystèmes que l'ECC, mais qui peuvent être néanmoins montées sur l'ECC : la première a été initialement proposée sur le RSA, la seconde sur l'AES.

3.2.4.1 Attaques initialement mises en évidence sur le RSA, et applicable sur l'ECC

Certaines attaques DFA initalement mises en évidence sur le RSA peuvent également s'appliquer sans modifications majeures sur l'ECC grâce au fait que la multiplication scalaire peut être vue comme une version additive d'une exponentiation modulaire, opération principale à effectuer lors du RSA. Parmi toutes les attaques DFA concernant le RSA, celles proposées par Bao *et al.* [BDH+97] font partie de celles qui peuvent s'adapter le mieux au contexte de l'ECC. Bao *et al.* proposent deux attaques différentes : la première consiste à fauter la clé, tandis que la seconde se concentre sur la perturbation d'une variable intermédiaire calculée lors de la multiplication scalaire.

Fauter un bit de clé. Lors de la multiplication scalaire, il est effectué :

$$Q = [k]P = \left[\sum_{i=0}^{\ell-1} k_i 2^i\right] P = \left[k_0 2^0 + k_1 2^1 + \cdots + k_i 2^i + \cdots + k_{\ell-1} 2^{\ell-1}\right] P.$$

Si un attaquant inverse la valeur du bit de clé k_i, alors le point Q verra sa valeur changée en $\hat{Q}$, tel que

$$\hat{Q} = \left[k_0 2^0 + k_1 2^1 + \cdots + \overline{k_i} 2^i + \cdots + k_{\ell-1} 2^{\ell-1}\right] P.$$

Ainsi, si l'attaquant calcule $\hat{Q} - Q$, alors il pourra obtenir :

$$\hat{Q} - Q = [2^i \overline{k_i} P - 2^i k_i] P.$$

Deux cas sont alors possibles, selon la valeur du bit de clé k_i :
- si $k_i = 0$, alors $\hat{Q} - Q = [2^i]P - \infty = [2^i]P$,
- si $k_i = 1$, alors $\hat{Q} - Q = \infty - [2^i]P = [-2^i]P$.

Ainsi, pour retrouver le bit de clé k_i, un attaquant doit tout d'abord effectuer un premier calcul de Q sans injecter de fautes. Ensuite, lors d'une seconde multiplication modulaire, il lui faut inverser la valeur du bit de clé k_i qu'il veut retrouver, et le résultat fauté $\hat{Q}$ récolté va lui servir à calculer $\hat{Q} - Q$: suivant la valeur de cette variable, il pourra déterminer la valeur de ce bit de clé. Pour retrouver les autres bits de clé, il devra réitérer toute cette procédure.

Il faut noter que cette attaque nécessite un dispositif d'injection de fautes très précis spatialement pour ne fauter qu'un bit de clé à la fois : seules les attaques par laser obéissent à cette contrainte.

Fauter la valeur d'un doublement. Pour cette seconde attaque de Bao *et al.*, il convient d'introduire la notation suivante :

$$\beta_i = [2^i]P.$$

En adoptant cette nouvelle notation, le calcul de Q s'écrit également :

$$Q = [k]P = (k_0\beta_0 + k_1\beta_1 + \cdots + k_i\beta_i + \cdots + k_{\ell-1}\beta_{\ell-1}).$$

Si un attaquant faute la variable β_i, alors le point Q verra sa valeur changée en $\hat{Q}$, tel que

$$\hat{Q} = \left(k_0\beta_0 + k_1\beta_1 + \cdots + k_i\hat{\beta}_i + \cdots + k_{\ell-1}\beta_{\ell-1}\right).$$

Ainsi, si l'attaquant calcule $\hat{Q} - Q$, alors il pourra obtenir :

$$\hat{Q} - Q = [k_i]\hat{\beta}_i - [k_i]\beta_i.$$

Deux cas sont alors possibles, selon la valeur du bit de clé k_i :
- si $k_i = 0$, alors $\hat{Q} - Q = \infty$,
- si $k_i = 1$, alors $\hat{Q} - Q = \hat{\beta}_i - \beta_i$.

Le protocole expérimental de cette attaque DFA est le même que celle décrite précédemment, hormis le fait que pour retrouver le bit de clé k_i, il doit fauter la variable β_i. De même, l'utilisation d'un laser est requise.

Enfin, dans le cadre des deux attaques proposées par Bao *et al.* [BDH+97], il faut également que le scalaire k reste constant pour toutes les multiplications scalaires attaquées, ce qui est le cas dans certains protocoles (par exemple l'ECIES [HMV04]), mais pas dans d'autres (par exemple l'ECDSA, cf. Algorithmes 14 et 15).

3.2.4.2 Attaques sur les machines d'état

Les attaques DFA présentées précédemment injectent des fautes sur le point de base ou les paramètres de courbe, dont les valeurs sont stockées dans des emplacements mémoires (registres). Choukri *et al.* [CT05] proposent quant à eux d'injecter des fautes sur la machine d'état ordonnant le calcul de la multiplication scalaire. Plus particulièrement, leur attaque se concentre sur la variable de la machine d'état comptant le nombre d'itérations restant à exécuter. Par exemple, si c'est l'algorithme du « doublement-et-addition » (cf. Algorithme 9) qui est choisi pour calculer la multiplication scalaire, l'étape 2 peut être implantée matériellement à l'aide d'un décrémenteur. Si ce décrémenteur est fauté par un attaquant, alors le nombre d'itérations de l'algorithme (normalement égal à ℓ) peut être drastiquement diminué, et le résultat fourni prématurément à l'attaquant peut lui permettre de retrouver la clé. Cette attaque a été initialement mise en évidence sur l'AES, et est aussi connue sous le nom d'attaques en faute par réduction de rondes.

L'exemple de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) peut être pris pour illustrer cette attaque. À la fin de l'étape 5 de la première itération, si $k_{\ell-1} = 1$,

alors $Q = P$, sinon $Q = \infty$. Du point de vue de l'attaquant, il lui suffit d'injecter une faute sur le décrémenteur comptant le nombre d'itérations restant à effectuer, afin que le cryptosystème lui fournisse le résultat du calcul de la variable Q dès la fin de la première itération : si ce résultat est égal à P, alors le bit de clé est égal à 1, sinon il est égal à 0.

Il faut noter que le succès de cette attaque est très dépendant de l'implantation matérielle du cryptosystème, et en particulier de la machine d'état et du compteur d'itérations associé (incrémenteur ou décrémenteur suivant les implantations). L'attaquant doit injecter des fautes spatialement très précises afin que le compteur (et le compteur seul) soit fauté et amené à une valeur voulue par l'attaquant : de ce point de vue, seuls les dispositifs d'injection de fautes par laser sont adaptés à cette attaque en fautes.

3.2.5 Bilan

Les attaques par perturbation portent atteinte au bon fonctionnement du cryptosystème afin d'obtenir des résultats fautés dont leur analyse peut permettre de retrouver la clé. Parmi tous les dispositifs de perturbation possibles pour un attaquant, le laser est le plus précis en matière de localisation spatiale des fautes injectées, et est donc adapté pour monter la plupart des attaques DFA mises en évidence sur l'ECC. Il faut préciser que c'est un procédé d'injection qui nécessite peu ou prou la décapsulation du cryptosystème attaqué, et en ce sens, c'est une attaque semi-invasive.

La multiplication scalaire $Q = [k]P$ est l'opération fondamentale dans le cadre de l'ECC : c'est donc elle qui est l'objet des attaques DFA. Il a été mis en évidence deux types principaux d'attaques DFA sur l'ECC : les attaques forçant le calcul de la multiplication scalaire sur une courbe cryptographiquement plus faible [BMM00] [CJ05] et celles maintenant ce calcul sur la courbe initiale considérée comme cryptographiquement forte [BOS06]. Viennent se greffer à ces deux types d'attaques DFA la version additive de l'attaque initialement proposée par Bao *et al.* sur le RSA [BDH$^+$97], ainsi que l'attaque visant la machine d'état cadençant le bon fonctionnement de la multiplication scalaire [CT05].

De par l'étude réalisée dans ce chapitre, ces attaques DFA peuvent être classées en trois catégories suivant leur probabilité de succès, selon qu'elles soient potentiellement réalisables, réalisables si l'implantation matérielle du cryptosystème le permet ou difficilement réalisables en pratique.

En ce sens, dans la première catégorie, un attaquant peut être tenté de placer les attaques de Biehl *et al.* consistant à fauter un seul bit (aléatoire) au début de la multiplication scalaire et à n'importe quel moment cette dernière, toutes les attaques de Ciet *et al.* ainsi que les attaques de Bao *et al.*. Un attaquant peut classer dans la deuxième catégorie les attaques de Blömer *et al.* et de Choukri *et al.* et dans la troisième catégorie, l'attaque de Biehl *et al.* consistant à fauter les deux coordonnées du point de base simultanément au début de la multiplication scalaire. Une fois ce classement établi, un attaquant peut donc dans une première phase se concentrer sur les attaques appartenant à la première catégorie. Ensuite, si il n'arrive pas à retrouver la clé malgré ses attaques, il

pourra tenter dans une seconde phase de monter les attaques appartenant à la deuxième, voire à la troisième catégorie.

Du point de vue du concepteur de circuits sécurisés, il faudra évidemment se prémunir des attaques DFA. Cet aspect sera plus détaillé dans la section 4.2, mais d'ors et déjà, une première liste de vulnérabilités peut être dressée : en effet, il faudra qu'il protège la clé, le chemin de données (plus précisément la valeur et le signe des points intermédiaires traités par le cryptosystème), la machine d'état, ainsi que le **point de base** (et ce, à n'importe quel moment de la multiplication scalaire) et les **paramètres de courbe**. Ainsi, si un concepteur de circuits doit sécuriser un cryptosystème ECC contre les attaques DFA, il doit protéger non seulement le stockage et la manipulation des paramètres privés, mais également des **paramètres publics**.

3.3 Combiner des attaques par observation et par perturbation

Les attaques par observation peuvent permettre d'améliorer la précision temporelle des attaques par perturbation : si un attaquant veut fauter une opération particulière d'un algorithme, il peut faire une première attaque SSCA afin qu'il puisse avoir une idée plus précise de l'implantation matérielle du cryptosystème, et ainsi qu'il puisse déclencher son attaque par perturbation au bon moment.

Il faut noter qu'à ce jour, les attaques par perturbation contribuant à améliorer les attaques par observation ont fait l'objet d'une seule publication écrite par Skorobogatov [Sko06].

En utilisant un laser pour illuminer une zone spécifique de la surface du cryptosystème, le courant circulant à travers un seul transistor peut être rendu visible dans les mesures de consommation du circuit. L'effet photoélectrique convertit la lumière en courant qui circule à travers un transistor supposé fermé. De cette manière, la contribution de ce transistor au courant global peut être modulé par la lumière. Ainsi, comparé à une attaque PA classique, cette technique permet à l'attaquant de retrouver non seulement les poids de Hamming des différentes variables utilisées par le cryptosystèmes, mais également la valeur individuelle de chaque bit constituant cette variable.

L'efficacité de cette attaque doit tout de même être nuancée car elle a été menée sur une mémoire utilisant une ancienne technologie de transistors (0,9 μm) : cette attaque doit être menée sur des technologies plus récentes afin que sa pertinence soit validée.

3.4 Conclusion de chapitre

Ce chapitre a présenté une vue d'ensemble des attaques exécutables sur les cryptosystèmes dans le cas général, puis dans le contexte particulier des courbes elliptiques. En résumé, deux types d'attaque ont été mises en évidence : les attaques dites « par ob-

servation », dans lesquelles l'attaquant se contente d'observer certaines caractéristiques pendant le fonctionnement du cryptosystème afin de récupérer la clé, et les attaques dites « par perturbation », dans lesquelles l'attaquant porte atteinte au fonctionnement du cryptosystème.

Dans la catégorie des attaques par observation, les attaques dites « simples » sont potentiellement les plus dangereuses, car elles permettent théoriquement à un attaquant de déterminer la clé d'un cryptosystème à l'aide d'un seul enregistrement du canal caché mesuré. Les attaques dites « différentielles » par observation nécessitent une plus grande quantité de relevés du canal caché mesuré, et requièrent donc pour un attaquant des moyens de calculs plus importants.

Il existe plusieurs moyens efficaces pour perturber le bon fonctionnement d'un cryptosystème dans le cadre d'une attaque. Parmi tous ces dispositifs, celui permettant d'avoir la plus grande précision d'injection pour un attaquant est le laser. Pour récupérer la clé *via* une attaque par perturbation, un attaquant peut perturber certains paramètres de courbe de façon à forcer le calcul de la multiplication scalaire sur une courbe cryptographiquement plus faible. Il peut également maintenir ce calcul sur la courbe initiale (considérée comme cryptographiquement forte), mais la faisabilité de cette attaque (SCFA) dépend de l'implantation matérielle du cryptosystème. Il peut également fauter des variables intermédiaires traitées dans le calcul de la multiplication scalaire, la machine d'état qui cadence le fonctionnement du cryptosystème ou encore la valeur de la clé.

Ainsi, si on se place du côté de l'attaquant, on s'aperçoit donc qu'il dispose de beaucoup de méthodes possibles pour récupérer la clé. Cependant, tous les attaquants n'ont pas la même efficacité potentielle. Celle-ci réside tout d'abord dans les capacités financières d'un attaquant : par exemple, si celui-ci a les moyens d'acquérir un laser pour mener des attaques DFA, il sera potentiellement plus efficace qu'un attaquant qui a juste les moyens d'investir dans un flash d'appareil photo. Mais l'efficacité d'un attaquant dépend également de ses capacités techniques : si il n'a aucune expérience dans l'utilisation d'un laser, alors il lui sera très difficile de monter une attaque par injection de fautes.

Si on se place du côté du concepteur de circuits sécurisés, on se rend compte de la difficulté de sa tâche. En effet, il devra prendre en compte toutes les vulnérabilités décrites dans ce chapitre pour pouvoir concevoir un cryptosystème résistant à toutes les attaques possibles, et ce, sans pénaliser outre mesure les performances initiales de celui-ci. C'est tout l'objet du chapitre 4.

Chapitre 4

Contre-mesures pour le cas particulier de l'ECC

Sommaire

Il a été présenté dans le chapitre 3 les différentes attaques possibles permettant de retrouver la clé sur des implantations matérielles de cryptosystèmes. Dans les attaques dites « par observation », l'attaquant se contente d'observer un ou plusieurs canaux cachés afin de retrouver les opérations effectuées par le cryptosystème ou les opérandes traitées par ces dernières. Dans les attaques dites « par perturbation », l'attaquant porte atteinte au bon fonctionnement du cryptosystème, et exploite les résultats potentiellement erronés fournis par celui-ci. Ces attaques peuvent suivant les cas être montées avec des dispositifs financièrement abordables, ce qui induit qu'elles doivent donc être prises au sérieux par les concepteurs de circuits sécurisés.

Notre but étant de protéger un cryptosystème ECC contre les attaques par observation et par perturbation, il convient donc de connaître l'état de l'art dans le domaine des contre-mesures afin de pouvoir le cas échéant en proposer de nouvelles. Ce chapitre détaille donc les différentes contre-mesures présentes dans la littérature. Il doit être rappelé que

c'est la multiplication scalaire $Q = [k]P$ (k étant la clé) qui est l'opération principale dans le cadre de l'ECC (cf. section 2.1.4) : **c'est donc ce calcul qui doit être protégé**.

De par la structure mathématique très riche des courbes elliptiques, de nombreuses contre-mesures logicielles ont été proposées dans la littérature : certaines d'entre elles sont efficaces, c'est-à-dire qu'elles ont un impact limité en termes de temps de calcul et de quantité de mémoire utilisée. Ainsi, suivant les contraintes de performance et de sécurité auxquelles doivent obéir le cryptosystème, un concepteur de circuits sécurisés pourra donc implanter une ou plusieurs contre-mesures adaptées, qu'elles soient logicielles ou matérielles.

Le chapitre V écrit par Joye [Joy05] dans le livre [BSS05] a été une source d'inspiration pour l'écriture de ce chapitre.

4.1 Protections contre les attaques par observation

Il a été vu dans la section 3.1 qu'une attaque par observation peut être simple ou différentielle. Une attaque SSCA (cf. section 3.1.2) permet théoriquement de déterminer la clé d'un cryptosystème à l'aide de l'enregistrement d'un seul motif du canal caché mesuré. Une attaque DSCA (cf. section 3.1.3) conduit à l'acquisition d'un plus grand nombre de relevés pour retrouver la clé.

4.1.1 Protections contre les attaques simples par observation

Dans le cadre de l'ECC, les attaques SSCA permettent de retrouver la clé à l'aide de l'observation d'une seule exécution de la multiplication scalaire $Q = [k]P$. Dans la section 3.1.2.1, il a été montré que les implantations des algorithmes de multiplication scalaire sont particulièrement vulnérables face aux attaques SSCA car :

- le chemin d'exécution de certains algorithmes de multiplication scalaire est déterminé par les bits de clé secrète (c'est par exemple le cas de l'algorithme du « doublement-et-addition » − cf. Algorithme 9),
- les formules pour calculer le doublement d'un point et l'addition de deux points sur une courbe de Weierstrass sont différentes (cf. relation 2.2).

Dans ces conditions, le cryptosystème engendre des traces de canaux cachés qui peuvent être facilement distinguables.

Dans le but de rendre plus difficile les attaques SSCA, un concepteur de circuits sécurisés peut donc faire en sorte que le cryptosystème exécute une séquence fixe d'opérations qui ne peut pas être reliée aux bits traités de k. L'information obtenue grâce aux canaux cachés ne permet donc pas de retrouver la clé (cf. Figure 4.1, à comparer avec la Figure 3.5).

Il a été mis en évidence deux méthodes différentes pour prémunir un cryptosystème ECC contre les attaques SSCA [Joy05] :

- utiliser un algorithme de multiplication scalaire régulier,
- rendre l'addition et le doublement de points indistinguables.

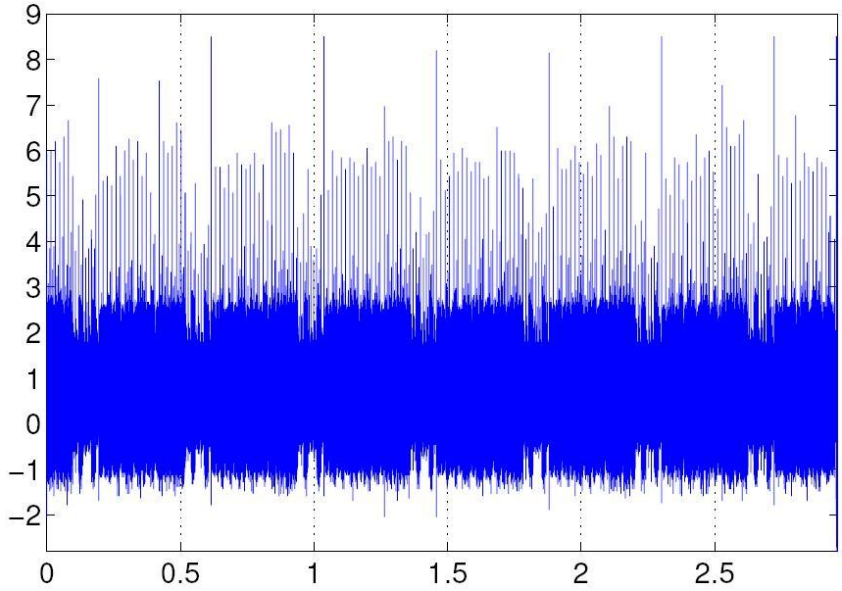

FIG. 4.1 – Relevé type de la consommation électrique d'un algorithme de multiplication scalaire protégé contre les attaques simples par observation [MOPV07].

Malheureusement, l'implantation de ces contre-mesures peut induire un surcoût important en termes de temps et de mémoire utilisée : cette section résumera ces différents surcoûts, et amènera des considérations supplémentaires quant au niveau de sécurité réel de ces contre-mesures, ainsi qu'à leur condition d'utilisation.

4.1.1.1 Utiliser un algorithme de multiplication scalaire régulier

Pour se prémunir des attaques SSCA, une contre-mesure possible est l'utilisation d'un algorithme de multiplication scalaire régulier. Cela veut dire que soit les branchements conditionnels (dépendants des bits de clé) sont absents de l'algorithme, soit il est exécuté la même séquence d'opérations quelque soit le bit de clé traité. Ainsi, plutôt que d'agir sur les formulations du doublement et de l'addition (ce sera l'objet de la section 4.1.1.2), le concepteur de circuits sécurisés se concentre ici sur une manière d'ordonnancer ces deux opérations de façon à obscurcir le lien entre les opérations exécutées et les bits de clé traités.

Algorithme du « doublement-et-toujours-addition ». Insérer des opérations inutiles dans les calculs sur les courbes elliptiques peut permettre de se protéger contre les attaques SSCA.

Dans cet esprit, Coron propose d'insérer des **opérations de points inutiles** afin d'écrire un algorithme de multiplication scalaire régulier appelé algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) [Cor99]. C'est une modification de l'algorithme

du « doublement-et-addition » (cf. Algorithme 9) qui exécute un doublement et une addition à chaque pas de l'algorithme, et ce, quelque soit la valeur du bit de clé traité. Ainsi, si $k_i = 1$ (respectivement $k_i = 0$), alors il est effectué un doublement à l'étape 3 de

Algorithme 16 Algorithme du « doublement-et-toujours-addition »

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow P$
2. **pour** $i = \ell - 2$ à 0 **faire**
3. $\quad R_0 \leftarrow [2]R_0$
4. $\quad b \leftarrow 1 - k_i$; $R_b \leftarrow R_b + P$
5. **fin pour**
6. **retourner** R_0

l'algorithme 16, puis l'addition de points $R_0 \leftarrow R_0 + P$ ($R_1 \leftarrow R_1 + P$). Par conséquent, si le comportement des algorithmes 9 et 16 sont comparés, il peut être constaté qu'il a été ajouté une addition de points inutile lorsque $k_i = 0$. Le canal caché mesuré apparaît désormais comme une succession régulière du motif {doublement + addition} : cela ne donne aucune information sur les bits de la clé.

Cette contre-mesure, qui est historiquement la première proposée dans le cadre de la securisation des cryptosystèmes ECC vis-à-vis des attaques SSCA, comporte cependant quelques inconvénients majeurs.

1. Tout d'abord, son implantation induit un surcoût important : l'algorithme 16 calculera $\ell D + \ell A$ or l'algorithme du « doublement-et-addition » (cf. Algorithme 9) effectuera en moyenne $\ell D + (\ell/2)A$, soit une augmentation moyenne du nombre d'opérations de points à effectuer de 33%. De plus, l'algorithme 16 nécessite trois éléments de mémorisation pour stocker les valeurs des variables R_0, R_1 et P, soit un de plus que dans le cadre de l'algorithme du « doublement-et-addition ».
2. Ensuite, cet algorithme peut être le siège d'une attaque dite « du doublement » [FV03], voire même de l'attaque similaire proposée par Yen *et al.* [YLMH05] (cf. section 3.1.2.3). Rappelons que ces attaques nécessitent seulement la comparaison des relevés de canal caché du calcul de $[k]P$ et de $[k]([2]P)$ pour récupérer la clé.

   Une contre-mesure possible à ces attaques consiste à choisir des nouvelles valeurs pour la clé ou les points intermédiaires à chaque exécution de la multiplication scalaire (cf. sections 4.1.2.1 et 4.1.2.2).
3. De plus, l'algorithme 16 est vulnérable face à une attaque par perturbation particulière, appelée « attaque par perturbation sans conséquence sur les calculs » (*Computational Safe-Error Attack*, CSEA) [YKLM01]. Cette attaque consiste à injecter une faute pendant une opération pour savoir si celle-ci est inutile ou non. Si c'est une opération inutile, alors la faute injectée ne change pas la valeur du point

final Q. Dans le cas contraire, la valeur de Q est fautée. Ainsi, l'attaquant peut distinguer les opérations utiles des opérations inutiles et peut donc retrouver la clé.

Dans le cas particulier de l'algorithme 16, pour récupérer le bit de clé k_i grâce à cette attaque, il faut fauter l'opération d'addition à l'étape i :
- si le résultat final Q est faux, l'opération d'addition effectuée à l'étape i a bien été utilisée pour la suite des calculs, donc $k_i = 1$,
- si le résultat final Q est juste, l'opération d'addition effectuée à l'étape i a été effectuée inutilement, donc $k_i = 0$.

Cette procédure doit être répétée pour chaque bit de clé k_i.

Il faut noter que si la valeur de la clé est choisie aléatoirement (cf. section 4.1.2.1), alors cette attaque peut être contrée.

4. Il faut également noter qu'un cryptosystème ECC implantant cet algorithme peut être le siège d'une attaque simple par analyse du rayonnement EM qu'il émet (cf. section 3.1.1.2). En effet, si un attaquant effectue des mesures locales de ce rayonnement en plaçant une sonde au-dessus de l'élément mémorisant la valeur de R_1, et si celle-ci mesure une activité EM lors du traitement du bit de clé k_i, alors il peut en conclure que la valeur de R_1 a été modifiée, et donc que $k_i = 0$.

   Un concepteur de circuits sécurisés peut contrer cette attaque en utilisant les techniques proposées par May *et al.* [MMS01] et Izu *et al.* [IIT03] consistant à charger le résultat de l'addition de points (étape 4 de l'algorithme 16) dans des registres différents à chaque itération.

5. Enfin, l'algorithme 16 est également vulnérable en l'état face à une variante d'attaque différentielle par observation visant l'adresse des registres (cette attaque et les contre-mesures associées seront décrites en section 4.3.1.2).

Pour résumer, l'implantation de cet algorithme entraîne beaucoup d'inconvénients en termes de performances et de sécurité. Heureusement pour les concepteurs de circuits sécurisés, la recherche dans le domaine des contre-mesures efficaces a conduit à l'élaboration de solutions plus performantes.

Échelle de Montgomery. L'échelle de Montgomery est un algorithme de multiplication scalaire qui a déjà été décrit dans la section 2.2.4.2. Rappelons que l'idée originale de Montgomery [Mon87] est basée sur le fait que la somme de deux points dont la différence est connue peut être calculée sans la coordonnée y de ces points (cette coordonnée pouvant par ailleurs être retrouvée si besoin). Utiliser cet algorithme de multiplication scalaire peut permettre de se prémunir des attaques SSCA, car quelque soit la valeur du bit de clé k_i traité, il est effectué une addition et un doublement de points.

Ce concept a été développé initialement pour les courbes d'équation $by^2 = x^3 + ax^2 + x$ [Mon87] [OS00]. Brier *et al.* [BJ02] ont par la suite fourni les formules de doublement et d'addition de points pour les courbes de Weierstrass (cf. relation 2.1). Le nombre d'opérations nécessaires au calcul de la coordonnée x de $Q = [k]P$ sur ces courbes s'élève à $19\ell M + 1I$ [15].

Comme cela a déjà été décrit dans la section 2.2.4.2, un des principaux avantages de l'échelle de Montgomery est que son calcul peut être effectué par deux UA fonctionnant en parallèle, ce qui permet de diminuer globalement son temps de calcul.

C'est ainsi que Izu *et al.* [IT02a] [IMT02] utilisent dans le cadre du calcul de l'échelle de Montgomery un troisième registre R_2 (en plus des registres R_0 et R_1) afin d'éliminer la dépendance des registres entre l'addition et le doublement de telle sorte que ces deux opérations puissent être exécutées en parallèle.

L'algorithme 12 donné par Joye *et al.* [JY02] n'utilise quant à lui que deux registres, et son implantation nécessite le temps de calcul de $1D + \ell A$, contre $\ell D + \ell A$ lorsqu'un seul processeur est utilisé dans l'algorithme original de l'échelle de Montgomery (cf. algorithme 11). Par conséquent, en utilisant les formules de doublement et d'addition mises en évidence par Brier *et al.* [BJ02] dans le cadre de l'utilisation de l'échelle de Montgomery, il peut être déduit que le temps de calcul de l'algorithme 12 est celui de $(9 + 11\ell)M$.

Quant à eux, Fischer *et al.* [FGKS02] montrent comment paralléliser les 19 multiplications nécessaires au calcul d'une addition suivie d'un doublement de point afin de se ramener à un temps de calcul équivalent à celui de $10M$ par bit de clé : le temps nécessaire pour calculer l'échelle de Montgomery est de $10\ell M$.

Un autre avantage de l'échelle de Montgomery est que cette protection contre les attaques SSCA ne contient aucune opération inutile, contrairement à l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) : il ne contient donc pas de vulnérabilités face aux attaques par injection de fautes de type CSEA (cf. remarque 3. dans la section intitulée « Algorithme du « doublement-et-toujours-addition » »).

Malgré tous ces avantages, il faut prendre en compte quelques inconvénients inhérents à l'utilisation de l'échelle de Montgomery.

1. Ensuite, il a été mis en évidence par Yen *et al.* [YKMH05] puis par Kim *et al.* [KQ08] que l'échelle de Montgomery peut être le siège d'une attaque dite « du doublement » [FV03].

   Cependant, si la clé ou les points intermédiaires traités par le cryptosystème sont rendus aléatoires (cf. sections 4.1.2.1 et 4.1.2.2), cette attaque peut être contrecarrée.

2. De plus, si un cryptosystème ECC implante les algorithmes 11 et 12, alors il peut être le siège d'une attaque simple par analyse du rayonnement EM qu'il émet (cf. section 3.1.1.2). En effet, si un attaquant effectue des mesures locales de ce rayonnement en plaçant une sonde au-dessus de l'élément mémorisant la valeur de R_1, et si celle-ci mesure une activité EM lors de l'addition de points effectuée pendant le traitement du bit de clé k_i, alors il peut en conclure que la valeur de R_1 a été modifiée, et donc que $k_i = 0$.

[15] Les multiplications par des constantes « petites » ont un coût égal à 1M dans cette estimation.

Là encore, un concepteur de circuits sécurisés peut contrer cette attaque en utilisant les techniques proposées par May *et al.* [MMS01] et Izu *et al.* [IIT03].

3. Enfin, l'algorithme 16 est également vulnérable en l'état face à une variante d'attaque différentielle par observation visant l'adresse des registres (cf. section 4.3.1.2 pour le descriptif de cette attaque et des contre-mesures associées).

Chaîne d'additions. Les protections contre les attaques SSCA décrites précédemment consistent à exécuter une même série d'opérations, et ce, indépendamment du bit de clé traité : l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) et l'échelle de Montgomery (cf. Algorithme 11) exécutent un doublement et une addition pour chaque bit de clé traité.

Une autre contre-mesure possible peut consister à n'exécuter que des additions de points. C'est ce que propose Clavier *et al.* [CJ01] en utilisant une chaîne d'additions. L'explication qui suit est directement tirée de la thèse de Clavier [Cla07].

Soit $C(k) = \{k^{(0)}, k^{(1)}, \cdots, k^{(r)}\}$ une chaîne d'additions pour l'exposant k. Ainsi pour tout $1 \leq i \leq r$, nous avons $k^{(i)} = k^{(j(i))} + k^{(l(i))}$ (avec $j(i), k(i) < i$). Cela fournit un moyen simple d'évaluer $Q = [k]P$: pour i allant de 1 à r, il suffit de calculer

$$[k^{(i)}]P = [k^{(j(i))}]P + [k^{(l(i))}]P$$

et de poser $Q = [k^{(r)}]P$. Ainsi, pour une chaîne d'additions de longueur r, rA sont alors nécessaires pour calculer Q.

Exemple 4.1. Une chaîne d'additions pour 25 est $C(25) = \{1, 2, 4, 5, 10, 20, 25\}$, car $[2]P = P + P$, $[4]P = [2]P + [2]P$, $[5]P = [4]P + P$, $[10]P = [5]P + [5]P$, $[20]P = [10]P + [10]P$, $[25]P = [20]P + [5]P$.
Ainsi, en utilisant la chaîne d'additions $C(25)$, $6A$ sont nécessaires pour calculer $[25]P$.

À l'étape i, $[k^{(i)}]P$ est évalué comme $[k^{(i)}]P = [k^{(j(i))}]P + [k^{(l(i))}]P$. En supposant que les valeurs $[k^{(j(i))}]P$ et $[k^{(l(i))}]P$ sont stockées dans les registres respectifs $R_{\alpha(i)}$ et $R_{\beta(i)}$ et que le résultat $[k^{(i)}]P$ est écrit dans le registre $R_{\gamma(i)}$, l'exposant k peut être représenté par la séquence de triplets de registres

$$\Gamma(k) = \{(\gamma(i); \alpha(i), \beta(i))\}_{1 \leq i \leq r},$$

chacun signifiant que $R_{\gamma(i)} = R_{\alpha(i)} + R_{\beta(i)}$ (par convention, la valeur $k = 1$ est représentée par $\Gamma(1) = \emptyset$).

Il peut en être déduit l'algorithme de multiplication scalaire 17. Il faut noter que $R_{\alpha(1)}$ et $R_{\beta(1)}$ sont initialisés à P car le deuxième élément d'une chaîne d'additions est toujours $k^{(1)} = 2$.

En supposant que l'addition $R_{\gamma(i)} \leftarrow R_{\alpha(i)} + R_{\beta(i)}$ ne fuit pas d'information secrète, Clavier [Cla07] (*et al.* [CJ01]) « prouve(nt) » la sécurité de cette implantation suivant plusieurs modèles généraux de sécurité. L'algorithme 17, appellé « algorithme universel »

Algorithme 17 Algorithme universel de multiplication scalaire

Entrées : P, $\Gamma(k) = \{(\gamma(i); \alpha(i), \beta(i))\}_{1 \leq i \leq r}$.
Sorties : $Q = [k]P$.

1. $R_{\alpha(1)} \leftarrow P$, $R_{\beta(1)} \leftarrow P$
2. **pour** $i = 1$ à r **faire**
3. $\quad R_{\gamma(i)} \leftarrow R_{\alpha(i)} + R_{\beta(i)}$
4. **fin pour**
5. **retourner** $R_{\gamma(r)}$

de multiplication scalaire[16] , peut émuler virtuellement toute méthode de multiplication scalaire. Il lit simplement des triplets de valeurs $(\gamma(i) : \alpha(i), \beta(i))$, chacun signifiant que le point contenu dans le registre $R_{\alpha(i)}$ doit être additionné au point contenu dans le registre $R_{\beta(i)}$ et que le résultat doit être écrit dans le registre $R_{\gamma(i)}$.

Utiliser cette méthode comporte plusieurs avantages.

Tout d'abord, d'un point de vue **sécuritaire**, l'utilisation de cette méthode ramène le problème d'examiner la sécurité de tout algorithme de multiplication scalaire à celle de l'opération élémentaire $R_{\gamma(i)} \leftarrow R_{\alpha(i)} + R_{\beta(i)}$ [Cla07] : cela rend le travail du concepteur de circuits sécurisés plus facile car il ne doit plus se préoccuper de la sécurité du cryptosystème au niveau macroscopique (l'implantation de la multiplication scalaire), mais de sa sécurité à un niveau plus bas (l'implantation des opérations modulaires).

La **rapidité** de l'algorithme de multiplication scalaire universel dépend de l'implantation de l'opération élémentaire $R_{\gamma(i)} \leftarrow R_{\alpha(i)} + R_{\beta(i)}$, mais également du nombre de fois que cette opération est effectuée : elle dépend donc de la longueur r de la chaîne d'additions $C(k)$. Schönnage [Sch75] démontre que la borne inférieure de r est égale à $(\ell + \log_2(\mathrm{HW}(k)) - 2{,}13)$, où $\mathrm{HW}(k)$ représente le poids de Hamming de k. Il montre également que trouver la chaîne d'addition la plus courte possible est un problème NP-complet. Cependant, différentes heuristiques efficaces ont été développées pour produire des chaînes d'additions relativement courtes. Par exemple, l'heuristique proposée par Walter [Wal98] conduit à la constitution de chaînes d'additions de longueur moyenne égale à $1{,}25\ell$. Cette heuristique a le double avantage de produire des chaînes d'addition de longueur proche de la borne inférieure mise en évidence par Schönnage, et d'être facilement implantable.

Ainsi, si on en revient aux performances de l'algorithme 17 utilisant une chaîne d'additions, celui-ci peut donc requérir en moyenne seulement $1{,}25\ell A$, contre $\ell D + \ell A$ dans le cadre de l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) : l'algorithme de multiplication scalaire universel utilisant une chaîne d'additions peut donc être très rapide.

[16] Dans [CJ01], cet algorithme est présenté sous sa version multiplicative dans le cadre du RSA. À l'origine, il est donc appelé « algorithme universel d'exponentiation ». Il a donc été adapté dans ce mémoire pour calculer une multiplication scalaire.

Malgré ses avantages d'utilisation en termes de sécurité et de performances, l'utilisation de cet algorithme entraîne quelques inconvénients.

1. Tout d'abord, il faut noter que le calcul de la chaîne d'additions doit être effectué en environnement sécurisé puisque sa seule connaissance permet de retrouver la clé k [Cla07].
2. Ensuite, cette méthode convient principalement à des scalaires k fixes. Si k varie (soit parce que le protocole ECDSA est utilisé − cf. Algorithmes 14 et 15, soit parce que des parades aux attaques différentielles par observation basées sur le choix aléatoire du scalaire sont implantées − cf. section 4.1.2.1), alors la chaîne d'additions doit être recalculée à la volée avant toute réexécution de l'algorithme 17.
3. De plus, il faut noter qu'il peut être obtenu $\alpha(1) = \beta(1)$ lors de l'établissement de $\Gamma(k)$, et dans ce cas, un point peut être additionné à lui-même lors de l'exécution de l'algorithme 17. Or, cet algorithme est résistant vis-à-vis des attaques SSCA justement parce qu'il n'est effectué que des additions de points. Utiliser cet algorithme nécessite donc que les opérations consistant à additionner deux points différents (donc effectuer l'addition de deux points) et à additionner deux points identiques (donc effectuer le doublement de ces deux points) soient indistinguables (les méthodes permettant de garantir cette condition seront décrites dans la section 4.1.1.2).

Pour conclure ce paragraphe, il faut noter que la recherche récente dans le domaine des algorithmes de multiplication scalaire utilisant des chaînes d'addition a produit des résultats intéressants. C'est ainsi que Méloni [Mel07a] propose d'utiliser des chaînes d'additions « euclidiennes » définies comme suit.

Une chaîne d'additions euclidienne $C'(k)$ est une chaîne d'additions de longueur r', où $k^{(0)} = 1$, $k^{(1)} = 2$, $k^{(2)} = k^{(0)} + k^{(1)} = 3$ et pour tout $3 \leq i \leq r'$ si $k^{(i)} = k^{(i-1)} + k^{(j(i))}$ pour un certain $j(i) < i - 1$, alors $k^{(i+1)} = k^{(i)} + k^{(i-1)}$ ou $k^{(i+1)} = k^{(i)} + k^{(j(i))}$. Trouver une chaîne d'additions euclidienne $C'(k)$ est relativement aisé : il suffit de fixer l'utilisation dans la chaîne $C'(k)$ d'un élément $k' = k^{(i)}$ premier avec k.

Exemple 4.2. Si on choisit de faire apparaître dans la chaîne d'additions euclidienne $C'(25)$ le nombre **16**, alors $C'(25)$ peut s'écrire : $C(25) = \{1, 2, 3, 5, 7, 9, \mathbf{16}, 25\}$,
car $[2]P = P + P$, $[5]P = [3]P + [2]P$, $[7]P = [5]P + [2]P$, $[9]P = [7]P + [2]P$, $[\mathbf{16}]P = [9]P + [7]P$, $[25]P = [\mathbf{16}]P + [9]P$.

Le choix de k' conditionne la longueur de la chaîne d'additions euclidienne, et donc, le nombre d'additions de points à effectuer par la suite. Il a été montré que choisir $k' = k/\phi$, où ϕ est le nombre d'or, permet d'obtenir une chaîne d'addition courte. L'utilisation de ces chaînes courtes, combinée à de nouvelles formules d'addition de points efficaces acceptant comme paramètres d'entrée deux points $P_1 = (X_1 : Y_1 : Z_1)$ et $P_2 = (X_2 : Y_2 : Z_2)$ avec

$Z_1 = Z_2$ permet de calculer $Q = [k]P$ grâce à $15{,}25\ell M$, ce qui en fait une contre-mesure SSCA compétitive.

Atomicité. L'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) ne contient pas de branchement conditionnel (dépendant des bits de clé) contrairement à l'algorithme du « doublement-et-addition » (cf. Algorithme 9) : ainsi, à chaque itération, un doublement est suivi par une addition de points, et ce, quelque soit la valeur du bit de clé traité. Cette idée a été généralisée et étendue par Chevallier-Mames *et al.* pour conduire au concept « d'atomicité » [CMCJ04] : lorsque le doublement et l'addition de points sont indistinguables *via* une attaque SSCA, le branchement conditionnel peut être éliminé de telle manière que l'algorithme de multiplication scalaire apparaisse comme une répétition d'additions de points[17]. Ce faisant, il peut être obtenu l'algorithme du « doublement-et-addition atomique » (cf. Algorithme 18).

Algorithme 18 Algorithme du « doublement-et-addition atomique »

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow P$; $R_1 \leftarrow P$; $i \leftarrow \ell - 2$; $b \leftarrow 0$
2. **tantque** $i \geq 0$ **faire**
3. $\quad R_0 \leftarrow R_0 + R_b$
4. $\quad b \leftarrow b \oplus k_i$; $i \leftarrow i + k_i - 1$
5. **fin tantque**
6. **retourner** R_0

Le principal avantage de cet algorithme est qu'il entraîne le calcul de seulement $\ell D + (\ell/2)A$ (en moyenne) : il a donc une complexité équivalente à celle de l'algorithme du « doublement-et-addition » (cf. Algorithme 9). Dans le cas où l'addition et la soustraction de points sont indistinguables *via* une attaque SSCA, l'algorithme 18 peut être remplacé à profit par sa version utilisant le $\mathrm{NAF}_2(k)$ [Joy05] (cf. section 2.2.3.2 pour une définition du NAF) : dans ce cas, le nombre d'opérations requis est $\ell D + (\ell/3)A$.

Bien que l'algorithme 18 soit très efficace, son utilisation implique cependant quelques contraintes.

1. Tout d'abord, son utilisation nécessite que le doublement et l'addition de points sont indistinguables (les méthodes permettant de garantir cette condition seront décrites dans la section 4.1.1.2).
2. Ensuite, la sécurité de l'algorithme 18 repose sur le fait que les opérations $b \leftarrow b \oplus k_i$ et $i \leftarrow i + k_i - 1$ (étape 4) n'apportent aucune information sur la valeur du bit de clé k_i. Si c'est toutefois le cas, des valeurs aléatoires peuvent être utilisées pour masquer la valeur de k_i : par exemple, l'opération $b \oplus k_i$ peut être remplacée par

[17] Il faut noter que Joye [Joy02] avait publié avant [CMCJ04] des algorithmes d'exponentiation (pour le RSA, version multiplicative de la multiplication scalaire) « atomiques ».

$b \oplus (k_i \oplus r) \oplus r$ pour un bit aléatoire r, et l'opération $i + k_i - 1$ peut être remplacée par $i + (k_i + t) - (t + 1)$ pour un nombre aléatoire t.

3. Enfin, cet algorithme fournit à l'attaquant le nombre d'additions de points effectuées, et donc potentiellement le poids de Hamming du scalaire k. Or, lorsque ce poids de Hamming est extrêmement grand ou petit, un attaquant peut retrouver k à l'aide d'une recherche exhaustive [CKQ03].

   Le poids de Hamming de k peut être masqué en choisissant aléatoirement le scalaire k à chaque itération (cf. section 4.1.2.1). Il peut également être utilisé un algorithme de multiplication scalaire utilisant le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3), car cela permet d'obtenir un poids de Hamming de $\mathrm{NAF}_w(k)$ constant quelque soit la valeur de k.

4. Enfin, il faut noter que vu que le scalaire est traité de la gauche vers la droite, l'algorithme 18 peut être le siège d'une attaque dite « du doublement » [FV03] [KQ08].

Recoder le scalaire pour utiliser un algorithme de multiplication scalaire régulier. Une autre stratégie pour protéger un cryptosystème ECC contre les attaques SSCA est d'utiliser un recodage du scalaire de manière à obtenir un algorithme de multiplication scalaire qui répète une séquence fixée d'opérations de points : l'observation de ces dernières ne fournit donc aucune information à l'attaquant car elles sont indépendantes de la clé.

Une première technique de recodage utilise la base 2^w, où w représente la longueur de la fenêtre traitée par l'algorithme de multiplication scalaire sous-jacent [Möl01] [OT03] [The06] [Möl02] [IT02b]. Il peut également être utilisé le $\mathrm{NAF}_2(k)$ (cf. section 2.2.3.2) [CJ03] [HM02b].

Le tableau 4.1 résume les propriétés des différentes techniques de recodage, à savoir le type de recodage (deuxième colonne), l'ensemble des chiffres utilisés par ce dernier (troisième colonne), et la séquence d'opérations de points obtenue (quatrième colonne) se répétant n fois, où n dépend notamment du nombre de chiffres traité à chaque itération de l'algorithme.

Référence	**Recodage**	**Chiffres utilisés**	**Séquence**
[HM02b]	$\mathrm{NAF}_2(k)$	$D_1 = \{-1, 0, 1\}$	DDA
[CJ03]	$\mathrm{NAF}_2(k)$	D_1	DDAD
[Möl01], [Möl02]	Fenêtre	$\{-2^w, \pm 1, \pm 2, \cdots, \pm(2^{w-1} - 1), 2^{w-1}\}$	$\underbrace{\mathrm{D} \cdots \mathrm{D}}_{w \text{ fois}}\mathrm{A}$
[IT02b], [OT03], [The06]	Fenêtre	$\{\pm 1, \pm 3, \cdots, \pm(2^w - 1)\}$	$\underbrace{\mathrm{D} \cdots \mathrm{D}}_{w \text{ fois}}\mathrm{A}$

Tab. 4.1 – Résumé des propriétés des différentes techniques de recodage du scalaire permettant d'utiliser un algorithme de multiplication scalaire régulier.

Certaines séquences d'opérations de points détaillées dans le tableau 4.1 sont obtenues grâce à l'insertion d'opérations de points inutiles. Par conséquent, celles-ci sont vulnérables face aux attaques CSEA [YKLM01]. Pour contre-carrer cette attaque, une première approche consiste à choisir la valeur de la clé aléatoirement (cf. section 4.1.2.1). Il peut également être utilisé des recodages permettant d'exécuter une séquence régulière d'opérations, ces dernières étant **toutes utilisées** dans le calcul de la multiplication scalaire [Möl01] [OT03] [The06].

Möller [Möl01] ainsi qu'Okeya *et al.* [OT03] proposent de calculer le recodage de k de la droite vers la gauche. Cette approche comporte deux inconvénients. Tout d'abord, il faut un emplacement mémoire pour stocker la valeur résultant du recodage de k. Ensuite, il faut attendre que le recodage soit terminé afin de commencer à exécuter l'algorithme de multiplication scalaire, ce dernier se calculant de la gauche vers la droite. C'est pour cela que Thériault [The06] propose un algorithme permettant le recodage de k de la gauche vers la droite, ce qui permet d'effectuer la multiplication scalaire au fur et à mesure que le recodage se calcule.

Enfin, il faut noter que si un concepteur de circuits sécurisés dispose de deux processeurs pouvant fonctionner en parallèle, alors il peut utiliser les méthodes plus efficaces proposées par Möller [Möl02] et Izu *et al.* [IT02b].

Le tableau 4.2 détaille pour chaque méthode de recodage le nombre de processeurs nécessaires (deuxième colonne), le temps d'exécution de ces méthodes (troisième colonne), ainsi que la vulnérabilité de ces dernières face aux attaques CSEA (quatrième colonne).

Référence	**Procs.**	**Temps d'exécution**	**CSEA ?**
[HM02b]	1	$17,78\ell M$	Oui
[CJ03]	1	$20\ell M$	Oui
[Möl01]	1	$16\ell M$	Non
[The06]	1	$16\ell M$	Non
[Möl02]	2	$(16\lfloor \ell/2 \rfloor + 55)M$	Non
[IT02b]	2	$5,85\ell M$	Non

Tab. 4.2 – Quelques propriétés des différents algorithmes de multiplication scalaire réguliers découlant d'un recodage du scalaire.

Tout d'abord, à la lecture de la cinquième colonne du tableau 4.2, il peut être déconseillé l'utilisation des méthodes décrites dans [HM02b] et dans [CJ03], car ces dernières sont vulnérables face aux attaques par perturbation de type CSEA [YKLM01].

La deuxième colonne du tableau doit être considérée comme une contrainte d'implantation : par exemple, si un concepteur de circuits sécurisés dispose d'au moins deux processeurs, alors il pourra choisir l'option proposée dans [IT02b] qui est la plus performante. Par contre, si il ne dispose que d'un processeur, alors son choix va se porter sur la méthode décrite dans [The06], qui a l'avantage de recoder le scalaire de la gauche vers la droite, contrairement à celle décrite dans [Möl01].

Premier bilan et autres algorithmes de multiplication scalaire réguliers. Les méthodes permettant d'obtenir un algorithme de multiplication scalaire régulier décrites dans cette section ont parfois quelques inconvénients.

1. Certains algorithmes contiennent des opérations inutiles, qui peuvent être le siège d'attaques CSEA [YKLM01] : c'est par exemple le cas de l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) et de certaines techniques visant à recoder le scalaire (cf. tableau 4.2).
2. Certains algorithmes sont vulnérables vis-à-vis des attaques dite « du doublement » car le scalaire y est traité de la gauche vers la droite [FV03] [KQ08].
3. Certains algorithmes nécessitent une phase d'initialisation pendant laquelle des précalculs sont effectués, ces derniers entraînant une latence supplémentaire au système, ainsi que l'utilisation d'un grand nombre de registres pour stocker le résultat de ces précalculs : c'est par exemple le cas des algorithmes de multiplication scalaire utilisant des chaînes d'addition (cf. Algorithme 17) et ceux recodant le scalaire (cf. tableau 4.2).

Pour éviter les deux premiers inconvénients, qui sont d'ordre sécuritaires, une première approche peut être d'utiliser malgré leurs défauts ces algorithmes dans lesquels seront implantés des contre-mesures supplémentaires adaptées aux attaques CSEA et « du doublement ». Cependant, l'ajout de ces contre-mesures peut dégrader les performances de l'algorithme initial. De plus, l'implantation d'une contre-mesure peut introduire des vulnérabilités supplémentaires dans le cryptosystème. C'est par exemple le cas de l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) qui conduit au calcul d'une addition de points supplémentaire (inutile) pour chaque bit de clé $k_i = 0$ par rapport à l'algorithme original du « doublement-et-addition » (cf. Algorithme 9) : l'algorithme 16 est ainsi protégé vis-à-vis des attaques SSCA, mais il est vulnérable face aux attaques par perturbation de type CSEA. Cela complique donc la tâche d'un concepteur de circuits sécurisés qui devra avant toute implantation d'une nouvelle contre-mesure reconsidérer la sécurité globale de son cryptosystème. Son choix va donc se porter préférentiellement sur des algorithmes qui n'induisent pas les inconvénients décrits précédemment. En particulier, il préférera implanter des algorithmes de multiplication scalaire réguliers qui, intrinsèquement :

1. ne contiennent aucune opération inutile ;
2. traitent le scalaire de la droite vers la gauche ;
3. entraînent l'utilisation d'un nombre raisonnable de registres pour stocker les variables intermédiaires, et n'entraînent ni le recodage du scalaire, ni même aucun précalcul.

Joye [Joy07] a proposé des algorithmes réguliers obéissant à toutes ces contraintes de sécurité et de performance. Parmi ceux-ci, il peut être cité l'algorithme du « doublement-addition » (cf. Algorithme 19) qui exécute un doublement et une addition pour chaque bit de clé traité.

Algorithme 19 Algorithme du « doublement-addition »

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow \infty$; $R_1 \leftarrow P$
2. **pour** $i = 0$ à $\ell - 1$ **faire**
3. $\quad b \leftarrow 1 - k_i$; $[2]R_b \leftarrow [2]R_b + R_{k_i}$
4. **fin pour**
5. **retourner** R_0

Cet algorithme nécessite le calcul de $\ell(D + A)$, comme l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16), sauf qu'il n'est pas vulnérable face aux attaques CSEA [YKLM01] (car il ne contient aucune opération inutile) et « du doublement » [FV03] (car les bits du scalaire sont traités de la droite vers la gauche) contrairement à ce dernier.

Un second algorithme proposé par Joye n'exécute que des additions de points : c'est l'algorithme de « l'addition-seulement » (cf. Algorithme 20).

Algorithme 20 Algorithme de « l'addition-seulement »

Entrées : P, $k = (1k_{\ell-2} \cdots k_0)_2$.
Sorties : $Q = [k]P$.

1. $R_0 \leftarrow P$; $R_1 \leftarrow P$; $R_2 \leftarrow [2]P$
2. **pour** $i = 0$ à $\ell - 1$ **faire**
3. $\quad b \leftarrow 1 - k_i$; $R_b \leftarrow R_b + R_2$
4. $\quad R_2 \leftarrow R_0 + R_1$
5. **fin pour**
6. $b \leftarrow k_0$; $R_b \leftarrow R_b - P$
7. **retourner** R_0

L'algorithme 20 conduit au calcul de $2A(\ell + 2)$, ce qui peut paraître prohibitif au premier abord, mais il faut constater que cet algorithme utilise seulement l'opération d'addition de points : le fait de ne pas utiliser le doublement de points amène des économies en termes de surface occupée par le circuit et de taille occupée par le code résultat de la programmation logicielle du cryptosystème.

Un bilan sur l'efficacité et la sécurité de l'ensemble des contre-mesures consistant à utiliser un algorithme de multiplication scalaire régulier qui ont été présentées dans cette section sera effectué en section 4.1.1.3.

La section suivante détaille les différentes variantes de la seconde contre-mesure possible contre les attaques SSCA pour l'ECC : rendre le doublement et l'addition de points indistinguables.

4.1.1.2 Rendre le doublement et l'addition de points indistinguables

Il existe deux manières de rendre l'addition et le doublement de points indistinguables : utiliser des formules d'addition de points (intrinsèquement) valables à la fois pour le doublement et l'addition (on parle de formules « unifiées »), et insérer des opérations modulaires (inutiles) dans les formules d'addition et de doublement afin qu'elles exécutent les mêmes opérations aux mêmes moments.

Formules d'addition de points valable à la fois pour le doublement et l'addition. Une méthode efficace pour protéger un cryptosystème ECC contre les attaques SSCA est d'utiliser des formules d'addition de points (intrinsèquement) valables pour le doublement et l'addition de points. Cette approche a été étudiée sur les courbes de Weierstrass (cf. relation 2.1) et sur d'autres types de courbes spécifiques.

Formules unifiées de Brier *et al.*. Une potentielle source de vulnérabilité des implantations des algorithmes de multiplication scalaire face aux attaques SSCA réside dans le fait que les formules utilisées pour l'addition de deux points $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$ et pour le doublement sont différentes. Plus précisément, c'est la formulation du paramètre λ (cf. relation 2.2) qui change suivant qu'il est calculé un doublement ou une addition.

L'idée de Brier *et al.* est de trouver une formulation de λ valable à la fois pour l'addition et le doublement [BJ02]. Ainsi, si l'on fait subir à λ quelques manipulations algébriques, on peut obtenir :

$$\begin{aligned}
\lambda &= \frac{y_2 - y_1}{x_2 - x_1} \quad [x_1 \neq x_2] \\
&= \frac{y_2 - y_1}{x_2 - x_1} \cdot \frac{y_1 + y_2}{y_1 + y_2} \quad [x_1 \neq x_2, y_1 \neq -y_2] \\
&= \frac{y_2^2 - y_1^2}{(x_2 - x_1)(y_1 + y_2)} \quad [x_1 \neq x_2, y_1 \neq -y_2] \\
&= \frac{(x_2^3 + ax_2 + b) - (x_1^3 + ax_1 + b)}{(x_2 - x_1)(y_1 + y_2)} \quad [x_1 \neq x_2, y_1 \neq -y_2] \\
&= \frac{(x_2 - x_1)(x_1^2 + x_1x_2 + x_2^2) + a(x_2 - x_1)}{(x_2 - x_1)(y_1 + y_2)} \quad [x_1 \neq x_2, y_1 \neq -y_2].
\end{aligned}$$

Finalement, il peut être écrit :

$$\lambda = \frac{x_1^2 + x_1x_2 + x_2^2 + a}{(y_1 + y_2)} \quad [y_1 \neq -y_2]. \tag{4.1}$$

Or, cette expression de λ reste utilisable quand $P_1 = P_2$ et peut être ainsi utilisée pour calculer le doublement d'un point. En effet, en remplaçant (x_2, y_2) par (x_1, y_1) dans la

relation 4.1, il peut être obtenu

$$\lambda \quad = \quad \frac{3x_1^2 + a}{2y_1},$$

ce qui correspond à la relation 2.2.

Il peut donc être obtenu des formules unifiées d'addition nécessitant les mêmes calculs d'opérations pour l'addition et le doublement. Ainsi, le coût de l'addition (ou du doublement) de points suivant les formules unifiées de Brier *et al.* est de $I + 5M$. Si le système de coordonnées $\mathcal{P}$ est utilisé pour éviter le calcul d'inversions modulaires (cf. section 2.2.2), le coût de l'addition est de $17M$ plus une multiplication par la constante a. Lorsque $a = -1$, l'addition ne coûte plus que $16M$.

Attaques sur les formules unifiées de Brier *et al.* Les formules unifiées de Brier *et al.* permettent de se prémunir des attaques SSCA, mais leur utilisation entraîne un inconvénient : leur formulation de λ est non définie pour $y_1 = -y_2$ (cf. relation 4.1). Ce désagrément apparemment anodin a été exploité par Izu *et al.* [IT03] pour monter une attaque.

Soient deux points P_1 et P_2 d'une courbe elliptique E, tels que $P_1 \neq \pm P_2$. Pour obtenir $P_3 = P_1 + P_2$ en coordonnées affines, les formules unifiées nécessitent notamment le calcul de l'inversion de $y_1 + y_2$ (cf. relation 4.1). Si $y_1 + y_2 = 0$, alors le cryptosystème va renvoyer une erreur lors de l'addition de ces deux points [18] [IT03]. Ainsi, une attaque dite « par analyse des messages d'erreur » (cf. section 3.1.1.4) peut être imaginée.

Supposons que ce soit l'algorithme du « doublement-et-addition » qui est choisi pour calculer la multiplication scalaire (cf. Algorithme 9). Un attaquant peut ainsi choisir un point P, et observer le comportement du cryptosystème. Si celui-ci retourne une erreur, cela veut dire qu'à l'itération $i = a$ du calcul de $Q = [k]P$, le cryptosystème a été dans l'incapacité de calculer une addition de points. Plus précisément, le point P est tel que $y(P) + y([m]P) = 0$, avec $m = \sum_{i=a}^{\ell-1} k_i 2^{i-a}$: il a ainsi être retrouvé $\ell - a$ bits de k.

Il peut y avoir plusieurs points P ayant cette propriété : ils peuvent être trouvés au préalable par l'attaquant pour des petites valeurs de m [IT03].

Cette attaque peut être contre-carrée si le scalaire k change à chaque itération de la multiplication scalaire (cf. section 4.1.2.1) ou si le point P subit un procédé « d'aveuglement » (cf. section 4.1.2.2).

Une autre attaque visant les formules unifiées de Brier *et al.* a été présentée par Walter [Wal04b] lorsque l'algorithme du « doublement-et-addition » est utilisé (cf. Algorithme 9). Cette attaque utilise le fait que certaines implantations de la multiplication modulaire

[18] Lorsque des coordonnées projectives sont utilisées (cf. section 2.2.2), les formules de Brier *et al.* induisent le calcul de la coordonnée $Z_3 = 2Z_1^3 Z_2^3 (Y_1 Z_2 + Y_2 Z_1)^3$ lors de l'addition des points $P_3 = P_1 + P_2$ [BJ02]. Ainsi, pour que le cryptosystème renvoie une erreur lors de cette addition, il faut que $P_3 = \infty$, soit $Z_3 = 0$, ou encore $Y_1 Z_2 + Y_2 Z_1 = 0$.

(modulo p) utilisent une soustraction conditionnelle finale afin d'obtenir un résultat dans l'intervalle $[0, p[$: c'est par exemple le cas de la multiplication modulaire de Montgomery (cf. Algorithme 6). Il est fait l'hypothèse qu'un attaquant peut détecter la présence de cette soustraction à l'aide d'une SSCA.

Les formules unifiées d'addition de points contiennent les multiplications intermédiaires suivantes :

$$U_1 = X_1 Z_2,\ U_2 = X_2 Z_1,\ S_1 = Y_1 Z_2,\ S_2 = Y_2 Z_1.$$

Lorsque les formules d'addition de Brier *et al.* sont utilisées pour effectuer un doublement, alors les opérandes sont les mêmes pour les calculs de U_1 et de U_2, ainsi que pour les calculs de S_1 et de S_2. Ce n'est pas le cas lors d'une addition de points. Ainsi, si un attaquant observe une soustraction pendant le calcul de U_1 et non pendant le calcul de U_2, alors il est certain qu'il est en train de se dérouler une addition de points (il peut tirer la même conclusion si il observe une soustraction pendant le calcul de S_1 et non pendant le calcul de S_2).

Cette attaque nécessite en théorie un seul relevé de canal caché, mais si l'attaquant veut identifier de manière certaine les différentes additions de points, alors il lui faut un plus grand nombre de courbes. Par exemple, lorsque les calculs se déroulent modulo p_{192} (cf. section 2.3.2.2), l'acquisition et l'analyse de 512 relevés permet de réduire le nombre de candidats possibles pour k à $2^{17,6}$ [Wal04b], ce qui est réalisable en pratique avec les moyens de calcul actuels.

Du point de vue des contre-mesures à ces attaques, Walter montre que les parades efficaces contre l'attaque d'Izu *et al.* (changer le scalaire k à chaque itération de la multiplication scalaire, « aveugler » le point P) sont inefficaces, et pire encore cela peut la faciliter.

Il peut par contre être noté que cette attaque n'est pas réalisable si un algorithme de multiplication scalaire utilisant le $\mathrm{NAF}_w(k)$ est implanté (cf. section 2.2.3). Une autre contre-mesure efficace à cette attaque est d'utiliser un algorithme de multiplication modulaire qui ne contient pas de soustraction conditionnelle.

Une dernière attaque possible sur les formules unifiées de Brier *et al.* utilise le fait que lorsque $P_1 = P_2$, des multiplications modulaires nécessaires au calcul de l'addition de points deviennent des carrés modulaires : ces derniers peuvent être observés à l'aide d'une attaque SSCA (en utilisant un modèle de consommation basé sur le poids de Hamming des variables intermédiaires, cf. section 3.1.3.3), et ainsi permettre de différencier les additions de points pour lesquelles $P_1 = P_2$ et $P_1 \neq P_2$ [SST04] [AFT+08]. L'exploitation de cette différentiation afin de récupérer la clé k_i dépend de l'algorithme de multiplication scalaire utilisant les formules unifiées de Brier *et al.* : si l'algorithme du « doublement-et-addition » est utilisé (cf. Algorithme 9), alors un motif de canal caché correspondant à l'addition de points avec $P_1 = P_2$ suivi d'un autre motif correspondant à l'addition de points avec $P_1 = P_2$ indique le traitement d'un bit de clé $k_i = 1$.

Cette attaque a été mise en évidence sur plusieurs méthodes de multiplication mo-

dulaire, en particulier la multiplication modulaire de Montgomery (cf. Algorithme 8). Il faut noter que cette attaque ne s'applique pas lorsque l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) ou l'échelle de Montgomery (cf. Algorithme 11) sont utilisés : une première contre-mesure à cette attaque peut donc consister à utiliser ces algorithmes de multiplication scalaire lorsque les formules unifiées de Brier *et al.* sont choisies. Une seconde contre-mesure possible est de changer le scalaire k à chaque itération (cf. section 4.1.2.1).

Formules unifiées généralisées de Déchène *et al.*. Pour éviter l'attaque d'Izu *et al.*, Déchène *et al.* [DBJ04] ont proposé une généralisation des formules unifiées données par Brier *et al.*.

Soient deux points $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$ d'une courbe de Weierstrass (cf. relation 2.1). On a donc

$$y_1^2 = x_1^3 + ax_1 + b \tag{4.2}$$

et

$$y_2^2 = x_2^3 + ax_2 + b. \tag{4.3}$$

Si l'opération $4.2 - 4.3$ est effectuée, on obtient

$$(y_1 + y_2)(y_2 - y_1) = (x_1^2 + x_1x_2 + x_2^2 + a)(x_2 - x_1). \tag{4.4}$$

Soit $m = m(x_1, y_1; x_2, y_2)$. En ajoutant $(x_1 - x_2)(y_1 - y_2)m$ de chaque côté de la relation 4.4, on obtient

$$(y_1 + y_2 + (x_1 - x_2)m)(y_2 - y_1) = (x_1^2 + x_1x_2 + x_2^2 + a + (y_1 - y_2)m)(x_2 - x_1). \tag{4.5}$$

En utilisant $\tilde{m} = m(x_2, y_2; x_1, y_1)$ et par symétrie, la relation 4.5 devient

$$(y_1 + y_2 + (x_2 - x_1)\tilde{m})(y_1 - y_2) = (x_1^2 + x_1x_2 + x_2^2 + a + (y_2 - y_1)\tilde{m})(x_1 - x_2). \tag{4.6}$$

Ainsi, λ obéit à deux relations différentes :

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \begin{cases} \dfrac{x_1^2 + x_1x_2 + x_2^2 + a + (y_1 - y_2)m}{y_1 + y_2 + (x_1 - x_2)m} & [y_1 + y_2 + (x_1 - x_2)m \neq 0] \\ \dfrac{x_1^2 + x_1x_2 + x_2^2 + a + (y_2 - y_1)\tilde{m}}{y_1 + y_2 + (x_2 - x_1)\tilde{m}} & [y_1 + y_2 + (x_2 - x_1)\tilde{m} \neq 0] \end{cases} \tag{4.7}$$

Il reste à montrer que λ est défini pour tous les points pour ne pas succomber à l'attaque d'Izu *et al.*. Les formules définies dans la relation 4.7 restent valables pour tous les points, sauf ceux qui satisfont : $y_1 + y_2 + (x_1 - x_2)m = 0 = y_1 + y_2 + (x_2 - x_1)\tilde{m}$. Cela implique dans un premier temps que si $x_1 = x_2$, alors $y_1 = -y_2$, donc $P_1 = -P_2$, ce qui est vrai par définition.

Dans un second temps, $[(x_1 \neq x_2) \Rightarrow (m + \tilde{m} = 0)]$. Ainsi la condition

$$[(m + \tilde{m} = 0) \Rightarrow (x_1 = x_2)] \tag{4.8}$$

assure que λ est entièrement défini. L'expression de λ est donc définie pour tous les points si m est choisi correctement. Une solution générale à la condition 4.8 est $m_k = (x_1 - x_2)^{2k}$. Pour des raisons d'efficacité, il peut être pris la solution particulière $m = 1$, et la relation 4.7 peut finalement s'écrire

$$\lambda = \begin{cases} \dfrac{x_1^2 + x_1 x_2 + x_2^2 + a + y_1 - y_2}{y_1 + y_2 + x_1 - x_2} & [x_2 \neq x_1 + y_1 + y_2] \\ \dfrac{x_1^2 + x_1 x_2 + x_2^2 + a + y_2 - y_1}{y_1 + y_2 + x_2 - x_1} & [\text{sinon}] \end{cases} \tag{4.9}$$

Il peut être noté que dans le cas particulier où $m = 0$ dans la relation 4.7, cette dernière correspond à l'expression de λ donnée par Brier *et al.* (cf. relation 4.1) : ainsi, il peut être considéré que l'expression de λ mise en évidence par Déchène *et al.* est une généralisation de celle donnée par Brier *et al.*.

Le coût de l'addition (ou du doublement) de points suivant les formules unifiées de Déchène *et al.* est de $I + 5M$. Si le système de coordonnées $\mathcal{P}$ est utilisé pour éviter le calcul d'inversions modulaires (cf. section 2.2.2), le coût de l'addition est de $19M$, soit seulement deux multiplications modulaires supplémentaires à calculer par rapport aux relations de Brier *et al.* (ou trois dans le cas où $a = -1$).

Il peut donc être conclu que l'utilisation des formules unifiées de Déchène *et al.* entraîne un surcoût raisonnable par rapport aux formules de Brier *et al.* et permet de surcroit de se prémunir de l'attaque décrite par Izu *et al.* : la contre-mesure de Déchène *et al.* peut donc être préférée à celle de Brier *et al.*.

Attaques sur les formules unifiées de Déchène *et al.*. Stebila *et al.* [ST06] ont montré que l'attaque proposée par Walter [Wal04b] sur les formules unifiées de Brier *et al.* pouvait également s'appliquer sur les formules unifiées de Déchène *et al.*. Comme Walter, Stebila *et al.* utilisent également le fait que certaines implantations de la multiplication modulaire (modulo p) utilisent une soustraction conditionnelle finale afin d'obtenir un résultat dans l'intervalle $[0, p[$. Mais ils montrent également que d'autres opérations dans les corps finis premiers qui utilisent une soustraction conditionnelle, comme **l'addition ou la soustraction modulaire**, peuvent également être le siège d'attaques.

Les formules unifiées d'addition de points de Déchène *et al.* contiennent les multiplications **et les soustractions** intermédiaires suivantes :

$$U_1 = X_1 Z_2,\ U_2 = X_2 Z_1,\ S_1 = Y_1 Z_2,\ S_2 = Y_2 Z_1,\ \boldsymbol{V = U_1 - U_2},\ \boldsymbol{N = S_1 - S_2}.$$

Il y a donc quatre événements conditionnels (dont un attaquant peut détecter la présence à l'aide d'une SSCA) qui peuvent permettre de distinguer une addition de points d'un doublement :

1. la soustraction conditionnelle dans le calcul soit de U_1 soit de U_2 mais pas dans l'autre,
2. la soustraction conditionnelle dans le calcul soit de S_1 soit de S_2 mais pas dans l'autre,
3. **l'addition conditionnelle dans le calcul de $\boldsymbol{V = U_1 - U_2}$,**
4. **l'addition conditionnelle dans le calcul de $\boldsymbol{N = S_1 - S_2}$.**

Ainsi, **l'observation des événements 3. et 4.** en plus des événements 1. et 2. permettent d'améliorer l'efficacité de l'attaque de Walter. Par exemple, lorsque les calculs se déroulent modulo p_{192} (cf. section 2.3.2.2), l'acquisition et l'analyse de 512 relevés permet de réduire le nombre de candidats possibles pour k à $2^{9,67}$, contre $2^{17,6}$ pour l'attaque de Walter.

Une contre-mesure efficace consiste à utiliser un algorithme de multiplication **et d'addition/soustraction** modulaire qui ne contiennent pas de soustraction conditionnelle.

Enfin, il faut noter que l'attaque mise en évidence par Amiel *et al.* [AFT+08] (cf. section précédente) peut être également montée sur les formules unifiées de Déchène *et al.*. Utiliser l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16), l'échelle de Montgomery (cf. Algorithme 11) ou changer le scalaire k à chaque itération (cf. section 4.1.2.1) sont des contre-mesures efficaces face à cette attaque.

Courbes possédant des formules d'addition de points intrinsèquement unifiées. Il a été détaillé précédemment des formules unifiées d'addition de points (valables également pour le doublement de points) pour les courbes de Weierstrass d'équation E_1 (cf. relation 2.1), ces formules étant le résultat d'une série de manipulations algébriques. D'autres courbes d'équation E_2 possèdent intrinsèquement des formules unifiées d'additions de points plus performantes : c'est par exemple le cas des courbes Jacobiennes, Hessiennes et d'Edwards. Ainsi, si il peut être trouvé des isomorphismes de courbes $\varphi : E_1 \rightarrow E_2$ (un isomorphisme est une isogénie de degré $m = 1$, cf. section 2.2.4.1), alors un concepteur de circuits sécurisés peut espérer améliorer les performances de son cryptosystème.

Il faut cependant noter que, dans la plupart des cas, ces isomorphismes n'existent que lorsque E_1 contient un (ou des) point(s) d'ordre deux ou trois (rappelons qu'un point est d'ordre n, si et seulement si $[n]P = \infty$, cf. section 2.1.4). Or, les courbes standardisées [FIP00] [SEC00b] (cf. section 2.3.2.1) ne contiennent pas de tels points. Ainsi, pour profiter des avantages liés à l'utilisation des courbes Jacobiennes, Hessiennes ou d'Edwards, il faudra que le concepteur de circuits sécurisés choisisse d'implanter des courbes elliptiques non-standardisées. Afin que cela ne se fasse pas au détriment de la sécurité, il faut donc qu'il choisisse les paramètres de courbe a, b et p de telle sorte que l'ECDLP soit difficile à résoudre sur ces courbes (cf. tableau 2.8 pour la liste des contraintes de sécurité sur ces paramètres). De manière générale, il peut être conseillé de choisir les paramètres a et p comme décrit dans la section 2.3.2, et de faire varier le paramètre b jusqu'à obtenir le (ou les) point(s) d'ordre deux ou trois nécessaires à l'existence de l'isomorphisme recherché,

tout en respectant les contraintes de sécurité élémentaires (notamment $\#E(\mathbb{F}_p) = hn$, avec $h \leq 4$).

Courbes Jacobiennes. Liardet *et al.* [LS01] ont montré que les formules d'addition et de doublement sur les courbes Jacobiennes définies par l'intersection de deux courbes quadratiques d'équations

$$\begin{cases} x^2 + y^2 = 1 \\ K^2x^2 + z^2 = 1 \end{cases}$$

sont intrinsèquement les mêmes. En utilisant des coordonnées projectives, un point (x, y, z) est représenté par le quadruplet $(X : Y : Z : T)$, et il obéit au système d'équations :

$$\begin{cases} X^2 + Y^2 = T^2 \\ K^2X^2 + Z^2 = T^2. \end{cases}$$

Une courbe elliptique E_1 obéissant à l'équation de Weierstrass (cf. relation 2.1) contenant **trois points d'ordre deux** sur $\mathbb{F}_p$ peut être isomorphique à cette forme de courbes Jacobiennes : dans ce cas, il peut donc être prouvé que $\#E_1 = hn$ (où $n = \mathrm{ord}_{E_1}(P)$, cf. section 2.1.4) avec $h \propto 4$ [LS01]. Le coût d'une addition (ou d'un doublement) de points en utilisant les formules données par Liardet *et al.* est de 16M (plus une multiplication par la constante K^2).

L'approche de Liardet *et al.* a été par la suite généralisée par Billet *et al.* [BJ03a] aux courbes elliptiques E_1 obéissant à l'équation de Weierstrass (cf. relation 2.1) et possédant **un seul point d'ordre deux** (ce qui implique que $\#E_1 = hn$ avec $h \propto 2$). Ils montrent que de telles courbes sont isomorphiques aux courbes E_2 ayant pour équation :

$$E_2 : y^2 = \epsilon x^4 - 2\delta x^2 + 1. \tag{4.10}$$

Dans la suite, il sera utilisé la version projective de 4.10 :

$$Y^2 = \varepsilon X^4 - 2\delta X^2Z^2 + Z^4. \tag{4.11}$$

Soient $P_1 = (X_1 : Y_1 : Z_1)$ et $P_2 = (X_2 : Y_2 : Z_2)$ deux points obéissant à la relation 4.11. L'opposé de P_1 s'écrit $-P_1 = (-X_1 : Y_1 : Z_1)$, et l'addition de $P_1 = (X_1 : Y_1 : Z_1)$ et de $P_2 = (X_2 : Y_2 : Z_2)$ est donné par :

$$\begin{cases} X_3 = X_1Y_2Z_1 + X_2Y_1Z_2 \\ Y_3 = (Z_1^2Z_2^2 + \varepsilon X_1^2X_2^2)(Y_1Y_2 - 2\delta X_1X_2Z_1Z_2) + 2\varepsilon X_1X_2Z_1Z_2(X_1^2Z_2^2 - X_2^2Z_1^2) \\ Z_3 = (Z_1^2Z_2^2 - \varepsilon X_1^2X_2^2) \end{cases}$$

Billet *et al.* montrent que ces formules sont également valables pour le doublement. Le coût de l'addition (ou d'un doublement) en utilisant cette formulation est de 13M, auquel il faut rajouter deux multiplications par la constante ε et une par la constante δ. Il est donc possible de diminuer ce coût en se plaçant dans le cas où $\varepsilon = 1$: il serait ainsi économisé les deux multiplications par ε.

Billet *et al.* montre que cela est possible en considérant comme Liardet *et al.* [LS01] le cas où une courbe elliptique de Weierstrass E_1 possède **trois points d'ordre deux** (et donc que $\#E_1 = hn$ avec $h \propto 4$). Dans ce cas, ils montrent que la courbe obéissant à la relation 4.11 est équivalente à la courbe quartique E_2' d'équation :

$$E_2' : Y^2 = X^4 - 2\rho X^2 Z^2 + Z^4. \tag{4.12}$$

Le premier avantage d'utiliser la courbe quartique E_2' plutôt que l'intersection de deux courbes quadratiques proposée par Liardet *et al.* est que seulement 13M (plus une par la constante ρ) sont nécessaires pour l'addition (ou le doublement) de points, soit 3M de moins par rapport à ce qui est proposé par Liardet *et al.*. Le second avantage est que les points appartenant à E_2' sont représentés par des triplets $(X : Y : Z)$, au lieu de quadruplets $(X : Y : Z : T)$ dans le cadre de la formulation proposée par Liardet *et al.* : le nombre de registres nécessaires est donc diminué.

Duquesne [Duq07] propose quant à lui des formules unifiées d'addition de points plus efficaces que celles décrites dans [BJ03a] en introduisant un nouveau système de coordonnées pour représenter les points de la courbe obéissant à la relation 4.11 : ses formules conduisent au calcul de $11M$, plus deux multiplications par la constante ε et une par δ (cf. relation 4.11). Ainsi, comme dans [BJ03a], il est intéressant de se placer dans le cas où $\varepsilon = 1$ afin d'économiser les deux multiplications par ε. Or, pour ce faire, et contrairement à ce qui est proposé dans [BJ03a], Duquesne indique qu'il n'est pas nécessaire (dans la plupart des cas) que la courbe elliptique de Weierstrass E_1 contienne trois points d'ordre deux. Finalement, les formules unifiées données par Duquesne nécessitent le calcul de $11M$ (plus 1M par la constante δ) et nécessitent que la courbe elliptique E_1 ne contienne seulement **un point d'ordre deux**.

Enfin, il faut noter également que l'approche de Billet *et al.* [BJ03a] a été généralisée par Olson [Ols04] : elle a montré comment il est possible de se placer sur une courbe quartique (utilisant des coordonnées projectives) sur laquelle l'addition et le doublement de points utilisent les mêmes formules, et ce, pour n'importe quelle courbe elliptique (dont celles de Weierstrass, cf. relation 2.1). Cette généralisation est coûteuse, puisqu'elle requiert dans le cas général le calcul de $31M$, mais si la courbe elliptique initiale contient des points aux propriétés particulières (par exemple un point $M = (\theta, 0)$ d'ordre deux, ou encore un point $M' = (0, \theta')$) alors les formules unifiées données par Olson peuvent être le cas échéant plus performantes.

Courbes Hessiennes. La plupart des courbes de Weierstrass d'équation E_1 contenant un point d'ordre 3 (donc $\#E_1 = hn$ et $h \propto 3$) peuvent être exprimées sous la forme Hessienne suivante :

$$X^3 + Y^3 + Z^3 = cXYZ$$

où $c \in \mathbb{F}_p$. Cette formulation a été proposée par Joye *et al.* [JQ01].

Soient $P_1 = (X_1 : Y_1 : Z_1)$ et $P_2 = (X_2 : Y_2 : Z_2)$ deux points d'une courbe Hessienne

H. L'opposé du point P_1 s'écrit $-P_1 = (Y_1 : X_1 : Z_1)$, et l'addition de points sur les courbes Hessiennes s'écrit, lorsque $P_1 \neq P_2$:

$$(X_1 : Y_1 : Z_1)+(X_2 : Y_2 : Z_2) = (X_2Y_1^2Z_2-X_1Y_2^2Z_1 : X_1^2Y_2Z_2-X_2^2Y_1Z_1 : X_2Y_2Z_1^2-X_1Y_1Z_2^2). \quad (4.13)$$

De plus, le doublement de points s'écrit :

$$[2](X_1 : Y_1 : Z_1) = (Y_1(X_1^3 - Z_1^3) : X_1(Z_1^3 - Y_1^3) : Z_1(Y_1^3 - X_1^3)).$$

Les formules calculant l'addition de deux points distincts et le doublement de points apparaissent à première vue différentes. Cependant, l'opération de doublement peut être réécrite en utilisant la formulation de l'addition de points (cf. relation 4.13) :

$$[2](X_1 : Y_1 : Z_1) = (Z_1 : X_1 : Y_1) + (Y_1 : Z_1 : X_1).$$

Ainsi, plus généralement, si l'addition de points est codée à l'aide d'une fonction nommée « HessianAdd »

$$\text{HessianAdd}(X_1 : Y_1 : Z_1, X_2 : Y_2 : Z_2) \;=\; P_1 + P_2$$

alors, il peut être déduit que :

$$\begin{aligned} \text{HessianAdd}(Z_1 : X_1 : Y_1, Y_1 : Z_1 : X_1) &= [2]P_1 \\ \text{HessianAdd}(X_1 : Y_1 : Z_1, Y_2 : X_2 : Z_2) &= P_1 - P_2 \end{aligned}$$

Le tableau 4.3 détaille toutes les possibilités d'implantation des formules unifiées Hessiennes.

Implantation	**Registres nécessaires**	**Coût d'une A (ou d'un D)**
Numéro 1	4	18M
Numéro 2	5	16M
Numéro 3	6	14M
Numéro 4	7	12M

TAB. 4.3 – Possibilités d'implantation des formules unifiées Hessiennes.

Le nombre de registres nécessaires (deuxième colonne du tableau 4.3) doit être considéré comme une contrainte d'implantation : par exemple, si un concepteur de circuits sécurisés ne dispose que de cinq registres, alors son choix sera restreint aux implantations numéro 1 et 2 du tableau 4.3, et il choisira donc la plus rapide des deux options qui est la numéro 2. En revanche, si il dispose d'un très grand nombre de registres, alors il pourra choisir l'option la plus performante qui est la numéro 4 du tableau 4.3. Il faut noter que cette dernière option d'implantation a un niveau de performance comparable aux formules Jacobiennes ayant un coût d'implantation minimale donnée dans [Duq07].

Enfin, il faut noter que les formules unifiées d'addition de points sur les courbes Hessiennes peuvent être calculées par trois processeurs en parallèle [Sma01] : dans ce cas, le temps nécessaire au calcul d'une addition de points est celui de $4M$.

Courbes d'Edwards. Les courbes d'Edwards [Edw07] ont déjà été décrites en section 2.2.4.3 (cf. relation 2.10). Rappelons que Bernstein *et al.* [BL07a] ont introduit un paramètre supplémentaire d par rapport aux courbes proposées par Edwards, ce qui permet de couvrir plus de courbes sur $\mathbb{F}_p$ (cf. relation 2.11).

Un grand avantage des courbes d'Edwards est que leur loi de groupe est **homogène** : il peut être calculé l'addition et le doublement de points grâce aux mêmes formules (le point à l'infini étant le point affine $\infty = (0, c)$).

Si une courbe elliptique de Weierstrass E_1 contient **un seul point d'ordre deux**, alors elle est équivalente à une courbe d'Edwards E_2 obéissant à la relation 2.11, et ayant un paramètre d qui n'est pas un carré sur $\mathbb{F}_p$ (il peut aussi être prouvé que si les points $P_1 = (x_1, y_1)$ et $P_2 = (x_2, y_2)$ sont sur la courbe obéissant à la relation 2.11, alors $dx_1x_2y_1y_2 \neq \pm 1$). Ce faisant, la loi de groupe sur E_2 est également **complète** : les formules unifiées d'addition (et de doublement) de points restent utilisables pour tous les points, ce qui n'est par exemple pas le cas des formules unifiées de Brier *et al.*. De plus, les formules unifiées d'addition de points sur les courbes d'Edwards ne nécessitent pas de réarrangement des coordonnées : il est calculé $[2](X_1 : Y_1 : Z_1) = (X_1 : Y_1 : Z_1) + (X_1 : Y_1 : Z_1)$ (contrairement aux courbes Hessiennes, où lorsqu'un doublement de point est effectué, il doit être calculé $[2](X_1 : Y_1 : Z_1) = (Z_1 : X_1 : Y_1) + (Y_1 : Z_1 : X_1)$).

Enfin, il faut noter que les formules unifiées d'addition de points sur les courbes d'Edwards sont très performantes, puisqu'elles nécessitent le calcul de seulement $12M$, comme les formules unifiées d'addition de points les plus efficaces sur courbes Jacobiennes [Duq07] et Hessiennes [JQ01].

Attaques sur les courbes possédant des formules d'addition de points intrinsèquement unifiées. Les formules d'addition de points intrinsèquement unifiées peuvent être le sujet des mêmes attaques que celles mises en évidence sur les formules unifiées valables sur les courbes de Weierstrass. Cette section est consacrée à un bilan des attaques possibles et des contre-mesures associées.

1. Tout d'abord, l'attaque dite « par analyse des messages d'erreur » mise en évidence par Izu *et al.* [IT03] n'est pas applicable sur les formules unifiées Jacobiennes, Hessiennes ou d'Edwards car la loi de groupe sur ces dernières est complète.

2. Ensuite, l'attaque de Walter [Wal04b] et la variante de cette dernière proposée par Stebila *et al.* [ST06] sont possibles sur ces formules unifiées lorsqu'un co-processeur ECC est constitué d'opérateurs arithmétiques modulaires (addition, soustraction, multiplication) utilisant des soustractions conditionnelles.

   Il faut noter que ces attaques ne sont pas réalisables si un algorithme de multiplication scalaire utilisant le $\mathrm{NAF}_w(k)$ est implanté (cf. section 2.2.3). Une autre contre-mesure efficace à ces attaques est d'utiliser un algorithme de multiplication

modulaire et d'addition/soustraction modulaire qui ne contiennent pas de soustraction conditionelle.

3. Enfin, les attaques décrites dans [SST04] [AFT+08] visant à discerner les multiplications des carrés modulaires sont possibles sur ces formules unifiées.

   Utiliser l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16), l'échelle de Montgomery (cf. Algorithme 11) ou changer le scalaire k à chaque itération (cf. section 4.1.2.1) sont des contre-mesures efficaces face à cette attaque.

Comparaison des formules unifiées. Le tableau 4.4 compare les différentes formules unifiées évoquées précédemment, en terme de nombre de processeurs nécessaires (deuxième colonne), ainsi qu'en terme de nombre de multiplications modulaires à calculer par addition ou par doublement de points (troisième colonne). La quatrième colonne de ce tableau précise également le cofacteur que doit posséder la courbe de Weierstrass initiale E_1 afin qu'un isomorphisme vers une courbe E_2 (Jacobienne, Hessienne ou d'Edwards) soit calculable.

Référence	**Processeurs**	**Coût de l'A**	**Cofacteur de E_1**
[BJ02]	1	$18M$ (cas général)	–
[BJ02]	1	$16M$ ($a = -1$)	–
[DBJ04]	1	$19M$ (cas général)	–
[DBJ04]	1	$17M$ ($a = -1$)	–
[LS01]	1	$17M$	$h \propto 4$
[BJ03a]	1	$16M$ (cas général)	$h \propto 2$
[BJ03a]	1	$14M$ ($\varepsilon = 1$)	$h \propto 4$
[Duq07]	1	$14M$ (cas général)	$h \propto 2$
[Duq07]	1	$12M$ ($\varepsilon = 1$)	$h \propto 2$
[JQ01]	1	$12M$	$h \propto 3$
[Sma01]	3	$4M$	$h \propto 3$
[BL07a]	1	$13M$ (cas général)	$h \propto 2$
[BL07a]	1	$12M$ ($c = 1$)	$h \propto 2$

Tab. 4.4 – Comparaison du coût de l'utilisation des différentes formules unifiées.

La deuxième colonne doit être vue comme une contrainte d'implantation. Par exemple, si un concepteur de circuits sécurisés ne peut utiliser qu'un seul processeur, alors il lui sera impossible d'implanter la solution détaillée dans [Sma01].

Si il est dans l'obligation d'utiliser des courbes standardisées [FIP00] [SEC00b], alors son choix est restreint aux solutions proposées dans [BJ02] et dans [DBJ04]. Vu que les formules unifiées proposées dans [BJ02] sont vulnérables face à l'attaque de Izu *et al.* [IT03], et plutôt que d'ajouter des contre-mesures spécialement pour cette attaque, il est plutôt conseillé de choisir les formules unifiées de Déchène *et al.* [DBJ04].

Si un concepteur de circuits sécurisés a toute latitude sur le choix de la courbe elliptique sur laquelle se déroulera la multiplication scalaire, et si il dispose d'au moins trois processeurs à sa disposition, alors il pourra implanter la solution très rapide proposée par Smart [Sma01]. Il faudra cependant qu'il trouve une courbe de Weierstrass E_1 avec $h \propto 3$.

Enfin, si il ne dispose que d'un seul processeur, alors il pourra choisir d'implanter les formules Hessiennes proposées par Joye *et al.* [JQ01], ces dernières entraînant le calcul de seulement $12M$ pour une addition ou un doublement de points comme celles proposées dans [Duq07] et [BL07a], mais nécessitant un plus petit nombre de registres (7).

Afin de rendre le doublement et l'addition de points indistinguables, une première méthode détaillée dans la section précédente réside dans l'utilisation de formules unifiées d'additions de points. Une autre méthode possible consiste à appliquer le principe d'atomicité à l'addition et au doublement de points.

Atomicité. Le concept d'atomicité a déjà été décrit précédemment comme un moyen d'obtenir un algorithme de multiplication scalaire régulier. Ainsi, l'algorithme 18 exécute une même opération élémentaire dite « atomique » à chaque itération (ici, une addition de points), et ce, quelque soit la valeur du bit de clé traité. Dans ce cas précis, le concept d'atomicité est considéré au niveau des opérations de point : le même raisonnement peut être appliqué au niveau des opérations modulaires impliquées dans le calcul de l'addition et du doublement de points.

C'est ce que proposent Trichina *et al.* [TB02], Gebotys *et al.* [GG02] et Chevallier-Mames *et al.* [CMCJ04]. L'addition et le doublement de points sont découpés en blocs élémentaires dits « atomiques » et notés Γ. Ainsi, même si l'algorithme de multiplication scalaire choisi ne calcule une addition de points que lorsque le bit de clé traité k_i a une certaine valeur (c'est par exemple le cas de l'algorithme du « doublement-et-addition », dans lequel une addition de points est effectuée uniquement si $k_i = 1$, cf. Algorithme 9), le canal caché mesuré apparaît désormais comme une succession régulière du motif $\{\Gamma\}$: cela ne donne aucune information sur les bits de la clé. Il faut noter que ce concept n'est valable que si un concepteur de circuits sécurisés autorise le fait que certaines opérations modulaires contenues dans le bloc Γ s'exécutent parfois inutilement.

Exemple 4.3. Considérons le cas où un concepteur de circuits sécurisés décide d'implanter les formules d'addition et de doublement sur les courbes de Weierstrass (cf. relation 2.2) sur le système coordonnées $\mathcal{A}$. Dans ces conditions, un doublement de points coûte $I + 4M$, tandis qu'une addition de points coûte $I + 3M$ (en négligeant le coût des additions modulaires par rapport aux multiplications modulaires, cf. tableau 2.3).
Ainsi, si un concepteur de circuits sécurisés choisit d'implanter le bloc atomique $\Gamma = \{IMMMM\}$, alors lorsqu'une addition de points sera effectuée, une des multiplications modulaires sera calculée inutilement.

Il a déjà été décrit précédemment des algorithmes de multiplication scalaire insérant des opérations de points inutiles afin d'exécuter la même séquence d'opérations, et ce quelque soit le bit de clé traité. C'est par exemple le cas de l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) : une addition de points inutile est effectuée lorsque $k_i = 0$. Un des principaux inconvénients de cette contre-mesure SSCA est son impact sur les performances du cryptosystème : le nombre moyen d'opérations de points à effectuer est augmenté de 33% par rapport à l'algorithme non-protégé du « doublement-et-addition » (cf. Algorithme 9). Ainsi, afin de limiter cet impact, il faut considérer l'atomicité au niveau des opérations modulaires et non au niveau des opérations de points.

Dans [GG02], il est implanté les formules d'addition et de doublement de points sur les courbes de Weierstrass (en considérant le cas particulier $a = -3$, cf. section 2.3.2.1) dans le système de coordonnées $\mathcal{J}$: le bloc atomique choisi Γ contient $9M$, 6 additions/soustractions modulaires, et 5 décalages vers la gauche permettant de calculer des multiplications modulaires par des puissances de 2. Dans ces conditions, une seule exécution du bloc Γ suffit pour calculer le doublement de points, tandis que l'addition de points nécessite le calcul de deux blocs successifs Γ. Ainsi, si le coût d'une addition modulaire est négligeable par rapport à celui d'une multiplication modulaire, le doublement de points coûte $9M$, tandis que l'addition de points coûte $18M$: le coût de cette dernière se voit donc augmenter de deux multiplications modulaires inutiles (cf. tableau 2.3).

Chevallier-Mames *et al.* [CMCJ04] améliorent ces résultats en considérant un bloc atomique Γ contenant une multiplication modulaire (qui peut être le cas échéant un carré modulaire), deux additions modulaires et une soustraction modulaire. Les adresses des registres pour chaque opération sont mémorisées sous forme de matrices. Contrairement à ce qui est proposé dans [GG02], où la multiplication modulaire par deux est effectuée à l'aide d'un décalage vers la gauche, elle est calculée dans [CMCJ04] à l'aide d'une addition modulaire ($2X \pmod p = X + X \pmod p$). Une autre différence notable par rapport à [GG02] est que le paramètre de courbe a peut prendre n'importe quelle valeur : la méthode détaillée dans [CMCJ04] est donc applicable à une classe plus importante de courbes. Finalement, le coût de l'implantation de l'addition (respectivement du doublement) de points sur les courbes de Weierstrass dans le système de coordonnées $\mathcal{J}$ est de 10 (16) blocs Γ. Ainsi, si le coût d'une addition modulaire est négligeable par rapport celui d'une multiplication modulaire, l'addition (respectivement le doublement) de points coûte $10M$ ($16M$) : l'implantation de cette contre-mesure n'a donc aucun impact sur les performances du cryptosystème, car seules des additions modulaires inutiles sont introduites, et non des multiplications modulaires comme c'est le cas dans [GG02].

Une dernière approche proposée par Mishra [Mis04] consiste à appliquer le principe d'atomicité de [CMCJ04] à deux processeurs fonctionnant en parallèle : cela permet de diminuer le temps de calcul global du doublement et de l'addition de points. Ils peuvent maintenant s'effectuer en six blocs Γ.

Pour résumer, l'atomicité permet qu'un attaquant observe une suite d'opérations iden-

tiques sans qu'il puisse les relier aux opérations de points effectuées, et donc aux bits de clé traités. De plus, afin de limiter l'impact de l'atomicité sur les performances du cryptosystème, ce concept peut être combiné avec un algorithme de multiplication scalaire utilisant le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3) : cette combinaison ne porte pas atteinte à la sécurité globale du cryptosystème face aux attaques SSCA.

Cependant, malgré tous ces avantages, l'utilisation de l'atomicité entraîne quelques inconvénients :

1. Tout d'abord, la méthode décrite dans [GG02] entraîne le calcul de multiplications modulaires inutiles, ce qui la rend vulnérable face aux attaques en fautes de type CSEA [YKLM01]. Cette attaque est également applicable sur l'atomicité décrite dans [CMCJ04] [Mis04], mais la précision requise est plus importante car les opérations inutiles exécutées sont des additions modulaires, opérations beaucoup plus rapides que les multiplications modulaires. Il faut noter que si la valeur de la clé est choisie aléatoirement (cf. section 4.1.2.1), alors cette attaque peut être contrée.

2. Ensuite, cet algorithme fournit à l'attaquant le nombre d'additions de points effectuées, et donc potentiellement le poids de Hamming du scalaire k. Or, lorsque ce poids de Hamming est extrêmement grand ou petit, un attaquant peut retrouver k à l'aide d'une recherche exhaustive [CKQ03].

   Le poids de Hamming de k peut être masqué en choisissant aléatoirement le scalaire k à chaque itératation (cf. section 4.1.2.1). Il peut également être utilisé un algorithme de multiplication scalaire utilisant le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3), car cela permet d'obtenir un poids de Hamming de $\mathrm{NAF}_w(k)$ constant quelque soit la valeur de k.

3. Enfin, il faut noter que la sécurité de cette contre-mesure est basée sur l'hypothèse que les opérations utiles et les opérations inutiles sont indistinguables, de même que les multiplications et les carrés modulaires. Or, cette dernière hypothèse n'est pas toujours vraie, comme en témoignent les attaques détaillées dans [SST04] [AFT+08]. Pour contre-carrer ces attaques, il peut être utilisé l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16), l'échelle de Montgomery (cf. Algorithme 11), ou encore des méthodes permettant de changer le scalaire k à chaque itération (cf. section 4.1.2.1).

4.1.1.3 Étude comparative

Le tableau 4.5 résume quelques propriétés de la plupart des contre-mesures détaillées dans les sections 4.1.2.1 et 4.1.1.2, et donc pour différents algorithmes de multiplication scalaire protégés ou non contre les attaques SSCA (première colonne) ainsi que pour différents types de formules unifiées (deuxième colonne). Ces propriétés sont le nombre de processeurs nécessaires (troisième colonne), le temps de calcul global de la multiplication scalaire (quatrième colonne), et la potentielle vulnérabilité du cryptosystème vis-à-vis des attaques par perturbation de type CSEA [YKLM01] (cinquième colonne).

L'étude du tableau 4.5 amène les conclusions suivantes :

Algo. $[k]P$	**Formules unifiées**	**Procs.**	**Coût de** $[k]P$	**CSEA ?**
Algo. 16 [Cor99]	–	1	$24\ell M$	Oui
Algo. 11 [BJ02]	–	1	$19\ell M + 1I$	Non
[FGKS02]	–	2	$10\ell M$	Non
[Mel07b]	–	1	$15,25\ell M$	Non
[Joy05]	[DBJ04] (cas général)	1	$25,33\ell M$	Non
[Joy05]	[DBJ04] ($a = -1$)	1	$22,66\ell M$	Non
[Joy05]	[JQ01]	1	$16\ell M$	Non
[HM02b]	–	1	$17,78\ell M$	Oui
[The06]	–	1	$16\ell M$	Non
[IT02b]	–	2	$5,85\ell M$	Non
Algo. 19 [Joy07]	–	1	$24\ell M$	Non
Algo. 20 [Joy07]	–	1	$32(\ell + 2)M$	Non
$\mathrm{NAF}_2(k)$	[DBJ04] (cas général)	1	$25,33\ell M$	Non
$\mathrm{NAF}_2(k)$	[DBJ04] ($a = -1$)	1	$22,66\ell M$	Non
$\mathrm{NAF}_2(k)$	[JQ01]	1	$16\ell M$	Non
$\mathrm{NAF}_2(k)$	[Sma01]	3	$5,33\ell M$	Non
$\mathrm{NAF}_2(k)$	[CMCJ04]	1	$13,33\ell M$	Oui
$\mathrm{NAF}_2(k)$	[Mis04]	2	$(8\ell - 2)M$	Oui

Tab. 4.5 – Étude comparative de l'ensemble des contre-mesures vis-à-vis des attaques simples par analyse de canaux cachés.

– si un concepteur de circuits sécurisés ne dispose que d'un seul processeur, alors il peut choisir d'implanter l'algorithme de multiplication scalaire $\mathrm{NAF}_2(k)$ avec les formules d'addition et de doublement de points indistinguables détaillées par Chevallier-Mames *et al.* [CMCJ04] car cette combinaison lui donne le plus petit temps de calcul global pour $[k]P$. Il devra cependant garder à l'esprit que le cryptosystème ainsi obtenu est vulnérable face aux attaques CSEA, même si cela requiert pour un attaquant d'avoir une très grande précision temporelle d'injection de fautes ;
– si il dispose de deux processeurs, alors il peut choisir d'implanter la méthode d'Izu *et al.* [IT02b], qui a le double avantage d'être la plus rapide et de n'être pas vulnérable face aux attaques CSEA ;
– si il dispose de trois processeurs, alors son choix pourra se porter sur la combinaison du $\mathrm{NAF}_2(k)$ avec les formules unifiées Hessiennes données par Smart [Sma01].

4.1.2 Protections contre les attaques différentielles par observation

Protéger un cryptosystème contre les attaques SSCA ne contribue pas à contrecarrer les attaques DSCA (cf. section 3.1.3). Par exemple, une attaque DSCA peut être montée sur l'échelle de Montgomery, protection possible vis-à-vis des attaques SSCA (cf. Algorithme 11) : si $k_{\ell-2} = 1$ (respectivement $k_{\ell-2} = 0$) alors la valeur $Q = [7]P$ ($Q = [5]P$) va être calculée. Ainsi, un attaquant peut monter une attaque DSCA en se concentrant sur le moment où est calculé potentiellement $Q_r = [7]P_r$. Il divisera les traces de consommation en deux groupes suivant la valeur prédictible d'un bit de $[7]P_r$. Si la courbe différentielle contient des pics, $Q = [7]P$ a bien été calculé. L'attaquant peut donc conclure que $k_{\ell-2} = 1$; sinon $k_{\ell-2} = 0$.

Ainsi, lorsqu'un attaquant essaie dans un premier temps de réaliser une attaque SSCA sur un cryptosystème et que celle-ci échoue, il peut dans un second temps se rabattre sur une attaque DSCA.

Pour monter une attaque DSCA, un attaquant doit être capable de modéliser le comportement théorique du canal caché mesuré, celui-ci dépendant de la valeur des variables intermédiaires traitées par le cryptosystème (cf. section 3.1.3). Or, un attaquant peut toutes les prédire, du moment qu'il connaît bien l'implantation matérielle du cryptosystème, le point de base P, la courbe E sur laquelle se déroule le calcul, et ce, pour différentes hypothèses de clé.

Autrement dit, dans le cas particulier de l'ECC, et afin de récupérer le secret k engagé dans le calcul de $Q = [k]P$ à l'aide d'une attaque DSCA, un attaquant doit s'assurer que le cryptosystème calcule $Q = [k]P$ pour un scalaire, un point de base et sur une courbe fixée et qu'il peut évaluer la valeur $g(Q_r)$, avec $Q_r = \left[\sum_{i=j}^{\ell-1} k_i 2^{i-j}\right] P_r$.

Il apparaît donc que l'implantation de trois contre-mesures peuvent, en changeant la valeur des variables intermédiaires, compliquer la tâche de l'attaquant : ajouter de l'aléa sur le scalaire k, sur le point de base P et sur l'équation de la courbe pendant le calcul de $Q = [k]P$.

Il faut noter que les attaques DSCA ne sont pas applicables sur certains protocoles cryptographiques impliquant le changement du scalaire k à chaque nouvelle exécution (comme par exemple l'ECDSA, cf. Algorithmes 14 et 15), ou imposant la valeur du point de base P. Cependant, ce type d'attaque peut être montée sur des multiplications scalaires impliquées dans des protocoles de chiffrement, comme par exemple l'ECIES [HMV04].

Il est donc détaillé dans cette section différentes méthodes permettant de rendre aléatoire les valeurs de k, de P et de l'équation de courbe E, tout en faisant attention à ne pas modifier le résultat final de la multiplication scalaire. La richesse mathématique des courbes elliptiques a permis la conceptualisation de nombreuses méthodes efficaces en termes de performance et de sécurité.

4.1.2.1 Agir sur la clé

Dans cette section, il est décrit différentes méthodes permettant de changer la valeur de la clé k à chaque multiplication scalaire $[k]P$, tout en ne modifiant pas le résultat final Q. Ainsi, il n'est plus possible de monter une attaque DSCA, car la valeur des différents points intermédiaires traités pendant la multiplication scalaire n'est pas connue de l'attaquant.

« Aveuglement » du scalaire. La contre-mesure consistant à « aveugler » le scalaire mise en évidence par Coron [Cor99] a été préalablement proposée pour le RSA, version multiplicative des courbes elliptiques, par Kocher [Koc96] et Kocher *et al.* [KJJ99].

Cette méthode utilise le fait que, dans la plupart des cas, la valeur de $n = \mathrm{ord}_E(P)$ est supposé être connue. Cette valeur peut permettre d'ajouter de l'aléa à la valeur de la clé k. Si on modifie $k^* = k + rn$, alors

$$Q = [k^*]P = [k + rn]P = [k]P + [rn]P = [k]P \tag{4.14}$$

car $[n]P = \infty$.

Ainsi, choisir aléatoirement r et renouveler ce choix à chaque nouvelle multiplication scalaire permet de rendre imprévisible la valeur de la clé, et donc sa représentation.

Coron estime qu'utiliser un nombre aléatoire r d'une longueur de 20 bits produit un niveau de sécurité suffisant.

Bien qu'efficace, cette méthode comporte quelques inconvénients.

1. Tout d'abord, l'utilisation de cette contre-mesure dégrade les performances d'un cryptosystème ECC, et ce par au moins deux facteurs différents. Le premier facteur de dégradation est dû à la nécessité d'implanter un générateur aléatoire de nombres, ce qui peut être coûteux (surtout si ce générateur doit produire des nombres **vraiment** aléatoires). Le second facteur de dégradation provient du fait que c'est k^* qui va être utilisé dans le calcul de la multiplication scalaire. Or, si on veut se limiter à un surcoût de 10%, autrement dit si r est de longueur 20 bits (si nous considérons un cryptosystème ECC utilisant une clé de 200 bits), alors k^* sera plus long que k de 20 bits : le temps nécessaire au calcul de la multiplication scalaire se voit donc augmenter. Plus précisément, en moyenne, cette contre-mesure nécessite le calcul de $20D$ (D pour doublement de points) et de $10A$ (A pour addition de points) supplémentaires.
2. Ensuite, l'utilisation directe de la relation 4.14 conduit à des problèmes de sécurité. Okeya *et al.* [OS00] ont par exemple montré que les bits de k^* donnaient suffisamment d'information pour pouvoir retrouver les bits de k. De même, il a été observé par Ciet [Cie03] qu'une partie des bits de k peuvent garder la même valeur après application de la relation 4.14. Ceci est dû au fait que les courbes elliptiques définies et standardisées sur $\mathbb{F}_p$ utilisent un nombre premier de Mersenne généralisé p de la forme $p = 2^n \pm 2^m \pm 1$, où m est pris relativement petit pour des raisons d'efficacité.

Ainsi, en utilisant le théorème d'Hasse (cf. relation 2.3), il peut être trouvé que la représentation binaire de $\#E(\mathbb{F}_p)$ peut s'écrire sous la forme d'un 1 suivi d'une longue série de zéros. Par exemple, la courbe standardisée « secp160k1 » [SEC00b] dispose d'un $\#E(\mathbb{F}_p)$ s'écrivant

$$\#E(\mathbb{F}_p) = (0\mathbf{1000000000000000000000}1B8FA16DFAB9ACA16B6B3)_{16}.$$

Ainsi, si il est calculé $k^* = k + rn$, alors k^* sera de la forme :

$$k^* = (r)_2 \| \boldsymbol{k_{\ell-1} \cdots k_{\ell-t}} \| \text{Autres bits de } k^*$$

Ainsi, les t bits de poids fort de k^* apparaissent en clair.

3. Enfin, Fouque *et al.* [FRVD08] ont montré que l'addition $k^* = k + rn$ peut être l'objet d'une attaque particulière : si celle-ci est effectuée à l'aide d'un additionneur séquentiel à propagation de retenue (cf. Figure 1.2), alors les retenues générées peuvent être détectées à l'aide d'une SSCA, et peuvent révéler les bits de k. Ainsi, si un concepteur de circuits sécurisés utilise la relation 4.14, alors il doit s'assurer que le calcul de celle-ci ne fuit aucune information.

   Cette attaque est intéressante car elle vise une contre-mesure, et non pas le cryptosystème qu'elle est censée protéger.

Pour toutes ces raisons, il est plutôt déconseillé d'utiliser cette contre-mesure.

Fission du scalaire. Le scalaire k peut également être l'objet d'une fission, c'est-à-dire qu'il peut être décomposé en deux parties (ou plus). Cette idée a tout d'abord été décrite de manière abstraite par Chari *et al.* [CJRR99], avant d'être concrètement proposée dans le cadre de l'ECC.

Plusieurs méthodes de fission sont à la disposition d'un concepteur de circuits sécurisés. La plus simple est la fission additive [CJ01] au cours de laquelle il est calculé

$$[k]P = [k - r]P + [r]P. \tag{4.15}$$

Un premier problème engendré par l'utilisation de la relation 4.15 est que pour cacher tous les bits de k, la taille de r doit être comparable à celle de k. Or, générer un nombre aléatoire de plusieurs dizaines de bits est coûteux, d'autant plus si r est un nombre **vraiment** aléatoire.

De plus, le temps de calcul de la relation 4.15 est approximativement le double du calcul de $[k]P$ sans fission, puisque $[k - r]P$ et $[r]P$ doivent être calculés.

Ainsi, si un concepteur de circuits sécurisés veut implanter une fission du scalaire plus efficace, il lui faudra choisir une méthode un peu plus complexe à mettre en œuvre. C'est dans ce sens qu'il peut utiliser une généralisation de la fission additive proposée par Chevallier-Mames [CM04]. Cette méthode était initialement destinée à la protection du RSA, mais elle peut également s'appliquer au cas de l'ECC. L'auteur détaille trois algorithmes différents. Il ne sera détaillé dans ce qui suit que le concept principal qui conduit à l'écriture de ces trois algorithmes.

Soit $k^{(i_j)} = k_{\ell-1 \to i_j} = (k_{\ell-1} \cdots k_{i_j})_2$, $1 \leq j \leq t$ et $i_1 > i_2 > \cdots > i_t$. En utilisant cette notation, il peut être écrit :

$$\begin{aligned}
[k]P &= [k^{(0)}]P, \\
&= [k^{(0)} - k^{(i_1)}]P + [k^{(i_1)}]P, \\
&= [k^{(0)} - k^{(i_1)} - k^{(i_2)}]P + [k^{(i_1)}]P + [k^{(i_2)}]P, \\
&= \cdots, \\
&= [k^{(0)} - k^{(i_1)} - k^{(i_2)} - \cdots - k^{(i_t)}]P + [k^{(i_1)}]P + [k^{(i_2)}]P + \cdots + [k^{(i_t)}]P.
\end{aligned}$$

Il est intéressant à noter ici que si les différentes valeurs de i_j sont choisies aléatoirement, il peut être décrit un algorithme de multiplication scalaire probabiliste (cf. [CM04], Fig. 3).

Cette méthode, même si elle apporte un haut niveau de sécurité vis-à-vis des attaques DSCA, conduit au calcul (d'au maximum) ℓ additions de points supplémentaires par rapport à l'algorithme du « doublement-et-addition » (cf. Algorithme 9). L'auteur donne ce résultat en considérant la combinaison de son concept avec le principe d'atomicité (cf. Algorithme 18). Rappelons que pour utiliser ce dernier, il faut s'assurer notamment que l'addition et le doublement de points sont indistinguables (cf. section 4.1.1.2).

Un concepteur de circuits sécurisés peut également choisir d'implanter la fission multiplicative [TB02] dans laquelle il est calculé

$$[k]P = [kr^{-1}]([r]P)$$

à l'aide d'un nombre aléatoire r ayant un inverse modulo $\mathrm{ord}_E(P)$.

Il peut enfin choisir la fission Euclidienne [Joy05], qui est encore plus complexe à implanter que la fission multiplicative, mais qui est une solution possible pour diminuer la taille de r, et ainsi limiter l'impact de ce type de contre-mesure :

$$[k]P = [k \bmod r]P + [\lfloor k/r \rfloor]([r]P).$$

Rendre aléatoire l'ordre de traitement des bits de clé. Une autre contre-mesure vis-à-vis des attaques DSCA agissant sur la clé consiste à rendre aléatoire l'ordre de traitement des bits de celle-ci.

Pour ce faire, le premier bit de k traité par l'algorithme de multiplication scalaire peut être rendu aléatoire. Cette méthode a été proposée par Messerges *et al.* [MDS99]. Dans un premier temps, l'algorithme de multiplication scalaire choisi est exécuté de la droite vers la gauche, d'un bit de clé k_j choisi aléatoirement vers le bit $k_{\ell-1}$. L'algorithme de multiplication scalaire utilise dans un second temps le résultat obtenu pour terminer son calcul de la gauche vers la droite, c'est à-dire du bit de clé k_{j-1} vers le bit k_0. Il faut noter que cette méthode n'amène pas un niveau de sécurité satisfaisant : elle ne doit donc pas être utilisée.

Les bits de clé peuvent également être traités dans un ordre aléatoire. Cette méthode

a été proposée par Trichina *et al.* [TB02]. Les auteurs utilisent le fait que, si les multiples de P sont précalculés ($[2^i]P$, avec $0 \leq i < \ell$), alors l'ordre de traitement des bits de k importe peu, du moment que le bit de clé k_i peut être associé à tout moment avec $[2^i]P$: les bits de clé peuvent être ainsi permutés avant chaque exécution de l'algorithme de multiplication scalaire. Il faut noter que cette méthode procure un haut niveau de sécurité, mais elle implique l'utilisation de ℓ registres (où ℓ est également le nombre de bits de la clé) pour mémoriser les différentes valeurs de $[2^i]P$. Surtout, cette méthode est vraiment efficace lorsque le point P ne change pas à chaque exécution du protocole sous-jacent.

Faire varier la représentation du scalaire à chaque multiplication scalaire. Dans cette section, il est détaillé différentes contre-mesures consistant à faire varier la représentation du scalaire à chaque multiplication scalaire, de façon à ce que la séquence de doublements et d'additions de points soit différente à chaque exécution de l'algorithme. La but d'une telle méthode est de rendre aléatoire la valeur des différents points intermédiaires traités par l'algorithme de multiplication scalaire : ce faisant, l'attaquant ne peut pas prédire la représentation interne des données, et ne peut donc pas monter une attaque DSCA.

Il pourra être constaté dans cette section que cet objectif n'est pas toujours atteint.

Utiliser une représentation redondante du scalaire. Utiliser une représentation redondante du scalaire est une idée avancée par Oswald *et al.* [OA01] et par Ha *et al.* [HM02a]. Si k utilise une représentation redondante (cf. section 1.1.2 et 1.1.3), alors la valeur de ses chiffres peut être changée à chaque nouvelle exécution de la multiplication scalaire, ce qui permet de modifier la valeur des points intermédiaires traités par l'algorithme. Les auteurs utilisent l'ensemble de chiffres D_1 (cf. section 1.1.2).

Une analyse approfondie de la sécurité de ces deux méthodes a été effectuée et conduit aux conclusions suivantes.

1. Si le doublement et l'addition de points sont distinguables *via* une attaque SSCA, alors la cryptanalyse de la méthode de Oswald *et al.* est possible [OS02] [KW03] [OS03] [EH03], de même que celle de Ha *et al.* [OH03].
2. Même lorsqu'une contre-mesure contre les attaques SSCA est utilisée, les méthodes proposées dans [OA01] et dans [HM02a] sont vulnérables face aux attaques dites « par collision » [FMPV04]. Dans ce type d'attaques, les relevés de canal caché de deux exécutions différentes d'un algorithme sont comparés au niveau d'un doublement de point : ainsi, sur le premier (respectivement sur le second) relevé, il est calculé $[2]A$ ($[2]B$). Cette comparaison ne permet pas à l'attaquant de remonter directement à la valeur de A et de B, mais elle lui permet toutefois de vérifier si $A = B$ ou pas (si c'est le cas, on dit qu'il y a « collision »), et cela est suffisant pour cette attaque qui se déroule comme suit.

   Pour chaque bit de clé k_i utilisant une représentation redondante, en détectant d'éventuelles collisions sur deux points intermédiaires successifs, un attaquant peut

diviser les relevés du canal caché mesuré en trois groupes, chacun correspondant à l'un des trois chiffres de l'ensemble D_1. Ce faisant, l'attaquant n'a pas besoin de savoir à quel chiffre correspond tel groupe : si le nombre de relevés dans un groupe est supérieur à la moitié du nombre total de relevés, alors $k_i = 0$, sinon $k_i = k_{i+1}$. En revanche, si les relevés sont également répartis dans les trois groupes, alors $k_i \neq k_{i+1}$.

Il faut noter que cette attaque n'est pas applicable si la fission ou l'aveuglement du scalaire est effectué (cf. sections précédentes).

Pour conclure sur l'efficacité réelle de ces contre-mesures, il faut considérer qu'elles impliquent des contraintes d'utilisation si importantes (doublement et addition indistinguables, ajout d'aléa supplémentaire sur la clé k [Wal04a] ou sur le point P [FMPV04]) qu'il est plutôt déconseillé de les utiliser.

Recodage du scalaire par fenêtres de taille aléatoire. Dans [LS01], Liardet *et al.* proposent non seulement des formules unifiées d'addition de points (cf. section 4.1.1.2), mais également une contre-mesure vis-à-vis des attaques DSCA. Leur idée consiste à recoder k à l'aide d'une méthode par fenêtres (cf. section 2.2.3.2), ces dernières étant de taille aléatoire inférieure ou égale à R. La clé k est recodée de la droite vers la gauche sous la forme de paires (b_i, e_i), où b_i est le chiffre et e_i est la taille de la fenêtre. La clé recodée est ensuite utilisée dans un algorithme de multiplication scalaire par fenêtres qui utilise 2^{R-2} points précalculés.

Il faut noter que la sécurité de la méthode proposée par Liardet *et al.* dépend de la manière dont sont implantés l'addition et le doublement de points. Si ces deux opérations sont distinguables grâce à une attaque SSCA, Walter [Wal02a] montre que l'utilisation répétée du même scalaire peut amener suffisamment d'information pour retrouver la clé : un attaquant peut utiliser le fait que le nombre de doublements de points consécutifs dépend de la taille de la fenêtre, et ainsi, il peut tirer profit de l'apparition irrégulière de la séquence $\{DA\}$. Avec 18 relevés du canal caché mesuré, et pour retrouver une clé de 192 bits, le nombre d'opérations à effectuer a une complexité de l'ordre de $O(2^{10})$, ce qui est possible à exécuter avec les moyens de calcul actuels.

Un autre algorithme similaire a été proposé par Ahn *et al.* [AHLM03] dans lequel $R = 3$, et où l'ensemble des chiffres utilisé est $\{0, 1, \cdots, 7\}$. Pour faire en sorte que leur méthode soit intrinsèquement protégée contre les attaques SSCA et l'attaque de Walter [Wal02a], trois doublements de points sont calculés à chaque itération de l'algorithme proposé (dont certains sont calculés inutilement), et ce quelque soit la longueur de la fenêtre traitée.

L'approche de Ahn *et al.* comporte deux inconvénients.

1. Tout d'abord, le calcul de doublements inutiles rend cette méthode vulnérable face aux attaques par perturbation de type CSEA [YKLM01] : si ces opérations sont découvertes, alors l'attaque décrite par Walter redevient réalisable [Wal02a].
2. Enfin, l'implantation de cette approche impacte les performances du cryptosystème :

pour $\ell = 160$, le temps de calcul de cette méthode est $(1, 29\ell + 3)D + (0, 43\ell + 3)A$.

Pour toutes ces raisons, il est plutôt déconseillé d'utiliser la méthode proposée par Ahn *et al.*. En revanche, si l'addition et le doublement de points sont rendus indistinguables, alors la méthode de Liardet *et al.* peut être utilisée.

Recodage du scalaire par fenêtres recouvertes. Une autre méthode de recodage du scalaire par fenêtres est détaillée par Itoh *et al.* [IYTT02]. Cette méthode consiste à considérer que deux fenêtres de taille w peuvent se recouvrir sur h bits, avec $h \geq w/2$ (h peut varier pendant le recodage de la clé, mais les auteurs conseillent de le maintenir fixe pour résister aux attaques SSCA). Le recodage s'effectue de la gauche vers la droite à l'aide de l'ensemble de chiffres $\{0, 1, \cdots, 2^{w-1}\}$: la valeur de la fenêtre courante est remplacée par une valeur aléatoire, et la différence entre ces deux valeurs (qui doit être inclue dans l'intervalle $[0,\ 2^{h-1}]$) est additionnée à la prochaine fenêtre (qui recouvre la fenêtre courante par h bits).

Ce recodage conduit à la mémorisation de 2^{w-1} points, et le coût des précalculs est de $(w-1)D + (2^w - w - 1)A$. Le temps d'exécution de cette méthode est égal à $(\ell - 1)D + [\ell/(w-h)]A$, tandis que son niveau de sécurité augmente pour des valeurs croissantes de la quantité $h/(w-h)$. Or, un concepteur de circuits sécurisés veut implanter une solution ayant un temps d'exécution minimal (il doit donc fixer une grande quantité $(w-h)$) et un AR minimal (il doit donc fixer une grande quantité $h/(w-h)$) : il cherchera donc à trouver le meilleur compromis performance/sécurité en choisissant la valeur des paramètres h et w qui lui convient.

L'algorithme MIST. Cet algorithme a été initialement proposé par Walter [Wal02b] pour protéger le RSA, mais il peut également être utilisé dans le cadre de l'ECC. Il consiste essentiellement à recoder la clé de la droite vers la gauche en utilisant un système de numération utilisant une base différente (choisie aléatoirement) à chaque étape de recodage. Les bases utilisées sont 2, 3 et 5. Le résultat de ce recodage est ensuite transformé progressivement en petites chaînes d'additions, ce qui permet de protéger l'ensemble de cet algorithme contre les attaques SSCA (cf. Algorithme 17).

L'analyse de la sécurité de cet algorithme a aussi été présentée par Walter [Wol02] : si le doublement et l'addition de points sont distinguables, alors il a montré qu'il est possible de réduire le nombre de clés possibles à environ $2^{3\ell/5}$, et même à $2^{\ell/3}$ sous certaines conditions. Sim *et al.* [SPL03] améliorent ce résulat puisqu'ils ont mis en évidence une attaque permettant de réduire ce nombre à $2^{0,0756\ell}$. Il faut également noter que pour des grandes tailles de clé, l'exécution de l'algorithme MIST peut donner à elle seule suffisamment d'information pour retrouver cette dernière [Wal03].

Ainsi, pour toutes ces raisons, il est plutôt déconseillé d'utiliser cette contre-mesure.

4.1.2.2 Agir sur le point de base ou ses multiples

Dans cette section, il est décrit différentes méthodes permettant de changer la valeur du point de base P ou de ses multiples à chaque multiplication scalaire $[k]P$, tout en

ne modifiant pas le résultat final Q. Ainsi, il n'est plus possible de monter une attaque DSCA, car la valeur des différents points intermédiaires traités pendant la multiplication scalaire n'est pas connue de l'attaquant.

« Aveuglement » du point de base. Cette technique a été proposée par Kocher *et al.* initialement dans le cadre du RSA [Koc96] [KJJ99], mais son principe peut également être utilisé dans le cadre des courbes elliptiques [Cor99]. Elle utilise un point $S = [k]R$ où R est un point secret :

$$Q = [k](P + R) - S = [k]P + [k]R - [k]R = [k]P. \tag{4.16}$$

Ainsi, il est calculé successivement $P^* = P + R$, $S = [k]R$ et $Q = [k]P^* - S$. Vu que R est secret, la représentation du point $S = [k]R$ est inconnue, tout comme celle de $P^* = P + R$. Le couple de points (R, S) contenu dans la mémoire de travail du cryptosystème peut être remplacé avant chaque nouvelle multiplication scalaire par ($[r]R$,$[r]S$), où r est un petit nombre aléatoire. Plus précisément, Coron [Cor99] suggère de calculer à chaque nouvelle exécution $R \leftarrow [(-1)^b 2]R$ et $S \leftarrow [(-1)^b 2]S$, où b est un bit de valeur aléatoire. En utilisant cette méthode de renouvellement, l'implantation de la relation 4.16 coûtera deux doublements (pour le calcul des nouvelles valeurs de R et S) et deux additions/soustractions de points (pour les calculs de $P^* = P+R$ et $Q = [k]P^*-S$).

Il faut noter que le rafraîchissement de la variable R proposé par Coron est vulnérable face à l'attaque dite « du doublement » (cf. section 3.1.2.3) [FV03]. En effet, lors d'une première application de la relation 4.16, il est calculé $[k]P^* = [k](P + R)$. Il suffit donc pour un attaquant de choisir $[2]P$ comme point d'entrée pour une seconde application de cette même relation : il sera ainsi effectué $[k]([2]P + [2]R) = [2][k](P + R) = [2]([k]P^*)$ (avec une probabilité de 1/2 du fait que b est un bit choisi aléatoirement). Finalement, l'attaquant obtenant les relevés de canal caché de $[k]P^*$ et de $[2]([k]P^*)$ a toutes les cartes en main pour réaliser l'attaque du doublement.

Pour empêcher cette attaque particulière, nous proposons de calculer plutôt $R \leftarrow [(-1)^b \mathbf{3}]R$ et $S \leftarrow [(-1)^b \mathbf{3}]S$: ainsi, les conditions pour réaliser l'attaque dite « du doublement » ne sont plus réunies. Notons que le coût global de l'implantation de cette contre-mesure devient donc égal à deux doublements et quatre additions/soustractions de points.

Isomorphisme de corps. Pour rendre imprévisible la représentation interne des nombres traités par un cryptosystème, un isomorphisme de corps peut être utilisé. Dans ce cas, connaissant un point P d'une courbe elliptique E définie sur un corps fini $\mathbb{F}_p$, un isomorphisme de corps aléatoire $\psi : \mathbb{F}_p \rightarrow (\mathbb{F}_p)^*$ est appliqué à P et E pour obtenir le point $P^* = \psi(P)$ on $E^* = \psi(E)$. Ainsi, la multiplication de point $Q = [k]P$ est évaluée grâce à $\psi^{-1}([k]P)$ [JT01].

Cette proposition de protection comporte deux inconvénients majeurs. Tout d'abord, les calculs dans des isomorphismes de corps peuvent être suivant les cas beaucoup plus

lents que dans des corps connus pour leur efficacité, comme par exemple ceux définis par un nombre premier de Mersenne (cf. section 2.3.2.2). De plus, il faut noter que cette contre-mesure peut être le sujet de l'attaque de Goubin [Gou03] et d'Akashita *et al.* [AT03] (cf. section 4.1.2.3).

Isomorphisme de courbes. Les isomorphismes de courbe sont des isogénies d'ordre 1 (cf. section 2.2.4.1). Il est possible de choisir aléatoirement un isomorphisme de courbes dans le but d'effectuer les calculs cryptographiques sur une courbe d'équation non-prédictible pour un attaquant : il n'aura donc aucune information sur la représentation interne des différentes variables intermédiaires traitées par le cryptosystème [JT01].

Deux courbes elliptiques d'équation $E : y^2 = x^3 + ax + b$ et $E' : y^2 = x^3 + a'x + b'$ sont isomorphiques si et seulement si il existe un nombre $u \neq 0$ tel que $u^4a' = a$ et $u^6b' = b$. L'isomorphisme de courbes est ainsi donné par :

$$\left\{ \begin{array}{rcccccccc} \varphi & : & E & \rightarrow & E', & P & = & (x,y) & \mapsto & P' = & (u^{-2}x, u^{-3}y) \\ \varphi^{-1} & : & E' & \rightarrow & E, & P' & = & (x',y') & \mapsto & P = & (u^2x', u^3y') \end{array} \right.$$

Ce faisant, il est utilisé la relation :

$$Q = \varphi^{-1}([k]\varphi(P)) = \varphi^{-1}([k]P') = \varphi^{-1}(Q') = Q.$$

Plus concrètement, il est calculé successivement $\varphi(P)$, $[k]\varphi(P)$ et $\varphi^{-1}([k]\varphi(P))$. Le coût total de l'utilisation de cette contre-mesure est égal à $1I + 10M$.

Il faut noter que cette contre-mesure peut être le sujet de l'attaque de Goubin [Gou03] et d'Akashita *et al.* [AT03] (cf. section 4.1.2.3).

Coordonnées projectives. L'idée développée par Coron est d'ajouter de l'aléa sur le point de base (ou sur ses multiples) exprimé(s) en coordonnées projectives [Cor99] [JT01]. Il faut rappeler que l'utilisation des coordonnées projectives permet d'éviter le calcul d'inversions modulaires coûteuses pendant le calcul de la multiplication scalaire (cf. section 2.2.2).

Lorsqu'un point est représenté en coordonnées projectives, il peut supporter plusieurs écritures différentes. Par exemple, dans le système de coordonnées projectives $\mathcal{P}$, les triplets $(\theta X : \theta Y : \theta Z)$ avec $\theta \neq 0$ représentent le même point. De même, dans les systèmes de coordonnées projectives $\mathcal{J}$, $\mathcal{J}^c$ et $\mathcal{J}^m$, les triplets $(\theta^2 X : \theta^3 Y : \theta Z)$ avec $\theta \neq 0$ représentent le même point. Il est donc possible de changer la représentation d'un point durant la multiplication scalaire, en modifiant la valeur de θ : cela coûte $3M$ (respectivement $5M$) si le système de coordonnées projectives $\mathcal{P}$ ($\mathcal{J}$, $\mathcal{J}^c$ ou $\mathcal{J}^m$) est utilisé.

Cette étape de modification de la représentation des points intermédiaires peut se dérouler au début (et à n'importe quel autre moment, choisi aléatoirement ou non) de la multiplication scalaire. Ainsi, le point de base P peut être représenté en coordonnées $\mathcal{P}$, et modifié avant chaque début d'une nouvelle exécution de l'algorithme de multiplication

scalaire. Ciet *et al.* [CJ03] proposent quant à eux de représenter le point P en coordonnées affines $\mathcal{A}$ afin de tirer profit de l'utilisation des coordonnées mixtes dans le cadre de l'addition de points (cf. section 2.2.2.2), et de modifier les coordonnées du premier point intermédiaire calculé par l'algorithme de multiplication scalaire implanté : par exemple, si c'est l'algorithme du « doublement-et-addition » (cf. Algorithme 9) qui est utilisé, ce sera la représentation du point $[2]P$ qui pourra être modifié. La même observation a été faite par Izu *et al.* [IMT02] pour l'échelle de Montgomery (cf. Algorithme 2.2.4.2).

Il faut noter que les inversions calculées à la fin de la multiplication scalaire permettant de retrouver les coordonnées affines du point résultat (par exemple, si le point $Q = [k]P = (X : Y : Z)$ est représenté en coordonnées projectives $\mathcal{P}$, alors il peut être calculé $x = X/Z$ et $y = Y/Z$) doivent être effectuées en environnement sécurisé : si le triplet $Q = (X : Y : Z)$ est accessible de l'extérieur, alors un attaquant peut retrouver la séquence des opérations de doublements et d'additions de points, et donc remonter à la clé. Cette attaque particulière détaillée par Naccache *et al.* [NSS04] peut être contre-carrée si le résultat est donné en coordonnées affines ou si la représentation de ce dernier est encore modifié en changeant la valeur de θ avant d'être envoyé à l'extérieur.

Il faut également noter que cette contre-mesure peut être le sujet de l'attaque de Goubin [Gou03] et d'Akashita *et al.* [AT03] (cf. section 4.1.2.3).

4.1.2.3 Attaque de Goubin et d'Akishita *et al.*

Toute la section 4.1.2 a été consacrée à la description de protections possibles contre les attaques différentielles par observation. Notamment, il a été question dans la section 4.1.2.2 de contre-mesures consistant à agir sur le point de base ou ses multiples. Or, Goubin [Gou03] a montré que l'implantation de certaines de ces contre-mesures peut amener de la vulnérabilité au cryptosystème ECC vis-à-vis d'une attaque particulière. En particulier, Goubin utilise le fait que certaines contre-mesures présentées dans la section 4.1.2.2 ne modifient pas la valeur d'un point $P_0 \in E$ ayant l'une de ses coordonnées affine ou projective nulle (par exemple Y) : c'est le cas de l'utilisation des isomorphismes de corps, de courbes ainsi que des coordonnées projectives. « L'attaque raffinée de Goubin » (*Goubin's Refined Attack*, GRA), se déroule comme suit : si un attaquant injecte comme point d'entrée $P = [c^{-1}(\mathrm{mod}\ \#E)]P_0 = [c^{-1}(\mathrm{mod}\ \#E)](X : 0 : Z)$ où c est un entier, alors une attaque DSCA pourra détecter le fait que le point $[c]P$ est calculé pendant la multiplication scalaire. L'attaquant peut ainsi retrouver la clé intégralement en adaptant cette attaque récursivement pour différentes valeurs de c.

Plusieurs courbes standardisées sur $\mathbb{F}_p$ contiennent des points de la forme $(X : 0 : Z)$ (ou (x, y)), la courbe estampillée « P-224 » étant une exception [FIP00] [SEC00b] : cette attaque particulière constitue ainsi une menace potentielle pour les cryptosystèmes ECC.

Dans [AT03], Akishita *et al.* décrivent une extension de la GRA. Cette attaque utilise les points provoquant le fait que des variables intermédiaires traitées lors de l'addition ou du doublement soient nulles. Dans le cadre du calcul du doublement de points, les points trouvés par Goubin dans [Gou03] peuvent être utilisés pour cette attaque. À ces points

s'en ajoutent d'autres dépendant de l'algorithme de multiplication scalaire et du système de coordonnées implantés. Par exemple, si l'algorithme du « doublement-et-addition » (cf. Algorithme 9) et les coordonnées Jacobiennes $\mathcal{J}$, $\mathcal{J}^c$ et $\mathcal{J}^m$ sont utilisées, les points satisfaisant les conditions $3x^2+a=0$ ou $5x^4+2ax^2-4bx+a^2=0$ conviennent. De même, si l'échelle de Montgomery (cf. Algorithme 11) est implantée (dans sa version omettant de calculer la coordonnée y), alors les points satisfaisant les conditions $x^2-a=0$ ou $x^2+a=0$ peuvent être utilisés dans le cadre de l'attaque d'Akishita *et al.*. Dans le cadre du calcul de l'addition de points, les points aboutissant à des variables nulles à l'itération $i=j$ dépendent de $k_{\ell-1\to j}$, et ne peuvent être trouvés que pour des petites valeurs de ce dernier.

Ces deux attaques nécessitent de sélectionner le point de base P utilisé par l'algorithme de multiplication scalaire, ce qui n'est possible que dans le cadre de l'utilisation de certains protocoles cryptographiques ECC, comme par exemple l'ECIES. Cependant, si l'injection d'un point d'entrée choisi est possible pour un attaquant, alors le concepteur de circuits sécurisés doit implanter des contre-mesures supplémentaires, comme par exemple l'ajout d'un aléa sur la clé (cf. section 4.1.2.1) ou sur le point de base (cf. section 4.1.2.2).

Smart [Sma03] a réalisé une étude détaillée de la GRA. Il en résulte deux conclusions différentes, suivant que l'ordre du point P_0 est petit ou grand. Si l'ordre du point P_0 est petit, alors cette attaque peut être contre-carrée facilement, en modifiant très légèrement les protocoles cryptographiques mis en jeu. En revanche, si il est grand, Smart [Sma03] propose d'utiliser des isogénies (cf. section 2.2.4.1). Cette contre-mesure est moins coûteuse que « l'aveuglement » du scalaire (cf. section 4.1.2.1), mais ce dernier est plus simple à implanter que la méthode proposée par Smart. Il faut également noter que, dans la plupart des cas, « l'aveuglement » du point de base est moins coûteuse que la contre-mesure basée sur les isogénies [Sma03].

Enfin, un concepteur de circuits sécurisés peut utiliser des contre-mesures spécifiques à la GRA et à l'attaque décrite par Akishita *et al.*. Il peut par exemple choisir d'implanter des courbes de Weierstrass pour lesquelles x^3+ax+b est irréductible sur $\mathbb{F}_p$ (c'est-à-dire que ce polynôme ne peut pas être réécrit à l'aide d'un produit de polynômes de degrés inférieurs) et telles que b n'est pas un carré sur $\mathbb{F}_p$: ce faisant, aucun point possédant une coordonnée nulle n'existe.

Dans ce qui suit, il est détaillé différents schémas de protection (attaqués ou non) mis en évidence contre les attaques de Goubin et d'Akashita *et al.*. La plupart de ces schémas intégrent également des contre-mesures contre les attaques SSCA générales et particulières (notamment l'attaque dite « du doublement » de Fouque *et al.* [FV03] et de Yen *et al.* [YLMH05]).

Schéma de protection contre les attaques de Goubin et d'Akashita *et al.* attaqué. Une contre-mesure proposée par Mamiya *et al.* [MMM04a] consiste à choisir aléatoirement un point de base initial $R_0 \neq \infty$ afin de calculer $R_1 = R_0 + [k]P$, puis $Q = R_1 - R_0$. Afin d'obtenir un algorithme (traitant les bits de k de la gauche vers la droite) résistant face aux attaques SSCA, il est considéré une représentation rendondante

de R dans laquelle l'ensemble des chiffres choisis est D_1 (cf. section 1.1.2) :

$$R + [k]P = [(1\underbrace{\overline{1}1\cdots\overline{1}}_{\ell \text{ fois}})_2]R + [(k_{\ell-1}\cdots k_1 k_0)_2]P.$$

Malheureusement, cet algorithme n'a pas été conçu initialement pour résister face à l'attaque du doublement particulière de Yen *et al.* [YLMH05] : si un point P_0 d'ordre 2 est injecté dans l'algorithme, alors le point intermediaire calculé à l'itération i est $-R$ si $k_i = 0$ ou $P_0 - R$ si $k_i = 1$. Cette attaque peut ainsi permettre de récupérer $\lceil \ell/w \rceil$ bits du scalaire k.

Schémas de protection contre les attaques de Goubin et d'Akashita *et al.* non-encore attaqués. Un premier schéma de protection encore non-attaqué à l'heure actuelle a été proposé par Itoh *et al.* [IIT04]. Celui-ci est basé sur le calcul de l'algorithme du « doublement-et-toujours-addition » (cf. Algorithme 16) exécuté de la droite vers la gauche, dans lequel la variable R_0 est initialement égale à la valeur d'un point R choisi aléatoirement ($R \neq \infty$). À la fin de l'exécution de l'algorithme, R est soustrait au résultat.

Les auteurs précisent que la sélection d'un nouveau point R avant chaque exécution de l'algorithme peut diminuer drastiquement les performances du cryptosystème : c'est pour cela qu'ils proposent comme alternative qu'il ait une valeur fixe, et que ce soit ses coordonnées projectives qui changent à chaque exécution (cf. section 4.1.2.2).

Une autre contre-mesure proposée par Itoh *et al.* [IIT04] est basée sur l'ajout d'une quantité (*offset*) sur les coordonnées des points traités durant la multiplication scalaire, ajout dont il faut tenir compte pour modifier en conséquence les formules d'addition et de doublement de points. Une nouvelle quantité est choisie aléatoirement à chaque exécution. Cette méthode, combinée avec le changement des coordonnées projectives (cf. section 4.1.2.2), permet d'éviter le calcul de points ayant une coordonnée nulle.

Cette contre-mesure induit des pénalités en terme de performances. L'implantation des formules d'addition (respectivement du doublement) de points servant à exécuter l'algorithme du « doublement-et-toujours addition » (cf. Algorithme 16) dans lequel le scalaire est traité de la gauche vers la droite coûte $18M$ ($15M$) pour une valeur de a quelconque. L'implantation du même algorithme dans lequel le scalaire est traité de la droite vers la gauche coûte $17M$ ($8M$) pour une valeur de a quelconque. Enfin, l'application de l'échelle de Montgomery (cf. Algorithme 11) coûte $25M$ par itération pour une valeur de a quelconque ($1M$ peut être économisé dans le cas où $a = -3$).

Enfin, un dernier schéma qui contre-carre la GRA [Gou03], l'attaque d'Akashita *et al.* [AT03], et même celle de Yen *et al.* [YLMH05], est proposé par Kim *et al.* [KHM+05]. Ils proposent de calculer un aveuglement de point particulier :

$$[k]P = [k](P + R) + [\#E - k]R,$$

dans lequel R est un point choisi aléatoirement.

À l'heure actuelle, ce schéma n'a pas encore été attaqué. Le coût moyen de son implantation est de $M(17, 5\ell + 20)$ lorsque les coordonnées projectives $\mathcal{J}$ sont utilisées.

4.1.2.4 Étude comparative

Cette section est consacrée à l'étude comparative des différentes contre-mesures vis-à-vis des attaques DSCA présentées dans ce mémoire. Dans un esprit de synthèse, nous classons ces différentes contre-mesures en trois catégories, selon qu'elles apportent un niveau de sécurité faible (c'est-à-dire qu'elles ont été elles-même attaquées), ou suffisant (c'est-à-dire qu'elles n'ont pas été elles-même attaquées). Une autre catégorie regroupe les parades dont l'utilisation peut être sujette à caution, soit à cause de leurs performances intrinsèques, soit à cause du manque de recul (et/ou d'études détaillées) sur la sécurité de celles-ci.

Tout d'abord, nous pouvons classer dans la catégorie des contre-mesures faibles d'un point de vue sécuritaire « l'aveuglement du scalaire » [Cor99], le choix aléatoire du premier bit de clé traité [MDS99], l'utilisation d'une représentation redondante du scalaire [OA01] [HM02a], le recodage du scalaire par fenêtres de taille aléatoire d'Ahn *et al.* [AHLM03], l'utilisation de l'algorithme MIST [Wal02b] [Wol02], l'utilisation d'un isomorphisme de corps ou de courbes [JT01], l'utilisation de la méthode de Mamiya *et al.* [MMM04a] : nous déconseillons donc leur utilisation.

Ensuite, nous pouvons classer dans la catégorie des contre-mesures sécurisées la fission du scalaire proposée par Chevallier-Mames [CM04], la méthode de Liardet *et al.* [LS01] (si le doublement et l'addition de points sont indistinguables), le recodage du scalaire par fenêtres recouvertes [IYTT02], « l'aveuglement » du point de base [Cor99] (avec un rafraîchissement des points R et S évitant l'attaque dite « du doublement » [FV03], cf. section 4.1.2.2).

Enfin, nous pouvons classer dans la catégorie des contre-mesures dont l'utilisation peut être sujette à caution d'une part les fissions du scalaire additive [Cor99] (car le temps de calcul de la multiplication scalaire est doublé), multiplicative [TB02] et Euclidienne [Joy05] (car il doit être calculé à chaque multiplication scalaire une inversion modulaire, opération coûteuse à effectuer dans $\mathbb{F}_p$ – cf. section 1.2.1.3) et le traitement des bits de clé dans un ordre aléatoire [TB02] (car cette méthode requiert ℓ registres, où ℓ est également le nombre de bits de la clé) pour des raisons de performances, et d'autre part les méthodes résistantes aux attaques du type de Goubin [IIT04] [KHM+05], ces dernières étant encore trop récentes et pas suffisamment étudiées pour se fier totalement à leur sécurité supposée.

Pour conclure, un concepteur de circuits sécurisés peut compliquer au maximum le travail d'un attaquant en agissant sur la clé **et** sur le point de base [Joy05]. Cependant, il devra par ailleurs étudier cette combinaison de contre-mesures pour vérifier qu'elle n'induit pas de vulnérabilités au cryptosystème. Par exemple, il peut être tentant de choisir l'implantation de la fission du scalaire proposée par Chevallier-Mames [CM04] combinée avec la technique « d'aveuglement » du point de base [Cor99], car cela coûtera seulement $(\ell + 2)D + (3\ell/2 + 4)A$, mais le cryptosystème n'est protégé qu'*a priori* contre toutes les attaques SSCA et DSCA connues à ce jour : le concepteur de circuits sécurisés

devra donc mener des études poussées sur le cryptosystème obtenu afin d'étudier les éventuelles faiblesses sécuritaires de ce dernier.

4.2 Protections contre les attaques par perturbation

Il a été détaillé dans la section 4.1 les différentes protections possibles contre les attaques (simples ou différentielles) par observation. Un concepteur de circuits sécurisés doit également protéger son cryptosystème face aux attaques par perturbation décrites dans la section 3.2.

Il n'est pas facile de choisir une protection particulière contre les attaques par perturbation parmi toutes celles qui sont possibles, car il faut prendre en compte dans cette décision le coût des différentes contre-mesures (en surface et en temps), ainsi que leur taux de détection d'erreurs. Il faut également que le concepteur de circuits sécurisés décide de la politique à mener en cas de détection d'une perturbation.

4.2.1 Détecter une perturbation

Cette section décrit les différentes contre-mesures permettant de détecter les effets d'une perturbation. Elle reprend le plan de la section 3.2, à savoir que la stratégie de détection varie selon que la multiplication scalaire se déroule sur une courbe différente de l'originale ou non. Par ailleurs, il a été montré dans la section 3.2.4.2 qu'il est impératif de protéger la machine d'état contrôlant le cryptosystème, et il est donc montré dans ce qui suit comment procéder. Enfin, des mécanismes plus généraux de détection de perturbation sont également détaillés dans cette section.

4.2.1.1 Détecter une perturbation conduisant au calcul de la multiplication scalaire sur une autre courbe

Les attaques par perturbation mises en évidence par Biehl *et al.* [BMM00], Ciet *et al.* [CJ05] ainsi que celle de Bao *et al.* visant la valeur d'un doublement de points [BDH+97] provoquent le calcul de la multiplication scalaire sur une autre courbe. Un concepteur de circuits sécurisés peut donc s'adapter en vérifiant que les différents points calculés par l'algorithme de multiplication scalaire sont sur la courbe initiale. Cela est donc équivalent pour les courbes obéissant à l'équation de Weierstrass (cf. relation 2.1) à calculer la quantité

$$(y^2 - x^3 - ax)$$

et à vérifier que

$$(y^2 - x^3 - ax) \stackrel{?}{=} b \pmod{p}. \tag{4.17}$$

L'implantation de cette vérification ne coûte que $3M$ (si la multiplication par le paramètre a est négligeable).

Le concepteur de circuits sécurisés doit effectuer cette vérification élementaire au moins une fois, avant de communiquer le point résultat Q (ou $\hat{Q}$ si ce dernier est fauté).

Il peut également le faire à tout autre instant de la multiplication scalaire, mais cela lui coûtera plus cher ($3M$ par vérification supplémentaire), mais surtout le nombre de vérifications implantées dépend de la politique qu'il a choisi d'appliquer en cas de détection d'une perturbation (cf. section 4.2.2).

Il faut enfin noter que cette contre-mesure est inefficace face à l'attaque SCFA [BOS06] et à l'attaque de Bao *et al.* visant un bit de clé [BDH+97] puisque le résultat fauté $\hat{Q}$ appartient dans les deux cas à la courbe initiale [19] .

De plus, afin de se protéger des attaques mises en évidence par Ciet *et al.* [CJ05], il a également été proposé d'implanter un procédé de vérification de l'intégrité des paramètres de courbes a et p *via* une somme de vérification (*Checksum* [Lal00]). Il faut cependant noter que cette vérification est inutile si la relation 4.17 est implantée. Surtout, le *Checksum* a une capacité de détection limitée : si seule cette contre-mesure est implantée, alors un attaquant peut tenter de dépasser les capacités de détection du *Checksum* associé à l'un des paramètres de courbe de sorte que ce dernier ne puisse pas constater de perturbation.

Enfin, l'attaque mise évidence par Bao *et al.* visant la valeur d'un doublement de points [BDH+97] étant une attaque par perturbation différentielle, un attaquant a donc besoin de perturber plusieurs multiplications scalaires utilisant le même scalaire k. C'est pour cela qu'un changement de la valeur de la clé à chaque nouvelle multiplication scalaire, parade possible contre les attaques différentielles par observation, peut être implanté afin de rendre plus difficile cette attaque. Il faut rappeler que la section 4.1.2.1 préconise l'utilisation de la fission additive généralisée proposée par Chevallier-Mames [CM04], qui est considérée comme plus sécurisée et plus performante que les autres techniques d'ajout d'aléa sur la clé.

4.2.1.2 Détecter une perturbation maintenant le calcul de la multiplication scalaire sur la courbe initiale

La section initiale a décrit des contre-mesures possibles contre les attaques par perturbation conduisant au calcul de la multiplication scalaire sur une autre courbe. Or, ces contre-mesures sont inefficaces contre les attaques par perturbation mantenant le calcul de la multiplication scalaire sur la courbe initiale : c'est le cas de l'attaque SCFA [BOS06] et de celle de Bao *et al.* visant un bit de clé [BDH+97].

Pour contre-carrer ces attaques, une solution possible consiste à changer la valeur de la clé à chaque nouvelle multiplication scalaire à l'aide de la fission additive généralisée proposée par Chevallier-Mames [CM04] (cf. section 4.1.2.1). En effet, ces attaques par perturbation étant différentielles, un attaquant a donc besoin de perturber plusieurs multiplications scalaires utilisant le même scalaire k pour les mener à bien.

De plus, Blömer *et al.* [BOS06] proposent d'appliquer l'astuce de Shamir [Sha99] [JPY01] (initialement destinée au RSA-CRT) aux courbes elliptiques afin de se protéger contre l'attaque SCFA qu'ils ont eux-mêmes proposé. Cette astuce consiste à exécuter

[19] Précisons que, concernant l'attaque de Bao *et al.* visant un bit de clé [BDH+97], le point fauté $\hat{Q}$ appartient à la courbe initiale car c'est un multiple du point de base P ($\hat{Q} = [\hat{k}]P$, ou $\hat{k}$ est la valeur fautée de k).

deux fois la multiplication scalaire, la seconde exécution s'effectuant sur un corps fini premier $\mathbb{F}_{p_0}$ (avec $p_0 < p$). La principale particularité de cette contre-mesure est que, contrairement à la duplication où il est calculé successivement (ou en parallèle) deux multiplications scalaires sur la même courbe et le même corps, le calcul de la seconde multiplication scalaire induit un surcoût inférieur car elle se déroule sur un corps contenant moins d'élements.

Plutôt que de calculer $Q = [k]P$ directement, il est effectué plutôt

$$Q_{p_0p} = [k]P_{p_0p} \text{ et } Q_{p_0} = [k]P_{p_0} \text{ où } P_{p_0p} = P \pmod{p_0p} \text{ et } P_{p_0} = P \pmod{p_0}.$$

À la fin, il faut vérifier si la condition

$$Q_{p_0} \stackrel{?}{=} Q_{p_0p} \pmod{p_0}$$

est vérifiée : si ce n'est pas le cas, cela veut dire qu'une faute a été induite.

Comparé à la duplication, cette contre-mesure induit dans la plupart des cas un surcoût moins important, car si p_0 est pris petit, alors le calcul de la multiplication scalaire modulo p_0p combiné à celui de son homologue modulo p_0 conduit à un nombre d'opérations mis en jeu inférieur. Mais surtout, la probabilité qu'un attaquant injecte une faute dans les deux multiplications scalaires sans que cela ne soit détecté est faible [BOS06], contrairement à la duplication (cf. section 4.3.2.1).

Enfin, pour se prémunir des attaques SCFA [BOS06], il a été proposé d'utiliser la version de l'échelle de Montgomery (cf. Algorithme 11 et 12) ne calculant pas la coordonnée y : vu que cette attaque agit sur le signe de cette coordonnée, le fait que cette dernière ne soit plus implantée empêche sa mise en pratique. L'implantation de l'échelle de Montgomery procure également l'avantage de protéger le cryptosystème ECC contre les attaques SSCA (cf. section 4.1.2.1). Mais cette approche comporte deux limitations. Tout d'abord, l'utilisation de l'échelle de Montgomery comporte des inconvénients d'ordre sécuritaire (cf. section 4.1.2.1 et 4.3.1.2). Ensuite, l'échelle de Montgomery n'est pas la contre-mesure SSCA la plus performante, notamment en terme de temps d'exécution (cf. tableau 4.5). Pour toutes ces raisons, l'utilisation de l'échelle de Montgomery ne calculant pas la coordonnée y n'est pas recommandée.

4.2.1.3 Détecter une perturbation dans une machine d'état

Il a été montré dans la section 3.2.4.2 qu'un attaquant pouvait récupérer la clé en injectant des fautes dans la machine d'état contrôlant le cryptosystème.

Ce type d'attaques peut être rendu plus difficile grâce à l'utilisation de codes détecteurs d'erreurs (par exemple le codage de Hamming [Lal00]) pour le recodage des différents états. Cette idée a été développée par Gaubatz *et al.* [GSS08], et elle possède deux avantages. Le premier est que le concepteur de circuits sécurisés peut choisir le compromis taux de détection d'erreurs/surface qui lui convient le mieux : par exemple, si il souhaite que sa machine d'état protégée contre les fautes ait une surface limitée, alors il se contentera d'implanter un codage de Hamming, mais il aura par conséquent

un faible taux de détection de fautes. En revanche, si il désire obtenir une machine d'état ayant un taux plus élevé de détection de fautes, alors il pourra implanter par exemple un codage Reed-Solomon [Lal00], mais le surcoût matériel pour la machine d'état sera par conséquent plus grand. Le second avantage est que la partie du circuit implantée dans la machine d'état servant à détecter d'éventuelles erreurs n'a aucun impact sur la fréquence de fonctionnement du cryptosystème.

4.2.1.4 Autres mécanismes plus généraux de détection de perturbation

La plupart des parades contre les attaques par perturbation décrites précédemment sont uniquement applicables au cas particulier des cryptosystèmes ECC (hormis le procédé de détection de perturbation dans une machine d'état, cf. section 4.2.1.3). Cependant, d'autres mécanismes plus généraux de détection de perturbation pouvant s'appliquer à tous les cryptosystèmes existent également.

Il a d'ailleurs déjà été evoqué précédemment le concept de duplication qui consiste à calculer successivement (respectivement en parallèle) deux fois la même opération [BECN+06] : cette duplication est dite temporelle (spatiale). Malheureusement, l'application de ce concept de duplication, en plus de diminuer les performances temporelles et/ou spatiales du cryptosystème, rend vulnérable le cryptosystème face aux attaques par perturbation du second ordre (cf. section 4.3.2.1).

Un autre mécanisme de protection consiste à calculer successivement une opération cryptographique, puis son inverse pour vérifier qu'aucune perturbation n'a été introduite [BECN+06] : c'est le principe dit « de vérification ». Il expliqué dans ce qui suit un exemple d'application de ce principe sur l'AES. Soit p un texte clair et k la clé privée servant au chiffrement, alors le texte chiffré résultat de l'exécution d'un cryptosystème AES noté c s'écrit :

$$c = \mathrm{AES}(p; k),$$

où $\mathrm{AES}(p; k)$ est elle-même une succesion d'opérations. Un concepteur de circuits sécurisés voulant sécuriser un tel cryptosystème contre les attaques par perturbation peut alors calculer $\mathrm{AES}^{-1}(c; k)$, où AES^{-1} est l'opération inverse de l'AES, et vérifier que :

$$\mathrm{AES}^{-1}(c; k) \stackrel{?}{=} p.$$

Si ce n'est pas le cas, des fautes ont été introduites.

Malheureusement, l'application de ce concept de vérification entraîne les mêmes inconvénients que la duplication, à savoir des performances diminuées pour le cryptosystème, ainsi qu'une vulnérabililité face aux attaques par perturbation du second ordre (cf. section 4.3.2.1).

4.2.2 Politique à mener en cas de détection

La section précédente a décrit différentes techniques permettant de détecter l'effet d'éventuelles perturbations. Une fois ces techniques détaillées, le concepteur de circuits

sécurisés doit maintenant concevoir la stratégie à mettre en place en cas de détection de fautes : faut-il communiquer à l'attaquant un résultat faux ? Faut-il que le cryptosystème se borne à lui communiquer un résultat correct ? Ou encore, faut-il inhiber ou désactiver temporairement ou définitivement le cryptosystème ? Dans cette section, il pourra être constaté que le choix de la politique à adopter en cas de détection influe sur celui de la contre-mesure et de l'implantation de cette dernière.

4.2.2.1 Communiquer à l'attaquant un résultat faux

Une première politique possible consiste à communiquer un résultat faux à l'attaquant si une perturbation est détectée. Cette technique est également connue sous le nom de « calculs infectieux ». Celle-ci a été appliquée notamment sur le RSA-CRT par Yen *et al.* [YKLM03], puis sur l'AES par Joye *et al.* [JMR07].

Il faut noter que l'utilisation de cette technique suscite une réserve importante : le résultat faux communiqué à l'attaquant ne doit pas lui donner d'informations exploitables pour récupérer la clé. C'est malheusement le cas de la plupart des schémas RSA-CRT utilisant ce procédé (par exemple Blömer *et al.* [BOS03] attaquent avec succès les deux algorithmes utilisant le concept des calculs infectieux proposés par Yen *et al.* [YKLM03]).

En fait, il est inutile et même dangereux de prendre le risque de communiquer à l'attaquant un point résultat faux qui pourrait paraître *a priori* inexploitable. C'est d'ailleurs l'une des raisons pour laquelle il n'est pas publié à ce jour de technique de calculs infectieux appliquée à l'ECC. Ainsi, pour cette raison, il est plutôt déconseillé d'appliquer cette politique.

4.2.2.2 Décider de ne communiquer que le résultat correct

Un concepteur de circuits sécurisés peut également faire en sorte que le cryptosystème qu'il implante communique quoi qu'il arrive le résultat correct.

Prenons l'exemple d'un cryptosystème implantant comme unique contre-mesure face aux attaques par perturbation la vérification que le point $Q = [k]P$ est bien sur la courbe initiale : si c'est le cas, il communique Q, sinon il recommence tout le calcul de la multiplication scalaire. Cette implantation amène deux remarques.

Tout d'abord, dans le cas où des opérations inutiles sont utilisées pour protéger le cryptosystème contre les attaques SSCA (cf. section 4.1.1), si un attaquant perturbe l'exécution de celles-ci une à une à chaque exécution de la multiplication scalaire, alors il sera dans la capacité de les différencier des opérations utiles, puisqu'il constatera (respectivement il ne constatera pas) un délai supplémentaire pour la communication du résultat si l'opération attaquée était (in)utile. Ce phénomène peut donc favoriser les attaques par perturbation de type CSEA [YKLM01]. Le fait d'effectuer cette vérification à intervalles réguliers (par exemple après chaque addition de points) ne change d'ailleurs rien à l'affaire, puisque les éventuels délais supplémentaires engendrés, même si ces derniers sont plus courts, pourront être quand même mesurés par un attaquant.

Ensuite, les attaques par perturbation et par observation peuvent être combinées pour façonner le concept d'attaque par analyse de comportement (*Differential Behavioral Ana-*

lysis, DBA). Ce principe a été appliqué sur le RSA par Joye *et al.* [JQYY02], et sur l'AES par Robisson *et al.* [RM07]. Dans ces deux publications, le comportement d'un cryptosystème comprend l'acceptation ou le rejet d'une signature ou d'un chiffrement, le temps total de calcul d'une signature ou d'un chiffrement et les relevés de divers canaux cachés (la consommation électrique, le rayonnement EM émis). Si ce comportement peut être observé pour différentes fautes induites à différents moments de l'exécution d'une multiplication scalaire avec un scalaire fixe, alors un attaquant peut être capable de récupérer la clé. C'est pour cela que si cette politique est choisie, il est vivement recommandé d'agir sur la représentation de la clé (cf. section 4.1.2.1).

4.2.2.3 Inhiber le cryptosystème temporairement ou définitivement

Un concepteur de circuits sécurisés peut également décider d'inhiber (ou de désactiver) le cryptosystème temporairement ou définitivement en cas de détection d'une perturbation.

Cette inhibition amène au moins deux conséquences importantes. La première est qu'elle conduit le cryptosystème à ne pas communiquer le point résultat fauté, ce qui contrarie les attaques DFA qui ont besoin de ce dernier pour retrouver la clé. La seconde est que, vu que le cryptosystème est rendu inutilisable, une attaque de type DBA sera difficile puisque cette dernière nécessite justement l'analyse de plusieurs opérations cryptographiques (signatures ou chiffrements par exemple), et non d'une seule.

C'est pour ces deux raisons que nous conseillons d'appliquer cette politique. Le but pour un concepteur de circuits sécurisés étant de donner le moins d'informations possibles à l'attaquant, il peut donc, au lieu d'interrompre le fonctionnement du cryptosystème dès qu'une perturbation est détectée (cette interrruption fait partie du comportement du cryptosystème, et peut donc être observée par un attaquant [JQYY02] [RM07]), forcer le cryptosystème à continuer le calcul de la multiplication scalaire (perturbé) jusqu'au bout, et ce, sans donner le résultat final. Ainsi, l'attaquant peut constater que son attaque a bien fonctionné, mais il n'a pas de certitude absolue sur l'endroit et le moment de la perturbation qu'il a induite.

4.2.3 Bilan

Il convient dans cette section de faire un bilan des contre-mesures à implanter pour protéger un cryptosystème ECC vis-à-vis des attaques par perturbation, et de mettre celles-ci en rapport avec la politique choisie par le concepteur de circuits sécurisés en cas de détection d'une telle attaque.

Pour résumer, la section précédente montre que le concepteur de circuits sécurisés doit appliquer une politique adaptée en cas de détection d'une perturbation sous peine de provoquer des différences observables dans le comportement du cryptosystème permettant à un attaquant de récupérer la clé. En d'autres termes, un cryptosystème dont la sécurité a été (apparemment) amélioré vis-à-vis d'une attaque particulière peut laisser échapper des informations observables qui peuvent permettre de retrouver la clé.

Par conséquent, il peut donc être conseillé d'inhiber le cryptosystème temporairement ou définitivement, et de continuer le calcul de la multiplication scalaire jusqu'au bout, qu'une faute ait été détectée ou pas, de façon à ce que l'attaquant ait le moins de détails possibles sur les raisons de la réussite de son attaque. Ainsi, la combinaison de ces deux actions implique qu'il peut être effectué une seule vérification de non-perturbation à la fin de la multiplication scalaire : il n'est donc pas conseillé d'implanter l'astuce de Shamir adaptée à l'ECC [BOS06], puisqu'elle consiste à calculer à tout instant deux multiplications scalaires. Par conséquent, pour détecter une perturbation conduisant le calcul de la multiplication scalaire sur une autre courbe, la vérification que le point résultat Q est bien sur la courbe initiale, et ce, uniquement à la fin de la multiplication scalaire suffit (cf. relation 4.17).

Il faut également pouvoir détecter les attaques par perturbation maintenant le calcul de la multiplication scalaire sur la courbe initiale, comme par exemple celle mise en évidence par Bao *et al.* visant un bit de clé [BDH+97], ou encore l'attaque SCFA [BOS06]. Or, ce sont des attaques par perturbation différentielles : un attaquant a donc besoin de perturber plusieurs multiplications scalaires utilisant le même scalaire k. C'est pour cette raison qu'il peut être conseillé de changer la valeur de k avant chaque nouvelle multiplication scalaire, et la méthode de fission du scalaire proposée par Chevallier-Mames [CM04] est toute indiquée pour cela.

Enfin, pour protéger la machine d'état d'éventuelles perturbations, la méthode de Gaubatz *et al.* [GSS08] doit être choisie, et le concepteur de circuits sécurisés ne doit d'ailleurs pas hésiter à utiliser un code détecteur d'erreurs possédant une grande capacité de détection, quitte à obtenir une machine d'état moins performante en terme de surface.

4.3 Variantes d'attaques et contre-mesures associées

La section 4.1 a décrit les contre-mesures possibles vis-à-vis des attaques SSCA et DSCA pour les cryptosystèmes ECC, tandis que la section 4.2 a détaillé celles relatives aux attaques DFA. Cette section décrit des variantes de ces attaques permettant de contourner potentiellement les sécurités mises en place par un attaquant. Ces nouvelles attaques conduisent le cas échéant à l'implantation de contre-mesures supplémentaires qui sont elles-mêmes détaillées dans ce qui suit.

4.3.1 Variantes d'attaques différentielles par observation

Dans cette section, il est décrit différentes variantes d'attaques DSCA. Parmi celles-ci, les attaques DSCA dites « d'ordre supérieur » consistent à analyser différentes portions d'un relevé de canal caché afin de contourner une contre-mesure implantée (cf. section 4.1.2). D'autres variantes d'attaques DSCA ne visent pas directement les variables intermédiaires traitées par le cryptosystème, mais plutôt l'adresse des registres qui les contiennent, ou encore le fait que ces variables soient réutilisées ou non. Ces variantes d'attaques vont conduire à un raffinement des contre-mesures décrites précédemment (cf.

section 4.1.2), et à la conceptualisation de nouvelles parades.

4.3.1.1 Attaque différentielle par analyse de canaux cachés d'ordre supérieur

Après avoir posé les bases de l'attaque DSCA (cf. section 3.1.3), Kocher *et al.* ont présenté une amélioration de celle-ci appelée attaque DSCA « d'ordre supérieur » (*High Order Differential Side-Channel Analysis*, HODSCA) [KJJ99]. Cette idée a ensuite été mise en pratique par Messerges [Mes00], et développée par Joye *et al.* [Joy04] [JPS05].

Au lieu d'étudier les variables intermédiaires traitées à un seul moment t de l'algorithme (comme c'est le cas lors d'une attaque DSCA classique, cf. section 3.1.3.2), l'idée sous-jacente de cette attaque est d'étudier plusieurs moments $t_1, \cdots, t_n$. Par exemple, pour une attaque DSCA d'ordre $n = 2$, un attaquant prédit la valeur de la variable intermédiaire Q_r à deux endroits différents de la courbe correspondant à deux instants t_1 et t_2. Si ces deux valeurs sont « masquées » de la même façon, c'est-à-dire qu'il a été effectué la même opération booléenne ou mathématique sur celles-ci avec la même variable aléatoire, alors ce masquage peut être éliminé pourvu qu'un attaquant applique une fonction de combinaison appropriée sur celles-ci (par exemple une différence ou une multiplication suivant la technique de masquage employée). Ainsi, l'effet du masquage étant éliminé, l'attaquant peut construire de nouvelles courbes qui sont le fruit de cette combinaison, et appliquer le protocole expérimental d'une attaque DSCA classique sur ces nouvelles courbes (cf. section 3.1.3.2).

Il peut donc être conclu qu'utiliser une seule protection (un seul masque) à un seul instant contre les attaques DSCA peut ne pas être suffisant, et qu'il faut en utiliser plusieurs à différents instants qui ne puissent pas être recombinés à l'aide d'une attaque HODSCA. Se protéger contre ce genre d'attaque est donc un exercice périlleux.

4.3.1.2 DSCA visant l'adresse des registres

Il existe une autre variante d'attaque DSCA qui vise non pas les variables intermédiaires traitées par le cryptosystème, mais plutôt l'adresse des registres qui les contiennent.

Dans le cadre du fonctionnement de certains cryptosystèmes, le chargement de données dans un registre possédant une adresse spécifique (qui est équivalent à effectuer l'opération $R_{\text{adresse}} \leftarrow$ données) peut se décomposer en deux étapes. Tout d'abord, l'adresse physique notée « adresse » est extraite et traverse un bus de communication (de la même façon que les données), pour qu'ensuite le chargement des données dans le registre concerné puisse être effectué. Or, le transit de ces adresses *via* ce bus amène de l'information, puisque des adresses différentes peuvent être distinguées à l'aide d'une attaque DSCA.

Dans [IIT02], Itoh *et al.* attaquent de cette manière un cryptosystème ECC implantant l'échelle de Montgomery (cf. Algorithme 11) comme parade contre les attaques SSCA, et le changement des coordonnées projectives (cf. section 4.1.2.2) comme contre-mesure vis-à-vis des attaques DSCA. Plus précisément, ils attaquent l'opération $R_0 \leftarrow [2]R_0$ (respectivement $R_1 \leftarrow [2]R_1$) qui est effectuée dans le cadre de l'échelle de Mongomery lorsque $k_i = 0$ ($k_i = 1$) : puisque le résultat de ce doublement de points est mémorisé

à une adresse fixe dépendante de la valeur du bit de clé traité k_i, un attaquant pourra retrouver ce dernier en effectuant une attaque DSCA sur le bus de communication afin de retrouver l'adresse du registre qui charge les données.

Ainsi, la réutilisation des registres peut donner le cas échéant suffisamment d'information à un attaquant pour retrouver la clé. Un concepteur de circuits sécurisés peut contrer ce type d'attaque en utilisant les techniques proposées par May *et al.* [MMS01] et Izu *et al.* [IIT03] consistant à charger les résultats d'opérations intermédiaires dans des registres différents à chaque itération.

4.3.1.3 DSCA par réutilisation des opérandes

Dans la section précédente, il a été décrit le principe d'une attaque DSCA utilisant une éventuelle réutilisation des registres du cryptosystème. Il peut également être monté une attaque DSCA par réutilisation des opérandes. Ce principe a été développé initialement sur le RSA [WT01] [Sch02], et il est également applicable sur les cryptosystèmes ECC comme suit.

Si l'algorithme de multiplication scalaire implanté utilise le $\mathrm{NAF}_w(k)$ (cf. section 2.2.3.2), alors l'utilisation de l'algorithme 10 conduira au précalcul des points P, $[3]P$, $\cdots$, $[2^{w-1}-1]P$, ainsi qu'à la mémorisation de ces derniers dans une table. Le problème est que l'utilisation d'une table mémorisant à tous les instants les mêmes éléments aux mêmes emplacements peut conduire à des attaques statistiques étudiant l'utilisation de ces points, et permettant ainsi de retrouver la clé. L'utilisation de la contre-mesure SSCA proposée par Möller [Möl01] consistant à recoder le scalaire pour utiliser un algorithme de multiplication scalaire régulier entraîne également ce problème de sécurité : un attaquant, en étudiant l'occurence d'utilisation des chiffres contenus dans l'ensemble $\{-2^w, \pm 1, \pm 2, \cdots, \pm(2^{w-1}-1), 2^{w-1}\}$, peut remonter à la clé [IMT02].

Cette variante d'attaque DSCA peut être rendue plus difficile si la représentation interne des points mémorisés dans la table est rendue aléatoire (par exemple, en changeant la valeur de leurs coordonnées projectives, cf. section 4.1.2.2), et si l'emplacement des différents points est changé à chaque utilisation de la table. Mais il faut noter que l'utilisation d'une telle contre-mesure entraîne des conséquences importantes pour le cryptosystème, notamment une augmentation de sa consommation électrique.

4.3.2 Variantes d'attaques par perturbation

La section précédente a présenté des variantes d'attaques DSCA. Il existe également deux variantes d'attaques par perturbation qui sont détaillées dans cette section. Tout d'abord, il est présenté celle dite « d'ordre supérieur » qui consiste à induire plusieurs perturbations pendant le déroulement d'un algorithme cryptographique afin de contourner des protections. Ensuite, il est détaillé une variante d'attaque SEA nommée MSEA qui utilise la perturbation de quelques bits du multiplicateur après leur utilisation dans une

multiplication modulaire à réduction intégrée (cf. section 1.2.1.2).

4.3.2.1 Attaque par perturbation d'ordre supérieur

Comme il existe des attaques DSCA d'ordre supérieur (cf. section 4.3.1.1), il existe également des attaques par perturbation du même type (*High Order Fault Analysis*, HOFA). L'idée sous-jacente de ce type d'attaque est d'induire n (> 1) perturbations distinctes (c'est donc une attaque par perturbation d'ordre n) afin de pouvoir dans une première phase fauter un résultat intermédiaire, puis dans une seconde phase faire en sorte que cette perturbation ne soit pas détectée par une éventuelle contre-mesure implantée. Cette idée a été pour la première fois mise en pratique par Kim *et al.* [KQ07]. Dans ce qui suit, le principe de cette attaque est illustré sur deux contre-mesures de base contre les attaques par perturbation que sont la duplication et la vérification (cf. section 4.2.1.4).

La duplication consiste à calculer successivement (respectivement en parallèle) deux fois la même opération, puis à vérifier à l'aide d'un opérateur « égalité » que les résultats des deux calculs sont bien les mêmes. Ainsi, un concepteur de circuits sécurisés implantant cette contre-mesure espère que si l'un des deux calculs est perturbé, alors celui-ci pourra être détecté [BECN+06]. Or, si un attaquant effectue une HOFA en perturbant successivement le calcul d'une des opérations et le fonctionnement de l'opérateur égalité, alors il peut espérer induire des fautes qui ne seront pas détectées, et ainsi récupérer une signature ou un texte (dé)chiffré erroné lui permettant de retrouver la clé. De plus, une autre HOFA peut être montée en injectant la même faute sur les deux calculs : ainsi, vu que l'opérateur égalité ne sert qu'à vérifier que les résultats des deux calculs sont les mêmes, il ne constatera pas de fautes de son point de vue. Le scénario de cette seconde HOFA est d'autant plus réaliste si des perturbations entraînant un délai supplementaire de traversée de portes logiques sont induites [FF07] : si les deux calculs sont effectués sur deux UA ayant une architecture parfaitement identique et fonctionnant en parallèle, alors un même délai supplémentaire induit sur les deux circuits va produire les mêmes fautes sur ces derniers, et l'opérateur égalité ne constatera pas de fautes de son point de vue.

Ainsi, l'émergence des attaques HOFA conduit à ne pas utiliser le principe de duplication comme contre-mesure vis-à-vis des attaques par perturbation.

La vérification consiste quant à elle à calculer successivement une opération cryptographique (notée $f(x)$), puis l'inverse de cette dernière (donc $f^{-1}(f(x))$), et à vérifier à l'aide d'un opérateur égalité que $(f^{-1}(f(x)) \stackrel{?}{=} x)$ [BECN+06]. Ainsi, un concepteur de circuits sécurisés implantant cette contre-mesure espère que si un attaquant perturbe le calcul de l'opération $f(x)$, celle-ci sera détectée. Or, comme pour le cas de la duplication, si un attaquant effectue une HOFA en perturbant successivement le calcul d'une des opérations et le fonctionnement de l'opérateur égalité, alors il peut espérer induire des fautes qui ne seront pas détectées, et ainsi récupérer une signature ou un texte (dé)chiffré erroné lui permettant de retrouver la clé. En revanche, contrairement à la duplication, si la même faute est injectée lors du calcul de f et de f^{-1}, alors un concepteur de circuits sécurisés peut espérer qu'il se produira avec une forte probabilité la relation $(f^{-1}(f(x)) \neq x)$:

l'attaque HOFA sera ainsi détectée.

Ainsi, le concept de vérification peut être utilisé pour rendre plus difficiles les attaques HOFA, mais cela induit un surcoût. Pour illustrer cet état de fait, il peut être pris l'exemple du chiffrement AES où il est effectué $c = \text{AES}(p; k)$, et où l'intégrité de ce dernier peut être vérifié en utilisant le déchiffrement correspondant $\text{AES}^{-1}(c; k)$ (cf. section 4.2.1.4). Le premier problème amené par cette approche est que même si le déchiffrement n'est pas requis pour l'application choisie, il doit quand même être implanté, ce qui conduit à une augmentation de la taille du code compilé dû au développement de cette nouvelle fonction. Le second problème est que, dans le cas spécifique de l'AES, le déchiffrement AES est plus lent que le chiffrement, ce qui induit un surcoût en terme de temps de calcul.

De plus, si un concepteur de circuits sécurisés veut utiliser le concept de vérification, il faudra qu'il sécurise l'opérateur égalité, ainsi que le résultat que ce dernier communique au cryptosystème[20]. Il peut par exemple décider d'effectuer deux comparaisons au lieu d'une, mais cela a pour conséquence d'augmenter la taille du code compilé ainsi que le temps de calcul. Il peut également appliquer une politique de calculs infectieux [YKLM03] [JMR07] : ce faisant, aucun test d'égalité n'est effectué, et si une faute est injectée, l'attaquant récupérera un point résultat faux, mais qu'il ne pourra pas exploiter pour retrouver la clé.

Cette section a détaillé le principe des attaques HOFA sur des contre-mesures de base pouvant être implantées pour n'importe quel cryptosystème (duplication et vérification). Pour le cas spécifique de l'ECC, la section 4.2.3 préconise, pour détecter une perturbation conduisant le calcul de la multiplication scalaire sur une autre courbe, de vérifier que le point résultat Q est bien sur la courbe initiale, et ce, **uniquement à la fin de la multiplication scalaire** : cela conduit donc **à un seul calcul** de la relation 4.17. Or, dans le cadre d'une attaque par perturbation d'ordre deux, un attaquant peut injecter une faute pendant le calcul de la multiplication scalaire, et perturber également le résultat **de l'unique calcul** de la relation 4.17 de telle sorte que le cryptosystème ne détecte pas d'attaque. Pour compliquer la tâche de l'attaquant, il peut donc être conseillé de l'obliger à effectuer une attaque par perturbation d'ordre n très grand : cela l'obligera à contourner un grand nombre de tests, ce qui sera plus difficile à réaliser. C'est pour cela que nous conseillons d'effectuer le calcul de la relation 4.17 **après chaque opération de points** (par exemple une addition ou un doublement) mise en jeu dans l'algorithme de multiplication scalaire implanté. Par exemple, si l'algorithme du « doublement-et-addition » utilisant le $\text{NAF}_2(k)$ est choisi (cf. Algorithme 10), alors il pourra être effectué $(\ell + \ell/3 = 4\ell/3)$ tests. Chaque test coûtant $3M$ (si la multiplication par le paramètre a est négligeable), alors $4\ell M$ seront calculées en plus.

Pour sécuriser ce dernier test, il n'est pas recommandé d'appliquer une politique de calculs infectieux, car celle-ci n'est pas applicable dans le cadre de l'ECC pour des raisons

[20] C'est d'ailleurs le cas dans les cartes à puces récentes, dans lesquelles les registres mémorisant les résultats de ces tests sensibles sont protégés par des mécanismes de redondance très efficaces tels que des *checksums* matériels et des codes correcteurs d'erreur [Lal00].

de sécurité (cf. section 4.2.2.1). En revanche, il peut être conseillé d'effectuer **deux tests successifs au lieu d'un**, ce qui porte à $8\ell M$ le surcoût engendré par cette contre-mesure. De même, pour rendre encore plus difficile la tâche d'un attaquant, ces tests peuvent être effectués **à des moments aléatoires**, et non pas forcément à la fin de chaque opération de points : cela va ainsi rendre plus difficile la synchronisation d'une attaque par perturbation.

Pour résumer, l'émergence des attaques HOFA conduit le concepteur de circuits sécurisés à étudier intensivement les vulnérabilités de son cryptosystème vis-à-vis de ces attaques, et à implanter des contre-mesures plus coûteuses et/ou plus subtiles.

4.3.2.2 Autre type d'attaque par perturbation sans conséquence sur les calculs

Il a déjà été évoqué précédemment un type d'attaque par perturbation sans conséquence sur les calculs appelée CSEA [YKLM01], qui utilise le fait qu'une faute aléatoire est induite durant une opération inutile exécutée par le cryptosystème. Il existe un autre type d'attaque par perturbation sans conséquence, appelée attaque MSEA, qui suppose que quelques bits du multiplicateur sont modifiés après avoir été utilisés dans un algorithme de multiplication modulaire croisée (par exemple, l'algorithme 21, qui est en fait l'algorithme 7 avec $\beta = 2$).

Algorithme 21 Exemple d'algorithme de multiplication modulaire croisée pouvant être le siège d'une attaque par perturbation sans conséquence sur les calculs visant quelques bits du multiplicateur

Entrées : $p = (p_{n-1} \cdots p_1 p_0)_2$, $A = (a_{n-1} \cdots a_1 a_0)_2$, $B = (b_{n-1} \cdots b_1 b_0)_2$.
Sorties : $S = \text{MUL}(A, B, p) = A.B \pmod p$.

1. $S \leftarrow 0$
2. **pour** $i = n - 1$ à 0 **faire**
3. $\quad S \leftarrow 2S + A.b_i \pmod p$
4. **fin pour**
5. **retourner** S

Le principe de l'attaque MSEA peut se résumer ainsi : la valeur du multiplicateur B sera correcte après l'operation $B \leftarrow A.B \pmod p$, même si des bits b_i du multiplicateur sont modifiés après avoir été employés dans l'algorithme de multiplication modulaire. Or, selon le mode d'utilisation de cet algorithme dans un algorithme de multiplication scalaire, cela peut donner suffisamment d'information à un attaquant pour retrouver la clé. Yen *et al.* [YJ00] ont montré un exemple d'application de cette attaque sur un algorithme d'exponentiation modulaire utilisable notamment dans le cadre du RSA.

Heureusement, pour contre-carrer ce type d'attaque, une parade à coût minime et simple à implanter a été proposée par Yen *et al.* [YJ00] : B peut jouer le rôle de multiplicande, c'est-à-dire qu'il peut être appelé dans l'algorithme de multiplication scalaire

la routine $B \leftarrow \text{MUL}(B, A, p)$, au lieu de $B \leftarrow \text{MUL}(A, B, p)$. Ainsi, puisque B est toujours appelé à chaque fois que l'étape 3 de l'algorithme 21 est effectuée, n'importe quelle perturbation de l'opérande B va toujours conduire à un résultat faux, et ce quelque soit le mode d'utilisation de cet algorithme de multiplication modulaire dans le cadre d'une multiplication scalaire. Une perturbation sur B ne va donc jamais être « sans conséquence sur les calculs », et par conséquent l'attaque MSEA se voit annihilée.

4.4 Conclusion de chapitre

Il a pu être constaté tout au long de ce chapitre qu'un grand nombre de contre-mesures a été proposé dans la littérature pour protéger un cryptosystème contre les attaques par observation (SSCA et DSCA) et par perturbation. Il convient donc dans cette conclusion de résumer les idées maîtresses de ces différentes parades, et d'en tirer des leçons quant à la conception de notre UA.

La première leçon qui peut être tirée de cette étude est que **les attaques par observation sont une menace réelle pour les cryptosystèmes ECC** contre lesquelles un concepteur de circuits sécurisés doit absolument se prémunir. Pour ce faire, contre les attaques SSCA, il doit faire en sorte que le cryptosystème implanté exécute une séquence fixe d'opérations qui ne peut pas être reliée *via* un relevé de canal caché aux bits traités du scalaire. Pour cela, il peut soit utiliser un algorithme de multiplication scalaire régulier (c'est-à-dire que ce dernier exécute intrinsèquement la même séquence d'opérations), soit rendre l'addition et le doublement de points indistinguables. L'étude de la section 4.1.1 a permis de constater que le concept d'atomicité [CMCJ04] est le concept de base le plus adapté pour lutter contre les attaques SSCA, car il est le moins coûteux en terme de temps de calcul. Un concepteur de circuits sécurisés devra cependant garder à l'esprit que le cryptosystème ainsi obtenu sera vulnérable face aux attaques par perturbation de type CSEA [YKLM01], même si cela requiert pour un attaquant d'avoir une très grande précision temporelle d'injection de fautes.

C'est ainsi qu'une deuxième leçon peut être tirée : **lorsqu'on protège un cryptosystème** contre un type d'attaque particulier (ici, les attaques par analyse de canaux cachés), cela peut **introduire de nouvelles vulnérabilités** (ici, la possibilité de monter une attaque CSEA). Autrement dit, à chaque fois qu'un concepteur de circuits sécurisés veut implanter une nouvelle contre-mesure vis-à-vis d'un type d'attaque particulier, il doit réévaluer la sécurité globale de son cryptosystème. Heureusement, les attaques CSEA sont rendues plus difficiles si une nouvelle valeur de la clé est choisie aléatoirement à chaque nouvelle multiplication scalaire (cf. section 4.1.2.1).

Or, modifier la valeur du scalaire fait également partie des parades implantables face aux attaques DSCA. Plus précisément, un concepteur de circuits sécurisés peut compliquer le montage d'une attaque DSCA **en agissant sur la clé ET sur le point de base** [Joy05], et c'est là la troisième leçon qu'un concepteur de circuits sécurisés peut

tirer de cette étude. Cependant, il devra par ailleurs étudier cette combinaison de contre-mesures pour vérifier qu'elle n'induit pas de vulnérabilités au cryptosystème. Il peut ainsi être conseillé de choisir l'implantation de la fission du scalaire proposée par Chevallier-Mames [CM04], qui intègre une contre mesure SSCA (l'atomicité [CMCJ04]) combinée avec la technique « d'aveuglement » du point de base [Cor99] : cela coûtera seulement $(\ell + 2)D + (3\ell/2 + 4)A$, mais le cryptosystème n'est protégé qu'*a priori* contre toutes les attaques SSCA et DSCA connues à ce jour : le concepteur de circuits sécurisés devra donc mener des études poussées sur le cryptosystème obtenu afin d'étudier les éventuelles faiblesses sécuritaires de ce dernier. Il faut également noter que l'implantation de ces deux contre-mesures DSCA (l'une modifiant la clé, l'autre le point de base) n'est pas vulnérable face à l'attaque DSCA particulière de Goubin [Gou03], de même qu'elle peut rendre plus difficile les attaques HODSCA, car les masques différents produits par ces deux parades paraissent difficiles à recombiner. Les concepteurs de circuits sécurisés devront toutefois se tenir informés de l'état de l'art sur ces attaques, car elles connaissent un développement continu ces dernières années [Joy04] [JPS05].

Enfin, les attaques DSCA basées sur la réutilisation des registres ou des opérandes peuvent être contrées en utilisant les techniques proposées par May *et al.* [MMS01] et Izu *et al.* [IIT03] qui consistent à charger les résultats d'opérations intermédiaires dans des registres différents à chaque itération.

La quatrième leçon qui peut être tirée de cette étude est que **les attaques par perturbation sont également une menace réelle pour les cryptosystèmes ECC**. Pour contre-carrer ce type d'attaques, il faut implanter une méthode à fort taux de détection de fautes, et appliquer une politique adaptée en cas de détection d'une perturbation sous peine de provoquer des différences observables dans le comportement du cryptosystème permettant à un attaquant de récupérer la clé (par exemple dans le cadre d'une attaque différentielle par analyse de comportement − DBA [RM07]). En d'autres termes, **un cryptosystème dont la sécurité a été (apparemment) amélioré vis-à-vis d'une attaque particulière peut laisser échapper des informations observables qui peuvent permettre de retrouver la clé**, et c'est là la cinquième leçon pouvant être déduite de cette étude. Il peut finalement être conseillé de continuer le calcul de la multiplication scalaire jusqu'au bout, qu'une faute ait été détectée ou pas, et de ne pas fournir un résultat fauté : l'attaquant récupère ainsi le moins de détails possibles sur les raisons de la réussite de son attaque.

Pour détecter une perturbation conduisant le calcul de la multiplication scalaire sur une autre courbe, la vérification que le point résultat Q est bien sur la courbe initiale est suffisante (cf. relation 4.17). Il faut également pouvoir détecter les attaques par perturbation mantenant le calcul de la multiplication scalaire sur la courbe initiale, comme par exemple celle mise en évidence par Bao *et al.* visant un bit de clé [BDH$^+$97], ou encore l'attaque SCFA [BOS06]. Or, ce sont des attaques par perturbations **différentielles** : un attaquant a donc besoin de perturber plusieurs multiplications scalaires utilisant le même scalaire k. C'est pour cette raison qu'il peut être conseillé de changer la valeur de k avant chaque nouvelle multiplication scalaire, or l'implantation de cette contre-mesure est

déjà conseillée pour protéger un cryptosystème ECC contre les attaques **différentielles** par observation. Enfin, pour protéger la machine d'état d'éventuelles perturbations, la méthode de Gaubatz *et al.* [GSS08] doit être choisie, et le concepteur de circuits sécurisés ne doit d'ailleurs pas hésiter à utiliser un code détecteur d'erreurs possédant une grande capacité de détection, quitte à obtenir une machine d'état moins performante en terme de surface.

Pour compliquer la tâche de l'attaquant, il peut également être conseillé de l'obliger à effectuer une attaque par perturbation d'ordre n très grand : cela l'obligera à contourner un grand nombre de tests, ce qui sera plus difficile à réaliser. C'est pour cela que nous conseillons d'effectuer le calcul de la relation 4.17 **après chaque opération de points** (par exemple une addition ou un doublement) mise en jeu dans l'algorithme de multiplication scalaire implanté. Par exemple, si l'algorithme du « doublement-et-addition » utilisant le $\mathrm{NAF}_2(k)$ est choisi (cf. Algorithme 10), alors il pourra être effectué ($\ell + \ell/3 = 4\ell/3$) tests. Chaque test coûtant $3M$ (si la multiplication par le paramètre a est négligeable), alors $4\ell M$ seront calculées en plus.

Pour sécuriser les différents calculs de la relation 4.17, il n'est pas recommandé d'appliquer une politique de calculs infectieux, car celle-ci n'est pas applicable dans le cadre de l'ECC pour des raisons de sécurité (cf. section 4.2.2.1). En revanche, il peut être conseillé d'effectuer **deux tests successifs au lieu d'un**, ce qui porte à $8\ell M$ le surcoût engendré par l'implantation de cette contre-mesure. De même, pour rendre encore plus difficile la tâche d'un attaquant, ces tests peuvent être effectués **à des moments aléatoires**, et non pas forcément à la fin de chaque opération de points : cela va ainsi rendre plus difficile la synchronisation d'une attaque par perturbation.

Pour conclure, nous pouvons écrire que l'émergence de variantes d'attaques et d'attaques d'ordre supérieur (c'est le cas des attaques HOFA, mais également des attaques HODSCA) conduit un concepteur de circuits sécurisés à étudier intensivement les vulnérabilités de son cryptosystème vis-à-vis de ces attaques, et à implanter des contre-mesures plus coûteuses et/ou plus subtiles. Ces attaques étant en constante évolution, nous assistons ainsi à une course continuelle à laquelle se livrent les concepteurs de circuits sécurisés et les attaquants dans le cas spécifique de l'ECC depuis une dizaine d'années, et cela n'est semble-t-il pas prêt de s'arrêter.

Deuxième partie

Conception d'une unité arithmétique hautes performances protégée contre les attaques par observation et par perturbation

Chapitre 5

Une unité arithmétique performante prenant en compte l'état de l'art

Sommaire

D'un point de vue théorique, les cryptosystèmes ECC disposent d'un avantage décisif sur ceux implantant le RSA : à niveau de sécurité équivalent, l'ECC requiert des clés beaucoup plus petites que le RSA. Rappelons par exemple qu'utiliser un RSA avec

une longueur de clé de 1024 bits apporte le même niveau de sécurité qu'un ECC avec une longueur de clé de 160 bits [SEC00a]. Il peut ainsi être espéré que le temps de calcul, la consommation électrique et l'espace mémoire mis en jeu soient réduits, et c'est pour toutes ces raisons que, théoriquement, l'ECC est un candidat crédible pour remplacer le RSA dans le cadre de la cryptographie à clé publique [Van04]. Il reste maintenant à vérifier que cet état de fait se confirme **d'un point de vue pratique**.

Pour implanter un cryptosystème, un concepteur de circuits sécurisés a le choix entre deux cibles technologiques principales : d'une part les processeurs, qu'ils soient généralistes (par exemple les Intel Pentium©) ou spécifiques (par exemple les processeurs graphiques – *Graphics Processing Unit*, GPU) et d'autre part les circuits intégrés, qu'ils soient dédiés ASIC ou programmables (par exemple les réseaux de portes programmables – *Field Programmable Gate Array*, FPGA). De nombreuses implantations (logicielles) de l'ECC sur processeurs sont présentes dans la littérature. Le tableau 5.1 détaille pour chaque cryptosystème (et sa référence bibliographique associée) la cible technologique choisie (deuxième colonne), le corps fini premier et la longueur du chemin de données implantés (troisième colonne), la fréquence de la cible (quatrième colonne) et le temps de calcul global de l'opération cryptographique principale à effectuer, à savoir soit une exponentiation modulaire pour le RSA soit une multiplication scalaire pour l'ECC (cinquième colonne). Un constat s'impose : les performances de la plupart des implantations

Cryptosystème	**Cible**	**Implantation**	**Fréquence**	**Temps**
ECC [BHLM01]	Intel Pentium2©	$\mathbb{F}_{p_{192}}$	400 MHz	3686 μs
ECC [Ber01]	Intel Pentium4©	$\mathbb{F}_{p_{224}}$	1,4 GHz	599 μs
ECC[21]	Intel Core2 Duo©	$\mathbb{F}_{p_{256}}$	2,13 GHz	669 μs
ECC [GT07]	Intel Core2 Duo©	$\mathbb{F}_{2^{255}-19}$	2,66 GHz	145 μs
ECC [SG08]	GPU Nvidia 8800 GTS©	$\mathbb{F}_{p_{224}}$	1,2 GHz	708 μs
RSA [BP01]	FPGA Xilinx©	1024 bits	45,2 MHz	3,1 ms
RSA [Suz07]	FPGA Xilinx©	1024 bits	400 MHz	1,71 ms
RSA [STCK04]	ASIC (0,5 μm CMOS)	1024 bits	64 MHz	46 ms

Tab. 5.1 – Quelques implantations logicielles de cryptosystèmes à clé publique.

(logicielles) de cryptosystèmes ECC présentées dans ce tableau surpassent celles relatives aux conceptions matérielles du RSA, ces dernières ayant qui plus est un niveau de sécurité inférieur. Ainsi, il peut être conclu que, **sur processeurs généralistes ou spécifiques**, l'ECC est une réelle alternative au RSA.

Parallèlement à ces recherches, la communauté arithmétique des ordinateurs a également montré que ce constat pouvait également s'appliquer **sur circuits dédiés ASIC** [Wol02] [ST03] **ou programmables de type FPGA** [OP01] [MMM04b] [DMKP05] [CDM05] [SPV06] [MSB+07] [BCM+07] [OBPV08] [GACG08] [GP08]. Dans ce chapitre,

[21] http ://www.ecrypt.eu.org/ebats/.

nous présentons une nouvelle implantation d'unité arithmétique (UA) pour courbes elliptiques sur circuits FPGA. L'objectif avoué de cette démarche est de surpasser les performances des différents cryptosystèmes présents dans la littérature, notamment en terme de temps de calcul. Notre UA ayant également vocation à être insérée dans des environnements à fortes contraintes industrielles (par exemple, les cartes à puces), celle-ci doit être également la plus compacte possible. Dans un second temps, nous la protégerons contre les attaques connues grâce à l'état de l'art (cf. chapitre 4).

5.1 Unités arithmétiques pour l'ECC présentes dans la littérature

Cette section décrit plusieurs architectures d'UA pour l'ECC présentes dans la littérature implantées soit sur FPGA, soit sur ASIC. Nous pourrons ainsi comparer objectivement les performances de notre UA avec celles présentes dans l'état de l'art.

Cette section peut être divisée en deux parties distinctes. Tout d'abord, nous détaillons le concept « d'architecture systolique » qui est utilisé par certaines implantations matérielles d'UA pour l'ECC de l'état de l'art. Ensuite, nous détaillons les architectures d'UA pour l'ECC que nous considérerons comme des références lorsqu'il sera temps d'effectuer des comparaisons avec notre UA. Nous pouvons classer les UA en trois catégories principales, selon qu'elles soient utilisables pour effectuer des calculs :

- seulement sur les corps finis premiers ($\mathbb{F}_p$) pour l'ECC,
- à la fois sur $\mathbb{F}_p$ et sur les corps finis binaires ($\mathbb{F}_{2^m}$) pour l'ECC,
- ou bien seulement sur $\mathbb{F}_p$ à la fois pour l'ECC et le RSA (version multiplicative de l'ECC).

Dans ce qui suit, les parties des différentes architectures de l'état de l'art dédiées à $\mathbb{F}_{2^m}$ et au RSA ne seront pas décrites, car notre UA est uniquement destinée à effectuer des calculs pour l'ECC sur $\mathbb{F}_p$. Notons également qu'à moins d'une mention contraire, les UA présentées dans cette section implantent l'algorithme du « doublement-et-addition » pour calculer la multiplication scalaire (cf. Algorithme 9).

5.1.1 Architectures systoliques : définition, avantages et inconvénients

Une architecture systolique est ainsi définie dans [AFQ83] : c'est un système parallèle spécialisé, fait de cellules construites sur quelques modèles simples, celles-ci étant connectées de façon régulière et locale et dans lequel les calculs effectués utilisent à la fois la notion de *pipeline* et de parallélisme et sont exécutés de façon synchrone.

Une telle architecture amène parfois quelques inconvénients, comme par exemple une augmentation du temps de calcul global de l'architecture ainsi qu'une augmentation de sa surface par rapport à sa version non-systolique. De plus, la rigidité d'une architecture systolique n'est pas toujours compatible avec la souplesse d'utilisation requise.

Cependant, son utilisation entraîne des avantages importants. Par exemple, les cellules de base sont souvent facile à concevoir. De plus, l'insertion d'étages de *pipeline* entre cellules adjacentes permet généralement d'augmenter la fréquence de fonctionnement et le débit de l'ensemble de l'architecture. Il faut enfin préciser que les algorithmes de multiplication modulaire dans lesquels il est effectué une série d'additions et de décalages, comme par exemple l'algorithme de Montgomery (cf. Algorithme 8), peuvent être facilement exprimés sous une forme permettant une implantation systolique. Pour toutes ces raisons, un grand nombre d'articles proposent des versions de multiplications modulaires systoliques initialement destinées au RSA [Eve90] [IMI92a] [IMI92b] [EW93] [Wal93] [SV93] [Kor94] [Oru95] [Tio98] [Blu99] [SHCW99] [Wal99] [Wal00] [TSW00] [TT01] [FP02]. Cependant, des versions systoliques d'algorithmes sont également implantées dans certaines ALUs pour l'ECC présentes dans la littérature [OP01] [OBPV08].

5.1.2 Unités arithmétiques pour l'ECC utilisables uniquement pour $\mathbb{F}_p$

Nous commençons notre tour d'horizon des UA pour l'ECC par celles qui sont uniquement utilisables pour effectuer des calculs sur $\mathbb{F}_p$.

D'après nos sources bibliographiques, la première UA pour l'ECC calculant sur $\mathbb{F}_p$ est proposée par Orlando *et al.* [OP01]. Cette UA, comme d'ailleurs la plupart des autres UA présentes de l'état de l'art [ST03] [MMM04b] [DMKP05] [CDM05] [SPV06] [BCM+07] [OBPV08] [GACG08] contient un multiplieur de Montgomery (cf. Algorithme 8). Le principal avantage de ce type de multiplieur a déjà été évoqué précédemment (cf. section 1.2.1.2) : c'est notamment son implantation matérielle simple (notamment lorsque la base β est petite) qui le rend vraiment intéressant par rapport aux autres algorithmes de multiplications modulaires, puisqu'elle peut se résumer en une série d'additions et de décalages. Cette UA contient, en plus d'un multiplieur de Montgomery, un additionneur ainsi qu'un grand ensemble de registres, ces trois types de composants pouvant travailler en parallèle sur différentes données. Le multiplieur implanté est une version particulière de la multiplication de Montgomery dont l'efficacité repose sur le précalcul de valeurs fréquemment utilisées au cours de la multiplication. Les auteurs implantent une version particulière de la méthode introduite par Orup [Oru95] permettant le calcul rapide de multiplications modulaires lorsque le multiplieur est traité dans une grande base, en utilisant le recodage de Booth (cf. section 1.1.3.1). Une partie du multiplieur a une architecture systolique (cf. section 5.1.1). Aucun détail n'est donné sur l'additionneur implanté. Quant au grand ensemble de registres instanciant les BRAM du FPGA (cf. annexe), il a deux rôles essentiels : il permet non seulement de mémoriser les valeurs précalculées dans le cadre du recodage de Booth, mais également d'implanter la version additive de la méthode d'exponentiation rapide nécessitant des précalculs sur le point de base P introduite dans [BGMW92]. L'inversion modulaire peut être effectuée à l'aide du petit théorème de Fermat (cf. section 1.2.1.3).

McIvor *et al.* [MMM04b] proposent quant à eux une version particulière d'un algorithme de multiplication de Montgomery proposée initialement par Koç *et al.* [KAJ96].

Cet algorithme était à l'origine destiné à des implantations logicielles : il consiste à découper des multiplications modulaires en une série d'additions et de multiplications dont les opérandes sont de taille adaptée à la cible technologique choisie. Les auteurs montrent que l'implantation (matérielle) d'un tel algorithme peut être également performante sur FPGA : ils utilisent 256 multiplieurs 18×18 et les lignes rapides dédiées aux propagations de retenue des Virtex-2 (cf. annexe) pour effectuer les différentes additions/soustractions et multiplications requises. Il faut noter que cette UA n'implante que l'opération de multiplication modulaire.

Öztürk *et al.* [OSS04] proposent quant à eux d'utiliser une arithmétique dite « à l'échelle » initialement proposée par Walter [Wal92], qui consiste à multiplier le modulo p ayant une forme particulière par un facteur d'échelle s dans le but d'effectuer toutes les opérations modulaires modulo $m = p.s$, où m a une forme spéciale autorisant notamment le calcul rapide des inversions modulaires. L'algorithme d'inversion modulaire proposé étant finalement très efficace, les opérations de points peuvent s'effectuer en coordonnées affines, ce qui permet notamment d'économiser un grand nombre de registres par rapport à l'utilisation de coordonnées projectives. Toutes les opérations sont effectuées en utilisant la représentation CS (cf. section 1.1.2.2), ce qui conduit à l'implantation d'une UA ayant un petit chemin critique, et donc une grande fréquence de fonctionnement. Les auteurs proposent un algorithme de réduction modulaire adapté à la forme particulière de m ainsi qu'à la représentation CS. Enfin, la multiplication modulaire est effectuée par additions-décalages et réductions croisées (cf. Algorithme 7).

Afin d'éviter de calculer les opérations coûteuses que sont les inversions modulaires (cf. section 1.2.1.3) à chaque addition ou doublement de points (cf. section 2.2), les coordonnées projectives peuvent être utilisées (cf. section 2.2.2). Or, si l'inversion implantée est rapide, alors il peut être envisagé de conserver les coordonnées affines. C'est le raisonnement que propose d'appliquer Daly *et al.* [DMKP05]. La plupart des UA pour l'ECC proposent d'effectuer des inversions modulaires à l'aide du petit théorème de Fermat (cf. section 1.2.1.3). Or, d'autres méthodes d'inversion sont plus rapides (cf. section 1.2.1.3) : à cet effet, nous pouvons par exemple citer les algorithmes utillisant des calculs de PGCD ou encore l'inversion de Montgomery. C'est cette dernière méthode d'inversion que Daly *et al.* [DMKP05] proposent d'implanter. Leur UA est conçue pour calculer l'opération $\left(\frac{A.B}{C}\right)$: $(A.B)$ est implanté à l'aide d'un algorithme de Montgomery non-systolique, et l'inverse modulaire de C est trouvé à l'aide de l'inversion de Montgomery. L'opération générique $\left(\frac{A.B}{C}\right)$ peut être utilisée pour calculer la division, l'inversion, le carré et la multiplication modulaire. Il faut noter qu'exécuter une telle opération générique entraîne un surcoût important, puisque cela entraîne l'implantation du matériel nécessaire au calcul à la fois d'une multiplication et d'une division modulaire. Les additions et les soutractions modulaires sont exécutées en complément à deux (cf. section 1.1.1.1). D'autres opérations sont disponibles, comme le calcul de l'opposé ($-A \pmod p$), et la multiplication par deux ($2A \pmod p$).

Byrne *et al.* [BCM+07] proposent quant à eux d'implanter un processeur pour l'ECC qui utilise la méthode de chaîne d'additions proposée par Méloni [Mel07a] pour effectuer

la multiplication scalaire. L'utilisation de cette méthode de calcul de $[k]P$ comporte au moins deux avantages. Tout d'abord, cette dernière est plus rapide que l'algorithme basique du « doublement-et-addition » (cf. Algorithme 9). Ensuite, les chaînes d'additions sont une protection possible contre les attaques SSCA (cf. section 4.1.2.1). Ils proposent également différentes versions de leur processeur dans lesquelles un grand nombre d'UA sont implantées afin de pouvoir paralléliser des calculs modulaires. Les multiplications modulaires sont effectuées à l'aide de l'algorithme de Montgomery (cf. Algorithme 8), et les additions modulaires sont effectuées en complément à deux (cf. section 1.1.1.1). Il faut noter que le processeur utilise des coordonnées projectives, mais il n'effectue pas d'inversion modulaire avant de donner le résultat de la multiplication scalaire, ce qui sous-entend que le processeur communique à l'extérieur le point résultat Q en coordonnées projectives. Or, nous devons rappeler qu'appliquer cette politique peut mettre en péril la sécurité du cryptosystème (cf. section 4.1.2.2, et plus particulièrement l'attaque de Naccache *et al.* [NSS04]).

Örs *et al.* [OBPV08] proposent également une UA pour l'ECC, dans laquelle la multiplication de Montgomery est implantée pour calculer l'opération de multiplication modulaire. Ce multiplieur a une architecture systolique (cf. section 5.1.1). Les additions et les soustractions modulaires sont exécutées en complément à deux (cf. section 1.1.1.1). Les inversions modulaires sont calculées à l'aide du petit théorème de Fermat (cf. section 1.2.1.3).

Ghosh *et al.* [GACG08] proposent une architecture d'UA dans laquelle quatre opérations modulaires peuvent s'exécuter en parallèle : l'addition, la soustraction, la multiplication et l'inversion. Le but avoué d'une telle démarche est de protéger cette UA contre les attaques SSCA, plus particulièrement les attaques PA (cf. section 3.1.2). L'addition et la soustraction modulaire se calculent en complément à deux (cf. section 1.1.1.1). La multiplication modulaire est effectuée par additions-décalages et réductions croisées (cf. Algorithme 7). À ce sujet, peu de détails sont donnés sur la manière dont sont réduits modulo p (cf. étapes 4 et 6 de l'algorithme 7) les résultats des différents décalages (cf. étape 3 de l'algorithme 7) et additions (cf. étape 5 de l'algorithme 7). L'inversion modulaire est effectuée à l'aide de méthodes utillisant des calculs de PGCD (cf. section 1.2.1.3). Vu que cette implantation d'inversion modulaire est rapide, les différentes opérations de points peuvent s'effectuer en coordonnées affines.

Enfin, Güneysu *et al.* [GP08] utilisent les BDSP contenus dans les Virtex-4 (cf. annexe) afin de pouvoir effectuer des opérations modulaires efficacement. Ainsi, les additionneurs (respectivement les opérateurs « multiplication/accumulation ») des BDSP sont instanciés afin de pouvoir effectuer un algorithme d'addition (respectivement de multiplication) modulaire rapide. L'UA utilise un modulo p ayant les propriétés des nombres de Mersenne généralisés afin d'accélérer la réduction modulaire (cf. section 1.2.2.3). Notons que l'opération d'inversion modulaire n'est pas implantée par les auteurs. Enfin, les ressources en BRAM du FPGA sont également utilisées, notamment pour accélérer les accès mémoires.

5.1.3 Unités arithmétiques pour l'ECC utilisables pour $\mathbb{F}_p$ et $\mathbb{F}_{2^m}$

La section précédente a détaillé des UA pour l'ECC présentes dans la littérature utilisables **uniquement** pour $\mathbb{F}_p$. Or, d'autres UA autorisent d'effectuer des calculs modulaires **à la fois** sur les corps finis premiers ($\mathbb{F}_p$) **et sur les corps finis binaires** (notés $\mathbb{F}_{2^m}$). L'intérêt porté sur ces derniers est dû essentiellement au fait que l'addition dans $\mathbb{F}_{2^m}$ se résume à un XOR bit-à-bit des opérandes : il n'y a donc aucune propagation de retenues, et cette propriété permet ainsi l'implantation d'une arithmétique rapide. Il peut paraître intéressant pour un industriel de proposer une UA permettant d'effectuer des calculs sur $\mathbb{F}_p$ et sur $\mathbb{F}_{2^m}$ afin de pouvoir couvrir un plus grand nombre de courbes elliptiques cryptographiquement fortes [SEC00b] : les UA en question sont ainsi plus flexibles que leurs homologues utilisables uniquement sur un seul type de corps fini. Historiquement, la recherche dans ce domaine s'est tout d'abord porté sur la conception d'opérateurs arithmétiques pouvant calculer la multiplication modulaire à la fois sur $\mathbb{F}_p$ et sur $\mathbb{F}_{2^m}$ [STK00] [Gro01], puis ce processus s'est généralisé aux UA pour l'ECC dans leur ensemble [Wol02] [ST03] [SPV06].

Wolkerstorfer [Wol02] décrit une architecture d'UA de ce type adaptée à des applications pour lesquelles la consommation électrique doit être minimale. Pour accélérer les calculs sur $\mathbb{F}_p$, la représentation CS est choisie (cf. section 1.1.2.2). Les additions/soustractions modulo p de deux opérandes A et B s'effectuent en prenant en compte le signe de $(A \pm B)$. La multiplication modulaire de deux opérandes A et B est effectuée à l'aide d'une méthode par additions-décalages et réductions croisées (cf. Algorithme 7). Ces étapes de réduction modulaire (cf. étapes 4 et 6 de l'algorithme 7) sont réalisées à l'aide d'estimations successives du quotient q mis en jeu dans l'opération $r = AB - qp \equiv AB \pmod{p}$ (pour illustration, rappelons que l'algorithme de Barrett est un algorithme de réduction modulaire basé sur ce principe, cf. Algorithme 5). L'inversion modulaire est effectuée à l'aide de méthodes utillisant des calculs de PGCD (cf. section 1.2.1.3). D'autres opérations sont disponibles, comme l'incrémentation ou le décalage d'une opérande d'un bit vers la droite ou vers la gauche. Au final, cette UA peut fonctionner à des fréquences élevées, et elle dispose également d'une architecture régulière, ce qui facilite son implantation notamment sur circuits dédiés ASIC.

Satoh *et al.* [ST03] proposent quant à eux une architecture d'UA pouvant fonctionner à la fois sur $\mathbb{F}_p$ et sur $\mathbb{F}_{2^m}$, dans laquelle la multiplication modulaire sur $\mathbb{F}_p$ s'effectue à l'aide d'une version particulière d'un algorithme de multiplication de Montgomery proposé initialement par Koç *et al.* [KAJ96]. Cette version est d'ailleurs celle utilisée par McIvor *et al.* [MMM04b] (l'UA proposée par ces derniers a déjà été décrite dans la section 5.1.2). Notons qu'il n'y a aucun additionneur modulaire implanté dans l'UA proposée par Satoh *et al.* : cela rend ainsi difficile son utilisation dans le cadre de l'ECC.

Plus récemment, Sakiyama *et al.* [SPV06] proposent d'implanter matériellement l'opération

$$T = A.B \pmod{p} \pm C.$$

La multiplication modulaire $A.B \pmod{p}$ est effectuée à l'aide de l'algorithme de Mont-

gomery (cf. Algorithme 8). Les différentes additions nécessaires au calcul de la multiplication de Montgomery sont effectuées à l'aide de « cellules 4 donne 2 » (cf. Figure 5.1). Les additions modulaires utilisent également ces cellules, de même que les soustractions

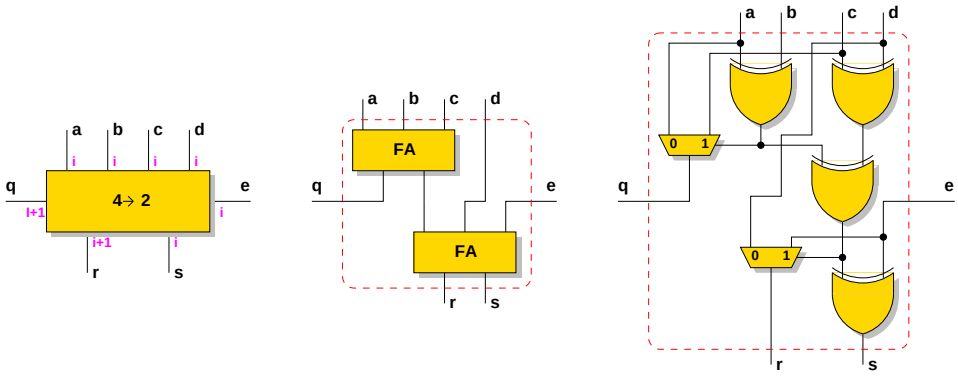

FIG. 5.1 – Cellule 4 donne 2.

modulaires, pour lesquelles C est représenté en complément à deux (cf. section 1.1.1.1). Aucun détail n'est donné sur l'implantation de l'opération d'inversion modulaire.

5.1.4 Unités arithmétiques utilisables pour l'ECC et pour le RSA

Bien que l'ECC soit théoriquement un candidat crédible pour remplacer le RSA dans le cadre de la cryptographie à clé publique [Van04], la popularité de ce dernier fait qu'il continue à être implanté dans les produits de sécurité, et il est probable que cette tendance persiste dans les prochaines années. C'est pour cette raison que certains auteurs proposent des architectures d'UA utilisables à la fois pour l'ECC et pour le RSA.

Crowe *et al.* [CDM05] proposent une UA supportant les calculs sur $\mathbb{F}_p$ à la fois pour l'ECC et pour le RSA. Les multiplications modulaires sont effectuées à l'aide de l'algorithme de Montgomery (cf. Algorithme 8). Les additions et les soutractions modulaires sont exécutées en complément à deux (cf. section 1.1.1.1). Les inversions modulaires sont calculées à l'aide du petit théorème de Fermat (cf. section 1.2.1.3).

Quant à l'article de Mentens *et al.* [MSB+07], il propose une UA utilisable d'une part sur $\mathbb{F}_p$ et $\mathbb{F}_{2^m}$ pour l'ECC, et d'autre part pour le RSA. Cette UA tire profit des capacités du FPGA (de la famille Virtex-2) : les multiplications modulaires sont implantées en instanciant des multiplieurs dédiés du FPGA, et environ 1Mb de BRAM est utilisée (cf. annexe). Le petit théorème de Fermat est utilisé pour calculer les inversions modulaires (cf. section 1.2.1.3). Notons qu'aucun détail n'est donné sur l'implantation de l'opération d'addition/soustraction modulaire.

5.2 Choix des différents paramètres d'implantation et de l'arithmétique modulaire utilisée

La section précédente a présenté l'état de l'art sur les circuits implantant les UA. Cette étude bibliographie nous permet de justifier les différents choix effectués dans le cadre de la conception de notre UA, et c'est justement ces choix qui sont détaillés dans cette section. Nous pouvons classer ces choix en deux catégories : ceux concernant les différents paramètres d'implantation et ceux concernant l'arithmétique modulaire.

5.2.1 Choix des différents paramètres d'implantation

Dans cette section, il est justifié les différents choix que nous avons effectués pour les différents paramètres d'implantation. Nous pouvons classer parmi ces derniers la représentation des nombres choisie, le modulo choisi pendant tout le calcul de la multiplication scalaire, et les opérations modulaires que nous avons choisies d'implanter dans notre UA.

5.2.1.1 Représentation des nombres choisie et conséquences

Nous pouvons constater qu'il y a essentiellement deux représentations des nombres implantées dans les UA de la littérature : la numération simple de position (cf. section 1.1.1.1) [OP01] [ST03] [MMM04b] [DMKP05] [CDM05] [BCM+07] [MSB+07] [OBPV08] [GACG08] [GP08] et la représentation à retenues conservées (CS, cf. section 1.1.2.2) [Wol02] [OSS04] [SPV06]. Ainsi, à notre connaissance, il n'y aucune UA pour l'ECC utilisant la représentation à retenues conservées **signées** (BS, cf. section 1.1.2.2). C'est pour cette raison que nous décidons d'explorer les possibilités offertes par cette dernière : notre but est de montrer qu'une UA dédiée à l'ECC implantant la notation BS peut également apporter un haut niveau de performances, et peut convenir à des environnements fortement contraints en espace.

5.2.1.2 Modulo choisi pendant tout le calcul de la multiplication scalaire

Après avoir adopté une représentation des nombres, il faut également choisir le modulo qui sera mis en jeu lors des calculs modulaires, et ce, pendant toute la multiplication modulaire.

Vu les performances reconnues de la multiplication de Montgomery (cf. section 1.2.1.2), nous décidons de l'implanter afin de pouvoir calculer des multiplications modulaires. Rappelons qu'à l'étape 7 de l'algorithme 8, une soustraction est nécessaire afin de ramener la variable S dans l'intervalle $[0; p[$. Plus précisément, si $A < p$ et $B < p$, la sortie S vérifie $S < 2p$: par conséquent, si $S \geqslant p$, on doit soustraire p à S une fois, et S peut ainsi être réutilisé comme entrée d'une prochaine multiplication modulaire de Montgomery. Or, cette soustraction conditionnelle, en plus d'être coûteuse matériellement, amène de la vulnérabilité au cryptosystème vis-à-vis des attaques SSCA (cf. section 4.1.1.2, et plus particulièrement [WT01] [Wal04b] [ST06]). Ainsi, pour des raisons de performance et de

sécurité, il conviendrait de supprimer cette étape de soustraction. Ce raisonnement a déjà été appliqué par Blum [Blu99], Walter [Wal02c], Gueron [Gue02] et Batina *et al.* [BM02], et nous proposons de faire la synthèse de ces différents travaux dans cette section.

Si nous voulons que les variables A, B et S aient le même intervalle de définition, de telle sorte qu'à la fin du calcul d'une multiplication de Montgomery, la sortie S puisse être réutilisée comme entrée d'une nouvelle multiplication de Montgomery. Cela signifie concrètement que

$$A < k_1 p, B < k_1 p \text{ et } S < k_1 p \tag{5.1}$$

où k_1 est un nombre entier. Rappelons que le résultat de la multiplication de Montgomery s'écrit

$$S = \frac{AB + mp}{\beta^n},$$

où la variable m vérifie $m < \beta^n$. Considérant $\beta^n > k_2 p$, où k_2 est un entier, nous pouvons finalement écrire :

$$S = \frac{AB + mp}{\beta^n} = \frac{AB}{\beta^n} + \left(\frac{m}{\beta^n}\right) p < \left(\frac{k_1^2}{k_2}\right) p + p = p\left(\frac{k_1^2}{k_2} + 1\right). \tag{5.2}$$

Précisons également que nous voulons une UA ayant la plus petite surface occupée possible. Or, si nous faisons croître le paramètre k_1, la taille des variables intermédiaires va augmenter, et par conséquent notre opérateur arithmétique va lui-même occuper une plus grande surface. C'est pour cette raison que nous choisissons le paramètre k_1 le plus petit possible. Notons que le choix $k_1 = 1$ n'est pas souhaitable : cela conduirait à la non-coïncidence des intervalles de définition de S décrit dans les relations 5.1 (dans laquelle $S < p$) et 5.2 (dans laquelle $S < 2p$). En revanche, si nous prenons $k_1 = 2$, l'inégalité 5.2 devient :

$$S < p\left(\frac{4}{k_2} + 1\right).$$

Ainsi, si $k_2 = 4$, alors $S < 2p$: par conséquent, ce résultat S peut ensuite être réutilisé comme entrée d'une prochaine multiplication de Montgomery. Finalement, $\beta^n > 4p > 2^{n+2}$, ce qui conduit à exécuter $n + 2$ itérations (au lieu de n initialement).

Tous ces éléments permettent de concevoir un algorithme de multiplication modulaire de Montgomery sans soustraction conditionnelle par p (cf. Algorithme 22).

5.2.1.3 Opérations disponibles et méthode de contrôle de celles-ci

Cette section justifie notre choix concernant les opérations calculables par notre UA dans le cadre de l'ECC, ainsi que la méthode de contrôle de celles-ci.

Tout d'abord, nous avons décidé d'implanter les trois opérations modulaires indispensables pour l'ECC que sont la multiplication modulaire ($A \times B \pmod{2p}$, avec $A \neq B$), l'addition modulaire ($A + B \pmod{2p}$, avec $A \neq B$), et la soustraction modulaire ($A - B \pmod{2p}$, avec $A \neq B$). Il est également important d'implanter l'inversion modulaire ($\frac{1}{X}$

Algorithme 22 Multiplication modulaire de Montgomery sans soustraction finale en base deux

Entrées : $p = (p_{n-1} \cdots p_1 p_0)_2$, $\text{pgcd}(p, 2) = 1$, $A = (a_n a_{n-1} \cdots a_1 a_0)_2 < 2p$, $B = (b_n b_{n-1} \cdots b_1 b_0)_2 < 2p$.

Sorties : $S = \text{MMM2}(A, B, 2p) = A.B.2^{-(n+2)} \pmod{2p}$

1. $S \leftarrow 0$
2. **pour** $i = 0$ à $n + 1$ **faire**
3. $\quad m_i \leftarrow s_0 \oplus a_i.b_0$
4. $\quad S \leftarrow (S + a_i.B + m_i.p)/2$
5. **fin pour**
6. **retourner** S

(mod p)), notamment pour éviter de communiquer à l'extérieur le point résultat de la multiplication scalaire $Q = [k]P$ en coordonnées projectives, et ainsi éviter l'attaque de Naccache *et al.* [NSS04] (cf. section 4.1.2.2). De plus, pour effectuer des opérations de points en coordonnées projectives (cf. section 2.2.2.1), nous avons également besoin d'implanter des opérations modulaires moins générales, afin d'améliorer l'efficacité globale de notre UA. L'exemple 2.1. justifie ainsi l'implantation du carré modulaire ($A^2 \pmod{2p}$), de la multiplication modulaire par 2 ($2 \times A \pmod{2p}$) et de la multiplication modulaire par le paramètre de courbe $a = -3$ ($-3 \times A \pmod{2p}$). Enfin, si notre UA est destinée à calculer des formules unifiées d'addition de points (cf. section 4.1.1.2), alors elle doit être capable le cas échéant d'effectuer un cube modulaire [BJ02] [DBJ04] ($A^3 \pmod{2p}$).

Ainsi, pour résumer, notre UA est capable de calculer huit opérations modulaires différentes. Nous devons maintenant expliquer comment notre UA est contrôlée afin de pouvoir exécuter cet ensemble de huit instructions. En général, pour contrôler le déroulement de cet ensemble, deux méthodes peuvent être choisies. La première consiste à implanter une machine d'états qui contrôle l'exécution d'opérations spécifiques, telles que celles que nous avons décrites précédemment, mais également d'autres plus complexes comme par exemple un doublement ou une addition de points ou encore une multiplication scalaire complète. Le principal problème amené par cette approche est qu'elle n'amène pas à l'UA une bonne flexibilité. Par exemple, si nous devons modifier le fonctionnement de notre UA de telle sorte qu'elle doive effectuer une nouvelle opération modulaire (par exemple, $A^4 \pmod{2p}$), ou une nouvelle opération de points (par exemple, le triplement − cf. section 2.2.4.1), il va donc falloir modifier l'architecture matérielle de la machine d'états, ce qui implique en réalité la création d'un circuit nouveau. À la place de cette première méthode, nous avons préféré implanter le contrôle de notre UA à l'aide d'un microcode mémorisé dans des blocs de mémoire morte (*Read-Only Memory Blocks*, BROMs) du FPGA. Une telle approche a déjà été proposée par Leong *et al.* [LL02] et Byrne *et al.* [BCM+07]. Les deux avantages principaux amenés par cette démarche sont un temps de développement de l'UA réduit, ainsi qu'une flexibilité accrue. En effet, en choisissant cette méthode, cela devient plus facile de mettre à jour l'ensemble des instructions exécutables par notre UA car il n'est pas nécessaire de recompiler l'ensemble de son architecture.

5.2.2 Choix concernant l'arithmétique modulaire

Une fois définis les différents paramètres 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 par notre UA.

5.2.2.1 Addition modulaire

Notre UA doit être capable d'effectuer l'opération d'addition modulaire $A + B \pmod{2p}$. En effet, nous nous servirons de ce calcul fondamental pour effectuer certaines opérations modulaires décrites dans la section 5.2.1.3, à savoir :

- la soustraction modulaire ($A - B \pmod{2p}$) grâce à $A - B \pmod{2p} = A + (-B) \pmod{2p}$,
- la multiplication modulaire par 2 ($2 \times A \pmod{2p}$) grâce à $2 \times A \pmod{2p} = A + A \pmod{2p}$,
- la multiplication modulaire par -3 ($-3 \times A \pmod{2p}$) grâce à $-2 \times A - A \pmod{2p}$.

Rappelons que nous avons choisi d'effectuer toutes ces opérations dans la représentation BS (cf. section 5.2.1.1). Or, cette dernière permet d'effectuer efficacement l'addition modulaire de deux opérandes A et B (cf. section 1.1.2.2, et plus particulièrement [TY92]). En effet, le calcul de $A + B \pmod{2p}$ peut être décomposé en deux étapes.

La première étape consiste à effectuer l'addition des opérandes $A = (a_{n+1}a_n \cdots a_0)_{BS}$ et $B = (b_{n+1}b_n \cdots b_0)_{BS}$, avec $-2p < A < 2p$, $-2p < B < 2p$ et $2^n \leq 2p < 2^{n+1}$. Le résultat $T = (t_{n+2}t_{n+1}t_n \cdots t_0)_{BS}$ satisfait la relation $T \equiv A + B \pmod{2p}$ et $-4p < T < 4p$.

La seconde étape est une correction dans le but d'obtenir un nombre $S = (s_n s_{n-1} \cdots s_0)_{BS}$ qui satisfait la relation $S \equiv T \pmod{2p}$, et $-2p < S < 2p$.

Tout d'abord, la valeur $tv = 4t_{n+2} + 2t_{n+1} + t_n$ qui utilise les trois chiffres de poids fort de T doit être calculée. Ensuite, selon la valeur de tv, trois cas se présentent :

- si $tv < 0$, alors on a $-4p < T < 0$, donc il faut calculer $S = T + 2p$,
- si $tv > 0$, alors on a $0 < T < 4p$, donc il faut calculer $S = T - 2p$,
- si $tv = 0$, alors on a $-2^n < T < 2^n$, soit encore $-2p < T < 2p$, donc il faut calculer $S = T + 0$.

Pour comprendre le dernier cas de figure, il faut utiliser les relations $2^n \leq 2p < 2^{n+1}$ et $-2^n < T < 2^n$ afin de construire l'inégalité

$$-2^{n+1} < -2p \leq -2^n < T < 2^n \leq 2p < 2^{n+1}. \tag{5.3}$$

Ainsi, on peut tirer de l'inégalité 5.3 que $-2p < T < 2p$. Par conséquent, il n'est pas nécessaire d'additionner $\pm 2p$ à T pour obtenir $-2p < S < 2p$. L'algorithme 23 détaille toute cette procédure.

Si nous comparons l'algorithme 23 prévu pour pouvoir calculer des additions modulaires modulo $2p$ avec son homologue qui exécute ces opérations modulo p (cf. Algorithme 4), nous pouvons constater qu'ils présentent seulement deux différences notables. La première différence réside dans les variables mises en jeu dans le calcul de tv, celles-ci étant

Algorithme 23 Addition modulaire de Takagi *et al.* modulo $2p$

Entrées : $2^n \leq 2p < 2^{n+1}$, $-2p < A = (a_{n+1}a_n \cdots a_0)_{BS} < 2p$, $-2p < B = (b_{n+1}b_n \cdots b_0)_{BS} < 2p$
Sorties : $S = A + B \pmod{2p}$, avec $S = (s_{n+1}s_n \cdots s_0)_{BS}$

1. $(T^+, T^-) \leftarrow \text{BSA}[(A^+, A^-), (B^+, B^-)]$
2. **si** $tv = 4t_{n+2} + 2t_{n+1} + t_n < 0$ **alors**
3. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (2p, 0)]$
4. **sinon si** $tv > 0$ **alors**
5. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (0, 2p)]$
6. **sinon si** $tv = 0$ **alors**
7. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (0, 0)]$
8. **finsi**
9. **retourner** S

décalées d'un chiffre vers la gauche : en effet, dans l'algorithme 4 (respectivement l'algorithme 23), il est calculé $tv = 4t_{n+1} + 2t_n + t_{n-1}$ ($tv = 4t_{n+2} + 2t_{n+1} + t_n$). La seconde différence est l'ajout conditionnel de $2p$ dans l'algorithme 23 au lieu de p dans l'algorithme 4. Nous pouvons donc constater que l'utilisation du modulo $2p$ ne rend pas plus difficile l'implantation matérielle de l'addition modulaire de Takagi *et al.* [TY92], puisque les modifications expliquées précédemment sont équivalentes à de simples décalages vers la gauche.

5.2.2.2 Multiplication modulaire

Notre UA doit être capable d'effectuer l'opération de multiplication modulaire $A \times B \pmod{2p}$. Ce faisant, nous pourrons calculer certaines opérations modulaires décrites dans la section 5.2.1.3, à savoir :

- le carré modulaire ($A^2 \pmod{2p}$) grâce à $A \times A \pmod{2p}$,
- le cube modulaire ($A^3 \pmod{2p}$) grâce à $A^2 \times A \pmod{2p}$.

L'algorithme de multiplication modulaire de Montgomery sans soustraction finale (cf. Algorithme 22) doit être modifié afin de pouvoir utiliser la représentation BS : l'algorithme 24 prend en compte tous ces changements. Pour obtenir ce nouvel algorithme, nous commençons tout d'abord par réécrire l'étape 3 de l'algorithme 22 :

$$\begin{aligned}
m_i &= [s_0 + a_i.b_0] \pmod 2 \\
&= [(s_0^+ - s_0^-) + (a_i^+ - a_i^-).(b_0^+ - b_0^-)] \pmod 2 \\
&= [(s_0^+ - s_0^-) \pmod 2 + [(a_i^+ - a_i^-).(b_0^+ - b_0^-)] \pmod 2] \pmod 2 \\
&= [(s_0^+ \oplus s_0^-) + [(a_i^+ - a_i^-) \pmod 2.(b_0^+ - b_0^-) \pmod 2] \pmod 2] \pmod 2 \\
&= [(s_0^+ \oplus s_0^-) + (a_i^+ \oplus a_i^-).(b_0^+ \oplus b_0^-)] \pmod 2 \\
&= (s_0^+ \oplus s_0^-) \oplus (a_i^+ \oplus a_i^-).(b_0^+ \oplus b_0^-).
\end{aligned}$$

Cette nouvelle valeur de m_i est calculée à l'étape 5 de l'algorithme 24.

À l'étape 6 de l'algorithme 22, quatre valeurs possibles peuvent être additionnées à S pour rendre de ce dernier pair : 0, B, p or $B + p$. Or, il faut faire en sorte que l'addition mise en jeu dans la nouvelle version de l'algorithme de Montgomery (cf. Algorithme 24) supporte la représentation BS. C'est pour cela qu'entre les étapes 6 et 18 de l'algorithme 24, six valeurs possibles peuvent être additionnées à S : 0, B, $-B$, p, $U = (B + p)$, $V = (-B + p)$. Cette addition garantit que S est pair à la fin de l'étape 18. Dans le cas général, cela veut dire que $(s_0^+,s_0^-) = (0,0)$ ou $(1,1)$. Or, il faut noter que le résultat d'une addition BS implique obligatoirement que $s_0^- = 0$. Ainsi, seul le cas $(s_0^+,s_0^-) = (0,0)$ est possible, et par conséquent, S^+ et S^- sont tous les deux pairs, et peuvent ainsi être décalés tous les deux vers la droite afin d'effectuer la division par deux (étape 18 de l'algorithme 24). Précisons enfin que les valeurs U et V nécessaires le cas échéant pour rendre la variable S paire sont précalculées aux étapes 1 et 2 de l'algorithme 24, et que la variable S est initialisée à l'étape 3 de ce même algorithme.

Algorithme 24 Multiplication modulaire de Montgomery sans soustraction finale utilisant la représentation à retenues conservées signées

Entrées : $p = (p_{n-1} \cdots p_1 p_0)_2$, $\text{pgcd}(p, 2) = 1$, $-2p < A = (a_{n+1} a_n \cdots a_0)_{BS} < 2p$, $-2p < B = (b_{n+1} b_n \cdots b_0)_{BS} < 2p$
Sorties : $(S^+, S^-) = (A^+, A^-).(B^+, B^-).2^{-(n+2)} \pmod{2p}$

1. $(U^+, U^-) \leftarrow \text{BSA}[(B^+, B^-), (p, 0)]$
2. $(V^+, V^-) \leftarrow \text{BSA}[(B^-, B^+), (p, 0)]$
3. $(S^+, S^-) \leftarrow \text{BSA}[(0, 0), (0, 0)]$
4. **pour** $i = 0$ à $n + 1$ **faire**
5. $\quad m_i \leftarrow (a_i^+ \oplus a_i^-).(b_0^+ \oplus b_0^-)$
6. $\quad$ **si** $(a_i^+, a_i^-, m_i) = (0, 0, 0)$ ou $(1, 1, 0)$ **alors**
7. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (0, 0)]$
8. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (1, 0, 0)$ **alors**
9. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (B^+, B^-)]$
10. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 1, 0)$ **alors**
11. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (B^-, B^+)]$
12. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 0, 1)$ ou $(1, 1, 1)$ **alors**
13. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (p, 0)]$
14. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (1, 0, 1)$ **alors**
15. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (U^+, U^-)]$
16. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 1, 1)$ **alors**
17. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (V^+, V^-)]$
18. $\quad$ **finsi**
19. $\quad (S^+, S^-) \leftarrow (S^+/2, S^-/2)$
20. **fin pour**
21. **retourner** (S^+, S^-)

5.2.2.3 Inversion modulaire

Notre UA doit également être capable de calculer l'opération d'inversion modulaire (rappelons que l'inverse de X modulo p s'écrit $X^{-1} = 1/X \pmod{p}$) afin de pouvoir exprimer le point résultat en coordonnées affines à la fin de la multiplication scalaire.

Rappelons qu'il existe trois méthodes principales d'inversion modulaire sur $\mathbb{F}_p$: le petit théorème de Fermat, les algorithmes utilisant le calcul du PGCD de deux nombres et l'inversion de Montgomery (cf. section 1.2.1.3). Cependant, pour pouvoir utiliser ces deux dernières méthodes ayant un temps d'exécution plus rapide que la première, il faut être capable de calculer des comparaisons. Or, puisque nous utilisons la représentation BS pour effectuer tous nos calculs, la comparaison de deux variables est une opération coûteuse à implanter.

C'est pour cela que nous avons choisi d'implanter le petit théorème de Fermat pour pouvoir calculer des inversions modulaires, puisque c'est une méthode qui ne nécessite aucune comparaison de variables. Elle implique juste le calcul d'une exponentiation modulaire (l'inverse de X s'écrivant $X^{-1} = X^{p-2} \pmod{p}$, si pgcd($X$,$p$) = 1 et avec p premier), cette dernière pouvant s'effectuer à l'aide d'une série de multiplications modulaires grâce à la version multiplicative de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) également appelée algorithme du « carré-et-multiplication » (cf. Algorithme 25).

Algorithme 25 Algorithme du « carré-et-multiplication » pour appliquer le petit théorème de Fermat

Entrées : p, $0 \leq X < p$, $E = (p-2) = (e_{n-1} \cdots e_1 e_0)_2$
Sorties : $X^E \pmod{p}$

1. $S \leftarrow X$
2. **pour** $i = n-1$ à 0 **faire**
3. $\quad S \leftarrow S \times S \pmod{p}$
4. $\quad$ **si** $e_i = 1$ **alors**
5. $\quad\quad S \leftarrow S \times X \pmod{p}$
6. $\quad$ **finsi**
7. **fin pour**
8. **retourner** S

Il faut noter que l'utilisation de cette méthode entraîne le calcul de $(\ell + \mathrm{HW}[(p-2)])M$, où $\mathrm{HW}[(p-2)]$ représente le poids de Hamming de $(p-2)$. Or, si les nombres de Mersenne généralisés mis en jeu dans les courbes standardisées sont utilisés (cf. section 2.3.2.2), cela peut conduire à un $\mathrm{HW}[(p-2)]$ proche de ℓ, et donc à un temps de calcul de l'inversion modulaire proche de $2\ell M$ (la seule exception notable est p_{256}, pour lequel $\mathrm{HW}[(p-2)] = \ell/2$, et par conséquent le temps de calcul de l'inversion est de l'ordre de $(3\ell/2)M$). Nous pouvons cependant rétorquer que notre UA peut accepter n'importe quelle valeur de p : rien n'empêche donc un concepteur de circuits de trouver un p ayant un faible poids de Hamming afin de diminuer le temps de calcul de l'inversion, et qui obéit aux contraintes de sécurité relatives à l'ECC (cf. tableau 2.8).

5.3 Notre unité arithmétique détaillée

Dans la section précédente, nous avons motivé nos choix concernant l'arithmétique modulaire qui sera implantée dans notre UA. Dans cette section, nous donnons plus de détails sur cette dernière : nous commençons par expliquer les dernières optimisations des algorithmes que nous avons effectuées avant l'implantation finale, puis nous décrivons l'architecture et le fonctionnement de notre UA.

5.3.1 Optimisations des algorithmes choisis

Nous voulons obtenir une UA ayant non seulement un temps de calcul minimal mais également une surface la plus compacte possible afin de pouvoir être insérée dans des environnements à fortes contraintes, comme par exemple les cartes à puces. Ainsi, si nous observons les algorithmes 23 et 24, nous pouvons imaginer concevoir une UA implantant toutes les opérations modulaires à l'aide d'un seul additionneur à retenues conservées signées. Cet additionneur effectuera l'opération

$$(C^+, C^-) \leftarrow \mathrm{BSA}[(A^+, A^-), (B^+, B^-)],$$

dans laquelle (A^+, A^-), (B^+, B^-) et (C^+, C^-) varieront suivant l'opération modulaire en cours. Il suffira donc d'implanter en amont de cet additionneur deux multiplexeurs (notés MUX1 et MUX2), chacun permettant de sélectionner un opérande adapté au calcul effectué parmi tous ceux possibles. Dans ce qui suit, les entrées de MUX1 (respectivement de MUX2) représentent toutes les valeurs possibles que peuvent prendre l'opérande (A^+, A^-) (respectivement (B^+, B^-)).

Si nous faisons un bilan des variables utilisées dans les algorithmes 23 et 24, nous pouvons remarquer que les valeurs 0, p et $2p$ coexistent parmi les entrées du MUX2. Or, si nous réécrivons ces algorithmes de telle sorte que la valeur 0 disparaisse, cela contribuera à diminuer la taille de MUX2, ainsi que potentiellement son chemin critique. C'est dans cet esprit que nous utilisons le caractère redondant de la représentation BS : dans l'algorithme 23 (respectivement 24), nous nous servons de $(2p, 0) = (4p, 2p)$, $(0, 2p) = (2p, 4p)$ et de $(0, 0) = (p, p)$ ($(p, 0) = (2p, p)$) afin d'éliminer la valeur 0 comme entrée possible du MUX2. Les algorithmes 26 et 27 prennent en compte ces ultimes optimisations (ces dernières étant indiquées en caractères gras).

5.3.2 Architecture de notre unité arithmétique

Notre UA est représentée en figure 5.2. Elle est composée de sept éléments appelés MUX1, MUX2, SELECTOR, MUX3, BSA, SHIFT, et DEMUX. Ses trois entrées sont (A^+, A^-), (B^+, B^-) et (p, p), et sa sortie est (OUT^+, OUT^-).

MUX1 est un multiplexeur qui sélectionne parmi les paires d'entrées (A^+, A^-), (B^+, B^-), (S^+, S^-) et (p, p) le couple de variables adapté à l'opération en cours selon la valeur de sa commande notée cmd_1 (codée sur deux bits). Il communique également au

Algorithme 26 Version de l'addition modulaire de Takagi *et al.* implantée dans notre unité arithmétique

Entrées : $2^n \leq 2p < 2^{n+1}$, $-2p < A = (a_{n+1}a_n \cdots a_0)_{BS} < 2p$, $-2p < B = (b_{n+1}b_n \cdots b_0)_{BS} < 2p$
Sorties : $S = A + B \pmod{2p}$, avec $S = (s_{n+1}s_n \cdots s_0)_{BS}$
1. $(T^+, T^-) \leftarrow \text{BSA}[(A^+, A^-), (B^+, B^-)]$
2. **si** $tv = 4t_{n+2} + 2t_{n+1} + t_n < 0$ **alors**
3. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (\mathbf{4p}, \mathbf{2p})]$
4. **sinon si** $tv > 0$ **alors**
5. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (\mathbf{2p}, \mathbf{4p})]$
6. **sinon si** $tv = 0$ **alors**
7. $\quad (S^+, S^-) \leftarrow \text{BSA}[(T^+, T^-), (\mathbf{p}, \mathbf{p})]$
8. **finsi**
9. **retourner** S

multiplexeur MUX2 la valeur des trois chiffres de poids fort de S permettant de calculer la valeur de tv (cf. étape 2 de l'algorithme 26).

MUX2 est un multiplexeur qui sélectionne parmi les paires d'entrées (B^+, B^-), (U^+, U^-), (V^+, V^-) et (p, p) le couple de variables adapté à l'opération en cours. Pour effectuer son choix, il se sert de la valeur de la variable tv transmise par MUX1 lorsque la commande notée cmd_2 (codée sur deux bits) lui indique qu'une addition modulaire est en cours de calcul. De même, il utilise la valeur de la variable m_i transmise par SELECTOR lorsque cmd_2 lui indique qu'une multiplication modulaire est en cours de calcul.

Il faut noter que, pour déterminer le signe de tv, nous avons implanté la méthode suivante. Nous scrutons ses chiffres en commençant par celui ayant le poids le plus fort pour aller ensuite vers celui ayant le poids le plus faible, de façon à trouver le premier chiffre non-nul qui indique le signe de tv : si le premier chiffre non-nul est égal à (1,0) (respectivement (0,1)), cela veut dire que la variable tv est positive (négative). De même, nous utilisons le fait que $tv = 0$ lorsque ses trois chiffres sont nuls, c'est-à-dire lorsqu'ils égaux soit à (0,0) soit à (1,1).

SELECTOR n'est présent que pour implanter l'étape 5 de l'algorithme 27. Ses entrées sont donc (b_0^+, b_0^-) et (a_i^+, a_i^-), ce dernier couple étant fourni à l'aide d'un registre à décalage. SELECTOR se résume à deux portes XORs deux entrées et à une porte AND deux entrées.

MUX3 sert à permuter le cas échéant les deux composantes d'une même opérande : ainsi, une entrée des entrées de MUX3 notée $(MUX1^+, MUX1^-)$ peut devenir $(MUX1^-, MUX1^+)$ lorsque la commande cmd_3 (codée sur deux bits) a une valeur particulière. Cette permutation peut permettre de calculer une soustraction à l'aide d'un BSA : si l'on veut effectuer la soustraction de la sortie de MUX1 par celle de MUX2, on peut effectuer $\text{BSA}[(MUX1^+, MUX1^-), (MUX2^-, MUX2^+)]$. MUX3 dispose de trois comportements différents suivant la valeur de cmd_3 : soit les deux

Algorithme 27 Version de la multiplication modulaire de Montgomery implantée dans notre unité arithmétique

Entrées : $p = (p_{n-1} \cdots p_1 p_0)_2$, $\text{pgcd}(p, 2) = 1$, $-2p < A = (a_{n+1} a_n \cdots a_0)_{BS} < 2p$, $-2p < B = (b_{n+1} b_n \cdots b_0)_{BS} < 2p$

Sorties : $(S^+, S^-) = \text{MMM3}(A, B, 2p) = (A^+, A^-).(B^+, B^-).2^{-(n+2)} \pmod{2p}$

1. $(U^+, U^-) \leftarrow \text{BSA}[(B^+, B^-), (\mathbf{2p}, \mathbf{p})]$
2. $(V^+, V^-) \leftarrow \text{BSA}[(B^-, B^+), (\mathbf{2p}, \mathbf{p})]$
3. $(S^+, S^-) \leftarrow \text{BSA}[(\mathbf{p}, \mathbf{p}), (\mathbf{p}, \mathbf{p})]$
4. **pour** $i = 0$ à $n + 1$ **faire**
5. $\quad m_i \leftarrow (a_i^+ \oplus a_i^-).(b_0^+ \oplus b_0^-)$
6. $\quad$ **si** $(a_i^+, a_i^-, m_i) = (0, 0, 0)$ ou $(1, 1, 0)$ **alors**
7. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (\mathbf{p}, \mathbf{p})]$
8. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (1, 0, 0)$ **alors**
9. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (B^+, B^-)]$
10. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 1, 0)$ **alors**
11. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (B^-, B^+)]$
12. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 0, 1)$ ou $(1, 1, 1)$ **alors**
13. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (\mathbf{2p}, \mathbf{p})]$
14. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (1, 0, 1)$ **alors**
15. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (U^+, U^-)]$
16. $\quad$ **sinon si** $(a_i^+, a_i^-, m_i) = (0, 1, 1)$ **alors**
17. $\quad\quad (S^+, S^-) \leftarrow \text{BSA}[(S^+, S^-), (V^+, V^-)]$
18. $\quad$ **finsi**
19. $\quad (S^+, S^-) \leftarrow (S^+/2, S^-/2)$
20. **fin pour**
21. **retourner** (S^+, S^-)

couples $(MUX1^+, MUX1^-)$ et $(MUX2^-, MUX2^+)$ restent inchangés, soit seules les composantes de $(MUX2^-, MUX2^+)$ sont permutées, soit les composantes des deux couples sont permutées.

BSA effectue l'addition à retenues conservées signées des sorties de MUX3 (cf. Figure 1.3).

SHIFT n'est présent que pour implanter l'étape 19 de l'algorithme 27 : selon la valeur du bit de commande cmd_4, ce composant décale ou non les composantes de la sortie du BSA.

DEMUX est un démultiplexeur : selon la valeur de la commande cmd_5 (codée sur deux bits), ce composant choisit la destination de ses entrées parmi ses quatre sorties possibles : (S^+, S^-), (U^+, U^-), (V^+, V^-) ou (OUT^+, OUT^-).

Précisons enfin que treize registres sont implantés dans notre UA afin de mémoriser les opérandes des calculs effectués ((A^+, A^-), (B^+, B^-) et (p, p)), les variables intermédiaires nécessaires ((S^+, S^-), (U^+, U^-), (V^+, V^-)) ainsi que sa sortie ((OUT^+, OUT^-)), et deux

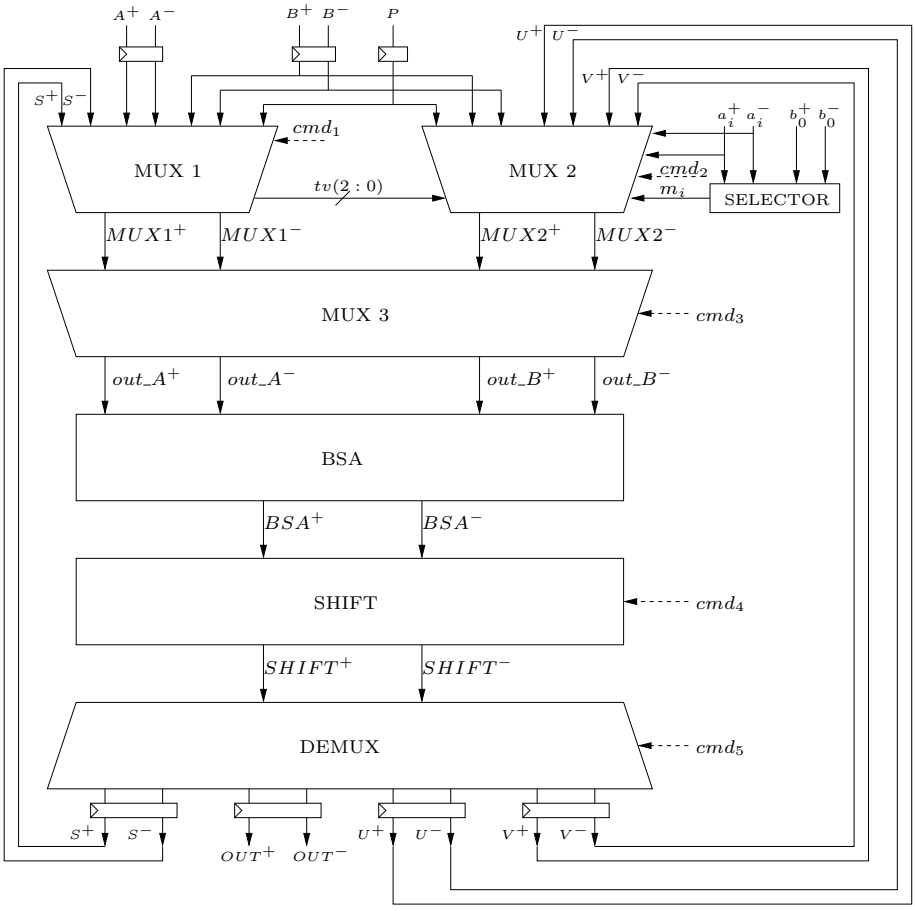

FIG. 5.2 – L'architecture de notre unité arithmétique

registres à décalage sont placés en amont de SELECTOR afin de lui fournir à chaque cycle d'horloge la valeur courante de a_i^+ et de a_i^-.

5.3.3 Exécution des différentes opérations modulaires par notre unité arithmétique

Cette section détaille la manière dont notre UA exécute les différentes opérations modulaires décrites dans la section 5.2.2.

5.3.3.1 Addition modulaire

Une addition modulaire ($A + B \pmod{2p}$) est calculée en deux cycles d'horloge. Lors du premier cycle d'horloge, MUX1 et MUX2 sélectionnent respectivement les couples (A^+,A^-) et (B^+,B^-), MUX3 n'effectue aucune permutation, et une première addition est calculée par BSA (BSA$[(A^+, A^-), (B^+, B^-)]$). SHIFT ne décale pas le résultat fourni

par BSA, et DEMUX envoie le résultat sur sa sortie (S^+,S^-). Lors du second cycle d'horloge, MUX1 sélectionne le couple (S^+,S^-) et MUX2 choisit la valeur qui doit être *a posteriori* additionnée à ce dernier selon la valeur de la variable tv. MUX3 n'effectue aucune permutation, et une seconde addition est calculée par BSA. SHIFT ne décale pas le résultat fourni par BSA, et DEMUX envoie finalement le résultat sur sa sortie (OUT^+,OUT^-).

La multiplication modulaire par 2 ($2 \times A \pmod{2p}$) s'effectue suivant la même procédure en imposant $A = B$.

La soustraction modulaire ($A - B \pmod{2p}$) utilise également la même procédure, excepté le fait que lors du premier cycle d'horloge, MUX3 permute les composantes de l'opérande B.

Enfin, la multiplication modulaire par -3 ($-3 \times A \pmod{2p}$) s'effectue en imposant $A = B$ dans notre UA, et en exécutant $-2 \times A - A \pmod{2p} = -2 \times A \pmod{2p} - A \pmod{2p}$. Cette dernière opération s'effectue en deux étapes.
La première étape est consacrée au calcul de $-2 \times A \pmod{2p}$. Lors du premier cycle d'horloge, MUX1 et MUX2 sélectionnent respectivement les couples (A^+,A^-) et (B^+,B^-) (rappelons que dans ce cas précis $A = B$), MUX3 permute les composantes des deux opérandes, et une première addition est calculée par BSA. SHIFT ne décale pas le résultat fourni par BSA, et DEMUX envoie le résultat sur sa sortie (S^+,S^-). Lors du second cycle d'horloge, MUX1 sélectionne le couple (S^+,S^-) et MUX2 choisit la valeur qui doit être *a posteriori* additionnée à ce dernier selon la valeur de la variable tv. MUX3 n'effectue aucune permutation, et une seconde addition est calculée par BSA. SHIFT ne décale pas le résultat fourni par BSA, et DEMUX envoie le résultat sur sa sortie (S^+,S^-). À la fin de ce processus, nous avons obtenu le résultat de $-2 \times A \pmod{2p}$: il reste maintenant dans une seconde étape à soustraire à ce résultat A afin d'obtenir $-3A \pmod{2p}$.
Lors du troisième cycle d'horloge, MUX1 et MUX2 sélectionnent respectivement les couples (S^+,S^-) et (B^+,B^-). MUX3 ne permute que les composantes du second couple avant d'effectuer une autre addition grâce au BSA. SHIFT ne décale pas le résultat fourni par BSA, et DEMUX envoie le résultat sur sa sortie (S^+,S^-). Lors du quatrième et dernier cycle d'horloge, le but est de réduire cette dernière modulo $2p$. C'est pour cela que MUX1 sélectionne le couple (S^+,S^-) et MUX2 choisit la valeur qui doit être *a posteriori* additionnée à ce dernier selon la valeur de la variable tv. MUX3 n'effectue aucune permutation, et une dernière addition est calculée par BSA. SHIFT ne décale pas le résultat fourni par BSA, et DEMUX envoie finalement le résultat sur sa sortie (OUT^+,OUT^-).

Pour résumer, la méthode d'addition modulaire implantée ($A + B \pmod{2p}$) peut être utilisée pour calculer la multiplication modulaire par 2 ($2 \times A \pmod{2p}$) et la soustraction modulaire ($A - B \pmod{2p}$) en deux cycles d'horloge, et pour exécuter la multiplication modulaire par -3 ($-3 \times A \pmod{2p}$) en quatre cycles d'horloge.

5.3.3.2 Multiplication modulaire

Une multiplication modulaire ($A \times B \pmod{2p}$) est calculée en ($n+5$) cycles d'horloge.
Les trois premiers cycles d'horloge sont consacrés au calcul des variables U, V ainsi

qu'à l'initialisation de la variable S à 0. Pour ce faire, MUX1 et MUX2 choisiront les opérandes adéquats, MUX3 n'agira que lors du calcul de V pour permuter les composantes de B (cf. étape 2 de l'algorithme 27), SHIFT ne décalera aucun des résultats des différentes additions, et DEMUX enverra sur les registres adéquats ces différents résultats.

Ensuite vient le temps du calcul des étapes de l'algorithme 27 allant de 4 à 20, du quatrième cycle d'horloge au $(n+5)^{\text{ème}}$. À chaque cycle d'horloge, une nouvelle valeur de m_i est calculée par SELECTOR. Cette valeur est envoyée à MUX2 (de même que celle de (a_i^+, a_i^-)) afin que ce dernier sélectionne la valeur à additionner à la variable S (qui est elle-même sélectionnée par MUX1). MUX3 n'exécute une permutation que lors de l'étape 11 de l'algorithme 27. SHIFT effectue un décalage d'un bit vers la droite des deux composantes résultats de l'addition BSA, et DEMUX envoie la sortie de SHIFT sur le registre S, excepté lors du dernier cycle d'horloge où il sélectionne la sortie de notre UA (OUT^+,OUT^-).

5.3.3.3 Inversion modulaire

Rappelons que nous avons choisi d'implanter le petit théorème de Fermat pour pouvoir calculer des inversions modulaires. Cette méthode implique le calcul d'une exponentiation modulaire, opération qui peut elle-même s'effectuer à l'aide d'une série de multiplications modulaires grâce à la version multiplicative de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) décrite dans l'algorithme 25. Il suffira donc à l'utilisateur de notre UA de la programmer afin qu'elle exécute une série de multiplications modulaires déterminée par la valeur fixe et connue de $E = p - 2$.

5.3.4 Exécution d'une multiplication scalaire par notre unité arithmétique

Après avoir détaillé l'exécution des différentes opérations modulaires calculables par notre UA, nous pouvons maintenant expliquer comment utiliser ces dernières pour exécuter une multiplication scalaire ($Q = [k]P$). Nous devons d'emblée préciser que nous nous sommes uniquement attachés dans nos travaux à optimiser l'implantation matérielle de notre UA, et à protéger cette dernière contre les attaques par observation et par perturbation. Plus précisément, nous ne sommes pas occupés d'implanter l'ensemble des registres et la logique de contrôle associée nécessaires au calcul de la multiplication scalaire dans son ensemble. De même, nous précisons qu'à la fin de la multiplication scalaire, notre UA fournit à l'extérieur les coordonnées du point résultat Q en représentation BS, et non en numération simple de position. Là encore, nous n'avons pas implanté une méthode ou un composant permettant de convertir un nombre de la représentation BS vers la numération simple de position, car différentes solutions sont proposées dans la littérature pour effectuer ces conversions.

En préambule, nous devons tout d'abord expliciter une importante contrainte d'utilisation de l'algorithme 27. En effet, ce dernier nécessite l'emploi d'une représentation particulière : lorsqu'on veut l'utiliser pour effectuer la multiplication modulo $2p$ de deux

opérandes A et B, alors le résultat S s'écrit :

$$(S^+, S^-) = \text{MMM3}(A, B, 2p) = (A^+, A^-).(B^+, B^-).2^{-(n+2)} \pmod{2p}.$$

Or, l'apparition du paramètre additionnel $2^{-(n+2)}$ peut être *a priori* un problème, notamment lorsque plusieurs opérations modulaires se succèdent.

Pour composer avec ce paramètre, nous appliquons la méthode suivante. Tout d'abord, nous plaçons les opérandes A et B mises en jeu dans une suite d'opérations modulaires dans une représentation particulière, appelée représentation de Montgomery (une opérande A devient ainsi $\tilde{A}$ dans cette représentation), en effectuant :

$$\begin{array}{rclcrcl} \tilde{A} & = & \text{MMM3}(A, 2^{(2n+4)}, 2p) & = & A.2^{(2n+4)}.2^{-(n+2)} \pmod{2p} & = & A.2^{(n+2)} \pmod{2p}, \\ \tilde{B} & = & \text{MMM3}(B, 2^{(2n+4)}, 2p) & = & B.2^{(2n+4)}.2^{-(n+2)} \pmod{2p} & = & B.2^{(n+2)} \pmod{2p}. \end{array}$$

Ensuite, tant que les opérandes sont dans cette représentation, nous pouvons exécuter des additions et des multiplications modulaires grâce aux propriétés suivantes [Mon85] :

$$\begin{cases} \tilde{A} \pm \tilde{B} \pmod{2p} = \left[A.2^{(n+2)} \pmod{2p}\right] \pm \left[B.2^{(n+2)} \pmod{2p}\right] = [A \pm B] .2^{(n+2)} \pmod{2p} \\ \text{MMM3}(\tilde{A}, \tilde{B}, 2p) = [A.2^{(n+2)}].[B.2^{(n+2)}].2^{-(n+2)} \pmod{2p} = A.B.2^{(n+2)} \pmod{2p}. \end{cases}$$

En d'autres termes, nous utilisons le fait que le résultat de l'addition modulaire ou de la multiplication de Montgomery de deux opérandes utilisant la représentation de Montgomery fournissent un résultat dans la même représentation. Enfin, lorsque la série d'opérations modulaires est terminée, il faut éliminer le facteur résiduel $2^{(n+2)}$ contenu dans le résultat $\tilde{S}$ de cette série. Ceci peut être fait grâce à

$$S \;=\; \text{MMM3}(\tilde{S}, 1, 2p) \;=\; S.2^{(n+2)}.2^{-(n+2)} \pmod{2p} \;=\; S \pmod{2p}.$$

Une fois ce préambule terminé, nous pouvons maintenant détailler l'exécution d'une multiplication scalaire à l'aide de notre UA. Elle peut se diviser en sept étapes distinctes.

1. Il faut commencer par convertir les coordonnées x_P et y_P du point P qui sont communiquées à notre UA en numération simple de position (cf. section 1.1.1.1) vers la représentation BS (cf. section 1.1.2.2). C'est une opération gratuite à effectuer. En effet, il faudra juste appliquer $x_P = (x_P^+, x_P^-) = (x_P, 0)$ et $y_P = (y_P^+, y_P^-) = (y_P, 0)$.
2. Avant de débuter le calcul de la multiplication scalaire, les coordonnées du point P initialement communiquées en coordonnées affines ($P = (x_P, y_P)$) doivent être converties en coordonnées projectives afin d'éviter de calculer des inversions modulaires à chaque addition ou doublement de points (cf. section 2.2.2). Par exemple, P doit être exprimé sous la forme $P = (X_P : Y_P : Z_P)$ lorsque le système de coordonnées $\mathcal{P}$ est utilisé (cf. tableau 2.2 pour consulter la forme projective du point P dans les autres systèmes). Là encore, le coût de cette opération est limité, puisqu'il faut effectuer lorsque $\mathcal{P}$ est choisi : $X_P = x_P, Y_P = y_P$ et $Z_P = 1$.
3. Conformément à la remarque préliminaire de cette section, nous avons décidé d'effectuer tous nos calculs en utilisant la représentation de Montgomery. Par consé-

quent, X_P et Y_P doivent être convertis dans cette représentation avant de débuter le calcul de la multiplication scalaire :

$$\begin{array}{rcrcl} \tilde{X_P} & = & \text{MMM3}(X_P, 2^{(2n+4)}, 2p) & = & X_P.2^{(2n+4)}.2^{-(n+2)} \pmod{2p} \\ & & & = & X_P.2^{(n+2)} \pmod{2p}, \\ \tilde{Y_P} & = & \text{MMM3}(Y_P, 2^{(2n+4)}, 2p) & = & Y_P.2^{(2n+4)}.2^{-(n+2)} \pmod{2p} \\ & & & = & Y_P.2^{(n+2)} \pmod{2p}. \end{array}$$

4. Une fois que les coordonnées du point de base P utilisent la représentation BS, celle de Montgomery, et sont converties en coordonnées projectives, nous pouvons effectuer la multiplication scalaire proprement dite. Rappelons que des algorithmes de multiplication scalaire sécurisés contre les attaques SSCA (respectivement non-sécurisés) sont détaillés dans la section 4.1.2.1 (dans la section 2.2.3).

5. À la fin de la multiplication scalaire, le point résultat $Q = (\tilde{X}_Q : \tilde{Y}_Q : \tilde{Z}_Q) = [k]P$ doit être converti en coordonnées affines, ne serait-ce que pour ne pas communiquer à l'extérieur Q en coordonnées projectives, et donc pour se prémunir de l'attaque mise en évidence par Naccache *et al.* (cf. section 4.1.2.2, et plus particulièrement [NSS04]). Par exemple, si le système de coordonnées projectives $\mathcal{P}$ est utilisé, il faut effectuer $\tilde{x}_Q = \text{MMM3}(\tilde{X}_Q, \tilde{Z}_Q^{-1}, 2p)$ et $\tilde{y}_Q = \text{MMM3}(\tilde{Y}_Q, \tilde{Z}_Q^{-1}, 2p)$ (cf. tableau 2.2 pour consulter les transformations à effectuer dans les autres systèmes de coordonnées projectives). Dans ce cas précis, une inversion modulaire et deux multiplications de Montgomery sont nécessaires.

6. Une fois tous les calculs modulaires terminés, $\tilde{x}_Q$ et $\tilde{y}_Q$ peuvent quitter la représentation de Montgomery. Conformément à la remarque placée en préambule de cette section, cela peut être fait grâce à :

$$\begin{array}{rcrcrcl} x_Q & = & \text{MMM3}(\tilde{x}_Q, 1, 2p) & = & x_Q.2^{(n+2)}.2^{-(n+2)} \pmod{2p} & = & x_Q \pmod{2p}, \\ y_Q & = & \text{MMM3}(\tilde{y}_Q, 1, 2p) & = & y_Q.2^{(n+2)}.2^{-(n+2)} \pmod{2p} & = & y_Q \pmod{2p}. \end{array}$$

   Il faut noter que, suite à ces deux multiplications de Montgomery, $x_Q < p$ et $y_Q < p$ [BM02]. Autrement dit, nous sommes assurés que x_Q et y_Q sont communiqués à l'extérieur modulo p.

7. Enfin, une conversion de x_Q et de y_Q de la représentation BS vers la numération simple de position doit être effectuée. Un procédé de conversion trivial consiste à effectuer $x_Q = x_Q^+ - x_Q^-$ à $y_Q = y_Q^+ - y_Q^-$ à l'aide d'un additionneur à retenue propagée, mais cela n'est pas très efficace car cela implique d'une part l'ajout d'un additionneur dans notre UA (ce qui est coûteux matériellement) et d'autre part un temps de calcul proportionnel à la taille des opérandes. Heureusement, des algorithmes permettant la conversion à la volée d'un nombre représenté en BS vers le complément à deux (cf. section 1.1.1.1) existent, et conduisent à un ajout de matériel minime [Tul86] [EL87]. À l'heure actuelle, nous n'avons pas encore implanté matériellement ces méthodes dans notre UA.

5.4 Résultats de l'implantation matérielle de notre unité arithmétique

Après avoir détaillé l'architecture de notre UA dans la section précédente, nous présentons les résultats de son implantation matérielle. Dans cette section, nous commençons tout d'abord par décrire notre mode opératoire, à savoir le langage de description matériel et l'outil et les options de synthèse utilisés. Puis, nous décrivons deux propriétés de notre UA qui la rend très compétitive par rapport à l'état de l'art : d'une part, son fonctionnement à une fréquence constante, et ce, quelque soit la taille des opérandes implantée, et d'autre part, sa facile mise à l'échelle. Enfin, nous comparons les performances de notre UA avec celles présentes dans l'état de l'art.

5.4.1 Mode opératoire

Dans cette section, nous décrivons le mode opératoire guidant nos expériences. Nous commençons par justifier le choix que nous avons effectué concernant le langage de description matérielle de notre UA. Puis, nous détaillons l'outil de synthèse physique ainsi que les options de ce dernier qui nous permettent de prototyper notre UA et d'évaluer ses performances.

5.4.1.1 Langage de description matérielle de notre UA

Différents langages permettent la description de circuits numériques. Actuellement, VHDL et Verilog constituent les principales alternatives. Nous avons décidé de coder notre UA en VHDL. Ce dernier offre quelques particularités facilitant la réutilisation de composants matériels, comme la généricité. Une fois le codage terminé, la simulation fonctionnelle permet une première vérification du système. Nous utilisons l'outil ModelSim de Mentor Graphics© lors de cette étape.

5.4.1.2 Outils et options de synthèse

Nous utilisons l'environnement de travail dédié aux FPGAs Xilinx©, à savoir ISE, ainsi que l'outil de synthèse intégré à ce dernier nommé XST.

Afin de pouvoir tirer toute la quintessence de la cible technologique choisie (ici, le FPGA), il faut non seulement bien connaître son architecture (cf. annexe), mais également l'outil de synthèse que nous utilisons.

Tout d'abord, puisque les architectures des FPGAs contiennent des LUTs, il est recommandé de décrire matériellement (lorsque cela est possible) la partie opérative de notre UA sous forme d'une ou de plusieurs tables. C'est cette stratégie que nous avons adopté pour décrire la partie de l'architecture de MUX2 dédiée à la recherche du signe de la variable tv dans le cadre du calcul d'une addition modulaire (cf. étape 2 de l'algorithme 26), ainsi que pour décrire BSA. Ensuite, il faut pouvoir utiliser la logique de contrôle efficace des FPGAs (cf. la partie notée « Carry & Control » représentée sur la

Figure 6.7). Pour ce faire, nous commençons tout d'abord par imiter le modèle de description matérielle d'un multiplexeur détaillé dans le manuel d'utilisation de XST [XST02]. Puis, nous utilisons les options de synthèse pour nous assurer que XST instanciera effectivement la partie du FPGA dédiée à la logique de contrôle. C'est ainsi que nous avons pu vérifier après synthèse que MUX1 et MUX2 utilisent bien des multiplexeurs. De plus, il est également possible d'instancier deux registres à décalage rapides pour émuler ceux qui sont chargés de fournir a_i^+ et a_i^- à l'entrée de SELECTOR. Pour ce faire, et comme dans le cas de la logique de contrôle, nous respectons le modèle de description matérielle d'un registre à décalage décrit dans le manuel d'utilisation de XST [XST02], et nous utilisons les options de synthèse. Là encore, cette démarche a été validée expérimentalement puisque nous avons pu vérifier que deux registres à décalage ont bien été instanciés à l'entrée de SELECTOR.

Pour résumer, nous avons décrit précédemment la manière dont peut être implanté efficacement chaque élément pris **indépendamment**. Il est également possible d'augmenter les performances de notre UA **globalement**. C'est ainsi par exemple que nous avons décidé d'autoriser l'outil de synthèse à optimiser notre architecture sans tenir compte de notre hiérarchie VHDL de départ. De même, il a pu être observé expérimentalement que l'outil de synthèse produisait des implantations matérielles de notre UA ayant le meilleur compromis surface occupée / fréquence d'utilisation lorsqu'il a comme priorité d'optimiser intensément cette dernière.

5.4.2 Avantages de notre UA

Après avoir décrit le mode opératoire de nos expériences, cette section présente les deux principaux avantages observables que possèdent notre UA : une fréquence de fonctionnement constante, et ce, quelque soit la taille des opérandes, et une mise à l'échelle facilitée. Cette dernière propriété sera définie en section 5.4.2.2.

5.4.2.1 Une fréquence de fonctionnement quasi-constante quelque soit la taille des opérandes

Nous avons effectué la synthèse de notre UA sur FPGA XCV2000 pour différentes tailles de clés (notées ℓ), et nous avons rassemblé les résultats dans le tableau 5.2. Cela nous permet par la même occasion d'exhiber un autre atout de la description matérielle de notre UA : nous considérons que ℓ (la taille des clés exprimées en nombre de bits) est un paramètre générique (au sens VHDL du terme), qui peut être changé par l'utilisateur de ce circuit avant toute nouvelle synthèse, et ce, sans avoir à modifier la taille des opérandes traitées par chaque composant.

Après une analyse du tableau 5.2, nous pouvons remarquer que notre UA a une fréquence de fonctionnement quasiment constante, et ce, quelque soit la taille des opérandes implantée. Ceci peut être un avantage décisif face aux autres UA présentes dans l'état de l'art. En effet, chaque ligne du tableau 5.2 représente un niveau de sécurité : lorsqu'il sera nécessaire de passer à un niveau de sécurité supérieur, notre UA conservera sa fréquence

ℓ	**Surface (*Slices*)**	**Fréquence**
160	3378	**93 MHz**
192	3909	**87 MHz**
224	4564	**91 MHz**
256	5210	**87 MHz**
384	7722	**91 MHz**
521	10548	**88 MHz**

TAB. 5.2 – Résultats après synthèse de notre unité arithmétique sur FPGA XCV2000

de fonctionnement du niveau inférieur, ce qui n'est pas forcément le cas des autres UA de l'état de l'art.

5.4.2.2 Une mise à l'échelle facilitée

La mise à l'échelle d'un composant (ici, notre UA) consiste essentiellement en son redimensionnement afin qu'il puisse être adapté à la taille des opérandes traitées. Une mise à l'échelle peut être facile ou difficile. Lorsqu'elle est facile (respectivement difficile), cela signifie que lorque la taille des opérandes augmente, les différents résultats observés qui sont dans notre cas la surface occupée et le chemin critique augmentent de manière raisonnable (déraisonnable).

Lorsque l'on veut évaluer l'influence conjointe de deux métriques d'implantation matérielle, il est coutume d'analyser le produit de ces deux dernières : c'est pour cela que nous avons choisi de tracer la courbe représentant l'évolution du produit (nombre de *slices* utilisés $\times$ chemin critique (en ns)) en fonction de la taille des opérandes.

Il s'avère que ce produit varie linéairement avec $\boldsymbol{\ell}$: il est donc plus facile de mettre à l'échelle notre UA que certaines autres présentes dans l'état de l'art implantant un ou des additionneurs à propagation séquentielle de retenue (cf. Figure 1.2) [OP01] [DMKP05] [CDM05] [BCM+07] [GACG08]. En effet, pour ces UAs, nous pouvons supposer que la surface va être une fonction linéaire des opérandes (un additionneur séquentiel sur ℓ bits ayant une architecture conforme à la figure 1.2 utilise ℓ cellules FAs), de même que pour le chemin critique (un additionneur à propagation séquentielle de retenue sur ℓ bits a comme chemin critique celui de ℓ cellules FAs) : dans ce cas, le produit (surface (ou *Slices*) $\times$ chemin critique) va être fonction de $\boldsymbol{\ell^2}$.

En plus d'une fréquence de fonctionnement constante quelque soit la taille des opérandes, la facile mise à l'échelle de notre UA est un avantage supplémentaire à prendre en considération face aux autres UA présentes dans l'état de l'art. En effet, lorsqu'il sera nécessaire d'augmenter la taille des opérandes traitées afin de passer à un niveau de sécurité supérieur, nous pouvons espérer que notre UA reste compétitive, ne serait-ce que face aux UA ayant comme référence bibliographique [OP01] [DMKP05] [CDM05] [BCM+07] [GACG08].

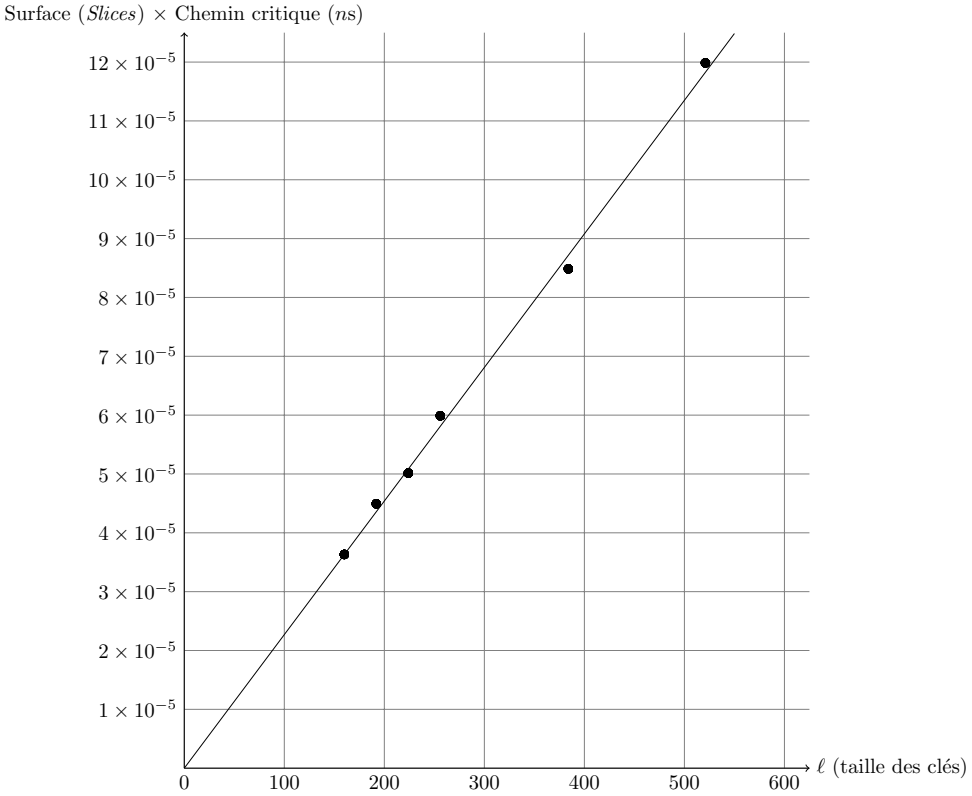

FIG. 5.3 – Illustration de la facile mise à l'échelle de notre unité arithmétique.

5.4.3 Comparaisons

Après avoir présenté quelques avantages intrinsèques de notre UA, nous proposons de comparer les performances de cette dernière avec celles présentes dans l'état de l'art. Ces comparaisons doivent être les plus objectives possibles : c'est pour cela qu'à chaque comparaison, nous avons décidé d'effectuer une synthèse de notre UA en utilisant le même FPGA que celui utilisé par l'UA de la littérature, et évidemment, en considérant la même taille des opérandes.

Nous rapportons les caractéristiques des UA en termes de :

- surface, en donnant le nombre de LUTs, de *slices*, de *Flip-Flops* (FFs), de BRAM, de BDSP et de multiplieurs (notés Mult.) utilisés par le FPGA,
- vitesse, en donnant la fréquence (notée fréq.) et le temps total nécessaire au calcul d'une multiplication scalaire (notée $[k]P$).

5.4.3.1 Comparaisons avec les UA de l'état de l'art non-protégées contre les attaques SSCA

Notre UA est globalement plus performante que celle proposée par Orlando *et al.* (cf. tableau 5.3) : elle utilise moins de LUTs, moins de *Flip-Flops* et elle peut fonctionner à une plus haute fréquence. Nous devons toutefois nuancer en précisant que leur UA dispose

UA	**LUTs**	**FFs**	**BRAM**	**Fréq.**
[OP01]	11416	5735	140 Kb	40 MHz
Notre UA	**3909**	**2327**	**0**	**87 MHz**

TAB. 5.3 – Comparaison entre la solution proposée par Orlando *et al.* et la nôtre sur FPGA XCV1000E avec $\ell = 192$

d'un grand ensemble de registres (qui peuvent servir par exemple à mémoriser les résultats des calculs modulaires impliqués dans une multiplication scalaire) lui permettant d'être un processeur à part entière. Les auteurs [OP01] estiment à **3 *ms*** le temps d'exécution par leur UA de la version additive de la méthode d'exponentiation rapide introduite dans [BGMW92]. Le temps de calcul par notre UA d'une multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{P}$ est de 8,943 *m*s. Or, cet algorithme est environ quatre fois plus lent que celui proposé dans [BGMW92]. D'où l'on en déduit finalement que notre UA mettrait environ **2,236 *ms*** à exécuter cet algorithme : notre UA est donc en tout point plus performante que celle proposée par Orlando *et al.*.

Notre UA est également globalement plus performante que celle proposée par McIvor *et al.* (cf. tableau 5.4) : elle utilise moins de *slices*, et elle dispose d'une fréquence de fonctionnement plus élevée. Cependant, lorsque nous comparons le temps d'exécution

UA	***Slices***	**Mult. 18×18**	**Fréq.**	**[k]P**
[MMM04b]	11992	256	45 MHz	**5,07 *ms***
Notre UA	**4987**	**0**	**168 MHz**	7,53 *ms*

TAB. 5.4 – Comparaison entre la solution proposée par McIvor *et al.* et la nôtre sur FPGA XC2VP avec $\ell = 256$

sur les deux UA d'une multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}$, nous observons que l'architecture proposée par McIvor *et al.* est la plus rapide. Nous pouvons rétorquer que l'UA en question utilise intensivement les ressources du FPGA en multiplieurs 18×18 (cf. section 6.4), et pas la nôtre : il est donc difficile de comparer objectivement les performances de ces deux UA.

L'UA proposée par Daly *et al.* utilise environ deux fois moins de *slices* que la nôtre, mais elle est aussi presque trois fois plus lente (cf. tableau 5.5) : notre UA possède donc un meilleur produit (*Slices* × chemin critique), et donc implique un meilleur compromis surface/temps. De plus, nous pouvons observer que l'exécution par notre UA d'une

UA	***Slices***	**Fréq.**	**[k]P**
[DMKP05]	**1854**	52 MHz	4,235 *m*s
Notre UA	3739	**155 MHz**	**3,477 *m*s**

Tab. 5.5 – Comparaison entre la solution proposée par Daly *et al.* et la nôtre sur FPGA XC2V2000 avec $\ell = 160$

multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}$ est plus rapide que l'exécution par l'UA de Daly *et al.* du même algorithme utilisant des coordonnées affines. Enfin, rappelons que notre UA peut être facilement mise à l'échelle, ce qui n'est pas forcément le cas de celle proposée par Daly *et al.* (cf. section 5.4.2.2). Ainsi, notre UA supporte globalement bien la comparaison avec celle décrite dans [DMKP05].

Il en est de même lorsque nous comparons les performances de notre UA avec celle proposée par Byrne *et al.* (cf. tableau 5.6). Notre UA utilise moins de *slices* et moins de

UA	***Slices***	**BRAM**	**Fréq.**	**[k]P**
[BCM$^+$07]	3735	129,6 Kb	38 MHz	6,43 *m*s
Notre UA	**3124**	**0**	**113 MHz**	**4,18 *m*s**

Tab. 5.6 – Comparaison entre la solution proposée par Byrne *et al.* et la nôtre sur FPGA XC2V6000 avec $\ell = 160$

BRAM (cf. section 6.4) que celle proposée par Byrne *et al.*. Cependant, pour ce qui est de l'utilisation des *slices*, notre propos doit être nuancé puisque l'UA décrite dans [BCM$^+$07] dispose d'un grand ensemble de registres lui permettant d'être considéré comme un processeur à part entière. Nous notons également que notre UA dispose d'une fréquence de fonctionnement plus élevée. Néanmoins, nous devons préciser que Byrne *et al.* fournissent des relevés de performance de leur architecture après le placement-routage de cette dernière, tandis que ceux de notre UA sont donnés avant cette étape. Or, le placement-routage de notre UA aura comme conséquence de diminuer sa fréquence de fonctionnement, ce qui fait que l'écart entre les fréquences de fonctionnement des deux UA va diminuer. De plus, lorsque nous comparons les temps d'exécution par les deux UA d'une multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}$, nous pouvons observer que notre UA est la plus rapide. Enfin, nous conjecturons que l'UA détaillée dans [BCM$^+$07] a des problèmes de mise à l'échelle

(cf. section 5.4.2.2), ce qui n'est pas le cas de la nôtre. Ainsi, pour toutes ces raisons, nous pouvons considérer que notre UA est plus performante que celle décrite par Byrne *et al.*.

L'UA proposée par Örs *et al.* [OBPV08] est en tout point moins performante que la nôtre (cf. tableau 5.7) : elle mobilise plus de *slices*, elle fonctionne à une fréquence moins élevée, et si nous comparons le temps de calcul par les deux UA d'une multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}^m$, nous pouvons constater que notre UA est plus rapide.

UA	***Slices***	**Fréq.**	**[k]P**
[OBPV08]	6055	91 MHz	14,843 ms
Notre UA	**3378**	**93 MHz**	**5,528 ms**

TAB. 5.7 – Comparaison entre la solution proposée par Örs *et al.* et la nôtre sur FPGA XCV1000E avec $\ell = 160$

En revanche, force est de reconnaître que l'UA proposée par Güneysu *et al.* est globalement plus performante que la nôtre (cf. tableau 5.8) : elle utilise moins de *slices*, moins de LUTs, moins de *Flip-Flops* et elle a une fréquence de fonctionnement supérieure (approchant même la fréquence maximale atteignable par un FPGA XC4VFX12-12, à savoir 500 MHz). De plus, si nous comparons le temps de calcul par les deux UA d'une multi-

UA	***Slices***	**LUTs**	**FFs**	**BDSP**	**BRAM**	**Fréq.**	**[k]P**
[GP08]	**1580**	**1825**	**1892**	26	11	**487 MHz**	**452 μs**
Notre UA	4361	3057	8237	**0**	**0**	219 MHz	4,555 ms

TAB. 5.8 – Comparaison entre la solution proposée par Güneysu *et al.* et la nôtre sur FPGA XC4VFX12-12 avec $\ell = 224$

plication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}^c$, nous pouvons constater que leur UA est presque dix fois plus rapide. Nous pouvons cependant expliquer la différence importante de performance existante entre les deux UA par le fait que les auteurs utilisent intensivement les ressources du FPGA en BDSP et en BRAM (cf. section 6.4). Il faut noter également que l'UA proposée dans [GP08] est utilisable uniquement pour effectuer des calculs sur les courbes elliptiques standardisées par le NIST [FIP00]. Plus précisément, le modulo implanté est un nombre de Mersenne généralisé (cf. section 1.2.2.3). Cela a l'avantage d'accélérer les étapes de réduction modulaire, mais l'inconvénient principal d'une telle démarche est que le modulo est fixe, et qu'il ne peut donc pas être changé par l'utilisateur de cette UA : une partie de la flexibilité de cette UA a donc été sacrifiée au profit de sa performance. Contrairement à l'architecture proposée dans [GP08], notre UA fonctionne pour n'importe quel modulo, ce dernier pouvant être choisi par l'utilisateur.

Comparée à l'UA proposée par Sakiyama *et al.* (cf. tableau 5.9), la nôtre mobilise légèrement moins de *slices*. À la décharge de Sakiyama *et al.*, il faut tout de même préciser

UA	***Slices***	**Fréq.**	**[k]P**
[SPV06]	4836	110 MHz	**6,272 *ms***
Notre UA	**4738**	**184 MHz**	7,064 *ms*

TAB. 5.9 – Comparaison entre la solution proposée par Sakiyama *et al.* et la nôtre sur FPGA XC2VP30 avec $\ell = 256$

que leur UA implante également des composants permettant d'effectuer des opérations arithmétiques sur $\mathbb{F}_{2^m}$. De plus, notre UA peut fonctionner à une plus haute fréquence que celle décrite dans [SPV06]. Néanmoins, il faut nuancer ce dernier point, puisque nous n'avons pas effectué le placement-routage de notre UA, contrairement à Sakiyama *et al.* : or, cette étape aura comme conséquence de diminuer la fréquence de fonctionnement de notre UA, et donc l'écart entre les deux fréquences de fonctionnement va également diminuer. Enfin, lorsque nous comparons le temps d'exécution par les deux UA d'une multiplication scalaire à l'aide de l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}^m$, nous pouvons observer que notre UA est un peu plus lente, même si l'écart entre les deux temps d'exécution (6,272 *ms* contre 7,064 *ms*) n'est pas prohibitif. En résumé, notre UA est tout-à-fait compétitive face à celle proposée dans [SPV06].

L'UA proposée par Crowe *et al.* [CDM05] utilise environ 11% moins de *slices* que la nôtre, mais a une fréquence de fonctionnement plus de trois fois inférieure (cf. tableau 5.10), ce qui fait que notre UA dispose d'un meilleur compromis *slices* occupés / chemin critique. De plus, si nous évaluons le temps de calcul sur les deux UA de l'algorithme du

UA	***Slices***	**Fréq.**	**[k]P**
[CDM05]	**5267**	44 MHz	28,72 *ms*
Notre UA	5861	**149 MHz**	**8,724 *ms***

TAB. 5.10 – Comparaison entre la solution proposée par Crowe *et al.* et la nôtre sur FPGA XC2V2000 avec $\ell = 256$

« doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}^c$, nous observons que notre UA calcule plus rapidement cet algorithme.

De même, l'UA proposée par Mentens *et al.* est globalement moins performante que la nôtre (cf. tableau 5.11) : elle consomme plus de *slices* (précisons tout de même que leur UA est un processeur à part entière), elle instancie plus de multiplieurs 16×16 et plus de BRAM (cf. section 6.4), et elle a une fréquence de fonctionnement moins élevée, même si il faut considérer que leur UA est placée-routée tandis que la nôtre n'est « que »

UA	*Slices*	Mult. 16×16	BRAM	Fréq.	[k]P
[MSB+07]	19304	66	1,056 Mb	66 MHz	26,8 ms
Notre UA	**4873**	**0**	**0**	**104 MHz**	**12,5 ms**

TAB. 5.11 – Comparaison entre la solution proposée par Mentens *et al.* et la nôtre sur FPGA XC3S5000 avec $\ell = 256$

synthétisée. Enfin, si nous exécutons sur les deux UA l'algorithme du « doublement-et-addition » (cf. Algorithme 9) sur le système de coordonnées $\mathcal{J}^c$, il est attendu que notre UA soit plus prompte à calculer le point résultat.

En résumé, notre UA est également très compétitive face aux UA détaillées dans [CDM05] et [MSB+07]. Il faut cependant préciser que le fait que ces deux UA soient utilisables dans le cadre de l'ECC **et du RSA** rend difficile la comparaison entre ces deux UA et la nôtre.

5.4.3.2 Comparaisons avec les UA de l'état de l'art protégées contre les attaques SSCA

Le tableau 5.6 a détaillé les performances de l'UA proposée par Byrne *et al.* [BCM+07] exécutant l'algorithme du « doublement-et-addition » (cf. Algorithme 9). Or, cette UA peut être également programmée pour implanter la contre-mesure SSCA proposée par Méloni [Mel07a] consistant à calculer la multiplication scalaire ($Q = [k]P$) à l'aide d'une chaîne d'additions (cf. section 4.1.1). Après placement-routage sur un FPGA XC2V6000, et avec $\ell = 160$, les auteurs estiment qu'il faut **5,14 ms** à leur UA pour pouvoir exécuter une multiplication scalaire par une chaîne d'additions.

De notre côté, nous utilisons les contre-mesures présentes dans la littérature afin de pouvoir protéger notre UA contre les attaques SSCA. À ce sujet, nous pouvons utiliser le tableau 4.5 qui indique que la contre-mesure la moins coûteuse (lorsqu'un seul processeur est utilisé) est celle qui consiste à combiner l'algorithme du « doublement-et-addition » utilisant le $\text{NAF}_2(k)$ (cf. Algorithme 10, en considérant $w = 2$) et le principe d'atomicité de Chevallier-Mames *et al.* [CMCJ04]. Après synthèse sur un FPGA XC2V6000, et avec $\ell = 160$, nous estimons qu'il faut **3,441 ms** à notre UA pour calculer une multiplication scalaire protégée contre la SSCA, ce qui est un temps bien inférieur par rapport à la solution proposée dans [BCM+07].

Enfin, notre solution est également plus intéressante que celle proposée par Ghosh *et al.* lorsqu'un concepteur de circuits sécurisés veut implanter une UA protégée contre les attaques SSCA (cf. tableau 5.12). Notre UA mobilise moins de *slices* (notons tout de même que l'UA décrite dans [GACG08] est un processeur à part entière), et elle fonctionne à une plus haute fréquence. De même, après synthèse sur un FPGA XC3S5000, et avec $\ell = 160$, nous estimons qu'il faut **3,781 ms** à notre UA pour calculer une multiplication scalaire protégée contre la SSCA (en combinant le $\text{NAF}_2(k)$ et l'atomicité décrite dans [CMCJ04]), ce qui est inférieur au temps nécessaire pour appliquer la solution proposée

UA	*Slices*	Fréq.	[k]P
[GACG08]	5289	45 MHz	4 ms
Notre UA	**3073**	**103 MHz**	**3,781 ms**

TAB. 5.12 – Comparaison entre la solution proposée par Ghosh *et al.* et la nôtre sur FPGA XC3S5000 avec $\ell = 160$

dans [GACG08] (**4 ms**). Cependant, de notre point vue, cet écart (3,781 ms contre 4 ms) n'est pas décisif car il faut prendre en compte le fait que les résultats concernant notre UA sont donnés après synthèse, tandis que ceux relatifs à l'UA décrite dans [GACG08] le sont après placement-routage.

5.5 Conclusion et perspectives

Il a été décrit dans ce chapitre une UA performante destinée à l'ECC. À cette occasion, nous avons proposé une nouvelle méthode d'implantation de l'algorithme de multiplication modulaire de Montgomery. Cet algorithme, qui est très souvent utilisé dans la cryptographie à clé publique, voit sa rapidité augmenter grâce à l'utilisation de la représentation à retenues conservées signées.

Au cours de nos travaux, nous avons pu constater toute l'étendue des avantages apportés par cette représentation. Tout d'abord, notre UA dispose d'un chemin critique court, et peut donc **fonctionner à une fréquence élevée** : si nous comparons la fréquence de fonctionnement de notre UA avec l'état de l'art (à cible technologique et à taille d'opérandes égales), nous pouvons observer que seule l'UA décrite dans [GP08] peut fonctionner à une fréquence plus élevée que la nôtre (rappelons que cela s'explique par le fait que cette UA utilise intensivement les BDSP du FPGA, ces derniers fonctionnant à une fréquence élevée). Cette haute fréquence de fonctionnement permet également d'obtenir **un temps de calcul de la multiplication scalaire** (opération principale mise en jeu dans l'ECC) inférieur ou proche des UA les plus rapides de la littérature (exception faite de l'UA décrite dans [GP08]).

De plus, notre UA peut être **facilement mise à l'échelle**, essentiellement grâce au fait que notre UA fonctionne à une fréquence constante, et ce, quelque soit la taille des opérandes. Ceci est un avantage décisif face aux UA de la littérature qui ne possèdent pas cette propriété, notamment lorsqu'il sera nécessaire de passer à un niveau de sécurité supérieur, et donc, lorsqu'il faudra augmenter la taille des opérandes implantées.

Notons également que puisque notre UA utilise le même chemin de données pour effectuer toutes les opérations modulaires, cela conduit à une **architecture compacte**, qui utilise peu de fonctions logiques, et qui est donc adaptée aux environnements fortement contraints en surface (par exemple les cartes à puce).

Une autre contribution importante de nos travaux consiste à proposer **une UA pour l'ECC protégée contre les attaques SSCA dont le temps de calcul de la mul-**

tiplication scalaire est le plus court de l'état de l'art.

Les perspectives de nos travaux sont nombreuses.

Tout d'abord, nous pouvons explorer la possibilité d'implanter **un multiplieur de Montgomery en grande base**, dans lequel les opérandes sont représentés avec un système d'Avizienis (cf. section 1.1.2.1, et plus particulièrement [Avi61]). Cette approche a déjà été testée lorsque ces dernières sont représentées en numération simple de position (cf. section 1.1.1.1) [EW93] [Oru95] [Blu99]. Dans ces dernières références bibliographiques, il a été montré que l'exécution d'une multiplication modulaire dans une base $\beta > 2$ nécessite moins de cycles d'horloge que son homologue dans la base $\beta = 2$, et l'augmentation obtenue du temps de cycle peut être modérée grâce à l'insertion d'étages de *pipeline*. Cependant, il faut noter que la surface occupée par ce type de multiplieur augmente. Peut-être que grâce à l'utilisation de la représentation à retenues conservées signées, nous pouvons obtenir un multiplieur dans une base $\beta > 2$ qui soit d'une part plus rapide que son homologue utilisant une base $\beta = 2$, et qui voit d'autre part sa surface augmenter raisonnablement afin de pouvoir être inséré dans un environnement fortement contraint en surface.

Ensuite, le calcul de l'opération d'inversion modulaire peut également être amélioré. Une piste d'amélioration possible consisterait à adapter l'inversion de Montgomery, ou les algorithmes utilisant des calculs de PGCD de telle manière que ces méthodes puissent manipuler des opérandes représentées en BS. Cela permettrait ainsi d'accélérer le temps de calcul de l'inversion modulaire, et si l'algorithme obtenu est très efficace, alors il pourra même être envisagé de conserver les coordonnées affines pendant l'exécution de la multiplication scalaire.

De plus, il pourrait être envisagé d'utiliser les ressources du FPGA cible en BDSP et en BRAM dans le but notamment d'envisager de combler (au moins partiellement) l'écart de performance existant entre notre UA et celle de Güneysu *et al.* [GP08].

En outre, il faut noter que notre UA ne peut pas être considéré comme un processeur à part entière : il ne dispose pas par exemple d'un ensemble de registres lui permettant de mémoriser les résultats d'opérations modulaires effectuées dans le cadre de la multiplication scalaire. Une autre piste de travail consisterait à doter notre UA d'un tel ensemble (implanté par exemple à l'aide des BRAM du FPGA cible) pour lui permettre de pouvoir exécuter une multiplication scalaire dans sa globalité. Dans le même esprit, il faudrait également implanter une méthode permettant d'effectuer la conversion des coordonnées du point résultat représenté en notation BS vers la numération simple de position. Pour ce faire, nous pouvons soit implanter un composant additionnel implantant un algorithme présent dans l'état de l'art [Tul86] [EL87], soit trouver une nouvelle méthode pour le faire nous permettant de réutiliser tout ou partie de notre UA.

Enfin, il faudra également vérifier que les bénéfices engendrés par l'utilisation de la représentation BS sont également visibles sur un circuit dédié ASIC. Une synthèse de notre UA sur cette cible technologique pourrait ainsi être envisagé, ne serait-ce que pour pouvoir comparer les performances de notre architecture avec celles étant implantées sur ASIC [Wol02] [ST03] [OSS04]. À ce propos, nous pouvons noter que la structure rela-

tivement régulière de notre UA pourrait faciliter son placement-routage. Nous pourrons également analyser la consommation électrique de notre UA sur ASIC pour confirmer que cette dernière puisse être déployée dans des environnements où cette contrainte est prépondérante, tels les cartes à puces.

Chapitre 6

Protection de notre unité arithmétique contre les attaques par perturbation

Sommaire

Le chapitre précédent a décrit une UA pour l'ECC rapide et programmable de telle sorte qu'elle puisse être protégée efficacement **contre les attaques SSCA**. Ce chapitre se concentre sur la protection de cette UA vis-à-vis des **attaques par perturbation**. Cette protection est basée sur l'utilisation de portes logiques préservant la parité (*Parity Preserving Logic Gates*, PPLGs). Dans cette étude, nous nous limitons à protéger la partie opérative de notre UA à savoir le BSA (cf. Figure 5.2). Nous verrons que l'implantation de cette contre-mesure induit un important taux de détection de fautes.

Il faut noter que l'utilisation des PPLGs a déjà été proposée mais dans le cadre de la **fiabilité** des circuits [Par06]. Nous proposons ici d'appliquer ce concept dans le cadre de la **sécurité** des circuits dédiés à la cryptographie. De plus, l'étude décrite dans [Par06] est en tout point **théorique** : nous décidons dans cette étude de fournir des résultats **pratiques** d'implantation en termes de surcoût et de détection de fautes.

Nous commençons dans ce chapitre par rappeler le contexte initial dans lequel a été proposé l'utilisation des PPLGs. Puis, nous proposons une méthode générale pour concevoir un circuit préservant la parité. Enfin, nous décrivons les différents résultats relatifs à la protection du BSA grâce à cette méthode.

6.1 Contexte initial

Parhami [Par06] a introduit une famille de portes logiques pour laquelle la parité des entrées est égale à la parité des sorties. Par exemple, une PPLG ayant deux bits en entrée (e_1, e_2) et deux bits en sortie (s_1, s_2) doit obéir à la relation :

$$e_1 \oplus e_2 = s_1 \oplus s_2.$$

Le but initial de Parhami était de pouvoir utiliser ces PPLGs dans le cadre de l'implantation de circuits dits « réversibles[22] » pouvant servir notamment à la (peut-être future) conception d'ordinateurs quantiques. Dans [Par06], Parhami montre que la seule porte logique réversible à 2 entrées (e_1, e_2) et à 2 sorties (s_1, s_2) qui est également une PPLG est celle qui implante ($s_1 = \overline{e_1}$, $s_2 = \overline{e_2}$). Ce n'est pas un résultat pleinement satisfaisant si nous voulons concevoir des circuits implantant des fonctions logiques complexes. Par conséquent, Parhami a cherché et trouvé les deux seules portes logiques réversibles à 3 entrées (e_1, e_2, e_3) et à 3 sorties (s_1, s_2, s_3) qui sont des PPLGs réversibles : la porte de Fredkin [FT82] (notée FRG, et représentée en figure 6.1) et la F2G (cf. Figure 6.2).

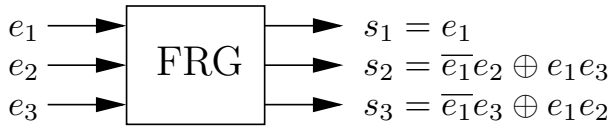

Fig. 6.1 – Porte de Fredkin.

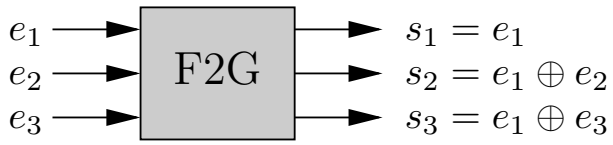

Fig. 6.2 – Porte F2G.

Il faut noter que si un concepteur de circuits sécurisés veut optimiser les performances d'un circuit implantant des PPLGs (réversibles ou non) en utilisant par exemple des

[22]Un circuit est dit « réversible » si on peut retrouver la valeur de ses sorties par la simple observation de ses entrées. Nous n'utilisons pas cette propriété dans cette étude.

sortances importantes (*fanouts*) ou des rebouclages, il doit à chaque optimisation vérifier que la parité du circuit est maintenue dans sa globalité. Ainsi, la conception d'un circuit préservant la parité (réversible ou non) est plus difficile à effectuer qu'un circuit conventionnel, et c'est pour cela que nous proposons dans la section suivante une méthode générale permettant la conception de tels circuits.

6.2 Méthode proposée pour concevoir un circuit préservant la parité

Il est proposé dans cette section une procédure générale permettant d'implanter des circuits préservant la parité. À chaque étape, nous nous ramenons à notre exemple d'application qui est la protection d'un BSA.

6.2.1 Choisir la partie du circuit à protéger

Un concepteur de circuits sécurisés doit tout d'abord diviser son circuit initial en n sous-circuits identiques (le sous-circuit étant noté $\mathcal{C}$), et ce seront ces derniers qui seront ensuite protégés à l'aide de PPLGs. Dans notre exemple, notre sous-circuit $\mathcal{C}$ a comme bits d'entrées a_i^+, b_i^+, a_i^-, b_i^- et c_i^+ et comme bits de sortie s_{i+1}^- et s_i^+ (cf. Figure 1.3).

6.2.2 Obtenir les équations logiques correspondantes

Le sous-circuit $\mathcal{C}$ doit être relativement simple afin de pouvoir récupérer les équations logiques qu'il implante facilement. Dans notre exemple, le résultat intermédiaire noté c_i^- (respectivement c_i^+) calculé par la cellule PPM qui traite les bits d'entrée a_i^+, b_i^+, a_i^- (a_{i-1}^+, b_{i-1}^+, a_{i-1}^-) s'exprime comme suit :

$$\begin{cases} c_i^- = a_i^+ \oplus b_i^+ \oplus a_i^- \\ c_i^+ = a_{i-1}^+.b_{i-1}^+ + a_{i-1}^+.\overline{a_{i-1}^-} + b_{i-1}^+.\overline{a_{i-1}^-}. \end{cases}$$

Finalement, les bits résultats s_{i+1}^- et s_i^+ calculés par la cellule PPM qui traite les bits d'entrée c_i^-, b_i^-, c_i^+ s'expriment comme suit :

$$\begin{cases} s_{i+1}^- = c_i^-.b_i^- + c_i^-.\overline{c_i^+} + b_i^-.\overline{c_i^+} \\ s_i^+ = c_i^- \oplus b_i^- \oplus c_i^+. \end{cases}$$

6.2.3 Exprimer ces équations dans un corps de Galois

Les PPLGs que sont F2G et FRG peuvent effectuer la fonction logique « NOT », et FRG peut implanter la fonction logique « AND ». En revanche, aucune de ces PPLGs ne peut calculer la fonction « OR ». C'est pour cela que les expressions logiques obtenues précédemment qui utilisent cette dernière fonction doivent être exprimées dans un corps de Galois grâce à la relation :

$$x + y = x \oplus y \oplus x.y.$$

Dans notre exemple, les variables c_i^+ et s_{i+1}^- peuvent finalement s'écrire :

$$\begin{cases} c_i^+ = a_{i-1}^+.b_{i-1}^+ \oplus a_{i-1}^+.\overline{a_{i-1}^-} \oplus b_{i-1}^+.\overline{a_{i-1}^-} \\ s_{i+1}^- = c_i^-.b_i^- \oplus c_i^-.\overline{c_i^+} \oplus b_i^-.\overline{c_i^+}. \end{cases}$$

Vérification. (pour s_{i+1}^-)
$s_{i+1}^- = c_i^-.b_i^- + c_i^-.\overline{c_i^+} + b_i^-.\overline{c_i^+}$
$= (c_i^-.b_i^- \oplus c_i^-.\overline{c_i^+} \oplus c_i^-.b_i^-.\overline{c_i^+}) + b_i^-.\overline{c_i^+}$
$= c_i^-.b_i^- \oplus c_i^-.\overline{c_i^+} \oplus c_i^-.b_i^-.\overline{c_i^+} \oplus b_i^-.\overline{c_i^+} \oplus c_i^-.b_i^-.\overline{c_i^+} \oplus c_i^-.b_i^-.\overline{c_i^+} \oplus c_i^-.b_i^-.\overline{c_i^+}$
$= c_i^-.b_i^- \oplus c_i^-.\overline{c_i^+} \oplus b_i^-.\overline{c_i^+}$
Cette vérification peut également s'appliquer pour la variable c_i^+.

6.2.4 Implanter ces équations grâce aux PPLGs.

Le circuit protégé va être uniquement constitué des PPLGs que sont FRG et F2G.

La cellule élémentaire du BSA **protégée contre les fautes** est représentée en Figure 6.3. Elle est elle-même constituée de deux composants notés PPM1 (cf. Figure 6.4) et PPM2 (cf. Figure 6.5).

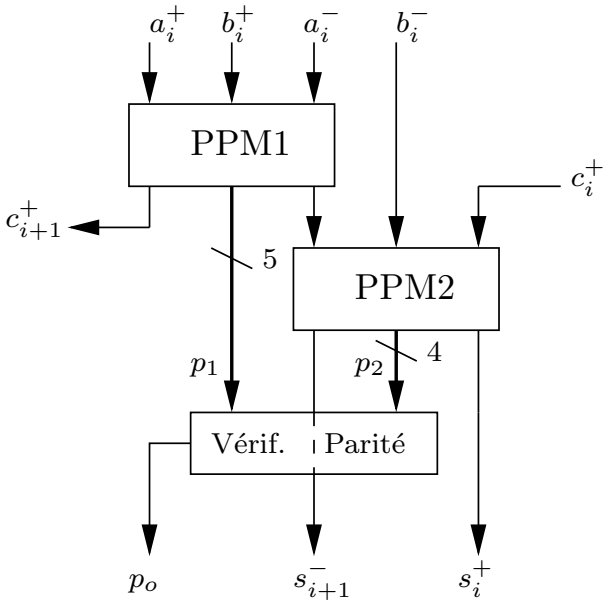

FIG. 6.3 – Cellule élémentaire du BSA protégée contre les fautes.

PPM1 calcule les deux bits nécessaires au calcul d'une BSA que sont c_{i+1}^+ et c_i^-, ainsi que p_1 qui est un signal additionnel servant à la préservation de la parité de PPM1.

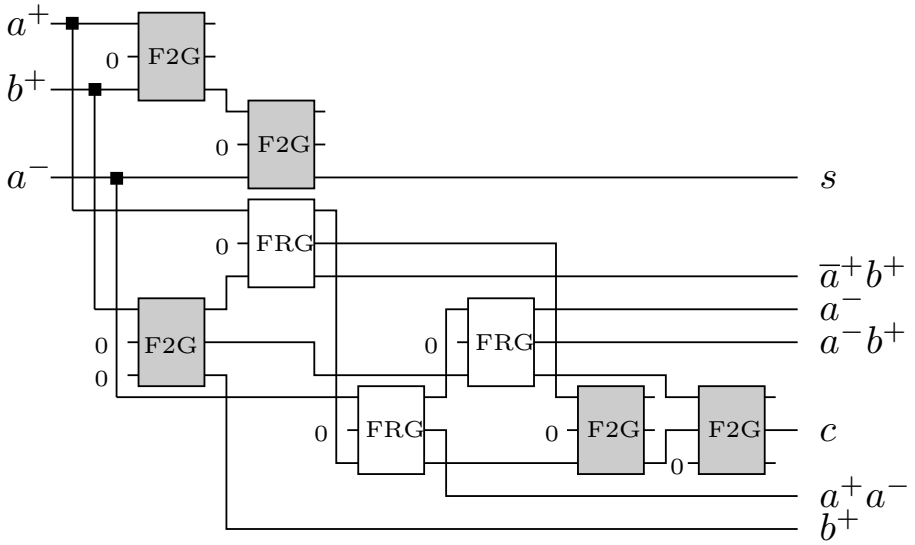

FIG. 6.4 – PPM1.

PPM2 calcule finalement les deux bits de sortie s_{i+1}^- et s_i^+, ainsi que p_2 qui est un signal additionnel servant à la préservation de la parité de PPM2. Le vérificateur de parité implante alors les trois fonctions logiques suivantes :

$$\begin{cases} par_1 = a_i^+ \oplus b_i^+ \oplus a_i^- \oplus c_i^- \\ par_2 = a_i^+ \oplus b_i^+ \oplus a_i^- \oplus p_1 \oplus c_{i+1}^+ \\ par_3 = c_i^+ \oplus b_i^- \oplus c_i^- \oplus s_{i+1}^- \oplus p_2 \oplus s_i^+. \end{cases}$$

Si le bit $p_o = par_1 + par_2 + par_3$ est égal à 1, cela signifie qu'une (ou des) faute(s) a (ont) été détectée(s).

Il doit être noté qu'aucune sortance supérieure à un n'est conseillée. Si un signal intermédiaire est utilisé par plus d'une PPLG, il est souhaitable de le dupliquer à l'aide d'une PPLG. Par exemple, si un signal x doit être dupliqué, une cellule F2G peut être implantée avec les entrées ($e_1 = \boldsymbol{x}$, $e_2 = 0$, $e_3 = 0$) : ses sorties seront alors égales à ($s_1 = \boldsymbol{x}$, $s_2 = \boldsymbol{x}$, $s_3 = \boldsymbol{x}$).

Si un signal de sortie générée par une PPLG n'est pas utilisé par la suite, il doit être envoyé en sortie du sous-circuit afin de maintenir sa parité globale. La conséquence directe de ce fait est que si deux signaux de sortie générés par une (ou des) PPLG(s) ont la même valeur et ne sont pas utilisés par d'autres PPLGs, ils peuvent être simplifiés (ou déconnectés). Par exemple, si une F2G a ses sorties égales à ($s_1 = \boldsymbol{x}$, $s_2 = \boldsymbol{x}$, $s_3 = \boldsymbol{x}$), et si s_2 et s_3 ne sont pas utilisés par d'autres PPLGs, ils peuvent être déconnectés au lieu d'être envoyés vers les sorties. PPM1 dispose de huit bits déconnectés (cf. Figure 6.4), et PPM2 six (cf. Figure 6.5).

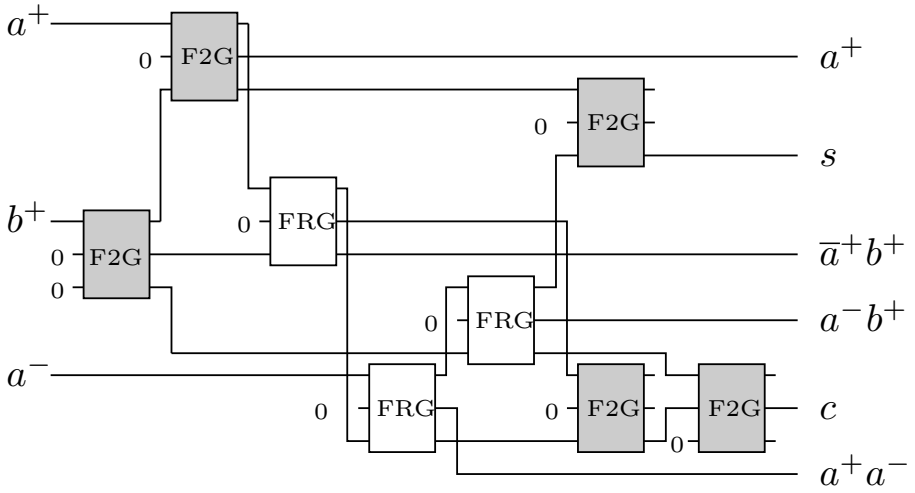

FIG. 6.5 – PPM2.

6.3 Résultats

Cette section présente des résultats relatifs à l'implantation de la préservation de la parité sur le BSA. Nous commençons tout d'abord par estimer l'impact de cette contre-mesure sur les performances de ce dernier. Puis, nous caractérisons le taux de détection de cette contre-mesure.

6.3.1 Impact sur les performances

Le tableau 6.1 précise l'impact de notre contre-mesure sur le BSA (avec n=160) en termes de surface occupée et de longueur du chemin critique. Ces résultats sont donnés après synthèse sur la technologie pour circuits dédiés ASIC C35 CORELIB. L'outil de synthèse utilisé est Design Vision.

Architecture	**Surface** (μm^2)	**Chemin critique** (ns)
BSA sans préservation de la parité	134440	1,39
BSA avec préservation de la parité	698157	5,69
Surcoût	**×5,2**	**×4,1**

TAB. 6.1 – Surcoût amené par la préservation de la parité sur le BSA (avec n=160)

La préservation de la parité amène donc un surcoût important : la surface occupée est multipliée environ par 5 et la longueur du chemin critique est multiplié environ par 4. Cependant, si nous insérons le BSA protégé dans la structure initiale de notre UA pour l'ECC (cf. Figure 5.2), le surcoût en terme de surface est acceptable (+ 38 %), mais le

chemin critique est encore très rallongé (+ 138 %). Il faut tout de même préciser que ce dernier peut être raccourci grâce à l'insertion d'étages de *pipeline*.

6.3.2 Taux de détection de fautes

Nous avons estimé le taux de détection de fautes de notre contre-mesure à l'aide de simulations sous Matlab. Dans ces simulations, deux opérandes de valeurs aléatoires représentés sur n bits (dans notre cas $n = 160$) sont envoyés en entrée du BSA protégé. Une ou deux fautes sont injectées durant le calcul du BSA. Il faut noter que nous ne pouvons pas aller au-delà de deux fautes injectées dans nos simulations car la puissance de calcul délivrée par Matlab n'est pas suffisante pour notre application. Le modèle de faute simulé est l'inversion de la valeur du ou des bits fautés (*bit-flip*). Ces fautes ne sont pas forcément contigues : elles peuvent être injectées n'importe où pendant le calcul du BSA. Les résultats sont détaillés dans le tableau 6.2.

Nombre de bits fautés	**1 bit**	**2 bits**
Fautes détectées	80%	86,8%
Fautes sans conséquence	14,3%	2,1%
Fautes non-détectées	**5,7%**	**11,1%**

TAB. 6.2 – Évaluation de la capacité de détection de fautes de notre contre-mesure.

Nous pouvons constater un premier fait intéressant : le nombre de fautes détectées augmente lorsque le nombre de bits fautés augmente (cf. première ligne du tableau 6.2). On peut par exemple comparer ce comportement avec celui d'un code détecteur d'erreurs linéaire tel que le codage de Hamming [Lal00] : pour de tels codes, tant que le nombre de fautes effectives ne dépasse pas les capacités de détection du code utilisé, ce dernier peut toujours détecter qu'il y a eu une (ou des) erreur(s) qui se sont produites, et son taux de détection d'erreurs est de 100%. En revanche, lorsque le nombre de fautes injectées dépasse les capacités du code, il ne peut détecter aucune erreur. Ainsi, dans le domaine de la sécurité des cryptosystèmes, et dans le cas où un attaquant ne dispose pas d'un dispositif d'attaque suffisamment sophistiqué pour injecter un petit nombre de fautes (cf. section 3.2.1.3), la préservation de la parité est plus adaptée que la simple utilisation de codes détecteurs d'erreurs linéaires.

Une deuxième observation découlant des résultats de nos simulations concerne des fautes sans conséquence pour le BSA (cf. tableau 6.2), c'est-à-dire que certaines fautes effectivement injectées n'induisent pas de résultat fauté pour l'addition. Ceci est dû notamment à la nécessaire duplication de certains signaux intermédiaires (cf. section 6.2.4), mais pas seulement : cela peut être également dû à un filtrage des fautes par les PPLGs elles-mêmes. Par exemple, si à l'entrée d'une F2G (cf. Figure 6.2) contenue dans notre BSA protégé, les valeurs des trois entrées e_1, e_2 et e_3 sont inversées (ce qui est un modèle de faute réaliste car ces trois entrées sont très proches les unes des autres, cf. section

3.2.1.3), seule la valeur de s_1 sera fautée en sortie de cette F2G. Notons que ces fautes sans conséquence ne sont pas exploitables dans le cadre d'attaques par perturbation de type CSEA puisqu'elles se produisent quelque soit l'opération effectuée, et donc **quelque soit le bit de clé traité**. Nous n'avons donc pas ajouté de vulnérabilité supplémentaire à notre cryptosystème.

Enfin, la troisième ligne du tableau 6.2 est celle qui doit retenir le plus notre attention, puisqu'elle nous renseigne sur le pourcentage de fautes non-détectées par notre dispositif. Le pourcentage de fautes non-détectées passe de 5,7% pour une faute induite à 11,1% pour deux fautes injectées dans notre BSA protégé. Il nous reste à porter nos simulations sur un logiciel plus adapté (par exemple PARI-GP) pour évaluer le nombre de fautes non-détectées lorsqu'il y en a un grand nombre d'injectées. De par les propriétés de la parité elle-même, il peut par exemple être espéré que le taux de détection d'erreurs lorsque 3 fautes sont injectées sera plus elevé que lorsque 2 fautes sont induites.

6.4 Conclusion et perspectives

Ce chapitre a proposé de protéger notre UA contre les attaques par perturbation à l'aide de la préservation de la parité, dont le concept théorique a été initialement proposé par Parhami [Par06]. Nous avons proposé au cours de notre étude une méthode générale simple pour concevoir un circuit préservant la parité, et nous avons fourni des résultats encourageants pour ce qui est de l'impact de cette contre-mesure, notamment en termes de surface occupée et de taux de détection de fautes.

Notre étude ouvre la voie vers de nombreuses perspectives. Tout d'abord, nous n'avons protégé que la partie opérative de notre UA à savoir le BSA. De futurs travaux pourraient consister à protéger grâce à la préservation de la parité l'ensemble de notre UA, en particulier la logique de contrôle ((dé)multiplexeurs).

Ensuite, nous pouvons essayer de diminuer l'impact de notre contre-mesure sur la surface occupée de notre BSA protégé. Ceci peut être fait par exemple en mettant en évidence de nouvelles PPLGs disposant de plus de 3 entrées-sorties, ou implantant des fonctions logiques plus complexes. Quant à l'impact de notre contre-mesure sur le chemin critique, celui-ci peut être atténué en insérant des étages de *pipeline*.

De plus, nous pouvons mener une étude théorique sur la manière dont le taux de fautes non-détectées peut être diminué grâce à la préservation de la parité en augmentant le taux de détection et/ou le taux de fautes sans conséquences. Cette étude pourra dans un deuxième temps servir à créer une suite logicielle permettant la création automatique d'un circuit préservant la parité à partir de l'analyse des fonctions logiques calculées par celui-ci. Nous pouvons également envisager d'inclure dans cette suite logicielle des notions d'optimisation de la surface occupée et de la longueur du chemin critique, voire même une option permettant de trouver le meilleur compromis respect des contraintes d'implantation/taux de détection d'erreurs maximal.

En outre, nous pouvons également étudier la manière dont se comporte notre contre-mesure face à des modèles de perturbation autres que le *bit-flip* comme le collage d'un

bit à une valeur précise par exemple. Le même type d'étude peut être menée lorsque des fautes de délai sont injectées [FF07]. De plus, l'évaluation de la capacité de détection de fautes de notre contre-mesure (cf. tableau 6.2) prend en compte un modèle de faute peu réaliste lorsque des fautes multiples sont injectées, puisque ces dernières peuvent se produire n'importe où dans le BSA protégé au cours de nos simulations. Or, il est beaucoup plus probable avec les moyens d'attaques actuels de fauter des fils proches ou contigus (cf. section 3.2.1.3). Il serait donc préférable dans des prochaines simulations d'injections de fautes de ne considérer que des fautes rapprochées spatialement, ce qui pourrait donner un taux de détection « réaliste » de notre contre-mesure plus important que le taux de détection « théorique » (ce dernier est donné dans le tableau 6.2). Il faudrait également pouvoir évaluer la capacité de détection de fautes de notre contre-mesure lorsque plus de deux fautes (contigues) sont injectées en même temps. Ceci nous permettra de caractériser complètement la préservation de la parité. Nous proposons d'effectuer nos calculs sur PARI-GP, au lieu de Matlab actuellement.

Une autre perspective consisterait à utiliser la propriété de réversibilité des PPLGs : vu que pour chaque jeu de sorties (s_1, s_2, s_3), il ne correspond qu'un jeu d'entrées (e_1, e_2, e_3) et un seul, dans le cas où le cryptosystème détecterait une faute au niveau d'une PPLG, celui-ci pourrait forcer la PPLG victime d'une attaque à recommencer son calcul. Si nous décidons d'utiliser le principe de réversibilité, il faut noter que d'autres portes ayant cette propriété peuvent être utilisées, comme par exemple la porte de Toffoli ou celle de Peres (cf. [Par06] pour plus de détails).

Enfin, nous pouvons également concevoir de protéger d'autres cryptosystèmes à l'aide de la préservation de la parité, comme par exemple l'AES.

Conclusion et perspectives générales

Ce travail de thèse est destiné à prodiguer des conseils exhaustifs à tout concepteur de circuit sécurisé devant implanter une unité arithmétique pour courbes elliptiques et devant concilier des impératifs de performance et de sécurité.

Nous résumons ci-dessous les trois objectifs principaux que nous avons réalisés durant cette thèse.

– Tout d'abord, nous avons conçu l'une des plus performantes unités arithmétiques pour courbes elliptiques présentes dans la littérature. Nous avons proposé de réécrire l'algorithme de multiplication modulaire de Montgomery en utilisant la représentation à retenues conservées signées, et les résultats obtenus sont très satisfaisants. Notamment, notre unité arithmétique peut fonctionner à une fréquence relativement plus élevée que la grande majorité de ses homologues, ce qui permet notamment d'obtenir un calcul de la multiplication scalaire (opération principale de la cryptographie sur courbes elliptiques) qui est l'un des plus rapides du marché.
De plus, notre unité peut être facilement mise à l'échelle, et ceci est un atout face à d'autres unités arithmétiques (présentes et futures) de l'état qui ne possèdent pas cette propriété, notamment lorsqu'il sera nécessaire d'augmenter la taille des opérandes implantées pour passer à un niveau de sécurité supérieur. En effet, ce faisant, nous pouvons espérer suivant les cas soit creuser l'écart de performances entre ces unités et la nôtre, soit le combler au moins en partie.
Enfin, notre unité artihmétique dispose d'une architecture compacte, qui utilise peu de ressources du FPGA, et qui est donc adaptée aux environnements fortement contraints en surface (par exemple les cartes à puce).

  Nous pouvons mettre en évidence plusieurs perspectives possibles pour ces travaux. Tout d'abord, il peut être exploré la possibilité d'implanter un multiplieur de Montgomery en grande base (autrement dit avec $\beta > 2$), dans lequel les opérandes sont représentés avec un système d'Avizienis (cf. section 1.1.2.1, et plus particulièrement [Avi61]). Nous pouvons ainsi espérer obtenir un multiplieur plus rapide que son homologue utilisant une base $\beta = 2$, car d'une part, il nécessiterait moins de cycles d'horloge pour calculer une multiplication modulaire et d'autre part sa fréquence de fonctionnement serait maintenue grâce à l'insertion d'étages de *pipeline*. Il faudrait juste vérifier que sa surface augmente raisonnablement afin de pouvoir être inséré dans un environnement fortement contraint en surface.
  Ensuite, il pourrait être envisagé d'accélérer le temps de calcul de l'opération d'inversion modulaire, en adaptant par exemple la méthode d'inversion de Montgomery, ou des algorithmes utilisant des calculs de PGCD à la représentation BS. Ainsi, si l'algorithme obtenu est très efficace, alors il pourrait être envisagé de conserver les coordonnées affines pendant l'exécution de la multiplication scalaire.
  Une voie d'amélioration possible réside également dans une utilisation des ressources du FPGA cible en BDSP et en BRAM. En effet, l'UA la plus performante de la littérature instancie ces dernières [GP08], et la seule manière de combler notre retard de performance consisterait à faire de même. Nous pourrions par exemple chercher si il est possible d'utiliser les BDSP afin d'effectuer des opérations arithmétiques

en notation redondante. De plus, les BRAM peuvent également être instanciés afin d'implanter l'ensemble des registres nécessaires à la mémorisation des variables intermédiaires impliquées dans le calcul de la multiplication scalaire. Cela permettrait d'une part de ne pas mobiliser plus de *Flip-Flops* et donc plus de *slices* de notre FPGA, et d'autre part d'accélérer les accès mémoires.
En outre, il faudrait également implanter une méthode permettant d'effectuer la conversion des coordonnées du point résultat représenté en notation BS vers la numération simple de position. De telles méthodes sont déjà présentes dans l'état de l'art [Tul86] [EL87], mais elles impliquent l'implantation d'un (petit) composant supplémentaire. Nous axerons plutôt nos recherches sur une nouvelle méthode permettant d'effectuer cette conversion en réutilisant tout ou partie de notre UA.
Enfin, pour pouvoir comparer les performances de notre UA sur circuits dédiés ASIC avec celles de l'état de l'art utilisant la même cible technologique [Wol02] [ST03] [OSS04], une implantation de notre UA (synthèse et placement-routage) sur ASIC peut être envisagée. Nous pourons aussi en profiter pour faire une analyse fine de sa consommation électrique, contrainte forte pour les circuits à alimentation autonome.

– Une autre contribution importante de nos travaux consiste à proposer une UA pour l'ECC protégée contre les attaques SSCA dont le temps de calcul de la multiplication scalaire est le plus court de l'état de l'art.

  Il est encore possible d'améliorer les performances de notre UA de deux manières différentes : d'une part, en maintenant notre stratégie de protection en l'état, et ce, en recherchant des contre-mesures plus performantes et/ou en implantant les opérations modulaires plus rapidement (cf. la remarque concernant notre première contribution), et d'autre part, en adoptant une autre stratégie de protection. Concernant ce dernier point, nous pouvons par exemple imaginer implanter plusieurs UAs (ou en tout cas plusieurs additionneurs) qui exécutent des calculs en parallèle de telle sorte que le cryptosystème soit protégé contre les attaques SSCA : c'est ce que proposent Ghosh *et al.* [GACG08], mais également Bajard *et al.* [BILT04].

– Enfin, notre dernière contribution réside dans la conceptualisation et la mise en pratique du principe de la préservation de la parité. L'application de ce principe permet de se prémunir face aux attaques par perturbation. L'implantation de la préservation de la parité dans notre UA apporte un haut niveau de détection de fautes ainsi qu'un impact sur les performances acceptable.

  Cette étude a mis en évidence de nombreuses perspectives possibles. Les principales consistent essentiellement à effectuer une étude plus poussée sur la manière dont le taux de fautes non-détectées peut être diminué (par l'augmentation du taux de détection et/ou du taux de fautes sans conséquences), et sur la façon de diminuer l'impact de cette contre-mesure sur la surface du circuit protégé (par la mise en évidence de nouvelles PPLGs disposant de plus de trois entrées ou implan-

tant des fonctions logiques plus complexes). Nous espérons que ces études pourront permettre d'automatiser la synthèse matérielle de cryptosystèmes complètement protégés à l'aide de la préservation de la parité.
Enfin, il peut être espéré que la réversibilité soit également une solution efficace pour la protection des cryptosystèmes face aux attaques par perturbation.

Annexe : introduction aux circuits FPGA

Cette annexe est une introduction aux circuits programmables, et plus précisément aux circuits FPGA. Nous commençons par détailler l'architecture de base d'un FPGA, puis ses différentes évolutions ou sophistications qui ont permis d'améliorer progressivement ses performances, et enfin les avantages d'utilisation qu'un FPGA procure notamment par rapport à un circuit ASIC.

Architecture d'un FPGA

Actuellement, trois fondeurs principaux se partagent le marché des circuits FPGA : Actel©, Altera©, et Xilinx©. Vu que la plupart des UA pour l'ECC présentes dans la littérature sont implantées sur des circuits FPGA Xilinx© [OP01] [MMM04b] [DMKP05] [CDM05] [SPV06] [MSB+07] [BCM+07] [OBPV08] [GACG08] [GP08], nous avons décidé de faire le même choix de cible technologique afin de pouvoir comparer plus facilement les performances de notre UA avec celles de l'état de l'art.

Nous choisissons donc de décrire les principales spécificités de la famille de FPGA commercialisée par Xilinx© depuis 1998 et qui va être la cible de notre implantation : la famille Virtex.

Un circuit Virtex, illustré par la figure 6.6, est principalement constitué de cinq types de ressources.

1. **Des blocs d'entrée/sortie programmables** (*Input/Output Blocks*, IOBs). Ils constituent l'interface entre les pattes (plots) du circuit et le cœur du FPGA. Il existe différents types d'entrées/sorties, notamment celles réservées à l'utilisateur, aux alimentations, à (ou aux) horloge(s), à l'adaptation des signaux, aux signaux de configuration et de programmation du FPGA, ainsi qu'aux signaux de test (*debug*).
2. **Des blocs logiques programmables** (*Configurable Logic Blocks*, CLBs). Composés de quatre cellules logiques (*Logic Cells*, LCs) réparties en deux tranches (*slices*) identiques (cf. Figure 6.7), ils servent à construire les circuits numériques implantés sur le FPGA. Ces CLBs sont disposées suivant une topologie matricielle. Chaque LC contient essentiellement **un générateur de fonctions à quatre entrées** réalisé à l'aide d'une LUT, **un élément de mémorisation** possédant des signaux

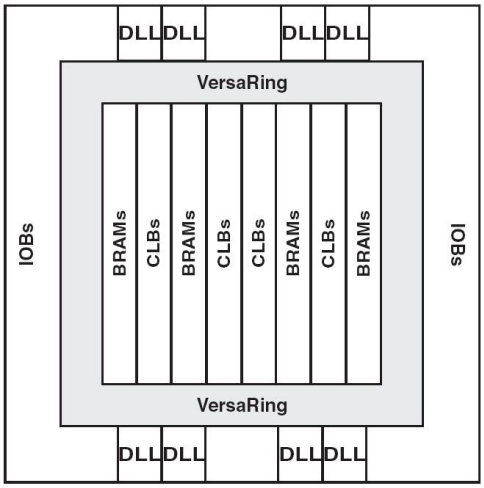

FIG. 6.6 – Vue d'ensemble de l'architecture d'un FPGA de type Virtex [X06].

d'initialisation (*set* et *reset*) synchrones ou asynchrones, ainsi que de la **logique de contrôle et de propagation de retenues** (notée « *Carry & Control* » sur la Figure 6.7) destinée notamment à la conception de circuits arithmétiques performants, comme par exemple les additionneurs.

3. **Un routage programmable.** Pour connecter les blocs logiques entre eux et les entrées/sorties, les Virtex disposent de toute une panoplie de ressources de routage programmables. En général, différentes connections situées à différents niveaux hiérarchiques peuvent être mises en jeu : des connections directes vers les voisins proches (c'est par exemple le rôle du dispositif *VersaRing* représenté sur la figure 6.6, qui offre les ressources nécessaires à l'interconnexion entre les IOBs et le cœur du composant), des connections générales à travers des matrices de routage voire même des connections à plus longue distance.
4. **Un générateur d'horloge programmable.** Un circuit Virtex possède également quatre lignes d'horloge disposant chacune d'une boucle à verrouillage de délai (*Delay-Locked Loop*, DLL, cf. Figure 6.6). Cette dernière permet de contrôler des décalages d'horloge (*clock skew*) à l'intérieur du FPGA ou entre plusieurs circuits, ainsi que de déphaser, doubler ou diviser une horloge.
5. **De la mémoire RAM.** Chaque LUT peut stocker de l'information : elle peut permettre la conception d'une mémoire synchrone à accès direct (*Random Access Memory*, RAM) 16×1 bits, tandis qu'une fois appariées, les deux LUTs d'une même tranche peuvent offrir une mémoire synchrone 16×2 bits, 32×1 bits ou 16×1 bits à double accès (*dual-port synchronous* RAM). Une CLB peut également fonctionner comme un registre à décalage de 16 bits. En résumé, on peut utiliser les CLBs pour implanter des petites mémoires, mais cela n'est pas très efficace.

   C'est pour cette raison qu'un circuit Virtex offre deux zones de blocs mémoires

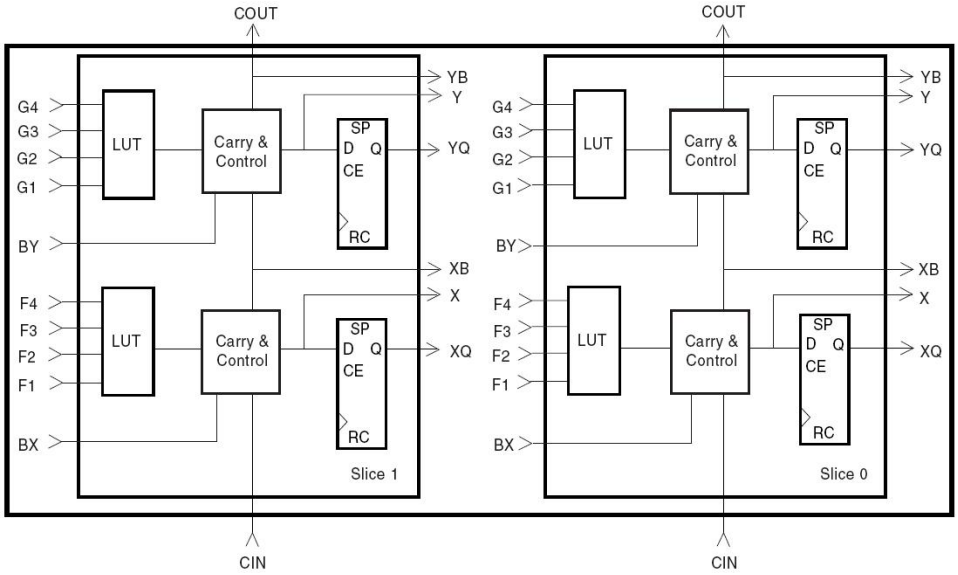

FIG. 6.7 – Schéma d'un bloc logique programmable d'un Virtex à deux tranches [X06].

(*Block* RAM, BRAM), organisées en blocs de 4096 bits, situées de part et d'autre de la matrice de CLBs. Ainsi, grâce aux outils de synthèse fournis par Xilinx© (ISE), nous pouvons instancier plusieurs blocs afin de construire des mémoires synchrones rapides et de taille désirée. Cette approche comporte deux avantages : les accès mémoires s'en trouvent accélérés, et les ressources en LUTs du FPGA sont économisées.

Les FPGA : des circuits en constante évolution

La section précédente a détaillé quelques éléments principaux de l'architecture d'un FPGA de la famille Virtex. Cette section décrit quelques évolutions notables qu'ont connues les FPGAs Virtex depuis leur création, certaines d'entre elles étant utilisées dans des UA performantes pour l'ECC présentes dans la littérature.

Tout d'abord, les Virtex tendent à disposer de plus en plus de BRAM, et certaines familles de FPGA se sont même spécialisées dans l'implantation de grandes capacités mémoires : c'est par exemple le cas de la famille Virtex-E Extended Memory qui est initialement destinée à la conception de circuits destinés aux applications gourmandes en mémoire, comme par exemple les processeurs de signal numérique (*Digital Signal Processor*, DSP). Notons également que ces BRAM ont des fréquences de fonctionnement de plus en plus élevées grâce à l'évolution des technologies d'intégration (par exemple 0.18 μm pour la famille Virtex-E lancée sur le marché en 2002, contre 0.22 μm pour la famille Virtex introduite en 2000).

Enfin, les FPGAs offrent des primitives pour la réalisation d'opérateurs arithmétiques :

c'est par exemple le cas de la famille Virtex-2 qui embarque des multiplieurs traitant deux nombres de 18 bits codés en complément à deux (cf. section 1.1.1). Plus récemment, les Virtex-4 sont équipés d'extensions arithmétiques matérielles dans le but d'accélérer plus particulièrement les applications effectuées par les DSPs. Ces blocs fonctionnels, que nous appellons « blocs DSP » (*Blocks* DSP, BDSP) contiennent chacun un multiplieur d'entiers signés de l_M bits, couplé avec un additionneur traitant des opérandes signées sur l_A bits (avec $l_A > l_M$). Ainsi, ces BDSP peuvent être programmés pour effectuer des fonctions arithmétiques de base telles que la multiplication, l'addition et la soustraction d'entiers signés ou non-signés, mais également la « multiplication/accumulation », opération très utilisée dans les DSPs. Pour tirer un profit maximal de ces blocs, des registres peuvent également être intercalés (*pipelines*) dans la structure des différents multiplieurs et additionneurs afin de réduire le chemin critique et ainsi augmenter la fréquence de fonctionnement et le débit de ces opérateurs arithmétiques.

Pour résumer, tous ces éléments mémoires ou arithmétiques supplémentaires progressivement embarqués dans les FPGAs, et initialement implantés pour être utilisés dans le cadre de la conception des DSPs, peuvent également servir à implanter des UA performantes pour l'ECC. C'est dans cet esprit par exemple qu'Orlando *et al.* [OP01] utilisent les BRAM d'un Virtex pour implanter un grand nombre de registres servant à effectuer une version additive de la méthode d'exponentiation introduite dans [BGMW92]. Cet algorithme gourmand en mémoire et qui nécessite des précalculs sur le point de base est toutefois quatre fois plus rapide que l'algorithme du « doublement-et-addition » (cf. Algorithme 9). Ils se servent également des BRAM pour mémoriser le résultat de précalculs servant à effectuer un algorithme de multiplication modulaire rapide. McIvor *et al.* [MMM04b] utilisent quant à eux les multiplieurs 18×18 intégrés dans les Virtex-2 pour implanter l'algorithme de Montgomery (cf. Algorithme 8), et les lignes rapides dédiées aux propagations de retenue du FPGA. Quant à eux, Mentens *et al.* [MSB+07] utilisent également les multiplieurs dédiés du FPGA et les BRAM. Enfin, Güneysu *et al.* [GP08] instancient les BDSP afin d'accélérer les calculs modulaires effectués, ainsi que les BRAM.

Avantages amenés par la conception de circuits sur FPGA

La section précédente a détaillé les principales évolutions qu'ont connues les FPGAs Virtex depuis leur création, et qui sont utilisées dans certaines UA présentes dans l'état de l'art (cf. section 5.1). Cette section décrit les principaux avantages amenés par la conception de circuits sur les FPGAs par rapport aux ASICs.

Tout d'abord, les FPGAs constituent un formidable **outil de prototypage**. En effet, ils permettent de valider des concepts ou de réaliser des solutions en un temps réduit et à moindre coût : une fois le système numérique décrit à l'aide d'un langage de description matériel, que ce soit celui dédié à la description matérielle de circuits intégrés à très hautes vitesses (*Very high speed integrated circuits Hardware Description Language*, VHDL) ou Verilog, nous disposons d'une suite logicielle (également appelés « outils de

synthèse et de placement/routage ») qui offre des informations très précises telles que le nombre de cellules requises, leur fonctionnalité et leurs interconnexions. Ainsi, au terme de ce processus nécessitant au plus quelques dizaines de minutes, nous pouvons étudier le comportement du circuit dans des conditions réelles.

Ensuite, grâce notamment aux cellules logiques destinées à l'optimisation d'opérateurs arithmétiques et aux mémoires synchrones facilitant le calcul par tables (cf. section 6.4), **les FPGAs concurrencent aujourd'hui les ASICs dans certaines applications** : les FPGAs ne sont donc plus seulement confinés dans le prototypage de systèmes numériques, et ils élargissent ainsi leur champ d'applications. Les résultats obtenus notamment par Beuchat [Beu01] indiquent que les FPGAs offrent une alternative sérieuse pour la réalisation de coprocesseurs cryptographiques, notamment du point de vue du débit. Il a par exemple montré en 2001 qu'un FPGA Xilinx XC4000 pouvait rivaliser avec les processeurs commerciaux de l'époque.

De plus, **les FPGAs s'avèrent moins onéreux que les ASICs** pour de petites séries et voient tout comme ces derniers leurs **performances augmenter** grâce à l'évolution des technologies d'intégration (on est ainsi parti en 2001 d'une technologie 220 nm pour les premiers Virtex-1, pour arriver en 2009 à une technologie 40 nm pour les Virtex-6). Cette diminution progressive de la taille des transistors entraîne également une augmentation continue de la densité d'intégration : les constructeurs peuvent ainsi implanter à chaque nouvelle famille de FPGAs plus de CLBs tandis que la surface du FPGA reste quasiment inchangée.

Enfin, l'utilisation des FPGAs entraîne **une plus grande flexibilité** que les ASICs. Par exemple, le concepteur n'a rien à faire pour générer et distribuer les horloges partout où elles sont nécessaires, tandis que c'est un problème complexe à résoudre dans les ASICs. De même, le dispositif de routage programmable VersaRing (cf. Figure 6.6) embarqué sur les Virtex-E offrant les ressources nécessaires à l'interconnexion des CLBs aux IOBs permet de modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier.

En résumé, **les FPGAs peuvent fournir des performances équivalentes aux petits ASICs, mais avec la souplesse de la programmation en plus.** Cependant, il faut noter que la conception d'un système performant sur FPGA *via* sa programmation dans un langage de description matériel nécessite non seulement **une bonne connaissance** de l'architecture du circuit cible afin d'exploiter judicieusement les ressources de ce dernier, mais également **du compilateur et de l'outil de synthèse utilisés** (cf. section 5.4.1.2).

Annexe : production scientifique

Communications avec actes dans un congrès international

[FRMTT08] J. Francq, J.-B. Rigaud, P. Manet, A. Tria et A. Tisserand. *Error-Detection for Borrow-Save Adders Dedicated to ECC Unit.* Dans les actes *IEEE-CS* de *Fault Diagnosis and Tolerance in Cryptography* – FDTC, pages 77–86, 2008.

[FF07] O. Faurax et J. Francq. *Security of several AES Implementations against Delay Faults.* Dans les actes de *Nordic Workshop on Secure IT* – NordSec, pages 61–72, 2007.

[MRFJTRQL06] P. Manet, J.-B. Rigaud, J. Francq, M. Jeambrun, A. Tria, B. Robisson, J. Quartana et S. Laabidi. *Integrated Evaluation Platform for Secured Devices.* Dans les actes de *International Workshop on Reconfigurable Communication-centric System-on-Chips* – ReCoSoC, pages 214–219, 2006.

Communications avec actes dans un congrès national

[FRM08a] J. Francq, J.-B. Rigaud et P. Manet. Optimiser ou protéger votre carte à puce peut la rendre plus vulnérable. Dans les actes des Journées Pédagogiques de la Coordination Nationale de la Formation en Micro et nanoélectronique – JP-CNFM, pages 215–218, 2008.

Communications orales sans actes dans un congrès international ou national

[Fra08] J. Francq. *Parity-Preserving Logic Gates as a Protection against Fault Attacks.* Pour *Yet Another Conference on Cryptography* – YACC, 2008.

[FRM08b] J. Francq, J.-B. Rigaud et P. Manet. Optimiser ou protéger votre carte à puce peut la rendre plus vulnérable. Pour les Journées Nationales du Réseau Doctoral en Microélectronique – JNRDM, 2008.

Communications par affiche dans un congrès international ou national

[FRMBT07] J. Francq, J.-B. Rigaud, P. Manet, J.-C. Bajard et A. Tisserand. Amélioration de la sécurité des circuits intégrés par codage de l'information. Pour les Journées Nationales du Réseau Doctoral en Microélectronique − JNRDM, 2007.

[FRMTB07] J. Francq, J.-B. Rigaud, P. Manet, A. Tisserand et J.-C. Bajard. Amélioration de la sécurité des cryptoprocesseurs par codage de l'information. Pour l'école thématique intitulée « Architectures des systèmes matériels enfouis et méthodes de conception associées » − ARCHI, 2007.

Distinction

Prix de la meilleure présentation orale pour la référence [FRM08b].

Productions scientifiques futures

À l'issu de la rédaction de ce mémoire, nous comptons déposer un article dans une revue internationale avec comité de lecture pour présenter notre unité arithmétique performante (cf. chapitre 5).

Bibliographie

[AARR02] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 2523, pages 29–45, 2002.

[AFQ83] F. André, P. Frison, and P. Quinton. Algorithmes systoliques : de la théorie à la pratique. *Rapport de recherche, Institut National de Recherche en Informatique et Automatique*, 1983.

[AFT+08] F. Amiel, B. Feix, M. Tunstall, C. Whelan, and W. P. Marnane. Distinguishing Multiplications from Squaring Operations. In *Proc. Selected Areas in Cryptography – SAC, LNCS*, volume 5381, pages 346–360, 2008.

[AHLM03] M. Ahn, J. Ha, H. Lee, and S. Moon. A Random M-ary Method Based Countermeasure against Side Channel Attacks. In *Proc. International Conference on Computational Science and Its Applications – ICCSA, LNCS*, volume 2668, pages 338–347, 2003.

[AK96] R. J. Anderson and M. Kuhn. Tamper Resistance - A Cautionary Note. In *Proc. USENIX Workshop on Electronic Commerce*, pages 1–11, 1996.

[AT03] T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In *Proc. Information Security – ISC, LNCS*, volume 2851, pages 218–233, 2003.

[Avi61] A. Avizienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, 10 :389–400, 1961.

[Bar87] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology – CRYPTO, LNCS*, volume 263, pages 311–323, 1987.

[BCM+07] A. Byrne, F. Crowe, W. P. Marnane, N. Meloni, A. Tisserand, and E. M. Popovici. SPA Resistant Elliptic Curve Cryptosystem using Addition Chains. *Int. J. High Performance Systems Architecture*, 1(2) :133–142, 2007.

[BCO04] É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 3156, pages 16–29, 2004.

[BDH+97] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the

Presence of Transient Faults. In *Proc. Security Protocols, LNCS*, volume 1361, pages 115–124, 1997.

[BDL97] D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology – EUROCRYPT, LNCS*, volume 1233, pages 37–51, 1997.

[BECN+06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IEEE Special Issue on Cryptography and Security*, 96(2) :370–382, 2006.

[Ber01] D. J. Bernstein. A Software Implementation of NIST P-224. In *Workshop on Elliptic Curve Cryptography – ECC*, 2001.

[Beu01] J.-L. Beuchat. *Étude et conception d'opérateurs arithmétiques optimisés pour circuits programmables*. Thèse de Doctorat, École Polytechnique Fédérale de Lausanne, 2001.

[BGMW92] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast Exponentiation with Precomputation : Algorithms and Lower Bound. In *Advances in Cryptology – EUROCRYPT, LNCS*, volume 658, pages 200–207, 1992.

[BGV94] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of Three Modular Reduction Functions. In *Advances in Cryptology – CRYPTO, LNCS*, volume 773, pages 175–186, 1994.

[BHLM01] M. Brown, D. Hankerson, J. López, and A. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In *Topics in Cryptology – CT-RSA, LNCS*, volume 2020, pages 250–265, 2001.

[BILT04] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia. Leak Resistant Arithmetic. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 3156, pages 62–75, 2004.

[BJ02] É. Brier and M. Joye. Weierstrass Elliptic Curves and Side-Channel Attacks. In *Proc. Public Key Cryptography – PKC, LNCS*, volume 2274, pages 335–345, 2002.

[BJ03a] O. Billet and M. Joye. The Jacobi Model of an Elliptic Curve and Side-Channel Analysis. In *Proc. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC, LNCS*, volume 2643, pages 34–42, 2003.

[BJ03b] É. Brier and M. Joye. Fast Point Multiplication on Elliptic Curves Through Isogenies. In *Proc. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC, LNCS*, volume 2643, pages 43–50, 2003.

[BK03] R. Bevan and E. W. Knudsen. Ways to Enhance Differential Power Analysis. In *Information Security and Cryptology – ICISC, LNCS*, volume 2587, pages 327–342, 2003.

[BL07a] D. J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In *Advances in Cryptology – ASIACRYPT, LNCS*, volume 4833, pages 29–50, 2007.

[BL07b] D. J. Bernstein and T. Lange. Inverted Edwards Coordinates. In *Proc. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC, LNCS*, volume 4851, pages 20–27, 2007.

[Bla83] G. R. Blakley. A Computer Algorithm for the Product AB modulo M. *IEEE Transactions on Computers*, 32(5) :497–500, 1983.

[Ble98] D. Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS# 1. In *Advances in Cryptology – CRYPTO, LNCS*, volume 1462, pages 1–12, 1998.

[Blu99] T. Blum. Modular Exponentiation on Reconfigurable Hardware. *Thèse de Mastère, Institut Polytechnique de Worcester, USA*, 1999.

[BM02] L. Batina and G. Muurling. Montgomery in Practice : How to Do it More Efficiently in Hardware. In *Topics in Cryptology – CT-RSA, LNCS*, volume 2271, pages 40–52, 2002.

[BMM00] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Advances in Cryptology – CRYPTO, LNCS*, volume 1880, pages 131–146, 2000.

[Boo51] A. D. Booth. A Signed Binary Multiplication Technique. *Quarterly Journal Mechanics of Applied Mathematics*, 4 :236–240, 1951.

[BOS03] J. Blömer, M. Otto, and J.-P. Seifert. A New CRT-RSA Algorithm Secure against Bellcore Attacks. In *Proc. Conference on Computer and Communications Security*, pages 311–320, 2003.

[BOS06] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Proc. Fault Diagnosis and Tolerance in Cryptography – FDTC, LNCS*, volume 4236, pages 36–52, 2006.

[BP01] T. Blum and C. Paar. High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware. *IEEE Transactions on Computers*, 50(7) :759–764, 2001.

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology – CRYPTO, LNCS*, volume 1294, pages 513–525, 1997.

[BSS05] I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography.* London Mathematical Society Lecture Note Series, Cambridge University Press, 2005.

[Cau40a] A. Cauchy. Sur les moyens de vérifier ou de simplifier diverses opérations de l'arithmétique décimale. Oeuvre complètes (tome 5, série 1) :443–455, 1840.

[Cau40b] A. Cauchy. Sur les moyens d'éviter les erreurs dans les calculs numériques. Oeuvre complètes (tome 5, série 1) :431–442, 1840.

[CC87] D. Chudnovsky and G. Chudnovsky. Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factoring Tests. *Advances in Applied Mathematics*, 7 :385–434, 1987.

[CDM05] F. Crowe, A. Daly, and W. P. Marnane. A Scalable Dual Mode Arithmetic Unit for Public Key Cryptosystems. In *Proc. International Conference on Information Technology : Coding and Computing − ITCC*, pages 568–573, 2005.

[CF06] H. Cohen and G. Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2006.

[CH04] J. Chung and M. A. Hasan. More Generalized Mersenne Numbers. In *Proc. Selected Areas in Cryptography − SAC, LNCS*, volume 3006, pages 335–347, 2004.

[Cie03] M. Ciet. *Aspects of Fast and Secure Arithmetics for Elliptic Curve Cryptography*. Thèse de Doctorat, Université Catholique de Louvain, 2003.

[CJ01] C. Clavier and M. Joye. Universal Exponentiation Algorithm - A First Step towards Provable SPA-Resistance. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 300–308, 2001.

[CJ03] M. Ciet and M. Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In *Information and Communication Security − ICICS, LNCS*, volume 2836, pages 348–359, 2003.

[CJ05] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography*, 36(1) :33–43, 2005.

[CJLM06] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading Inversions for Multiplications in Elliptic Curve Cryptography. *Designs, Codes and Cryptography*, 39(2) :189–206, 2006.

[CJRR99] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptography − CRYPTO, LNCS*, volume 1666, pages 398–412, 1999.

[CKQ03] J. Cathalo, F. Koeune, and J.-J. Quisquater. A New Type of Timing Attack : Applications to GPS. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2779, pages 291–303, 2003.

[Cla07] C. Clavier. *De la sécurité physique des crypto-systèmes embarqués*. Thèse de Doctorat, Université de Versailles Saint-Quentin-en-Yvelines, 2007.

[CM04] B. Chevallier-Mames. Self-Randomized Exponentiation Algorithms. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2964, pages 236–249, 2004.

[CMCJ04] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis : Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6) :760–768, 2004.

[CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In *Proc. ASIACRYPT, LNCS*, volume 1514, pages 51–65, 1998.

[Cor99] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 1717, pages 292–302, 1999.

[Cra92] R. Crandall. Method and Apparatus for Public Key Exchange in a Cryptographic System. US Patent 5159632. 1992.

[CRR02] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *Proc. Cryptographic Hardware and Embedded Systems – CHES, LNCS*, volume 2523, pages 13–28, 2002.

[CT05] H. Choukri and M. Tunstall. Round Reduction using Faults. In *Proc. Fault Diagnosis and Tolerance in Cryptography – FDTC*, pages 13–24, 2005.

[CWF07] H. Chen, W. Wu, and D. Feng. Differential Fault Analysis on CLEFIA. In *Information and Communication Security – ICICS, LNCS*, volume 4861, pages 284–295, 2007.

[DBJ04] I. Déchène, É. Brier, and M. Joye. Unified Point Addition Formulae for Elliptic Curve Cryptosystems. In *Embedded Cryptographic Hardware : Methodologies and Architectures – Nova Science Publishers*, pages 247–256, 2004.

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22 :644–654, 1976.

[DI06] C. Doche and L. Imbert. Extended Double-Base Number System with Applications to Elliptic Curve Cryptography. In *Progress in Cryptology – INDOCRYPT, LNCS*, volume 4329, pages 335–348, 2006.

[DIK06] C. Doche, T. Icart, and D. R. Kohel. Efficient Scalar Multiplication by Isogeny Decompositions. In *Proc. Public Key Cryptography – PKC, LNCS*, volume 3958, pages 191–206, 2006.

[DIM05] V. Dimitrov, L. Imbert, and P. K. Mishra. Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains. In *Advances in Cryptology – ASIACRYPT, LNCS*, volume 3788, pages 59–78, 2005.

[DJM98] V. Dimitrov, G. A. Jullien, and W. C. Miller. An Algorithm for Modular Exponentiation. *Information Processing Letters*, 66(3) :155–159, 1998.

[DKL+00] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, and J.-L. Willems. A Practical Implementation of the Timing Attack. In *Proc. Smart Card Research and Advanced Application – CARDIS, LNCS*, volume 1820, pages 167–182, 2000.

[DMKP05] A. Daly, W. P. Marnane, T. Kerins, and E. M. Popovici. An FPGA Implementation of a GF(p) ALU for Encryption Processors. *Microprocessors and Microsystems (Special Issue on FPGAs : Applications and Designs) – Elsevier*, 28(5–6) :253–260, 2005.

[Duq07] S. Duquesne. Improving the Arithmetic of Elliptic Curves in the Jacobi Model. *Information Processing Letters – Elsevier*, 104(3) :101–105, 2007.

[Eck85] W. Van Eck. Electromagnetic Radiation from Video Display Units : An Eavesdropping Risk ? *Computers and Security*, 4 :269–286, 1985.

[Edw07] H. M. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American Mathematical Society*, 44 :393–422, 2007.

[EH03] N. Ebeid and M. A. Hasan. Analysis of DPA Countermeasures Based on Randomizing the Binary Algorithm. *Research Report CORR-03-14, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada*, 2003.

[EL87] M. D. Ercegovac and T. Lang. On-the-Fly Conversion of Redundant into Conventional Representations. *IEEE Transactions on Computers*, 36(7) :895–897, 1987.

[EL04] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Elsevier, 2004.

[ELM03] K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2612, pages 343–354, 2003.

[Eve90] S. Even. Systolic Modular Multiplication. In *Advances in Cryptology − CRYPTO, LNCS*, volume 537, pages 619–624, 1990.

[EW93] S. E. Eldridge and C. D. Walter. Hardware Implementation of Montgomery's Modular Multiplication Algorithm. *IEEE Transactions on Computers*, 42(6) :693–699, 1993.

[Fan06] H. Fanet. *Micro et nano-électronique : Bases, composants, circuits*. Dunod, 2006.

[FF07] O. Faurax and J. Francq. Security of several AES Implementations against Delay Faults. In *Proc. Nordic Workshop on Secure IT Systems − NordSec*, pages 61–72, 2007.

[FGKS02] W. Fischer, C. Giraud, E. W. Knudsen, and J.-P. Seifert. Parallel Scalar Multiplication on General Elliptic Curves over $\mathbb{F}_p$ Hedged against Non-Differential Side-Channel Attacks. *IACR e-Print Archive 007*, 2002.

[FIP93] Data Encryption Standard (DES), Federal Information Processing Standards Publication 46–2, National Institute of Standards and Technology. 1993.

[FIP00] Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186–2, National Institute of Standards and Technology. 2000.

[FIP01] Announcing the Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, National Institute of Standards and Technology. 2001.

[FMPV04] P.-A. Fouque, F. Muller, G. Poupard, and F. Valette. Defeating Countermeasures Based on Randomized BSD Representations. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 312–327, 2004.

[FP99] W. Freking and K. Parhi. A Unified Method for Iterative Computation of Modular Multiplications and Reduction Operations. In *Proc. International Conference on Computer Design − ICCD*, pages 80–87, 1999.

[FP02] W. Freking and K. Parhi. Performance-Scalable Array Architectures for Modular Multiplication. *The Journal of VLSI Signal Processing*, 31(2) :101–116, 2002.

[FRVD08] P.-A. Fouque, D. Réal, F. Valette, and M. Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 5154, pages 198–213, 2008.

[FT82] E. Fredkin and T. Toffoli. Conservative Logic. *International Journal of Theoretical Physics*, 21(3–4) :219–253, 1982.

[FV03] P.-A. Fouque and F. Valette. The Doubling Attack − Why Upwards Is Better Than Downwards. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2779, pages 269–280, 2003.

[GACG08] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta. Parralel Crypto-Devices for GF(p) Elliptic Curve Multiplication Resistant against Side-Channel Attacks. *Computers and Electrical Engineering − Elsevier*, 35(2) :329–338, 2008.

[GG02] C. Gebotys and R. Gebotys. Secure Elliptic Curve Implementations : An Analysis of Resistance to Power-Attacks in a DSP. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 114–128, 2002.

[GHM89] A. Guyot, Y. Herreros, and J.-M. Muller. JANUS, an On-line Multiplier/Divider for Manipulating Large Numbers. In *Proc. IEEE Symposium on Computer Arithmetic − ARITH*, pages 106–111, 1989.

[GMO01] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis : Concrete Results. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 251–261, 2001.

[Gou03] L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Proc. Public Key Cryptography − PKC, LNCS*, volume 2567, pages 199–211, 2003.

[GP08] T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 5154, pages 62–78, 2008.

[Gro01] J. Grossschädl. A Bitserial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m). In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 206–223, 2001.

[GS92] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2nd edition, 1992.

[GSS08] G. Gaubatz, E. Savas, and B. Sunar. Sequential Circuit Design for Embedded Cryptographic Applications Resilient to Adversarial Faults. *IEEE Transactions on Computers*, 57(1) :126–138, 2008.

[GT04] C. Giraud and H. Thiebeauld. A Survey on Fault Attacks. In *Proc. Smart Card Research and Advanced Application − CARDIS, IFIP*, volume 153, pages 159–176, 2004.

[GT07] P. Gaudry and E. Thomé. The $\mathsf{mp}\mathbb{F}_p$ Library and Implementing Curve-Based Key Exchanges. In *Software Performance Enhancement for Encryption and Decryption − SPEED*, pages 49–64, 2007.

[Gue02] S. Gueron. Enhanced Montgomery Multiplication. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 46–56, 2002.

[HM02a] J. Ha and S. Moon. Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 551–563, 2002.

[HM02b] Y. Hitchcock and P. Montague. A New Elliptic Curve Scalar Multiplication Algorithm to Resist Simple Power Analysis. In *Proc. Information Security and Privacy, LNCS*, volume 2384, pages 214–225, 2002.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[IIT02] K. Itoh, T. Izu, and M. Takaneka. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 129–143, 2002.

[IIT03] K. Itoh, T. Izu, and M. Takaneka. A Practical Countermeasure against Address-Bit Differential Power Analysis. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2779, pages 382–396, 2003.

[IIT04] K. Itoh, T. Izu, and M. Takenaka. Efficient Countermeasures against Power Analysis for Elliptic Curve Cryptosystems. In *Proc. Smart Card Research and Advanced Application − CARDIS, IFIP*, volume 153, pages 99–114, 2004.

[Imb08] L. Imbert. *Arithmexotiques*. Habilitation à diriger des recherches, Université Montpellier 2, 2008.

[IMI92a] K. Iwamura, T. Matsumoto, and H. Imai. High-Speed Implementation Methods for RSA Scheme. In *Advances in Cryptology − EUROCRYPT, LNCS*, volume 658, pages 221–238, 1992.

[IMI92b] K. Iwamura, T. Matsumoto, and H. Imai. Systolic-Arrays for Modular Exponentiation using Montgomery Method. In *Advances in Cryptology − EUROCRYPT, LNCS*, volume 658, pages 477–481, 1992.

[IMT02] T. Izu, B. Möller, and T. Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side-Channel Attacks. In *Progress in Cryptology − INDOCRYPT, LNCS*, volume 2551, pages 296–313, 2002.

[IT02a] T. Izu and T. Takagi. A Fast Parallel Elliptic Curve Multiplication Resistant Against Side-Channel Attacks. In *Proc. Public Key Cryptography − PKC, LNCS*, volume 2274, pages 280–296, 2002.

[IT02b] T. Izu and T. Takagi. Fast Elliptic Curve Multiplication with SIMD Operations. In *Information and Communication Security − ICICS, LNCS*, volume 2513, pages 217–230, 2002.

[IT03] T. Izu and T. Takagi. Exceptional Procedure Attack on Elliptic Curve Cryptosystems. In *Proc. Public Key Cryptography − PKC, LNCS*, volume 2567, pages 224–239, 2003.

[IYTT02] K. Itoh, J. Yajima, M. Takaneka, and N. Torii. DPA Countermeasures by Improving the Window Method. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 303–317, 2002.

[JMR07] M. Joye, P. Manet, and J.-B. Rigaud. Strengthening Hardware AES Implementations against Fault Attacks. *IET Information Security*, 1(3) :106–110, 2007.

[Joy95] M. Joye. Introduction élémentaire à la théorie des courbes elliptiques. *Technical report, CG-1995/1, UCL Crypto Group, Louvain-la-Neuve*, 1995.

[Joy02] M. Joye. Recovering Lost Efficiency of Exponentiation Algorithms on Smart Cards. *Electronics Letters*, 38(19) :1095–1097, 2002.

[Joy04] M. Joye. Smart-Card Implementation of Elliptic Curve Cryptography and DPA-type Attacks. In *Proc. Smart Card Research and Advanced Application − CARDIS, IFIP*, volume 153, pages 115–125, 2004.

[Joy05] M. Joye. *Defences against Side-Channel Analysis*. Advances in Elliptic Curve Cryptography, Cambridge University Press, 2005.

[Joy07] M. Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4727, pages 135–147, 2007.

[JPS05] M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3659, pages 293–308, 2005.

[JPY01] M. Joye, P. Paillier, and S.-M. Yen. Secure Evaluation of Modular Functions. In *Proc. International Workshop on Cryptology and Network Security*, pages 227–229, 2001.

[JQ01] M. Joye and J.-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 402–410, 2001.

[JQYY02] M. Joye, J.-J. Quisquater, S.-M. Yen, and M. Yung. Observability Analysis − Detecting when Improved Cryptosystems Fail. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2271, pages 17–29, 2002.

[JT01] M. Joye and C. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography − An Algebraic Approach. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 377–390, 2001.

[JY00] M. Joye and S.-M. Yen. Optimal Left-to-Right Binary Signed-Digit Exponent Recoding. *IEEE Transactions on Computers*, 49(7) :740–748, 2000.

[JY02] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 291–302, 2002.

[KA98] M. Kuhn and R. J. Anderson. Soft Tempest : Hidden Data Transmission using Electromagnetic Emanations. In *Information Hiding*, volume 1525, pages 124–142, 1998.

[Kah67] D. Kahn. *The Codebreakers − The Story of Secret Writing*. Scribner, 1967.

[KAJ96] C. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3) :26–33, 1996.

[Kal95] B. S. Kaliski. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8) :1064–1065, 1995.

[KHM+05] C. Kim, J. Ha, S. Moon, S.-M. Yen, W.-C. Lien, and S.-H. Kim. An Improved and Efficient Countermeasure against Power Analysis Attacks. *IACR e-Print Archive 022*, 2005.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology − CRYPTO, LNCS*, volume 1666, pages 388–397, 1999.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers by Automata. *Soviet Physics-Doklady*, 7 :595–596, 1963.

[Kob87] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177) :203–209, 1987.

[Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems. In *Advances in Cryptology − CRYPTO, LNCS*, volume 1109, pages 104–113, 1996.

[Kor94] P. Kornerup. A Systolic, Linear-Array Multiplier for a Class of Right-Shift Algorithms. *IEEE Transactions on Computers*, 43(8) :892–898, 1994.

[KQ07] C. H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA : New Attacks, New Results, and New Countermeasures. In *Proc. Workshop in Information Security Theory and Practices − WISTP, LNCS*, volume 4462, pages 215–228, 2007.

[KQ08] C. H. Kim and J.-J. Quisquater. Method for Detecting Vulnerability to Doubling Attacks. In *Information and Communication Security − ICICS, LNCS*, volume 5308, pages 97–110, 2008.

[KT05] M. E. Kaihara and N. Takagi. Bipartite Modular Multiplication. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3659, pages 201–210, 2005.

[KW03] C. Karlof and D. Wagner. Hidden Markov Model Cryptanalysis. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2779, pages 17–34, 2003.

[Lal00] P. K. Lala. *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann Publishers, 2000.

[LCC+06] T.-H. Le, J. Clédière, C. Canovas, B. Robisson, C. Servière, and J. L. Lacoume. A Proposition for Correlation Power Analysis Enhancement. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4249, pages 174–186, 2006.

[LCC08] T.-H. Le, C. Canovas, and J. Clédière. An Overview of Side Channel Analysis Attacks. *Proc. Symposium on Information, Computer and Communications Security − ASIACCS, ACM*, pages 33–43, 2008.

[LF80] R. E. Ladner and M. J. Fischer. Parallel Prefix Computation. *Journal of the ACM*, 27(4) :831–838, 1980.

[LL02] P. Leong and I. Leung. A Microcoded Elliptic Curve Processor using FPGA Technology. *IEEE Transactions on VLSI Systems*, 10(5) :550–559, 2002.

[LNVCC07] T.-H. Le, Q. T. Nguyen-Vuong, C. Canovas, and J. Clédière. Novel Approaches for Improving the Power Consumption Models in Correlation Analysis. *IACR e-Print Archive 306*, 2007.

[LS01] P.-Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 391–401, 2001.

[MDS99] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smart Cards. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 1717, pages 144–157, 1999.

[Mel07a] N. Meloni. *Arithmétique pour la Cryptographie basée sur les Courbes Elliptiques*. Thèse de Doctorat, Université Montpellier II, 2007.

[Mel07b] N. Meloni. New Point Addition Formulae for ECC Applications. In *Proc. International Workshop on the Arithmetic of Finite Fields − WAIFI, LNCS*, volume 4547, pages 189–201, 2007.

[Mes00] T. S. Messerges. Using Second Order Power Analysis to Attack DPA Resistant Software. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 1965, pages 238–251, 2000.

[Mil86] V. Miller. Use of Elliptic Curve in Cryptology. In *Advances in Cryptography − CRYPTO, LNCS*, volume 218, pages 417–426, 1986.

[Mis04] P. K. Mishra. Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 328–342, 2004.

[MM61] P. Morrison and E. Morrison. *Charles Babbage and his Calculating Engines : Selected Writings*. Dover Publications Inc., 1961.

[MMM04a] H. Mamiya, A. Miyaji, and H. Morimoto. Efficient Countermeasures against RPA, DPA, and SPA. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 343–356, 2004.

[MMM04b] C. McIvor, M. McLoone, and J. V. McCanny. FPGA Montgomery Modular Multiplication Architectures Suitable for ECCs over GF(p). In *Proc. IEEE International Symposium on Circuits and Systems − ISCAS*, volume 3, pages 509–512, 2004.

[MMS01] D. May, H. Muller, and N. P. Smart. Random Register Renaming to Foil DPA. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 28–38, 2001.

[MO90] F. Morain and J. Olivos. Speeding Up the Computations on an Elliptic Curve using Addition-Subtraction Chains. *Theoretical Informatics and Applications*, 24 :531–543, 1990.

[Möl01] B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In *Proc. Information Security, LNCS*, volume 2200, pages 324–334, 2001.

[Möl02] B. Möller. Parallelizable Elliptic Curve Point Multiplication Method with Resistance against Side-Channel Attacks. In *Proc. Information Security, LNCS*, volume 2433, pages 402–413, 2002.

[Mon85] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170) :519–521, 1985.

[Mon87] P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177) :243–264, 1987.

[MOPV07] E. De Mulder, S. B. Örs, B. Preneel, and I. Verbauwhede. Differential Power and Electromagnetic Attacks on a FPGA Implementation of Elliptic Curve Cryptosystems. *Computers and Electrical Engineering − Elsevier*, 33 :367–382, 2007.

[MSB+07] N. Mentens, K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede. A Side-Channel Attack Resistant Programmable PKC Coprocessor for Embedded Applications. In *Proc. International Conference on Systems, Architectures, MOdeling and Simulation − IC-SAMOS*, pages 194–200, 2007.

[Mul89] J.-M. Muller. *Arithmétique des ordinateurs*. Masson, 1989.

[MVO96] A. Menezes, S. Vanstone, and P. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[NS03] P. Q. Nguyen and I. E. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Designs, Codes and Cryptography*, 30 :201–217, 2003.

[NSS04] D. Naccache, J. Stern, and N. P. Smart. Projective Coordinates Leak. In *Advances in Cryptology − EUROCRYPT, LNCS*, volume 3027, pages 257–267, 2004.

[OA01] E. Oswald and M. Aigner. Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 39–50, 2001.

[OBPV08] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware Implementation of an Elliptic Curve Processor over GF(p) with Montgomery Modular Multiplier. *International Journal of Embedded Systems*, 3(4) :229–240, 2008.

[OH03] K. Okeya and D.-G. Han. Side Channel Attack on Ha-Moon's Countermeasure of Randomized Signed Scalar Multiplication. In *Progress in Cryptology − INDOCRYPT, LNCS*, volume 2904, pages 334–348, 2003.

[Ols04] L. D. Olson. Side-Channel attacks in ECC : A General Technique for Varying the Parametrization of the Elliptic Curve. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 220–229, 2004.

[Omu90] J. Omura. A Public Key Cell Design for Smart Card Chips. In *IT Workshop*, pages 983–985, 1990.

[OP01] G. Orlando and C. Paar. A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 348–363, 2001.

[Oru95] H. Orup. Simplifying Quotient Determination in High-Radix Modular Multiplication. In *Proc. IEEE Symposium on Computer Arithmetic − ARITH*, pages 193–199, 1995.

[OS00] K. Okeya and K. Sakurai. Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack. In *Progress in Cryptology − INDOCRYPT, LNCS*, volume 1977, pages 178–190, 2000.

[OS02] K. Okeya and K. Sakurai. On Insecurity of the Side Channel Attack Countermeasure using Addition-Subtraction Chains under Distinguishability between Addition and Doubling. In *Proc. Information Security and Privacy, LNCS*, volume 2384, pages 420–435, 2002.

[OS03] K. Okeya and K. Sakurai. A Simple Power Attack on Randomized Addition-Subtraction Chains Method for Elliptic Curve Cryptosystems. *IEICE Transactions*, E-86(A) :1171–1180, 2003.

[OSS04] E. Özturk, B. Sunar, and E. Savas. Low-Power Elliptic Curve Cryptography using Scaled Modular Arithmetic. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 3–26, 2004.

[Osw02] E. Oswald. Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 82–97, 2002.

[OT03] K. Okeya and T. Takagi. The Width-w NAF Method provides Small Memory and Fast Elliptic Scalar Multiplication Secure against Side Channel Attacks. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2612, pages 328–342, 2003.

[OW99] P. Van Oorschot and M. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12 :1–28, 1999.

[Par06] B. Parhami. Fault-Tolerant Reversible Circuits. In *Proc. IEEE Asilomar Conference on Signals, Systems and Computers − ACSSC*, pages 1726–1729, 2006.

[Pol78] J. Pollard. Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, 32 :918–924, 1978.

[QS01] J.-J. Quisquater and D. Samyde. Electromagnetic Analysis (EMA) : Measures and Countermeasures for Smart Cards. In *Proc. Smart Card Programming and Security, LNCS*, volume 2140, pages 200–210, 2001.

[QS02] J.-J. Quisquater and D. Samyde. Eddy Current for Magnetic Analysis with Active Sensor. In *Proc. International Conference on Research in SmartCards − E-Smart*, pages 185–194, 2002.

[Rei60] G. Reitwiesner. Binary Arithmetic. *Advances in Computers*, 1 :231–308, 1960.

[RM07] B. Robisson and P. Manet. Differential Behavioral Analysis. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4727, pages 413–426, 2007.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21 :120–126, 1978.

[SA02] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 2–12, 2002.

[Sch75] A. Schönhage. A Lower Bound for the Length of Addition Chains. *Theoretical Computer Science*, 1 :1–12, 1975.

[Sch85] R. Schoof. Elliptic Curves over Finite Fields and the Computation of Square Roots mod p. *Mathematics of Computation*, 44 :483–494, 1985.

[Sch95] R. Schoof. Counting Points on Elliptic Curves over Finite Fields. *Journal de théorie des nombres de Bordeaux*, 7 :219–254, 1995.

[Sch02] W. Schindler. A Combined Timing and Power Attack. In *Proc. Public Key Cryptography − PKC, LNCS*, volume 2274, pages 263–279, 2002.

[SEC00a] SEC 1 : Elliptic Curve Cryptography. Standard for Efficient Cryptography. The SECG Group. 2000.

[SEC00b] SEC 2 : Recommended Elliptic Curve Domain Parameters. Standard for Efficient Cryptography. The SECG Group. 2000.

[SG08] R. Szerwinski and T. Güneysu. Exploiting the Power of GPUs for Asymmetric Cryptography. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 5154, pages 79–99, 2008.

[Sha99] A. Shamir. Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attack. *United States Patent*, (5991415), 1999.

[SHCW99] C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu. An Improved Montgomery's Algorithm for High-Speed RSA Public-Key Cryptosystem. *IEEE Transactions on VLSI Systems*, 7(2) :280–284, 1999.

[SK00] E. Savas and C. K. Koç. The Montgomery Inverse−Revisited. *IEEE Transactions on Computers*, 49(7) :763–766, 2000.

[Sko06] S. P. Skorobogatov. Optically Enhanced Position-Locked Power Analysis. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4249, pages 61–75, 2006.

[Sla60] J. Slansky. Conditional Sum Addition Logic. *IRE Transactions on Electronic Computers*, EC(9) :226–231, 1960.

[Sma01] N. P. Smart. The Hessian Form of an Elliptic Curve. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2162, pages 118–125, 2001.

[Sma03] N. P. Smart. An Analysis of Goubin Refined Power Analysis Attack. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2779, pages 281–290, 2003.

[Sol99] J. Solinas. Generalized Mersenne Numbers. *Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada*, 1999.

[SPL03] S. G. Sim, D. J. Park, and P. J. Lee. New Power Analysis on the Ha-Moon Algorithm and the MIST Algorithm. In *Information and Communication Security − ICICS, LNCS*, volume 3269, pages 291–304, 2003.

[SPV06] K. Sakiyama, B. Preneel, and I. Verbauwhede. A Fast Dual-Field Modular Arithmetic Logic Unit and Its Hardware Implementation. In *Proc. IEEE International Symposium on Circuits and Systems − ISCAS*, pages 787–790, 2006.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7 :281–292, 1971.

[SST04] H. Sato, D. Schepers, and T. Takagi. Exact Analysis of Montgomery Multiplication. In *Progress in Cryptology − INDOCRYPT, LNCS*, volume 3348, pages 290–304, 2004.

[ST03] A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Transactions on Computers*, 52(4) :449–460, 2003.

[ST06] D. Stebila and N. Thériault. Unified Point Addition Formulae and Side-Channel Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4249, pages 354–368, 2006.

[STCK04] E. Savas, A. F. Tenca, M. E. Ciftcibasi, and C. K. Koç. Multiplier Architectures for GF(p) and GF(2^n). *IEE Proc. Computers and Digital Techniques*, 151(2) :147–160, 2004.

[Ste67] J. Stein. Computational Problems Associated with Racah Algebra. *Journal of Computational Physics*, 1 :397–405, 1967.

[STK00] E. Savas, A. F. Tenca, and C. K. Koç. A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m). In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 1965, pages 281–296, 2000.

[Suz07] D. Suzuki. How to Maximize the Potential of FPGA Resources for Modular Exponentiation. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 4727, pages 272–288, 2007.

[SV93] M. Shand and J. Vuillemin. Fast Implementations of RSA Cryptography. In *Proc. IEEE Symposium on Computer Arithmetic − ARITH*, pages 252–259, 1993.

[TB02] E. Trichina and A. Bellezza. Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 98–113, 2002.

[The06] N. Theriault. SPA Resistant Left-to-Right Integer Recodings. In *Proc. Selected Areas in Cryptography − SAC, LNCS*, volume 3897, pages 345–358, 2006.

[Tio98] A. A. Tiountchik. Systolic Modular Exponentiation via Montgomery Algorithm. *Electronics Letters*, 34(9) :874–875, 1998.

[Too63] A. L. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Math*, 4 :714–716, 1963.

[TSW00] W.-C. Tsai, C. B. Shung, and S.-J. Wang. Two Systolic Architectures for Modular Multiplication. *IEEE Transactions on VLSI Systems*, 8(1) :103–107, 2000.

[TT01] E. Trichina and A. A. Tiountchik. Scalable Algorithm for Montgomery Multiplication and its Implementation on the Coarse-Grain Reconfigurable Chip. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2020, pages 235–249, 2001.

[Tul86] D. M. Tullsen. A Very Large Scale Integration Implementation of an On-Line Arithmetic Unit. *Thèse de Mastère, Université de Los Angeles, USA*, 1986.

[TY92] N. Takagi and S. Yajima. Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem. *IEEE Transactions on Computers*, 41(7) :887–891, 1992.

[Van04] S. Vanstone. ECC Holds Key to Next-Gen Cryptography. *Rapport technique, Certicom©*, 2004.

[Wal92] C. D. Walter. Faster Modular Multiplication by Operand Scaling. In *Advances in Cryptology − CRYPTO, LNCS*, volume 576, pages 313–323, 1992.

[Wal93] C. D. Walter. Systolic Modular Multiplication. *IEEE Transactions on Computers*, 42(3) :376–378, 1993.

[Wal98] C. D. Walter. Exponentiation using Division Chains. *IEEE Transactions on Computers*, 47(7) :757–765, 1998.

[Wal99] C. D. Walter. Montgomery's Multiplication Technique : How to Make it Smaller and Faster. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 1717, pages 80–93, 1999.

[Wal00] C. D. Walter. An Improved Linear Systolic Array for Fast Modular Exponentiation. *IEEE Computers and Digital Techniques*, 147(5) :323–328, 2000.

[Wal02a] C. D. Walter. Breaking the Liardet-Smart Randomized Exponentiation Algorithm. In *Proc. Smart Card Research and Advanced Application − CARDIS*, pages 59–68, 2002.

[Wal02b] C. D. Walter. MIST : An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2271, pages 53–66, 2002.

[Wal02c] C. D. Walter. Precise Bounds for Montgomery Modular Multiplication and some Potentially Insecure RSA Moduli. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2271, pages 30–39, 2002.

[Wal03] C. D. Walter. Seeing through MIST given a Small Fraction of an RSA Private Key. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2612, pages 391–402, 2003.

[Wal04a] C. D. Walter. Security Constraints on the Oswald-Aigner Exponentiation Algorithm. In *Topics in Cryptology − CT-RSA, LNCS*, volume 2964, pages 208–221, 2004.

[Wal04b] C. D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 3156, pages 191–204, 2004.

[Wol02] J. Wolkerstorfer. Dual-Field Arithmetic Unit for GF(p) and GF(2^m). In *Proc. Cryptographic Hardware and Embedded Systems − CHES, LNCS*, volume 2523, pages 500–514, 2002.

[Wri87] P. Wright. *Spy Catcher : The Candid Autobiography of a Senior Intelligence Officer*. Viking Press, 1987.

[WS56] A. Weinberger and J. L. Smith. A one-Microsecond Adder using one Megacycle Circuitry. *IRE Transactions on Electronic Computers*, EC(5) :65–73, 1956.

[WT01] C. D. Walter and S. Thompson. Distinguishing Exponent Digits by Observing Modular Subtractions. In *Topics in Cryptology – CT-RSA, LNCS*, volume 2020, pages 192–207, 2001.

[Wu00] H. Wu. On Modular Reduction. *Research Report CORR-2000-36, CACR, University of Waterloo, Canada*, 2000.

[X06] Production Product Specification of Virtex-E Field Programmable Gate Arrays, Xilinx. 2006.

[XST02] Xilinx Synthesis Technology (XST) User Guide, Xilinx. 2002.

[YJ00] S.-M. Yen and M. Joye. Checking before Output May not be Enough against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9) :967–970, 2000.

[YKLM01] S.-M. Yen, S. Kim, S. Lim, and S. Moon. A Countermeasure against one Physical Cryptanalysis May Benefit Another Attack. In *Information Security and Cryptology – ICISC, LNCS*, volume 2288, pages 414–427, 2001.

[YKLM03] S.-M. Yen, S. Kim, S. Lim, and S. Moon. RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis. *IEEE Transactions on Computers*, 52(4) :461–472, 2003.

[YKMH05] S.-M. Yen, L.-C. Ko, S. Moon, and J. Ha. Relative Doubling Attack Against Montgomery Ladder. In *Information Security and Cryptology – ICISC, LNCS*, volume 3935, pages 117–128, 2005.

[YLMH05] S.-M. Yen, W.-C. Lien, S. Moon, and J. Ha. Power Analysis by Exploiting Chosen Message and Internal Collisions-Vulnerability of Checking Mechanism for RSA-Decryption. In *Progress in Cryptology – MYCRYPT, LNCS*, volume 3715, pages 183–195, 2005.

Glossaire

AES (*Advanced Encryption Standard*) : norme de chiffrement avancé.

ASIC (*Application Specific Integrated Circuit*) : circuit intégré pour application spécifique.

BDSP (*Block* DSP) : bloc DSP.

BRAM (*Block* RAM) : bloc RAM.

BS (*Borrow-Save*) : retenues conservées signées.

BSA (*Borrow-Save Addition*) : addition à retenues conservées signées.

CLB (*Configurable Logic Block*) : bloc logique programmable.

CMOS (*Complementary Metal Oxide Semiconductor*) : logique complémentaire pour semiconducteurs métal-oxyde.

CRT (*Chinese Remainder Theorem*) : théorème chinois des restes.

CS (*Carry-Save*) : retenues conservées.

CSA (*Carry-Save Addition*) : addition à retenues conservées.

CSCA (*Correlation Side-Channel Analysis*) : attaque par analyse de canaux cachés par corrélation.

CSEA (*Computational* SEA) : attaque par perturbation sans conséquence sur les calculs.

DBA (*Differential Behavioral Analysis*) : attaque différentielle par analyse de comportement.

DBNS (*Double-Base Number System*) : système de numération à double base.

DES (*Data Encryption System*) : système de chiffrement des données.

DFA (*Differential Fault Analysis*) : attaque différentielle par injection de fautes.

DSCA (*Differential Side-Channel Analysis*) : attaque différentielle par analyse de canaux cachés.

DSP (*Digital Signal Processor*) : processeur de signal numérique.

ECC (*Elliptic Curve Cryptography*) : cryptographie basée sur les courbes elliptiques.

ECDH (*Elliptic Curve Diffie-Hellman*) : processus de Diffie-Hellman pour courbes elliptiques.

ECDLP (*Elliptic Curve Discrete Logarithm Problem*) : problème du logarithme discret sur courbes elliptiques.

ECDSA (*Elliptic Curve Digital Signature Algorithm*) : algorithme de signature numérique sur courbes elliptiques.

ECIES (*Elliptic Curve Integrated Encryption System*) : système de chiffrement intégré pour courbes elliptiques.

ECMQV (*Elliptic Curve Menezes-Qu-Vanstone*) : protocole pour courbes elliptiques de Menezes-Qu-Vanstone.

EM : électromagnétique.

EMA (*ElectroMagnetic Analysis*) : attaque par analyse du rayonnement électromagnétique.

FA (*Full-Adder*) : cellule d'addition complète.

FPGA (*Field Programmable Gate Array*) : réseau de portes programmables.

FRG (*Fredkin Gate*) : porte de Fredkin.

GA (*Glitch Attack*) : attaque par modification de la fréquence d'horloge du circuit.

GPU (*Graphics Processing Unit*) : processeur graphique.

GRA (*Goubin's Refined Attack*) : attaque raffinée de Goubin.

HDL (*Hardware Description Language*) : langage de description matérielle.

HODSCA (*High Order Differential Side-Channel Analysis*) : attaque par analyse de canaux cachés d'ordre supérieur.

HOFA (*High Order Fault Analysis*) : attaque en fautes d'ordre supérieur.

IOB (*Input/Output Block*) : bloc d'entrée/sortie.

LC (*Logic Cell*) : cellule logique.

LUT (*Look-Up Table*) : table.

MSEA (*Memory* SEA) : attaque par perturbation sans conséquence visant des emplacements mémoires.

NAF (*Non-Adjacent Form*) : forme non-adjacente.

NIST (*National Institute of Standards and Technology*) : institut américain des standards et de la technologie.

NSA (*National Security Agency*) : agence de sécurité nationale américaine.

PA (*Power Analysis*) : attaques utilisant la mesure de la consommation électrique.

PGCD : plus Grand Commun Diviseur.

PP (*Partial Product*) : produit partiel.

PPLG (*Parity-Preserving Logic Gate*) : porte logique préservant la parité.

PPM : plus Plus Moins.

PSCA (*Partitioning Side-Channel Analysis*) : attaque par analyse de canaux cachés par partitionnement.

RAM (*Random Access Memory*) : mémoire synchrone à accès direct.

RF (*Random Fault*) : modèle de faute aléatoire.

RSA : Rivest Shamir Adleman.

SA (*Spike Attack*) : attaque par injection de pics (de courant ou de tension).

SCFA (*Sign-Change Fault Attack*) : attaque en fautes par changement de signe.

SSCA (*Simple Side-Channel Analysis*) : attaque simple par analyse de canaux cachés.

TA (*Timing Attack*) : attaque par analyse du temps de calcul.

UA : Unité Arithmétique.

VHDL (*Very high speed integrated circuits* HDL) : langage de description matérielle de circuits intégrés à très hautes vitesses.

Title and Abstract

Title

Design and Securization of High-Performance Arithmetic Units for ECC (Ph.D. in Computer Science earned in 2009 from the Université Montpellier 2).

Abstract

Elliptic Curve Cryptography (ECC) is more and more used in public-key cryptosystems, especially because it delivers the highest strength-per-bit of any public-key cryptography system known today. Consequently, ECC-based cryptosystems are smaller than RSA-based cryptosystems, thus ECC is more convenient for very contrained circuits (e.g. smart cards). Besides, ECC has gained some benefits from the improvement of computer and curve arithmetic, which helps it to be a viable alternative to RSA in the industrial world. Although cryptosystem designers must improve continuously the performance of their devices, they must also protect them against physical attacks which can be a real threat for their security. Indeed, some efficient attacks called "side-channel" and "fault" attacks have been intensively developed. Thus, cryptosystem designers must embed some countermeasures to these attacks. Nevertheless, attention must be paid that these countermeasures must not add new vulnerabilities to the device and should induce a limited overhead to its global performance.

It has been proposed during this Ph.D. thesis a new arithmetic unit architecture for ECC. Its performance are better than most of the published designs. This is mainly due to the choice of the used number representation, which is redundant (borrow-save representation). Another contribution of this study comes from the protection of this arithmetic unit against side-channel attacks : thanks to the state-of-the-art, the proposed side-channel-protected circuit becomes the quickest published ECC arithmetic unit. Finally, the parity-preservation principle has been studied in order to prevent our design from fault attacks. This latter contribution leads to encouraging results.

Keywords

Cryptography, RSA, elliptic curves, computer arithmetic, performance, security, physical attacks, tradeoff security/performance, parity preservation.

Résumé

La cryptographie basée sur les courbes elliptiques (ECC) est de plus en plus utilisée dans les cryptosystèmes à clé publique, notamment parce qu'à niveau de sécurité équivalent, la taille nécessaire des clés ECC est nettement inférieure à ce que son prédecesseur, le RSA, requiert. L'ECC conduit donc à implanter des circuits plus compacts que pour le RSA, ce qui indique qu'elle est plus adaptée aux circuits fortement contraints (cartes à puce, etc.). L'ECC a d'ailleurs bénéficié de l'amélioration continue de l'arithmétique (des ordinateurs et des courbes) ces dernières années, ce qui lui permet de se positionner comme un remplaçant crédible au RSA dans le monde industriel. Il est vrai qu'un concepteur de circuits cryptographiques doit chercher à améliorer les performances de son cryptosystème, mais il doit également protéger ce dernier contre des attaques physiques pouvant compromettre sa sécurité. En effet, des attaques efficaces dites « par observation » et « par perturbation » ont été mises en évidence. Le concepteur de circuits cryptographiques doit donc implanter des parades à ces attaques, également appelées contre-mesures. Cependant, l'ajout de ces contre-mesures ne doit pas d'une part ajouter de nouvelles vulnérabilités au cryptosystème, et d'autre part diminuer drastiquement ses performances.

Ces travaux de thèse proposent une nouvelle architecture d'unité arithmétique pour l'ECC. Il se trouve que les performances de cette dernière sont meilleures que la plupart de celles présentes dans la littérature. Ceci est essentiellement dû à l'utilisation d'une représentation redondante des nombres, appelée représentation à retenues signées. Le second résultat principal de ces travaux provient de la protection de cette unité contre les attaques par observation à l'aide de l'état de l'art : ce faisant, nous proposons là encore la solution la plus performante de la littérature. Enfin, nous avons exploré la possibilité de protéger notre circuit contre les attaques par perturbation à l'aide du principe de la préservation de la parité. Cette dernière contribution amène des résultats encourageants.

Discipline

Thèse de Doctorat en Informatique obtenue à l'Université Montpellier 2 en 2009.

Mots-clés

Cryptographie, RSA, courbes elliptiques, arithmétique des ordinateurs, performances, sécurité, attaques physiques, compromis sécurité/performances, préservation de la parité.

Intitulés et adresses des laboratoires

– Centre de Microélectronique de Provence - Georges Charpak, Laboratoire Systèmes et Architectures Sécurisés (SAS), 880 avenue de Mimet, 13541 Gardanne.
– LIRMM, Équipe ARITH, UMR 5506 - CC477, 161 rue Ada, 34095 Montpellier Cedex 5.