Support logiciel robuste aux attaques passives et actives pour l'arithmétique de la cryptographie asymétrique sur des (très) petits coeurs de calcul

Audrey Lucas

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
Mathématique et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Audrey LUCAS

Support logiciel robuste aux attaques passives et actives pour l'arithmétique de la cryptographie asymétrique sur des (très) petits cœurs de calcul

Thèse présentée et soutenue à LANNION , le 19 Décembre 2019
Unité de recherche : IRISA UMR 6074

Rapporteurs avant soutenance :

Roselyne CHOTIN	Maître de Conférences, Sorbonne Université
Laurent-Stéphane DIDIER	Professeur, Université de Toulon, IMATH

Composition du jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition ne comprend que les membres présents

Président :	Emmanuel CASSEAU	Professeur, Université de Rennes 1 - ENSSAT, IRISA
Examinateurs :	Roselyne CHOTIN	Maître de Conférences, Sorbonne Université
	Laurent-Stéphane DIDIER	Professeur, Université de Toulon, IMATH
Dir. de thèse :	Arnaud TISSERAND	Directeur de Recherche CNRS, Las-STICC
Invité(s)	Benoît GÉRARD	Expert, DGA

REMERCIEMENTS

Après plusieurs années, un chapitre de ma vie se termine au moment d'écrire ces lignes. Ce fut une expérience enrichissante, tant sur le plan professionnel que personnel. Pour cela, je remercie mon directeur de thèse, Arnaud Tisserand, pour m'avoir proposé ce sujet de thèse et pour son encadrement. Grâce à ses conseils, j'ai pu m'épanouir professionnellement et développer plusieurs de mes qualités.

J'adresse également toute ma reconnaissance aux membres du jury : Emmanuel Casseau qui présidât le jury, les rapporteurs, Roselyne Chotin et Laurent-Stéphane Didier et enfin, Benoît Gérard qui fut invité.

Durant cette thèse, j'ai eu l'occasion de rencontrer de nombreuses personnes qui par leur présence m'ont beaucoup apporté. Je tiens, évidemment, à remercier l'équipe CAIRN dans laquelle j'ai réalisé ma thèse, mais aussi l'équipe GRANIT, qui fut également très présente. C'était un véritable plaisir de venir travailler chaque jour. Je pourrais citer bien des personnes, mais ne pouvant pas mettre tout le monde, je m'abstiendrais afin de ne pas faire de jaloux. Bien sûr, j'ai une pensée toute particulière pour le groupe de HIIT qui perdure malgré l'abandon de certains (ils se reconnaîtront, à n'en pas douter). Plus récemment, j'ai intégré une nouvelle équipe. Je tiens à les remercier, et particulier Bernard, de leur bienveillance, de leur compréhension et de leur soutien durant cette dernière ligne droite.

Ensuite, j'aimerais aussi remercier mes amis et ma famille qui m'ont encouragée. Des amis qui m'ont (surtout) permis de décompresser à de très nombreuses reprises et dont certaines ont fait le déplacement pour assister à ma soutenance. J'aimerais tout particulièrement remercier Cyrielle, mon éternelle binôme. Les discussions quotidiennes, tant sur le plan personnel et professionnel, ont égaillées mes journées et m'ont débloquée moult fois.

Merci à ma famille, entre autres choses, pour leur indulgence et leur compréhension vis-à-vis de mes absences (physiquement ou mentalement), que moi-même je ne me pardonne pas encore.

Enfin, j'aimerais remercier la personne la plus merveilleuse que je connaisse et qui partage ma vie depuis plus de 12 ans maintenant : Clément. Son soutien au quotidien comme dans les moments de doute et d'épuisement fut salvateur. Il fut mon piller durant toutes ces années et l'est encore aujourd'hui et je l'espère pour encore de très très nombreuses années.

TABLE DES MATIÈRES

LISTE DES ALGORITHMES

TABLE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE

`ADD`	Addition de points sur une courbe elliptique
AES	Advanced Encryption Standard
BS	Borrow-save
CFA	Analyse par collision
CPA	Analyse corrélée de consommation
CRT	Théorème chinois des restes
CS	Carry-save
DA	Doublement et addition
DAA	Doublement et addition toujours
DBA	Analyse comportementale différentielle
`DBL`	Doublement de point sur une courbe elliptique
`DBL+V`	Doublement de point avec vérification sur un courbe elliptique
DE	Division euclidienne
DES	Data Encryption Standard
DFA	Analyse différentielle de faute
DLP	Problème du logarithme discret
DPA	Analyse différentielle de consommation
E_E	Courbe d'Edwards
E_M	Courbe de Montgomery
E_W	Courbe de Weierstrass
ECC	Cryptographie basée sur les courbes elliptiques
ECDH	Échange de clé de Diffie-Hellman
ECDLP	Problème du logarithme discret sur les courbes elliptiques
ECDSA	Signature sur ECC pour *elliptic curve digital signature agreement*
EFD	site *explicit formulas database*
EM	Électromagnétique
FA	Attaque en faute
`fadd`	Addition sur le corps
`fmul`	Multiplication sur le corps
$\mathbb{F}_p$	Corps fini en grand caractéristique p
FSA	Analyse de la sensibilité à la faute
HW	Poids de Hamming

IC	Compteur d'itérations
ICC	Compteur d'itérations avec cswap
IM	Information mutuelle
MIA	Analyse d'information mutuelle
MLD	Échelle de Montgomery
MLD_V	Échelle de Montgomery avec vérification
NaCl	Networking and Cryptography library
NAF	Forme non-adjacente
$\mathcal{O}$	Élément neutre d'une courbe
PKI	Infrastructure à clé publique
PV	Vérification de point
RPA	Analyse raffinée de consommation
RR	Réduction de ronde
RSA	Algorithme de Rivest, Shamir et Adleman
SCA	Attaque par observation
SEA	Attaque *Safe-error*
SM	Multiplication scalaire
SPA	Analyse simple de consommation
SSA	Représentation statique à affectation unique
UAL	Unité arithmétique et logique
μ**NaCl**	Networking and Cryptography library for microcontrollers
wadd	addition de w bits
wmul	multiplication de w bits
XZ	Type de coordonnées pour les courbes de Montgomery
ZPA	Attaque du point *zero-value*

INTRODUCTION

Dans notre société actuelle, tout le monde (ou presque) utilise la *cryptographie*, parfois sans s'en rendre compte. Elle est présente au quotidien, lors d'un paiement par carte bancaire ou par internet, de l'envoi d'un email, d'utilisations de réseaux sociaux ou d'applications. Bien évidemment, elle est aussi utilisée dans d'autres contextes tels que la santé ou dans le cadre militaire. Dans l'antiquité, les chefs militaires comprenaient déjà l'importance de cacher des informations à leurs ennemis. À Babylone, les messages étaient inscrits sur le crâne d'esclaves envoyés en messagers après que les cheveux eurent repoussé. Ce type de méthode, tenant plus de la stéganographie que de la cryptographie, atteint rapidement ses limites. En effet, une fois le message découvert, rien n'empêche de le lire et de le comprendre. Pour cela, des techniques ont été développées afin de garantir que seul le (ou les) destinataire(s) légitimes(s) puisse(nt) comprendre le message. Cela correspond à la *confidentialité* des données. Le message initial est chiffré afin de devenir incompréhensible pour des tierces personnes. Lorsque le destinataire reçoit le message, il doit le déchiffrer pour le comprendre.

Bien évidemment, la confidentialité seule ne suffit pas. Par exemple, Bob (l'un des deux interlocuteurs) reçoit un message lui donnant rendez-vous dans le parc près de chez lui. Il se rend au rendez-vous, en pensant retrouver Alice, sa meilleure amie. Une fois sur place, il constate que la personne lui ayant envoyé le message n'est pas Alice mais est un tueur en série. Ainsi, savoir qui est l'émetteur d'un message est souvent capital et correspond à l'*authenticité*.

Malheureusement, la confidentialité et l'authenticité du message ne suffissent toujours pas. Par exemple, Bob envoie à Alice le message : « Je te dois 10 euros ». Une tierce personne pourrait modifier le message pour qu'Alice reçoive : « Je te dois 1000 euros ». Il est donc important d'être sûr que le message reçu est bien celui qui a été envoyé, ce qui correspond à l'*intégrité* des données.

Ces trois éléments ; confidentialité, authenticité et intégrité, sont des objectifs majeurs de la cryptographie. Cependant, avec l'utilisation d'applications numériques, telles que les signatures électroniques, le concept de non-répudiation a été développé. L'émetteur d'un message ne pourra pas nier le fait de l'avoir émis.

La cryptographie est l'ensemble des principes et méthodes dont l'application assure le chiffrement et le déchiffrement des données dans le but d'en préserver la confidentialité, l'intégrité, l'authenticité et la non-répudiation. Le développement des ordinateurs, du numérique, des systèmes de communication et des outils informatiques, rend primordiale la protection des com-

munications et des données des différents utilisateurs. C'est pourquoi, la cryptographie subit un essor considérable afin de répondre aux besoins des utilisateurs, qu'ils soient institutionnels, industriels ou privés.

La cryptographie peut se scinder en différentes catégories : *symétrique* et *asymétrique* (respectivement appelées cryptographique à clé privée et à clé publique), qui ont chacune leurs avantages et leurs inconvénients. Lorsque deux protagonistes utilisent la cryptographie symétrique, un secret commun est utilisé pour chiffrer et déchiffrer les communications. Ce secret commun est appelé la *clé secrètes*. Cette clé ne doit être connue que des participants de la discussion. Ainsi, une entité n'ayant pas cette clé ne pourra pas comprendre les communications. Ce type de cryptographie permet un chiffrement et un déchiffrement rapide. Cependant, la clé doit être échangée au préalable à chacun des participants de la communication. Les cryptosystèmes tels que DES, 3DES, AES sont des exemples connus de cryptographie symétrique.

Contrairement à la cryptographie symétrique, deux protagonistes, se servant de la cryptographie asymétrique, ne possèdent pas de secret commun. Au lieu d'avoir une seule clé, chacun d'eux détient deux clés : une *clé publique* et une *clé privée*. La clé privée est gardée précieusement par son propriétaire, alors que la clé publique est connue de tous, ce qui permet d'éviter l'échange de clés préalables. Si un interlocuteur désire envoyer un message à un autre, il lui suffit de le chiffrer avec la clé publique du destinataire. L'exemple le plus connu de cryptographie asymétrique est le cryptosystème RSA [32, 59], dont la sécurité repose sur le problème de la factorisation de grands nombres. La cryptographie basée sur les courbes elliptiques (ECC) [87, 77] est un autre exemple dont la sécurité repose sur le problème du logarithme discret (DLP). Dans les systèmes embarqués, le coût de ECC (surface de silicium et consommation d'énergie) est moindre que celui de RSA. De plus, la taille des clés nécessaires, publiques et privées, est bien plus petite que celles de RSA. Par exemple, pour assurer un niveau de sécurité de 110 bits, RSA demande des clés de taille 2048 bits, contre seulement 224 bits pour ECC. Pour cette raison, ECC tend à remplacer RSA.

Avec l'essor de la cryptographie, des méthodes se sont développées dans le but de retrouver des informations protégées sans utiliser la clé de déchiffrement. Cet ensemble de techniques est appelé *cryptanalyse*. Le protagoniste malveillant usant de la cryptanalyse est alors nommé *attaquant* ou *adversaire*. Cette entité malveillante peut avoir différents objectifs : retrouver le message (totalement ou partiellement), retrouver la clé de déchiffrement, ou encore distinguer deux messages.

Pour arriver à ses fins, l'attaquant peut cibler différents aspects du cryptosystème : l'aspect mathématique, l'aspect logique, l'aspect protocolaire et/ou l'aspect implantation logicielle ou matérielle du système. La factorisation de grands nombres est une attaque mathématique possible contre RSA. Un grand nombre d'autres attaques mathématiques existent : linéaire,

différentielle, statistique, *etc*. C'est pour cela que les algorithmes de chiffrement doivent être sûrs mathématiquement.

La bonne utilisation d'un algorithme est aussi capitale que la robustesse de l'algorithme en lui-même. En effet, une mauvaise utilisation (p. ex. mauvais choix dans les paramètres) peut permettre de réaliser des attaques protocolaires, comme l'attaque de l'homme du milieu (le tueur en série, de l'exemple plus tôt). Cet *homme du milieu* se fait passer pour un correspondant qu'il n'est pas, afin d'obtenir des informations dont il n'est pas le destinataire légitime. Ainsi le protocole, qui utilise l'algorithme sûr, doit également être sûr. Par exemple, un protocole demande de choisir un nombre m au hasard pour la suite de son fonctionnement. Ce nombre doit-il être choisi une bonne fois pour toutes, ou doit-il être généré aléatoirement pour chaque exécution du cryptosystème ? Le choix d'un m parfaitement aléatoire mais définitif est souvent une faille exploitable.

Une fois que les algorithmes et les protocoles sont définis, ils doivent être implantés. Une mauvaise implantation peut aussi être une source de vulnérabilités qu'un adversaire peut exploiter. Il est important que le protocole et l'algorithme soient bien implantés. Des attaques logicielles comme le *buffer overflow* sont aussi possibles.

Le dernier aspect attaquable est l'implantation matérielle ou logicielle du système. En effet, les algorithmes sûrs mathématiquement ne le sont pas toujours en pratique. Certains composants interagissent entre eux et peuvent être influencés par l'environnement. Ainsi, des informations sensibles peuvent se déduire du comportement du circuit. Ces attaques sont nommées les *attaques physiques*. Ces attaques, réalisables sur les circuits embarqués à cause de leur proximité avec les utilisateurs, dont certains malveillants, se déclinent en plusieurs catégories : rétro-ingénierie, observation ou *side channel attacks* (SCA) et perturbation, ou *fault attacks* (FA).

La rétro-ingénierie consiste à extraire chaque étape du système afin d'en comprendre le fonctionnement. C'est comme décortiquer une boîte noire, dont le contenu est initialement inconnu, pour obtenir une analyse détaillée de chaque étape de fabrication et de chaque élément de son contenu.

Les *attaques par observation*, aussi appelées *attaques passives*, utilisent des observations de paramètres physiques telles que le temps d'exécution, la consommation électrique ou encore le rayonnement électromagnétique. Ces observations sont mesurées durant l'exécution du système sur le circuit. Ces mesures peuvent ensuite être analysées grâce à des outils statistiques afin de déduire des relations entre les paramètres physiques mesurés et des valeurs secrètes internes.

Les *attaques par perturbation*, aussi appelées *attaques actives*, perturbent le circuit, grâce par exemple à des variations de la tension d'alimentation, des radiations électromagnétiques ou encore des illuminations laser. L'objectif de ces perturbations est d'induire un comportement

non spécifié du circuit pendant l'exécution des algorithmes. Ce comportement est ensuite exploité pour déduire des valeurs secrètes internes.

Les systèmes embarqués sont omniprésents dans notre société. Il est donc primordial de les protéger contre les différents types d'attaques. Plusieurs protections et contre-mesures existent contre les attaques physiques à différents niveaux : mathématiques (p. ex. une modification du calcul principal) ; algorithmique (p. ex. une modification de l'algorithme, ou encore l'ajout d'opérations inutiles) ; protocolaire (p. ex. l'ajout d'utilisation de valeurs aléatoires dans le protocole d'utilisation) ; matérielle (p. ex. l'ajout de détecteurs d'intrusion). En général, ces protections ne considèrent qu'un seul type d'attaque physique.

Contre les SCA, le but des protections est de faire disparaître ou de réduire fortement les dépendances entre les paramètres physiques observables et les informations sensibles. Pour atteindre cet objectif, des méthodes d'uniformisation ou de randomisation peuvent s'employer. Les protections contre les FA sont basées sur la détection et/ou la correction de l'erreur engendrée lors de l'attaque.

Comme, en général, la plupart de ces protections sont spécifiques, elles ne protègent pas simultanément des attaques SCA et FA. Par exemple, des schémas d'uniformisation, efficaces contre des SCA, ne le seront pas contre les FA. Pire, en plus de ne pas être efficaces, certaines protections contre un type d'attaque peuvent rendre l'implantation vulnérable à l'autre type d'attaque. L'ajout d'opérations inutiles, dites factices, permettant d'uniformiser les calculs, est un exemple de méthode de protection contre des SCA rendant le système vulnérable à un certain type de FA. Ces opérations peuvent rendre le système vulnérable à des attaques de type *safe-error* qui utilise le fait que ces opérations ne sont pas employées pour le calcul. Il est donc capital de tenir compte simultanément des SCA et des FA lors de l'élaboration et de l'implantation de protections, ce qui n'est généralement pas le cas dans la littérature.

L'objectif de cette thèse est d'étudier et de concevoir des *protections logicielles* pour ECC en se préoccupant simultanément des FA et des SCA. Ces protections ciblent, plus particulièrement, des dispositifs embarqués, à petit ou très petit cœur de calcul, tels que des microcontrôleurs (p. ex. Cortex-M0 32 bits ARM).

Cette thèse est composée de deux parties. Une première partie, composée de quatre chapitres, sera dédiée à l'état de l'art du domaine de la cryptographie. Le premier chapitre introduira les courbes elliptiques utilisées en cryptographie. Plus particulièrement, les courbes de Weierstrass et de Montgomery seront décrites ainsi que d'autres courbes telles que les courbes d'Edwards. De plus, ce chapitre introduira différents algorithmes de multiplication scalaire ainsi que certaines formules pour le doublement et l'addition de points.

Le deuxième chapitre décrira les principales attaques par observation présentes dans la littérature. Certaines sont spécifiques à ECC, aux cryptosystèmes asymétriques ou aux cryptosystèmes symétriques, alors que d'autres sont possibles sur les deux types de cryptosystème

(symétrique et asymétrique). Quelques contre-mesures contre ces attaques seront également détaillées dans ce chapitre dans le cadre ECC. Elles sont principalement basées sur des principes d'uniformisation et de randomisation.

Le troisième chapitre présentera des attaques par perturbation, ainsi que des attaques dites hybrides qui associent des injections de fautes à des attaques SCA. Des contre-mesures relatives à ces attaques seront également explicitées. Elles reposent essentiellement sur le principe de randomisation ou de redondance de calcul.

Le dernier chapitre de cette partie sera consacré à l'arithmétique modulaire pour des microcontrôleurs. Les opérations cryptographiques s'effectuent sur des grands nombres. Par conséquent, un microcontrôleur ne peut pas les réaliser directement, mais applique des algorithmes pour les faire. Des algorithmes de multiplication, d'addition et de réduction modulaire seront décrits dans ce chapitre.

La seconde partie de cette thèse portera sur les travaux réalisés. Cette partie sera divisée en deux chapitres. Le premier portera sur des *contre-mesures combinées* contre des attaques SCA et FA. Les protections proposées sont une uniformisation de la multiplication scalaire grâce à une *vérification de point* (PV) et un *compteur d'itération* (IC). Ces protections permettent de protéger la multiplication scalaire contre les FA tout en ayant un comportement uniforme, agissant ainsi contre les SCA du type *simple power analysis* (SPA).

Le second chapitre portera sur un *simulateur arithmétique de l'activité* théorique. L'objectif de ce simulateur est d'identifier et évaluer des fuites liées à l'arithmétique (algorithmes de calcul et représentation des nombres) lors de l'exécution de la multiplication scalaire afin de s'assurer que les protections, proposées dans le chapitre précédent, sont robustes aux FA et aux SPA.

PREMIÈRE PARTIE

État de l'art

CHAPITRE 1

CRYPTOGRAPHIE BASÉE SUR LES COURBES ELLIPTIQUES

La cryptographie basée sur les *courbes elliptiques* (ECC pour *elliptic curve cryptography*) est introduite dans les années 80 par Miller [87] et Koblitz [77]. Actuellement, ECC tend à remplacer RSA pour fournir un support de cryptographie à clé publique (PKC) en particulier dans les systèmes embarqués. En effet, pour un même niveau de sécurité, ECC est plus performant et nécessite des clés plus petites que RSA [32, 59]. Par exemple, des clés d'environ 220 bits pour ECC assurent une sécurité similaire à RSA avec des clés d'environ 2000 bits. De plus, les coûts, en surface de silicium et en énergie consommée, sont bien moindres que pour RSA.

Des protocoles d'échange de clé, de chiffrement ou encore de signature utilisent les courbes elliptiques. En particulier l'échange de clé de Diffie-Hellman (ECDH) [41] et la signature ECDSA [72] (pour *elliptic curve digital signature agreement*) sont deux protocoles couramment employés.

Lors de l'échange de clé ECDH, illustré à la figure 1.1 entre deux protagonistes Alice et Bob, la courbe E et le point P sont des paramètres publics. Alice et Bob choisissent chacun un grand entier aléatoire, respectivement a et b, et calculent respectivement aP et bP. Alice envoie aP à Bob, lui permettant de calculer $abP = b(aP)$ et Bob transmet bP à Alice, lui permettant de calculer, à son tour, $abP = a(bP)$. Ainsi, Alice et Bob partagent le même secret abP et ils peuvent communiquer en utilisant cette information secrète sans que l'interception des valeurs aP et bP par un attaquant puisse lui donner une quelconque information sur abP.

Une signature numérique permet d'assurer l'authentification de l'expéditeur d'un message. Généralement, elle consiste à chiffrer un message avec la clé privée et à le déchiffrer avec la clé publique. Par exemple, si Alice veux s'authentifier auprès de Bob (c.-à-d. Bob s'assure que Alice est bien son interlocutrice), alors elle chiffre un message m avec sa clé privée. Bob n'aura plus qu'à déchiffrer ce message (chiffré au préalable par Alice) avec la clé publique d'Alice. Le protocole ECDSA fonctionne sur ce principe de base. Néanmoins, d'autres étapes, décrites au tableau 1.1, sont réalisées durant la signature. La courbe E d'ordre l et le point P d'ordre n, sont encore des paramètres publics [59, 32].

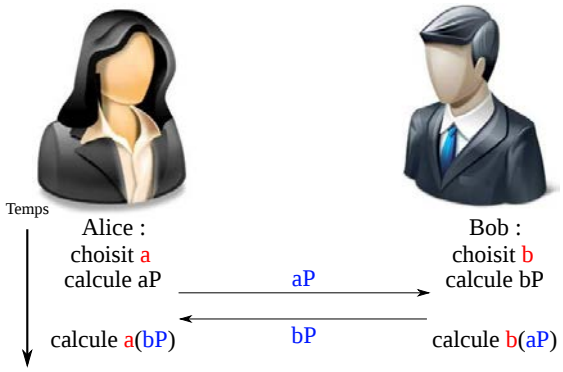


FIGURE 1.1 – Échange de clé ECDH

Signature	Vérification
Données : m est un message ; H une fonction de hachage, l l'ordre de E, n l'ordre de P, (k_A, K_A) les clés privée et publique d'Alice	
Calculer $e = H(m)$	Calculer $e = H(m)$
Choisir au hasard $r \in [1, l]$	Vérifier que $u, s \in [1, n-1]$
Calculer $Q = rP = (x_Q, y_Q)$	Calculer $w = s^{-1} \bmod n$
Ne garder que x_Q	Calculer $u_1 = ew \bmod n$ et $u_2 = uw \bmod n$
$u = x_Q \bmod n$	Calculer $Q = (x_Q, y_Q) = u_1 P + u_2 K_a$
Calculer $s = r^{-1}(e + k_A u) \bmod n$	Ne garder que x_Q
Envoyer (m, u, s)	Vérifier que $u = x_Q \bmod n$

TABLE 1.1 – Signature ECDSA [92]

L'opération principale de ECC est la *multiplication scalaire* (SM) (pour *scalar multiplication*. Cette opération consiste à effectuer la multiplication $[k]P$ sur une courbe E, où P est un point de E et k un grand nombre entier qui peut être public ou privé en fonction du protocole. ECC possède trois niveaux de calculs illustrés à la figure 1.2 : au niveau de la courbe ($[k]P$), au niveau des points de la courbe (addition et doublement de points, notés `ADD` et `DBL`), et enfin au niveau du corps fini sur lequel la courbe est définie (p. ex. addition, soustraction, multiplication et inversion dans le corps).

La sécurité de ECC repose sur le *problème du logarithme discret sur les courbes elliptiques* (ECDLP). Pour ce problème, il est toujours actuellement infaisable en pratique de retrouver k à partir d'un couple de points $(P, Q = [k]P)$.

Dans cette partie, les courbes elliptiques de Weierstrass et de Montgomery seront définies. Seules des courbes sur $\mathbb{F}_p$ seront décrites et utilisées dans ce document. De plus, certaines formules d'addition (`ADD`) et de doublement (`DBL`) de points et des algorithmes de SM seront introduits ainsi que leurs coûts de calcul. Ces coûts sont évalués en terme de nombre d'opérations sur $\mathbb{F}_p$: multiplication `M`, carré `S` et addition/soustraction `A`.

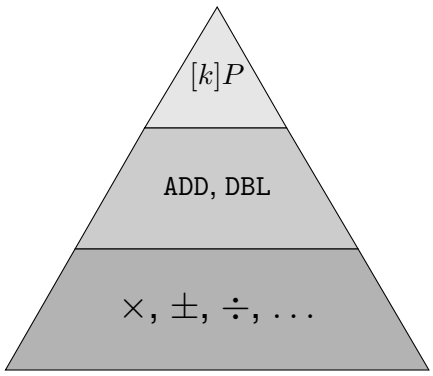


FIGURE 1.2 – Niveaux de calculs sur ECC

1.1 Courbes elliptiques

Différents types de courbes elliptiques existent dans la littérature. Ces courbes ont des propriétés qui varient en fonction de leurs types. De plus, certaines sont plus performantes que d'autres en terme de vitesse de calcul d'une SM. Dans cette section, certaines courbes seront décrites, en particulier les courbes de Weierstrass et de Montgomery sur $\mathbb{F}_p$, ainsi que différents types de coordonnées de points de ces courbes et différentes formules d'addition et de doublement de points.

1.1.1 Coordonnées des points d'une courbe elliptique

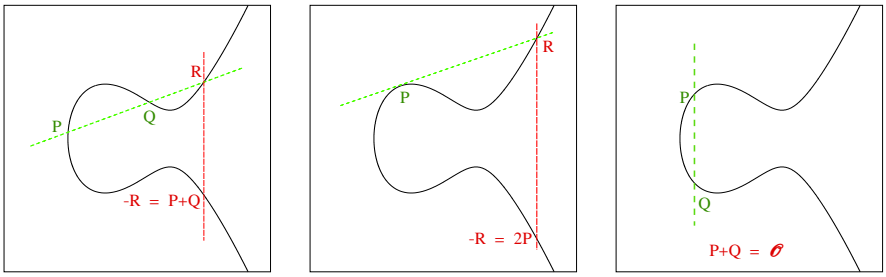
Un point d'une courbe est défini par ses *coordonnées*. Différentes représentations, ou types de coordonnées, existent pour écrire les points d'une courbe. Les *coordonnées affines* sont la plus basique d'entre-elles et consistent à représenter un point par le couple (x, y) où $x, y \in \mathbb{F}_p$. En général, elles ne sont pas utilisées pour des implantations pratiques car une inversion est nécessaire lors de `ADD` et de `DBL`, les rendant coûteuses.

D'autres types de coordonnées permettent d'éviter ces inversions durant la SM. Toutes ne seront pas décrites. Lorsque les coordonnées x et y, d'un point, sont remplacées par $\frac{x}{z}$ et $\frac{y}{z}$, le triplé (x, y, z) correspond au point représenté en *coordonnées projectives*.

Les *coordonnées* XZ sont similaires aux coordonnées projectives à la seule différence que la coordonnée y est ignorée. Par conséquent, un point P est représenté par le couple (x, z).

Les *coordonnées jacobiennes* sont également constituées d'un triplé (x, y, z) et sont obtenues en substituant, respectivement, les coordonnées x et y par $\frac{x}{z^2}$ ou $\frac{y}{z^3}$.

Tous ces représentations de points, mis à part les coordonnées affines, permettent d'éviter les inversions présentes dans `ADD` et `DBL`, les rendant plus performantes. Une normalisation des coordonnées x et y permet de revenir en coordonnées affines, à partir des coordonnées projectives, jacobiennes et XZ.



(a) $P + Q$ (b) $2P$ (c) $P - P = \mathcal{O}$

FIGURE 1.3 – Illustration graphique de la Loi de groupe des points d'une courbe elliptique sur $\mathbb{R}$

1.1.2 Courbes de Weierstrass

Les *courbes de Weierstrass courtes* [59, 32] E_W sur $\mathbb{F}_p$ sont définies par l'équation :

$$E_W : y^2 = x^3 + ax + b. \tag{1.1}$$

Les coordonnées x et y dans $\mathbb{F}_p$ correspondent à un point de la courbe en coordonnées affines. Les variables a et b, définies dans $\mathbb{F}_p$, sont des paramètres qui définissent la courbe et qui respectent des conditions spécifiques [59, 32].

Le choix de ces paramètres est capital. En effet, un mauvais choix de ces paramètres peut engendrer une courbe moins robuste que d'autres [10]. Dans certains cas, un choix judicieux permet d'avoir une SM plus efficace (p. ex. $a = -3$).

L'ensemble défini par $E_W(\mathbb{F}_p)$ est l'ensemble des points (x, y) qui satisfont l'équation 1.1 et auquel le point $\mathcal{O}$ est ajouté. Ce point est appelé l'*élément neutre* de la courbe. L'ensemble $E_w(\mathbb{F}_p)$ possède une structure de groupe. La figure 1.3 est la représentation géométrique cette loi de groupe sur $\mathbb{R}$ où deux cas de figure doivent être discernés : $P + Q$ et $2Q$. De plus, $P + Q$ n'est possible que si $Q \neq \pm P$. En général, des formules différentes sont nécessaires pour effectuer ces deux opérations. Dans le premier cas, pour faire $P + Q$, une formule d'addition (`ADD`) de points est utilisée. Dans le second cas, une formule de doublement (`DBL`) de point est employée afin de calculer $2Q$.

La plupart des formules pour effectuer `ADD` et `DBL` sont disponibles sur le site de *explicit formulas database* (EFD) [9]. Seules les formules de `ADD` et `DBL` pour les coordonnées projectives et jacobiennes sont rappelées au tableau 1.2.

Coord.	Jacobiennes	Projectives
	$X = \frac{x}{z^2}, Y = \frac{y}{z^3}$	$X = \frac{x}{z}, Y = \frac{y}{z}$
ADD	$ZZ_1 = Z_1^2$	$U_1 = X_1 \cdot Z_2$
	$ZZ_2 = Z_2^2$	$U_2 = X_2 \cdot Z_1$
	$U_1 = X_1 \cdot ZZ_2$	$S_1 = Y_1 \cdot Z_2$
	$U_2 = X_2 \cdot ZZ_1$	$S_2 = Y_2 \cdot Z_1$
	$S_1 = Y_1 \cdot ZZ_2Z_2$	$ZZ = Z_1 \cdot Z_2$
	$S_2 = Y_2 \cdot ZZ_1Z_1$	$T = U_1 + U_2$
	$H = U_2 - U_1$	$TT = T^2$
	$I = (2 \cdot H)^2$	$M = S_1 + S_2$
	$J = H \cdot I$	$R = TT - U_1 \cdot U_2 + a \cdot ZZ^2$
	$r = 2(S_2 - S_1)$	$F = ZZ \cdot M$
	$V = U_1 \cdot I$	$L = M \cdot F$
	$X_3 = r^2 - J - 2 \cdot V$	$LL = L^2$
	$S = 2 \cdot S_1 \cdot J$	$G = (T + L)^2 - TT - LL$
	$Y_3 = r \cdot (V - X_3) - S$	$W = 2 \cdot R^2 - G$
	$T = ZZ_1 + ZZ_2$	$X_3 = 2 \cdot F \cdot W$
	$U = (Z_1 + Z_2)^2$	$Y_3 = R \cdot (G - 2 \cdot W) - 2 \cdot LL$
	$Z_3 = (U - T) \cdot H$	$Z_3 = 4 \cdot F^3$
DBL	$XX = X_1^2$	$XX = X_1^2$
	$YY = Y_1^2$	$ZZ = Z_1^2$
	$t_0 = YY^2$	$w = a \cdot ZZ + 3 \cdot XX$
	$ZZ = Z1^2$	$s = 2 \cdot Y_1 \cdot Z_1$
	$t_1 = (X1 + YY)^2$	$ss = s^2$
	$S = 2 \cdot (t_1 - XX - t_0)$	$Z_3 = s \cdot ss$
	$M = 3 \cdot XX + a \cdot ZZ^2$	$R = Y_1 \cdot s$
	$X_3 = M^2 - 2 \cdot S$	$RR = R^2$
	$U = S - X_3$	$B = (X_1 + R)^2 - XX - RR$
	$Y_3 = M \cdot U - 8 \cdot t_0$	$h = w^2 - 2 \cdot B$
	$V = YY + ZZ$	$X_3 = h \cdot s$
	$Z_3 = (Y_1 + Z_1)^2 - V$	$Y_3 = w \cdot (B - h) - 2 \cdot RR$

TABLE 1.2 – Formules de ADD et DBL pour les courbes de Weierstrass courtes [9]

1.1.3 Courbes de Montgomery

Les *courbes de Montgomery* [88] sont définies par l'équation 1.2, avec a et b des paramètres dans $\mathbb{F}_p$ qui respectent des conditions spécifiques [59, 32] :

$$E_M : by^2 = x^3 + ax^2 + x. \tag{1.2}$$

La représentation de coordonnées XZ est la plus employée avec les courbes de Montgomery. Elle consiste à mettre les coordonnées du point en projectif et à ignorer la coordonnée y. Un point affine $P = (x, y)$ se transforme en coordonnées XZ en calculant $x = \frac{X}{Z}$. Le point P devient alors $P = (X, Z)$.

Comme pour les courbes de Weierstrass, différentes formules, disponibles sur le site EFD [9], sont employées pour calculer `ADD` et `DBL`. Dans cette section, seules les formules pour la représentation XZ sont rappelées à l'équation 1.3, où la valeur de $a_{24} = \frac{a+2}{4}$ est précalculée au début de la SM. Lors de l'addition de points P_1 et P_2, la variable x_P est la coordonné x de $P_2 - P_1$.

$$\texttt{DBL} : \begin{cases} A &= (x_1 + z_1)^2 \\ B &= (x_1 - z_1)^2 \\ u_0 &= A - B \\ x_3 &= AB \\ z_3 &= u_0(B + a_{24}u_0) \end{cases} \qquad \texttt{ADD} : \begin{cases} u_0 &= (x_2 - z_2)(x_1 + z_1) \\ u_1 &= (x_2 + z_2)(x_1 - z_1) \\ x_3 &= (u_0 + u_1)^2 \\ z_3 &= x_P(u_0 - u_1)^2 \end{cases} \tag{1.3}$$

1.1.4 Autres types de courbes

Les courbes de Weierstrass et de Montgomery ne sont pas les seuls types de courbes elliptiques existants. D'autres types de courbes ont des caractéristiques particulières qui peuvent avoir des avantages en matière de protection contre des attaques physiques. Cela est le cas des courbes d'Edwards (E_E) [44] qui ont des formules uniformes pour `ADD` et `DBL`. La séquence de calculs pour effectuer l'opération $P + Q$ est également utilisée pour réaliser l'opération $2P$.

Les courbes d'Edwards sont définies sur $\mathbb{F}_p$ par l'équation 1.4, où c et d sont des paramètres dans $\mathbb{F}_p$ définissant la courbe et qui respectent des conditions spécifiques ($cd(1 - c^4d) \neq 0$ et d n'est pas un carré).

$$E_E : x^2 + y^2 = c^2(1 + dx^2y^2) \tag{1.4}$$

Les coûts de certaines formules de `ADD` et `DBL`, pour les courbes décrites plus tôt, sont récapitulés au tableau 1.3. Les courbes de Montgomery sont les plus performantes en nombre d'opérations sur $\mathbb{F}_p$. Les courbes d'Edwards sont plus coûteuses que ces dernières, mais ont

	ADD	DBL
Weierstrass, Proj.	$11\mathtt{M} + 6\mathtt{S} + 16\mathtt{A} + 1 \times a$	$5\mathtt{M} + 6\mathtt{S} + 12\mathtt{A} + 1 \times a$
Weierstrass, Jac.	$11\mathtt{M} + 5\mathtt{S} + 13\mathtt{A}$	$1\mathtt{M} + 8\mathtt{S} + 16\mathtt{A} + 1 \times a$
Edwards, Proj.	$10\mathtt{M} + 1\mathtt{S} + 7\mathtt{A} + 1 \times c + 1 \times d$	$3\mathtt{M} + 4\mathtt{S} + 6\mathtt{A} + 3 \times c$
Montgomery XZ	$3\mathtt{M} + 2\mathtt{S} + 6\mathtt{A}$	$2\mathtt{M} + 2\mathtt{S} + 4\mathtt{A} + 1 \times a_{24}$

TABLE 1.3 – Coût des opérations sur $\mathbb{F}_p$, où S=1M (tirée de [9])

l'avantage d'avoir des formules uniformes pour ADD et DBL. Les courbes de Weierstrass, qui sont utilisées dans des normes (p. ex. FIPS 186-4 [1] du NIST), sont les plus coûteuses.

1.2 Multiplication scalaire

La *multiplication scalaire* (SM) est l'opération principale dans les protocoles ECC à la fois en terme de coût et de sécurité (p. ex. manipulation d'une clé privée). Elle consiste à effectuer $[k]P$, ce qui équivaut à multiplier un point P par un scalaire k. De nombreuses opérations ADD et DBL sont utilisées pour réaliser la SM. Dans la plupart des cas, $[k]P$ se calcule par une succession de ADD et DBL. De nombreux algorithmes, avec différentes caractéristiques, permettent d'effectuer la SM. Dans cette partie, quelques-uns seront décrits.

1.2.1 Doublement et addition (DA)

L'algorithme le plus simple pour réaliser une SM est le `doublement et addition` (DA) décrit dans l'algorithme 1. Le scalaire k est considéré en binaire, et s'écrit $k = (k_{m-1}, \ldots, k_0)_2$ où m est la taille de k. À l'étape d'initialisation de DA, le *point intermédiaire* Q est fixé au point à l'infini ($\mathcal{O}$). Ensuite, le point Q est modifié en fonction de la valeur de k_i. Si $k_i = 0$, alors Q est doublé ($Q \leftarrow 2Q$). Alors que si $k_i = 1$, le point Q est doublé puis le point P lui est additionné ($Q \leftarrow 2Q + P$). Lorsque tous les bits de k ont été utilisés, le résultat de la SM Q est retourné.

Le coût de cette SM est d'environ $m\mathtt{DBL} + \frac{m}{2}\mathtt{ADD}$. En effet, une clé k sera considérée comme cryptographiquement robuste si elle contient à peu près autant de bits à 0 et à 1.

1.2.2 Échelle de Montgomery (MLD)

L'algorithme le plus couramment utilisé, pour les courbes de Montgomery, est `l'échelle de Montgomery` (MLD), décrit dans l'algorithme 2. Tout comme DA, le scalaire est considéré en binaire. À l'étape d'initialisation, deux points intermédiaires Q_1 et Q_2 sont respectivement fixés

1. http ://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

Algorithme 1 : SM - `doublement et addition`

Entrées : P et $k = (k_{m-1}, \ldots, k_0)_2$
Résultat : $[k] \cdot P$

```
1 Q ← O
2 pour i = m − 1 à 0 faire
3     Q ← 2 · Q                 /*DBL*/
4     si k_i = 1 alors
5         Q ← Q + P             /*ADD*/
6 retourner Q
```

à $\mathcal{O}$ et à P. Au cours de l'itération, les deux points Q_1 et Q_2 sont modifiés de manière similaire, comme illustré dans l'algorithme 2.

Comme les mêmes opérations sont effectuées quelle que soit la valeur de k_i, la SM est dite *régulière* et coûte $m\texttt{ADD} + m\texttt{DBL}$. L'intérêt de MLD par rapport à DA est qu'il n'y a plus de dépendance entre les opérations `ADD` exécutées et les bits k_i du scalaire à `1`.

Lors de `ADD` de Q_1 et Q_2, la valeur de la coordonnée x de $Q_2 - Q_1$ doit être connue pour pouvoir effectuer le calcul de $Q_1 + Q_2$. Cette contrainte ne pose pas de problème puisque $Q_2 - Q_1 = P$ à chaque itération dans MLD. En effet, à chaque itération i de la SM, $Q_1 = \alpha P$ et $Q_2 = (\alpha + 1)P$. Durant l'itération suivante $i + 1$, ces valeurs deviennent :

— Si $k_i = 0$: $Q_1 \leftarrow 2Q_1$ et $Q_2 \leftarrow Q_1 + Q_2$

$$\begin{cases} Q_1 &= 2\alpha P \\ Q_2 &= (2\alpha + 1)P \end{cases} \Rightarrow Q_2 - Q_1 = P \tag{1.5}$$

— Si $k_i = 1$: $Q_1 \leftarrow Q_1 + Q_2$ et $Q_2 \leftarrow 2Q_2$

$$\begin{cases} Q_1 &= (2\alpha + 1)P \\ Q_2 &= (2\alpha + 2)P \end{cases} \Rightarrow Q_2 - Q_1 = P \tag{1.6}$$

1.2.3 Forme non-adjacente (NAF)

Pour certains scalaires, changer la représentation du nombre peut être avantageux. Dans ce cas, k n'est plus considéré en binaire. Des méthodes de recodage permettent de transformer la manière d'écrire k. La représentation signée appelée *forme non adjacente* (NAF) est l'une de ces méthodes. La notation $\bar{1} = -1$ est alors introduite. AInsi, le nombre k s'écrit tel que $\text{NAF}(k) = \sum_{i=0}^{m-1} k_i 2^i$ où $k_i \in \{\bar{1}, 0, 1\}$. L'algorithme 3 réalisant la SM est similaire au DA.

Algorithme 2 : SM - `échelle de Montgomery`

Entrées : P et $k = (k_{m-1}, \ldots, k_0)_2$
Résultat : $[k] \cdot P$

$Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$
pour $i = m-1$ **à** 0 **faire**
  si $k_i = 1$ **alors**
    $Q_1 \leftarrow Q_1 + Q_2$ /*ADD*/
    $Q_2 \leftarrow 2 \cdot Q_2$ /*DBL*/
  sinon
    $Q_2 \leftarrow Q_1 + Q_2$ /*ADD*/
    $Q_1 \leftarrow 2 \cdot Q_1$ /*DBL*/
retourner Q_1

Exemple 1. *le nombre* 31 *s'écrivant* $(11111)_2$ *en binaire peut s'écrire* $(10000\bar{1})$ *en représentation NAF. Avec la méthode DA, une SM coûterait* $5\mathit{ADD} + 5\mathit{DBL}$ *alors qu'avec l'algorithme 3 elle coûterait* $6\mathit{DBL} + 1\mathit{ADD} + 1\mathit{SUB}$.

L'opération `SUB` est similaire à `ADD`. En effet, pour soustraire un point P à Q, il suffit d'additionner $-P$ à Q. Le calcul de l'opposé $-P$ est vraiment très simple. Par exemple, pour les courbes de Weierstrass et de Montgomery, l'opposé de P s'obtient en faisant $y = -y$.

Algorithme 3 : SM - NAF

Entrées : P et $k = \sum_{i=0}^{l-1} k_i 2^i$ avec $k_i \in \{0, \pm 1\}$ en représentation NAF
Résultat : $[k] \cdot P$

$Q \leftarrow \mathcal{O}$
pour $i = m-1$ **à** 0 **faire**
  $Q \leftarrow 2 \cdot Q$ /*DBL*/
  si $k_i = 1$ **alors**
    $Q \leftarrow Q + P$ /*ADD*/
  si $k_i = -1$ **alors**
    $Q \leftarrow Q - P$ /*SUB*/
retourner Q_1

1.2.4 w-NAF

Une variante par fenêtrage de la méthode NAF existe et est appelée $w-$NAF. Les bits sont traités par paquet de taille w bits où w est la taille de la fenêtre. Le but de cette méthode est de réduire les apparitions de chiffres différents de 0 dans l'écriture du nombre. Par conséquent, le nombre de chiffres égaux à 0 augmente.

Au commencement de la SM avec la méthode du w-NAF, décrite dans l'algorithme 5, les points P_i sont précalculés tels que $P_i \leftarrow [i]P$ avec $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$. Ces précalculs se réalisent simplement grâce à l'algorithme 4 et nécessitent au plus $2^{w-1} - 1$ ADD et 1 DBL. Plus la taille de la fenêtre w est grande, plus le nombre de précalculs est important.

Durant le reste de la SM, le P_i adéquat est ajouté (resp. soustrait) au point intermédiaire Q si $k_i > 0$ (resp. $k_i < 0$). Le coût, sans les précalculs, de cette SM est de m DBL $+ \dfrac{m}{w+1}$ ADD.

Algorithme 4 : Précalculs des points P_i

Entrées : P et w, où w est la taille de la fenêtre
Résultat : Liste des points pré-calculés P_i

$Q \leftarrow P$
$L = [0, \cdots, 0]$
$L[0] \leftarrow Q$
pour $i = 1$ **à** $2^{w-1} - 1$ **faire**
    $Q \leftarrow 2 \cdot P + Q$
    $L[i] \leftarrow Q$
retourner L

Algorithme 5 : SM - w-NAF

Entrées : P et $k = \sum_{i=0}^{m-1} k_i 2^i$ en représentation w-NAF
Résultat : $[k] \cdot P$

Pré-calculer $P_i = [i]P$ avec $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
$Q \leftarrow \mathcal{O}$
pour $i = m - 1$ **à** 0 **faire**
    $Q \leftarrow 2 \cdot Q$ /*DBL*/
    si $k_i > 0$ **alors**
        $Q \leftarrow Q + P_{k_i}$ /*ADD*/
    si $k_i < 0$ **alors**
        $Q \leftarrow Q - P_{k_i}$ /*SUB*/
retourner Q_1

D'autres méthodes de recodage du scalaire (p. ex. chaîne d'additions [28], représentation à bases multiples [42], représentation de Zeckendorf [118]) et d'autres algorithmes de SM (p. ex. fenêtre glissante), existent. Ils permettent d'avoir des SM plus performantes ou avec certaines propriétés (p. ex. uniformité). Mais elles nécessitent de recoder les scalaires ce qui peut prendre du temps pour certaines représentations.

1.3 Conclusion

De nombreuses méthodes, plus ou moins coûteuses, permettent d'effectuer des SM. Certaines d'entre elles sont plus performantes que d'autres. Certaines, parfois plus coûteuses, possèdent des propriétés spécifiques (p. ex. uniformité). Certains algorithmes sont connus pour être peu sûrs (p. ex. DA qui est vulnérable aux attaques SPA, voir chap. 2). Lorsque la SM doit être implantée, les différentes caractéristiques de ces algorithmes doivent être prises en considération.

Les systèmes embarqués, sur lesquels sera implantée la SM, doivent faire face à différents types d'attaques et en particulier aux attaques physiques. Ces dernières se déclinent en plusieurs catégories qu'il est possible de combiner [102] : rétro-ingénierie, observation et perturbation. Elles sont totalement différentes des attaques logiques ou mathématiques et requièrent des protections spécifiques.

CHAPITRE 2

ATTAQUES PAR OBSERVATION ET CONTRE-MESURES

Les *attaques par canaux auxiliaires* (*side channel attacks* en anglais) (SCA), encore appelées attaques passives ou par observation, utilisent des observations de paramètres physiques appelés *grandeurs observables*. Ces grandeurs observables sont mesurées durant l'exécution du cryptosystème sur le circuit. Ces mesures sont couramment appelées des *traces*. Elles peuvent représenter : le temps d'exécution [78], la consommation électrique [85], le rayonnement électromagnétique (EM) [1] par exemple.

Les SCA exploitent des *corrélations potentielles* entre les traces obtenues par des mesures externes et des informations secrètes manipulées dans le circuit durant l'exécution. Dans la littérature, les SCA sont souvent divisées en deux catégories. D'une part, celles qui n'utilisent qu'une seule trace (p. ex. *simple power analysis* (SPA)). D'autre part, celles qui utilisent de nombreuses traces et des outils statistiques (p. ex. *differential power analysis* (DPA)) ou d'apprentissage. Ces deux types de SCA existent également pour les autres paramètres physiques mesurés (p. ex. le rayonnement EM).

Les protections contre les SCA ont pour objectif de supprimer, ou de fortement réduire, les dépendances entre les valeurs sensibles et les variations observables des paramètres physiques. Pour cela, deux principes peuvent être utilisés : l'*uniformisation* et la *randomisation*. L'uniformisation consiste à rendre les séquences d'opérations indistinguables quelle que soit la valeur secrète manipulée dans le circuit. L'ajout, au calcul initial, d'opérations inutiles, pour combler les différences de comportement au niveau des opérations est un exemple de méthode d'uniformisation. Ces opérations inutiles sont également appelées *opérations factices*. La randomisation consiste à générer de l'activité aléatoire dans le but de brouiller fortement la ou les trace(s) du calcul. En effet, comme les outils statistiques considèrent cette activité aléatoire comme de vraies données, leurs résultats s'en trouvent perturbés. Parmi les méthodes pour ajouter ce bruit, l'ajout d'opérations factices est aussi possible ici.

Dans cette partie, différentes grandeurs observables, des attaques par observation, ainsi que des protections associées sont décrites.

2.1 Grandeurs observables

Lors du fonctionnement d'un circuit intégré, plusieurs types de paramètres physiques sont observables. Dans la littérature, les termes *canaux cachés*, ou *canaux auxiliaires*, sont souvent utilisés pour parler de ces paramètres observables. Les plus couramment utilisés sont la consommation électrique [85], le rayonnement électromagnétique (EM) [1] et le temps d'exécution [78, 40, 19]. Le rayonnement thermique (ou dissipation de température) [65] et le bruit [51] peuvent aussi être employés pour réaliser des SCA.

Ces paramètres sont mesurables grâce à différents instruments, tel qu'un oscilloscope couplés à des dispositifs de mesures (p. ex. des sondes). Un oscilloscope permet de mesurer l'intensité du courant, grâce à une résistance insérée en série dans le circuit d'alimentation. Comme la puissance consommée est proportionnelle à l'intensité du courant, la consommation électrique peut être déduite.

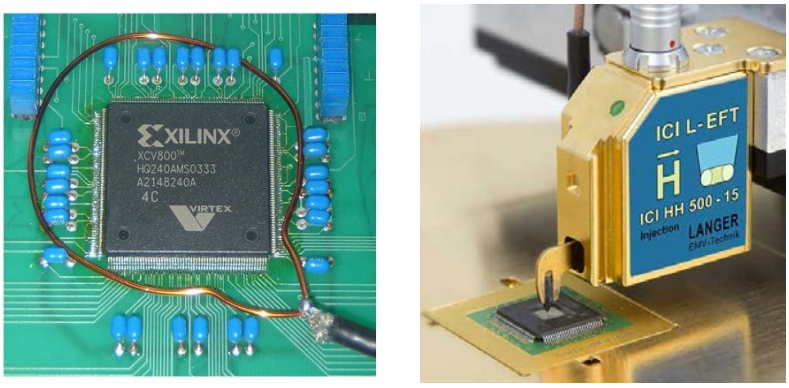
Le principe est similaire pour le rayonnement EM qui nécessite l'utilisation de sondes électromagnétiques. Deux méthodes de mesure sont possibles : mesure globale ou mesure locale. Ces deux méthodes nécessitent deux types de sondes différentes. Une sonde en forme de boucle, illustrée à la figure 2.1(a), permet une mesure globale. Au contraire, une toute petite sonde ou micro-sonde, comme illustrée à la figure 2.1(b), placée à un endroit spécifique du circuit, permet une mesure locale.

Le rayonnement thermique est corrélé avec le modèle électrique. Par conséquent, il peut servir pour réaliser une attaque SCA, comme illustré à la figure 2.2. Cette figure schématise une attaque réalisée dans [65] grâce au rayonnement thermique. La dissipation de température augmente si l'activité du circuit est élevée. Au contraire, elle diminue lorsque l'activité est faible. Comme la variation de la température, dans le circuit et le boîtier contenant le circuit, est en général assez lente, les mesures sont souvent peu précises pour des circuits avec des fréquences d'horloge rapides.

La récupération des mesures est la première étape d'une SCA. Une fois que l'attaquant a en sa possession les traces, il peut appliquer différentes méthodes pour retrouver des informations secrètes.

2.2 Attaques par observation

Maintenant que notre adversaire possède des traces mesurées lors de l'exécution du système, il lui faut les exploiter. Pour cela, plusieurs méthodes s'offrent à lui. Ces méthodes correspondent à des attaques différentes qui sont décrites dans cette section où seule la consommation électrique est considérée. Néanmoins, la plupart de ces attaques sont réalisables avec les différents paramètres physiques mesurables.



(a) Sonde globale [90] (b) Sonde locale [81]

FIGURE 2.1 – Sonde EM pour une mesure globale et locale

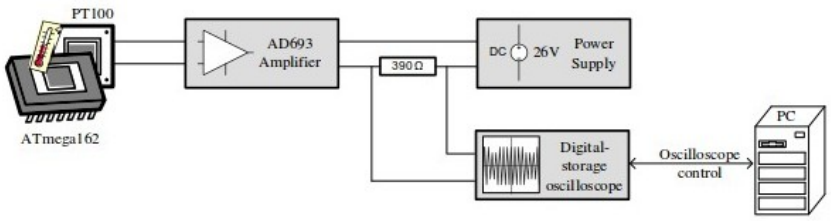


FIGURE 2.2 – Schéma de SCA observant les variations de température [65]

2.2.1 Analyse simple de consommation (SPA)

L'*analyse simple de consommation* [80, 79], ou *Simple Power Analysis* (SPA), est possible lorsque la séquence d'opérations exécutée dépend d'informations secrètes (p. ex. la clé privée). Dans ce cas, une seule trace (ou très peu) est nécessaire.

ECC, comme RSA, est particulièrement sensible à cette attaque. En effet, lorsque les formules d'addition (`ADD`) et de doublement (`DBL`) sont différentes, les distinguer sur une trace est possible. Par conséquent, si la multiplication scalaire (SM) est effectuée avec un algorithme comme le `doublement et addition` (DA), alors l'adversaire peut retrouver la clé par une lecture de la trace.

La figure 2.3 représente le principe d'une SPA sur ECC. Deux motifs sont distinguables ; d'une part `ADD` et d'autre part `DBL`. Ainsi, lorsque deux motifs de `DBL` sont observés consécutivement (aux itérations i et $i+1$), l'attaquant sait que le i-ème bit est égal à 0. Malgré sa simplicité, une attaque SPA a des limites, en particulier lorsque les différences (entre les deux motifs) sont bien plus petites que le bruit (fonctionnement et mesure).

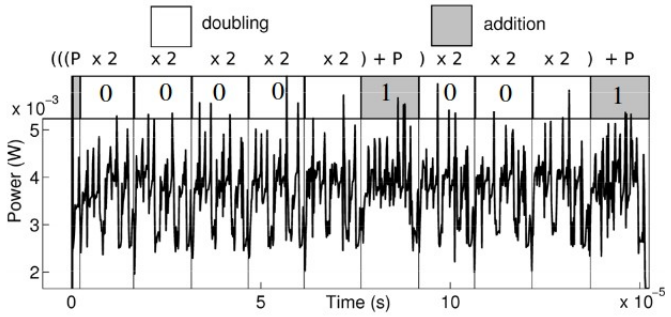


FIGURE 2.3 – Trace d'un *doublement et addition* [98]

2.2.2 Analyse différentiel de consommation (DPA) et attaque *address-bit DPA*

Une *analyse différentielle de consommation* (DPA) [80, 79], nécessite un grand nombre de traces de consommation t_i pour différents messages m_i. Pour réaliser cette attaque, l'attaquant doit être capable de faire rejouer partiellement l'algorithme ciblé avec des hypothèses de clé gérables (p. ex. 1 octet). Cette portion de l'algorithme doit comporter un résultat prédictible r_p.

Le déroulement de l'attaque est le suivant :

— Récupération des couples (t_i, m_i) ;
— Exécution, pour chaque m_i, de la portion de l'algorithme pour une hypothèse de clé gérable ;
— Récupération de r_p pour chaque m_i ;
— Répartition, dans différents groupes, des traces t_i en fonction de la valeur de r_p ;
— Application de la *différence des moyennes* ;
— Si un pic de consommation est remarqué, alors l'hypothèse est correcte ;
— Sinon, recommencer avec une nouvelle hypothèse ;
— Recommencer pour une autre partie de la clé gérable.

La figure 2.4 représente le résultat d'une attaque DPA sur 1 octet de clé d'un DES, réalisée, par moi-même, grâce aux traces du « DPA contest » de 2008/2009 [68]. Lors de cette attaque, la fuite utilisée est la distance de Hamming sur des mots de 4 bits, à la sortie de la SBox du premier tour du DES. La portion du code correspondant à cette fuite permet d'obtenir un résultat prédictible r_p entre 0 et 4. Les traces ont ensuite été réparties en 5 paquets (un paquet pour chaque valeur de r_p) puis moyennées afin d'appliquer la différence des moyennes. La courbe avec le pic le plus important (courbe verte), correspond à la bonne hypothèse de clé. En plus du pic de consommation correspondant à la bonne hypothèse de clé, d'autres pics sont visibles. Ce sont des *pics factices*. Plus le nombre de traces, utilisées lors de l'attaque, est important, moins de pics factices apparaissent, et donc plus la précision de l'attaque augmente.

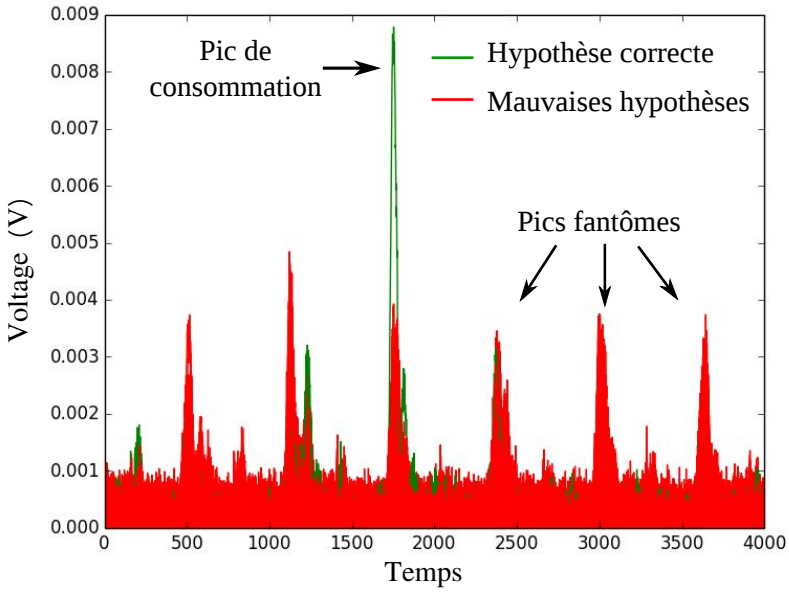


FIGURE 2.4 – Attaque DPA sur un DES

Une analyse différentielle par rayonnement EM (DEMA) de ECC est réalisée avec 2000 traces dans [90]. La méthode est la même que pour une DPA, seul le canal auxiliaire est modifié. L'algorithme utilisé pour la SM est le `doublement et addition toujours` (voir sec. 2.3) implanté sur un FPGA Xilinx Virtex XCV800 à une fréquence d'horloge de 300MHz. Malgré la réussite de l'attaque dans [90], les attaques DPA peuvent être compliquées à mettre en œuvre dans ECC. En effet, comme la clé privée de ECC n'est utilisé que rarement (un à deux fois, p. ex. deux fois pour un échange de clé de type ECDH), obtenir de très nombreuses traces pour un même scalaire secret n'est toujours possible.

L'attaque *address-bit DPA* [69] est une variante de l'attaque DPA. Cette attaque repose sur le fait que lorsqu'une donnée est chargée sur différentes adresses d'une mémoire, alors la consommation électrique varie en fonction de la différence du poids de Hamming des adresses. Ainsi, l'attaque est possible si une dépendance existe entre les données secrètes et les adresses des adresses consultées. Dans [69], les auteurs réalisent cette attaque sur ECC et réussissent à retrouver la clé secrète avec 500 traces.

2.2.3 Analyse corrélée de consommation (CPA)

Une attaque par *analyse corrélée de consommation* (CPA) [17] se déroule de la même manière qu'une attaque DPA. L'outil statistique utilisé dans cette attaque est le *coefficient de corrélation de Pearson*. Ce coefficient permet de quantifier la dépendance linéaire entre deux

variables. Lorsque l'attaquant fait rejouer la portion de l'algorithme qui lui permet d'obtenir le résultat prédictible r_p, il calcule la distance de Hamming[1] H_{i,r_p} entre r_p et m_i. Par la suite, l'ensemble des H_{i,r_p} sera noté H et l'ensemble des t_i sera noté T. Avec ces notations, le coefficient de Pearson s'écrit :

$$\rho_{TH} = \frac{cov(T,H)}{\sigma_T \cdot \sigma_H} \tag{2.1}$$

où $cov(T,H)$ est la covariance entre T et H et σ_T (resp. σ_H) est l'écart-type de T (resp. de H) Une approximation $\widetilde{\rho_k}$, de ce coefficient, est donnée par la formule suivante avec k l'hypothèse de clé gérable et n le nombre de traces :

$$\widetilde{\rho_k} = \frac{n\sum t_i \cdot H_{i,r_p} - \sum t_i \cdot \sum H_{i,r_p}}{\sqrt{n\sum t_i^2 - (\sum t_i)^2} \cdot \sqrt{n\sum H_{i,r_p}^2 - (\sum H_{i,r_p})^2}} \tag{2.2}$$

L'hypothèse de clé qui maximise $\widetilde{\rho_k}$ (c.-à-d. qui renvoie $|\widetilde{\rho_k}|$ le plus proche de 1) est la bonne. Grâce à cette méthode, l'apparition de pics factices est réduite ainsi que le nombre de traces nécessaires lors de l'analyse.

2.2.4 Analyse d'information mutuelle (MIA)

L'attaque par *analyse d'information mutuelle* (MIA) [52] est plus générique que les attaques décrites précédemment. Contrairement à une attaque CPA qui ne considère que les relations linéaires, l'attaque MIA prend également en compte des relations non linéaires. Le calcul de l'information mutuelle (IM) nécessite de connaître les définitions de l'entropie de Shannon $H(X)$ et de l'entropie conjointe.

Définition 2.2.4.1. *L'entropie de Shannon $H(X)$ et l'entropie conjointe $H(X,Y)$ sont définies par les équations suivantes, où $\mathbb{P}(X_i)$ est la probabilité qu'un évènement X_i se réalise.*

$$H(X) = -\sum_i \mathbb{P}(X_i) \cdot \log(\mathbb{P}(X_i)) \tag{2.3}$$

$$H(X,Y) = -\sum_i \mathbb{P}(X_i, Y_j) \cdot \log(\mathbb{P}(X_i, Y_j)) \tag{2.4}$$

Définition 2.2.4.2. *L'information mutuelle est définie par :*

$$IM(X,Y) = H(X) + H(Y) - H(X,Y) \tag{2.5}$$

Les étapes suivantes constituent le déroulement d'une attaque MIA :

1. Calcul de l'entropie pour chaque trace ;

1. Le nombre de bits différents entre deux mots.

i	d_i	calcul de dP	calcul de $d(2P)$
6	1	2×0 $0 + P$	2×0 $0 + 2P$
5	0	$2 \times P$ $2P + P$	$2 \times 2P$ $4P + 2P$
4	0	$2 \times 2P$ $4P + P$	$2 \times 4P$ $8P + 2P$
3	1	$2 \times 4P$ $8P + P$	$2 \times 8P$ $16P + 2P$
2	1	$2 \times 9P$ $18P + P$	$2 \times 18P$ $36P + 2P$
1	1	$2 \times 19P$ $38P + P$	$2 \times 38P$ $76P + 2P$
0	0	$2 \times 39P$ $78P + P$ retourner $78P$	$2 \times 78P$ $156P + 2P$ retourner $156P$

TABLE 2.1 – Calcul de $[78] \cdot P$ et de $[78] \cdot (2P)$, issu de [47] ($d = k$)

2. Exécution, pour chaque m_i, de la portion de code pour une hypothèse de clé gérable ;
3. Récupération de r_p pour chaque m_i ;
4. Répartition des (t_i, m_i) en fonction de la valeur de r_p ;
5. Estimation de l'entropie conjointe $H(t_i, r_p)$ pour chaque r_p ;
6. Calcul de IM pour chaque hypothèse de clé.

L'hypothèse de clé qui maximise IM correspond à la bonne clé.

2.2.5 Attaque du doublement et analyse comparative de consommation

L'*attaque du doublement* [47] est une variante de SCA. Elle cible particulièrement la SM de ECC en coordonnées affines. Plus précisément, cette attaque exploite des collisions possibles lors de la SM réalisée avec un DA, commençant par le bit de poids fort vers le bit de poids faible.

Deux traces d'exécution (p. ex. consommation) avec le même scalaire k et des points P différents (choisis par l'attaquant) sont nécessaires. La première des deux SM réalisée est $Q_1 = [k]P$ avec P comme point de base. Ensuite, la seconde SM réalisée est avec $2P$ comme point de base et correspond à $Q_2 = [2]Q_1 = [k]([2]P)$. Les différentes étapes du calcul lors de ces deux SM sont décrites au tableau 2.1 (avec $k = 78$).

Une collision se produit entre les calculs de Q_1 et de Q_2, et est mise en évidence par les encadrés dans le tableau 2.1 lorsque $k_i = 0$. En effet, le i-ème DBL de Q_1 et le même que le

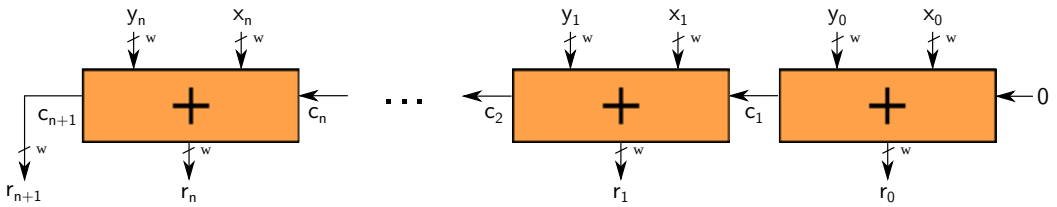


FIGURE 2.5 – Addition de grands nombres X et Y

$i+1$-ème DBL de Q_2 si et seulement si $k_i = 0$. Grâce à une comparaison des traces, le scalaire peut être retrouvé.

Une généralisation de cette attaque, nommée l'*analyse comparative de consommation* [61], fut utilisée contre une implémentation d'un RSA-1024 utilisant la multiplication de Montgomery sur un FPGA Virtex-II. Pour réaliser cette attaque, les quatre premiers bits ont été supposés connus. Les bits suivants sont retrouvés récursivement.

2.2.6 Attaque par analyse de fuite de retenue

L'attaque, basée sur la *fuite de retenue* [46] (ou *carry-leakage based* en anglais), est utilisée contre les algorithmes asymétriques implantés avec la première contre-mesure de Coron [2] [34] (voir sec. 2.3.2). Dans cette attaque, la cible est la contre-mesure elle-même et plus particulièrement, la première addition avec le scalaire et un nombre aléatoire S_a.

Dans un circuit, les nombres sont découpés en mots de taille w. Pour réaliser une addition entre deux nombres X et Y, les mots x_i et y_i sont additionnés successivement grâce à un additionneur comme illustré à la figure 2.5.

Lorsque la clé secrète k est additionnée avec S_a, une retenue c_i peut apparaître pour chaque additionneur. Un drapeau de retenue est alors levé. La probabilité, qu'un drapeau de retenue soit levé, ne dépend que de la valeur de k, qui est fixée. Par conséquent, une corrélation existe entre la valeur de la clé et l'apparition de retenue (qu'il est possible d'observer). Un circuit avec un petit additionneur (p. ex. 8 bits) et qui manipule une grande clé (p. ex. 521 bits) sera plus sensible qu'un circuit qui possède un plus gros additionneur (p. ex. 32 bits). En effet, plus w est de petite taille, plus le nombre d'apparitions de retenues possibles augmente.

Une fois l'information des levées de drapeaux obtenue pour l exécutions, l'attaquant a deux possibilités : tenter une attaque exhaustive (ES) ou combiner l'analyse (CA) des drapeaux avec une DPA. Les complexités de ces attaques pour RSA et ECC sont récapitulées dans le tableau 2.2.

2. *randomisation du scalaire*

Implémentation	Attaque	Mesures	Coût de calcul
RSA 1024 - additionneur 8 bits	ES	2^{16}	1
RSA 1024 - additionneur 8 bits	CA	$10,000$	2^{32}
RSA 1024 - additionneur 16 bits	CA	$10,000$	2^{32}
RSA 2048 - additionneur 32 bits	CA	$10,000$	2^{32}
ECC 160 - additionneur 16 bits	ES	2^{16}	2^{40}
ECC 160 - additionneur 32 bits	ES	2^{20}	2^{55}

TABLE 2.2 – Complexité [46] des attaques sur RSA et ECC

2.2.7 Analyse raffinée de consommation et attaque du point *zero-value*

Lors d'une SM, des points particuliers de la courbe, différents de $\mathcal{O}$, peuvent apparaître. Ces points ont la spécificité d'avoir une de leurs coordonnées égale à zéro. L'attaque par *analyse raffinée de consommation* [55], spécifique à ECC, exploite l'apparition de ces points. De plus, elle reste efficace malgré des protections contre des SCA.

Pour réaliser cette attaque, l'adversaire doit être en mesure de trouver un de ces points spéciaux :

— Un point spécial, de la forme $P = (0, y)$, existe si et seulement si b est un résidu quadratique modulo p.

— Un point spécial, de la forme $P = (x, 0)$, existe si et seulement si $x^3 + ax + b = 0$ a au moins une racine dans $\mathbb{F}_p$.

L'attaquant doit également connaître les $m-i-1$ premiers bits de poids fort de k pour retrouver le $i-$ème bit. Grâce à cela, le point P peut être généré tel que :

$$P = [(k_{m-1}, \ldots, k_{i+1}, 1)_2^{-1} (\mathrm{mod}\, \sharp E)] \cdot P_0 \tag{2.6}$$

où P_0 est un point spécial et $\sharp E$ est l'ordre de la courbe. Une fois le point P généré, l'adversaire récupère des mesures d'exécution de $[k]P$ dont il calcule la courbe moyenne d'exécution. Si un pic est présent sur cette courbe, alors l'hypothèse de clé est correcte.

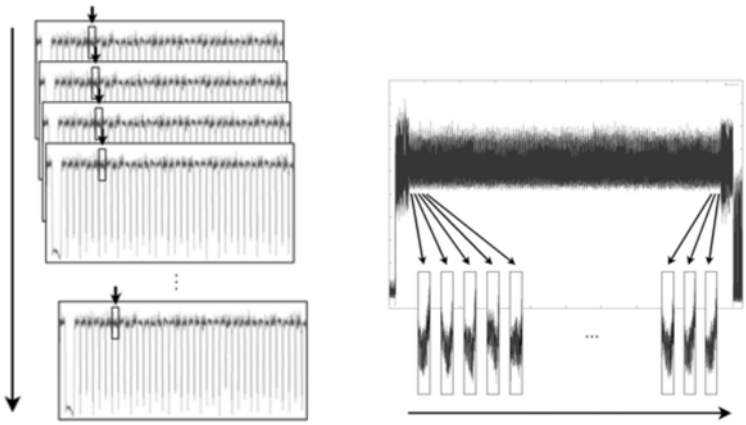
L'attaque du point *zero-value* (ZPA) [2] est une amélioration/généralisation de cette méthode. En effet, dans certains cas, les points spéciaux décrits plus tôt n'existent pas. Cependant, certains points sont tels que, lors des opérations `ADD` et `DBL`, des valeurs intermédiaires sont nulles. Ces points sont alors dits *zero-value*. Ainsi, une attaque ZPA considère plus de points particuliers qu'une attaque RPA.

2.2.8 Attaque horizontale

La plupart des attaques décrites précédemment sont des attaques dites *verticales*. De nombreuses traces sont nécessaires pour retrouver la clé. Contrairement à ces attaques, les at-

taques dites *horizontales* [30] emploient une ou de très peu de traces pour retrouver la clé comme illustré à la figure 2.6. La trace utilisée est fractionnée en plusieurs parties puis chaque partie est traitée.

La SM est réalisée grâce à des opérations sur la courbe (`ADD` et `DBL`). Ces opérations, au niveau courbe, sont elles-mêmes composées d'opérations sur le corps fini de calcul (`A`, `M`, *etc.*). L'attaque *Big Mac*, une attaque horizontale introduite dans [112] puis améliorée dans [36], utilise la distance euclidienne afin de déterminer si deux multiplications de nombres ont un opérande en commun. Par conséquent, pour deux multiplications $A \times B$ et $C \times D$, où $A \neq C$, un adversaire, capable de savoir si $D = B$, pourra retrouver la clé secrète. Le coefficient de Pearson [29] peut également être employé comme outil statistique, au lieu de la distance euclidienne.



(a) Attaque verticale [30] (b) Attaque horizontale [30]

FIGURE 2.6 – Différents types de SCA

La *désynchronisation* des traces peut compromettre la réussite des SCA. Les attaques horizontales fractionnent une trace pour obtenir de nombreux morceaux de trace. Ces morceaux ne sont, en général, pas alignés les uns par rapport aux autres. Une étape de réalignement des traces est donc nécessaire avant la phase d'attaque. Des méthodes, qui ne sont pas décrites dans ce manuscrit, permettent d'effectuer un réalignement des traces comme par exemple les méthodes ci-dessous :

— corrélation de phase seulement (*Phase-Only Correlation*) (POC) [63] ;
— déformation temporelle dynamique (*Dynamic Time Warping*) (DTW) [110] ;
— corrélation d'amplitude seulement (*Amplitude Only Correlation*) (AOC) [56] ;
— resynchronisation par moment (*Resynchronization by Moments*) (RM) [38].

2.2.9 Attaque par profilage

Lors d'une attaque par *profilage* ou *template* [23, 99], l'utilisation d'un clone du dispositif attaqué est nécessaire. Ce clone ne contient aucune valeur secrète. Cependant, l'attaquant a un total contrôle dessus. L'attaque s'articule en deux parties : la phase d'entraînement et la phase d'attaque en elle-même.

Phase d'entraînement

Durant la phase d'entraînement ou d'apprentissage, l'attaquant exécute une opération n fois (ici la SM) pour des hypothèses de clé gérables connues. Les consommations de chaque SM sont enregistrées, puis les traces moyennes M_i, pour chacune des hypothèses de clé, sont calculées. Après une soustraction deux à deux des traces moyennes, les l différences les plus grandes sont sélectionnées. Ce sont les points d'intérêt $P_1, \ldots, P_l$. Les traces sont alors réduites aux seuls points d'intérêts. Le vecteur de bruit, d'une hypothèse de clé, est ensuite déterminé par $B_j = t_j - M_i$ pour chaque trace mesurée. La dernière étape de cette phase est le calcul de la matrice de covariance, nommée Σ, avec l'équation 2.7.

$$\Delta_i = \frac{1}{n-1} \sum_{j-1}^{n} B_j^t B_j \tag{2.7}$$

Le couple (M_i, Δ_i) correspond au *template* pour l'hypothèse de clé k_i.

Phase d'attaque

Lors de la phase d'attaque, des traces S sont mesurées durant l'exécution de quelques SM, avec le dispositif et des clés secrètes inconnues. Une comparaison est ensuite faite entre les traces S et les caractéristiques de consommation profilées lors de la phase précédente. Cela permet de trouver à quel groupe de profil la trace se rapproche le plus.

Ensuite, la trace doit être classifiée. Pour cela, l'attaquant doit savoir si elle provient d'une opération déjà testée en calculant sa probabilité d'observation. Calculer cette probabilité est équivalent à calculer le vecteur de bruit N en utilisant le signal moyen du *template*. Ce vecteur est défini par $N = S - M_i$ où M_i est la moyenne des traces correspondant à S. Ensuite, la probabilité d'observation de N est déterminée par l'équation 2.8, où Σ est la matrice de covariance, calculée précédemment, et l le nombre de points d'intérêt. L'hypothèse de clé la plus probable est celle pour laquelle la probabilité $p_{k_i}(N)$ est la plus grande.

$$p_{k_i}(N) = \frac{1}{\sqrt{(2\pi)^l det(\Delta)}} exp(-\frac{1}{2} N^T \Delta^{-1} N) \tag{2.8}$$

La désynchronisation des traces peut compromettre la réussite d'une attaque *template*. Les méthodes de resynchronisation (voir sec. 2.2.8) peuvent s'appliquer avant la phase d'entraînement. Les réseaux neuronaux convolutifs (CNN) [20], utilisant des techniques de *machine learning*, rendent le réalignement des traces non nécessaire et permettent d'avoir des points d'intérêt moins précis.

2.2.10 Conclusion

Les attaques SCA sont nombreuses et ne sont pas toutes décrites dans ce document. Elles sont réalisables à cause de dépendances entre les paramètres physiques observables et les informations sensibles. Certaines sont contrées par des protections qui permettent de réduire ou de supprimer ces dépendances. Cependant, malgré ces protections, des attaques restent efficaces, par exemple les SCA d'ordre supérieur [79, 86] qui correspondent à des attaques sur des systèmes ayant une protection contre une attaque classique.

2.3 Protections contre les attaques par observation

Lorsque les variations des paramètres physiques sont dépendantes des données secrètes, alors des informations sensibles peuvent fuir en exploitant le comportement du circuit. Les protections contre les SCA se basent sur l'uniformisation et la randomisation. Leur but est de supprimer ou d'atténuer les dépendances entre les données sensibles et le comportement du circuit. L'objectif de l'uniformisation est de rendre les séquences d'opérations indistinguables quelle que soit la valeur manipulée dans le circuit. L'ajout d'opérations, ou encore à la modification de l'algorithme sont des exemples de méthodes d'uniformisation. La génération d'une activité aléatoire permet de brouiller les mesures prises lors de l'exécution. Comme les outils statistiques utilisés considèrent cette randomisation comme de vraies données, certaines attaques sont évitées ou se compliquent. Dans cette section, différentes contre-mesures basées sur l'uniformisation et la randomisation seront décrites.

2.3.1 Uniformisation des données

L'objectif de l'uniformisation est de « gommer » les dépendances entre les valeurs sensibles et les mesures effectuées durant les attaques par observation. Plusieurs techniques permettent d'atteindre ce but. Certaines consistent à modifier l'algorithme de calcul alors que, d'autres ajoutent des opérations inutiles dites *factices*.

Doublement et addition toujours

L'algorithme 6 de `doublement et addition toujours` (DAA) [34] effectue des `ADD` factices lorsque le bit de clé manipulé est égal à 0. Ainsi, une `ADD` est faite à chaque itération de la SM. Le coût de cette méthode est de $m\texttt{ADD} + m\texttt{DBL}$, soit un surcoût d'environ 33 %.

Même si cet algorithme régulier est résistant aux SPA, l'ajout d'opérations factices le rend très vulnérable aux attaques du type *safe-error* (SEA) [115]. De plus, il reste vulnérable à l'attaque du doublement [47]. Enfin, il faut être certain que l'implantation de la sélection de la bonne variable contenant Q_{k_i} en ligne 5 ne présentera pas de fuite (entraînant p. ex. une attaque *adress-bit* sur cette variable). Ceci peut se faire en randomisant l'adresse de Q_{k_i} par exemple.

Algorithme 6 : SM - `doublement et addition toujours`

Entrées : P et $k = (k_{m-1}, \ldots, k_0)_2$
Résultat : $[k] \cdot P$

1 $Q_0 \leftarrow P$
2 **pour** $i = m - 2$ **à** 0 **faire**
3 $\quad Q_0 \leftarrow 2 \cdot Q_0$ /*DBL */
4 $\quad Q_1 \leftarrow Q_0 + P$ /*ADD */
5 $\quad Q_0 \leftarrow Q_{k_i}$
6 **retourner** $Q[0]$

L'échelle de Montgomery

`L'échelle de Montgomery`, décrite dans la section 1.2.2 avec l'algorithme 2, permet de réaliser une SM régulière. En effet, une `ADD` et un `DBL` sont effectuées systématiquement pour chaque bit de clé. Cependant, contrairement à l'algorithme `doublement et addition toujours`, aucune opération factice n'est ajoutée, évitant l'attaque SEA. Néanmoins, cet algorithme reste également vulnérable à l'attaque du doublement [117].

Recodage de clé

Avec les algorithmes de SM précédents, le scalaire est représenté en binaire. Cela n'est pas le cas pour les algorithmes NAF et $w-$NAF (voir sec. 1.2.3 et sec. 1.2.4) où le scalaire est représenté dans une base qui est différente de deux. Le changement de la représentation du scalaire peut permettre de protéger la SM contre des SCA.

Le changement de base n'est pas la seule possibilité de recodage de clé. En effet, des chaînes d'addition correspondant au scalaire peuvent s'appliquer.

Définition 2.3.1.1. *Une chaîne d'addition, correspondant au scalaire k, est une séquence $C(k)$ s'écrivant :*

$$C(k) = \{k^{(0)}, k^{(1)}, \ldots, k^{(r)}\} \tag{2.9}$$

où $k^{(0)} = 1$ et $k^{(r)} = k$ et tel que pour tout $1 \leq i \leq r$, il existe $j < i$ et $l < i$ tel que $k^{(i)} = k^{(j)} + k^{(l)}$.

Exemple 2. $C(5) = \{1, 2, 3, 5\}$ *est une chaîne d'addition correspondant à 5.*

De nombreuses méthodes existent pour obtenir des chaînes d'addition plus ou moins courtes, comme les méthodes euclidiennes, de Fibonacci, ou encore de Zeckendorf. Ces chaînes d'addition peuvent alors être utilisées par des algorithmes [91] de SM régulier, tel que l'`algorithme universel` [31]. Ainsi, seules des `ADD` sont effectuées durant l'exécution, permettant de contrer la SPA.

Bien sur, le calcul, de la chaîne d'addition, doit se faire dans un environnement sécurisé. De plus, la rapidité de la SM dépendra de la longueur de la chaîne d'addition. En effet, plus une chaîne sera longue, plus la SM le sera aussi. L'inconvénient majeur de cette méthodes est que si la clé est modifiée à chaque exécution, alors le calcul de la chaîne d'addition s'effectue à chaque fois ce qui est souvent assez coûteux.

Courbes unifiées

Certains types de courbes, tels que les courbes d'Edwards ou Hessiennes, possèdent des formules de `ADD` et de `DBL` unifiées. Ces courbes, définies par une équation différente des courbes de Weierstrass, ne doivent pas être choisies n'importe comment. En effet, certaines sont plus sûres que d'autres [10]. Le choix des paramètres définissant ces courbes est donc crucial. De plus, si ces paramètres respectent certaines conditions [9], alors un isomorphisme existe entre ces courbes unifiées et les courbes de Weierstrass.

Courbes non unifiées - Nouvelle formule

Avec des courbes non unifiées, les opérations `ADD` et `DBL` sont différentes et sont vulnérables vis-à-vis des SPA. Un moyen de contrer les SPA est d'employer la même formule pour le calcul de `ADD` et de `DBL`, tout en restant sur la courbe d'origine. Cette méthode, introduite dans [18], puis améliorée dans [100], implique de modifier algébriquement le calcul de `ADD` et de `DBL`.

Courbes non unifiées - Atomicité

La notion d'*atomicité* fut introduite dans [25] et étendue dans [53]. Son objectif est d'avoir des `ADD` et des `DBL` indistinguables sans modification algébrique des formules de `ADD` et de `DBL`. Des opérations sont introduites dans les `ADD` et `DBL` pour dégager des blocs atomiques, comme

Algorithme 7 : SM - doublement et addition atomique

Entrées : $P, k = (k_{n-1}, ..., k_1, k_0)_2$
Résultat : $Q = k \cdot P$

$Q_0 \leftarrow P$; $Q_1 \leftarrow P$; $i \leftarrow n-2$; $b \leftarrow 0$
tant que $i \geq 0$ **faire**
  $Q \leftarrow Q_0 + Q_b$
  $b \leftarrow k_i$; $i \leftarrow i + k_i - 1$
retourner Q_0

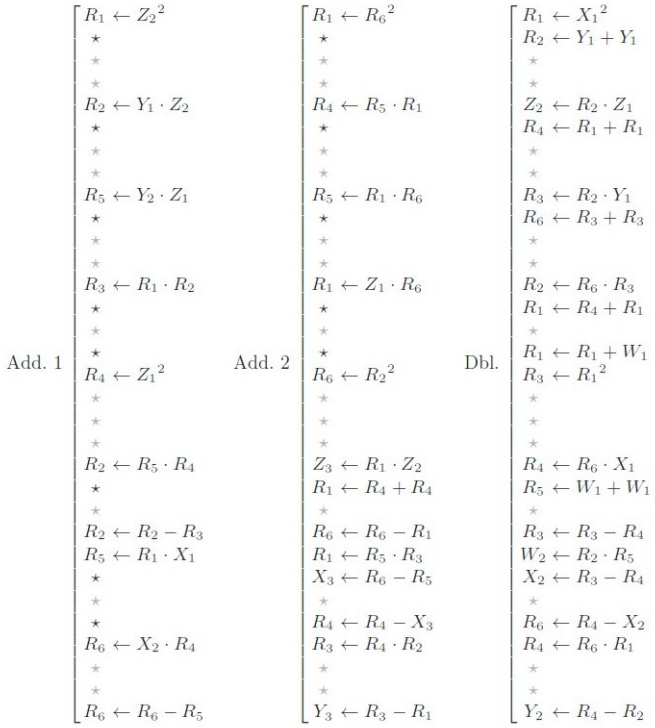


FIGURE 2.7 – Blocs atomiques [53]

illustré à la figure 2.7. Les étoiles de cette figure correspondent à des opérations factices. Si `ADD` et `DBL` sont indiscernables, alors la SM, effectuée avec l'algorithme 7, peut apparaître comme une suite de motifs similaires. La trace d'exécution de la figure 2.8 illustre ces motifs similaires. Cette protection, contre les SPA, est vulnérable à l'attaque du doublement, ainsi qu'aux attaques *safe-error* (voir sec. 3.2.2).

2.3.2 Randomisation des données

La randomisation permet de contrer certaines SCA. Le changement de la clé à chaque exécution est une méthode classique, au niveau protocolaire, pour contrer des attaques phy-

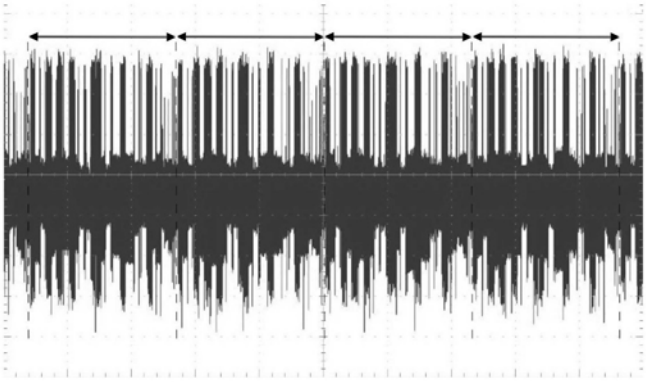


FIGURE 2.8 – Traces d'exécutions de blocs atomiques [53]

siques. La randomisation des données génère de l'activité aléatoire dans le but de brouiller les outils statistiques utilisés lors d'attaques. Ces outils considèrent cet aléa comme de vraies données. Différentes méthodes permettent de produire cette activité aléatoire, comme l'ajout d'opérations leurres ou de masques aléatoires. Dans cette section, différentes méthodes de randomisation des données sont décrites.

L'aveuglement du scalaire

L'aveuglement du scalaire [34] a pour but de randomiser la clé à chaque exécution. Pour cela, $\widetilde{k} = k + r\sharp E$ est calculé au lieu de $[k]P$, où $\sharp E$ est l'ordre de la courbe et r un nombre aléatoire d'une taille minimale. Le calcul de la SM devient alors :

$$Q = [\widetilde{k}]P = [k + r\sharp E]P = [k]P + [r\sharp E]P = [k]P.$$

Cette contre-mesure implique d'implanter un générateur de nombres aléatoires qui doit générer r dans un environnement sûr. Comme $\widetilde{k}$ sera plus grand que k, le temps d'exécution de la SM augmentera. Par exemple, pour un r aléatoire de 20 bits [3], le surcoût sera de 20 `DBL` et 10 `ADD`. Plusieurs attaques [94] de cette contre-mesure existent, tels que par exemple l'attaque du drapeau de retenue [46], (voir sec. 2.2.6).

Le fractionnement additif du scalaire

Le *fractionnement additif du scalaire* [31] est une méthode similaire à l'aveuglement du scalaire. Elle consiste à décomposer la clé en plusieurs parties. Le calcul de la SM devient alors : $[k]P = [k - r]P + [r]P$ avec r un nombre aléatoire de la même taille que k. L'inconvénient

3. Jugé suffisant dans [34] en 1999. Actuellement, une trentaine de bit, minimum est conseillé

Algorithme 8 : SM aléatoire avec pré-calculs

Entrées : $A = [2^{n-1}P, \ldots, 2P, P]$ liste des multiples de P, $k = (k_{n-1}, ..., k_1, k_0)_2$
Résultat : $[k] \cdot P$

1 $Q \leftarrow 0$
2 $W \leftarrow$ permutation$([n-1, \ldots, 0])$ /* W contient les indices des permutations aléatoires */
3 **pour** *tous les éléments* j *dans* W **faire**
4 $\quad Q \leftarrow Q + d_{W[j]} \times A[W[j]]$
5 **retourner** *Retourner* Q

de cette méthode est l'augmentation du temps d'exécution de la SM. En effet, avec cette méthode, il est doublé. D'autres types de fractionnement existent dans la littérature comme p. ex. le fractionnement multiplicatif [109].

Ordre des calculs aléatoires

Le scalaire n'est pas le seul élément manipulable afin d'introduire de l'aléa dans la SM. En l'effet, l'ordre des calculs, des `ADD` et des `DBL`, peut être manipulé [109]. L'équation suivante permet de rendre l'ordre des opérations de courbe aléatoire :

$$kP = k_{n-1}2^{n-1}P + \ldots + k_1 2P + k_0 P \tag{2.10}$$

Des multiples du point de base sont précalculés puis lors de la SM sont associés au bon bit de clé, comme illustré à l'algorithme 8. Par conséquent, les bits de clé peuvent se permuter aléatoirement avant chaque exécution. Cette méthode a un coût conséquent en précalculs, ainsi qu'en espace mémoire, ce qui peut la rendre inadaptée à une implantation sur un système embarqué.

Recodage du scalaire - représentation redondante

Le recodage de clé, en plus de pouvoir s'utiliser dans un but d'uniformisation, peut s'employer dans un but de randomisation. L'objectif est de rendre aléatoire la valeur des points intermédiaires [57] grâce à une représentation redondante de k. En effet, si k a plusieurs écritures alors, plusieurs séquences d'opérations (au niveau courbe) correspondent à une même valeur de k. Ainsi, ces séquences peuvent être modifiées à chaque exécution. Cela implique que les valeurs des points intermédiaires changent également. Malheureusement, cette méthode reste vulnérable aux attaques par collision [48].

Recodage du scalaire - fenêtrage aléatoire

Une méthode de recodage par fenêtrage aléatoire de taille w est proposée dans [84]. La représentation du scalaire est modifiée grâce à l'algorithme 9 tel que k s'écrit :

$$k = \sum_{i=0}^{d-1} b_i 2^{e_i}$$

où

$$\begin{cases} e_i & \in & \mathbb{Z}_{\geq 0} \\ b_i & \in & \{-2^{w-1}+1, -2^{w-1}+3, \ldots, 2^{w-1}-3, 2^{w-1}-1\} \end{cases}$$

L'opération à la ligne 6, de l'algorithme 9 représente une assignation aléatoire de la variable r. Une fois le recodage de la clé effectué, un algorithme de SM par fenêtrage peut s'appliquer avec des précalculs des points intermédiaires nécessaires. Ces points intermédiaires dépendent du recodage de la clé effectué auparavant.

Algorithme 9 : Algorithme de décomposition de k pour fenêtrage aléatoire

Entrées : $k = (k_{n-1}, ..., k_1, k_0)$
Résultat : Une liste de paires $\{(b_i, e_i)\}_{i=0}^{d-1}$

1 $d \leftarrow 0$; $j \leftarrow 0$
2 **tant que** $j \leq n$ **faire**
3 **si** $k_j = 0$ **alors**
4 $j \leftarrow j + 1$
5 **sinon**
6 $r \leftarrow_{\mathcal{R}} \{1, \ldots, w\}$ /* assignation aléatoire de r */
7 $t \leftarrow \min\{n, j + r - 1\}$
8 $h_d \leftarrow (k_t k_{t-1} \ldots k_j)_2$
9 **si** $h_d > 2^{r-1}$ **alors**
10 $b_d \leftarrow h_d - 2^r$
11 $(k_n k_{n-1} \ldots k_{t+1})_2 \leftarrow (k_n k_{n-1} \ldots k_{t+1})_2 + 1$
12 **sinon**
13 $b_d \leftarrow h_d$
14 $e_d \leftarrow j$
15 $d \leftarrow d + 1$
16 $j \leftarrow t + 1$
17 **retourner** *la séquence* $(b_0, e_0), (b_1, e_1), \ldots, (b_{d-1}, e_{d-1})$

Aveuglement du point de base

L'aveuglement de point de base [34] est similaire à l'aveuglement du scalaire. Il consiste à calculer $Q = [k](P + R) - S = [k]P + [k]R - S = [k]P$ avec $S = [k]R$ et R un point aléatoire. Les points R et S sont stockés en mémoire. À chaque nouvelle exécution les points R et S sont mis à jour tel que $R = [(-1)^b 2]R$ et $S = [(-1)^b 2]S$ où b est un bit de valeur aléatoire.

Malheureusement, la mise à jour, des points R et S, est vulnérable à l'attaque du doublement. Lors de la SM, le calcul $[k](P+R)$ est effectué. Si un attaquant prend le point $[2]P$ comme entrée, alors $[k]([2]P + [2]R) = [2][k](P + R) = [2]([k]P)$ sera effectué avec une probabilité de $\frac{1}{2}$. En effet, si $b = 0$ alors $R \leftarrow [2]R$. Pour éviter cela, la mise à jour du point doit être différente.

Aléa sur des coordonnées projectives

L'ajout d'aléa sur les coordonnées projectives [34] est une autre méthode de randomisation. En effet, un point en coordonnées projectives est représenté par $(\theta X, \theta Y, \theta Z)$. Pour randomiser ce point, il suffit de modifier la valeur de θ de manière aléatoire. Dans [27], les auteurs proposent de garder le point de base P en coordonnées affines pour pouvoir profiter des coordonnées mixtes. Cette méthode fut adaptée à `l'échelle de Montgomery` dans [70].

2.3.3 Conclusion

Les protections contre les SCA sont diverses, mais elles sont toutes basées sur l'uniformisation ou sur la randomisation. Certaines des contre-mesures décrites précédemment sont plus efficaces que d'autres. En effet, certaines d'entre-elles rendent ou laissent le cryptosystème vulnérable à d'autres attaques. Le tableau 2.3 récapitule les protections, vues dans cette partie, ainsi que des attaques réalisables sur ces dernières. La liste des attaques fonctionnant sur

Contre-mesure	Cible	Attaques restantes
`Doublement et addition toujours`	SPA	[80, 79, 115, 47]
`L'échelle de Montgomery`	SPA, SEA	[47, 69]
Courbes unifiées	SPA, SEA	[71, 114, 106, 3]
Formules unifiées	SPA	[71, 106]
Atomicité	SPA	[115, 22, 3, 47]
Aveuglement du scalaire	DPA	[94]
Recodage du scalaire	SPA, DPA	[75, 93, 48]
Fenêtrage aléatoire	DPA	[116, 113]
Aveuglement du point	DPA	[71, 114, 47]
Coordonnées projective	DPA	[55, 2]

TABLE 2.3 – Des attaques malgré des protections

ces protections n'est pas exhaustive. Néanmoins, cela illustre que le choix de la contre-mesure est capital. Par exemple, la contre-mesure `doublement et addition toujours` protège la SM des attaques SPA. Cependant, elle rend le système vulnérable aux attaques SEA et le laisse vulnérable aux attaques du doublement. Finalement, le choix de la contre-mesure doit prendre en compte ces vulnérabilités.

CHAPITRE 3

ATTAQUES PAR PERTURBATION ET CONTRE-MESURES

Les *attaques par perturbation*, ou *attaque en fautes* (FA), sont inspirées du domaine spatial où les satellites subissent des radiations qui peuvent perturber leur bon fonctionnement. Ainsi, les FA sont des attaques physiques, qui exploitent, de manière directe ou indirecte, un comportement non spécifié du circuit. Plusieurs angles d'attaque se distinguent. Un attaquant peut choisir de cibler les paramètres d'entrée, le flux de contrôle, la mémoire ou encore les données durant l'exécution. D'autres critères existent, comme la précision de l'attaque ou sa reproductibilité. Ils permettent une classification des attaques par perturbation [111, 5].

Les perturbations seront désignées comme des fautes qui peuvent être *temporaires* ou *permanentes*. Les fautes temporaires ne détériorent pas le circuit. Lors d'une exécution ultérieure, le circuit fonctionnera de nouveau normalement. À l'inverse, les fautes permanentes sont irrémédiables. Le circuit est endommagé et ne fonctionnera plus normalement. Ces fautes peuvent toucher un ou plusieurs bits et ont divers effets :

- *bit flip* : la valeur du bit est inversée ;
- *bit set* : la valeur du bit est forcée à 1 ;
- *bit reset* : la valeur du bit est forcée à 0 ;
- *bit stuck-at* ou collage : la valeur du bit est maintenue à sa valeur précédente.

Les perturbations peuvent avoir plusieurs causes, comme par exemple des variations de température, de tension d'alimentation, d'horloge interne, *etc*. Différents moyens de provoquer des perturbations seront décrits dans ce chapitre. En fonction des fautes provoquées, un attaquant peut employer différentes méthodes pour obtenir des données sensibles.

Deux types de protection existent contre les FA : les schémas de *détection* et les schémas de *correction d'erreur*. Les schémas de détection permettent d'appliquer différentes politiques d'action lorsqu'une attaque a lieu. La destruction des données sensibles ou l'arrêt de l'exécution pour la relancer sont des exemples de réactions possibles. Plusieurs niveaux de détection sont possibles : au niveau matériel en utilisant des capteurs d'intrusion ou encore au niveau algorithmique grâce à des calculs redondants ou de l'utilisation de codes détecteurs d'erreurs.

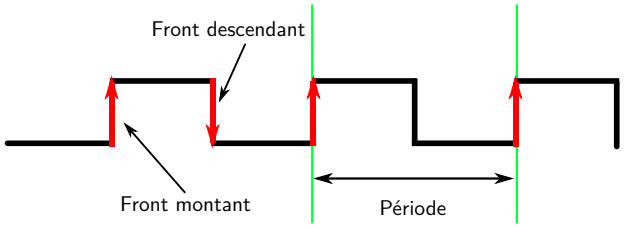


FIGURE 3.1 – Signal d'horloge interne d'un circuit

Les schémas de correction sont capables de corriger la faute provoquée lors de l'attaque. Pour cela, ils utilisent des méthodes exécutant les opérations attendues même en présence de fautes (p. ex. l'utilisation de voteurs majoritaires) ou de codes correcteurs d'erreurs.

Dans ce chapitre, différents moyens de provoquer et d'exploiter des fautes sont décrits dans un premier temps. Ensuite, des contre-mesures contre les FA sont détaillées.

3.1 Comment injecter des fautes ?

Les perturbations peuvent être provoquées de différentes manières, que ce soit par des variations de l'environnement ou encore par des injections électromagnétiques ou laser. Dans cette section, des méthodes de perturbation sont décrites.

3.1.1 Violation de contraintes temporelles

Les circuits sont cadencés par une *horloge interne* qui donne une référence temporelle, à intervalle régulier, aux différents composants du circuit. Les données des registres sont mises à jour à chaque front montant (ou descendant) de l'horloge. Pour assurer le bon déroulement de l'exécution, la période d'horloge doit être plus longue que le temps du chemin critique (chemin le plus long avec une marge pour les variations de fabrication et de fonctionnement). Le front d'horloge arrive toujours après la fin de la propagation des données les plus lentes. La modification cette cadence peut occasionner des perturbations.

Une diminution de la période modifie la cadence interne du circuit, pouvant entraîner une mauvaise sauvegarde des données. Avec cette méthode, un attaquant peut réaliser des attaques, dites, d'*Overclocking* et de *Glitch* d'horloge. Une diminution de la tension d'alimentation est une autre source de modification de la cadence d'horloge. Elle a pour conséquence l'augmentation du chemin critique. Ainsi le cycle d'horloge peut devenir trop court pour sauvegarder les données. Une hausse de la température ou une variation électromagnétique peut avoir les mêmes implications qu'une diminution de la tension d'alimentation.

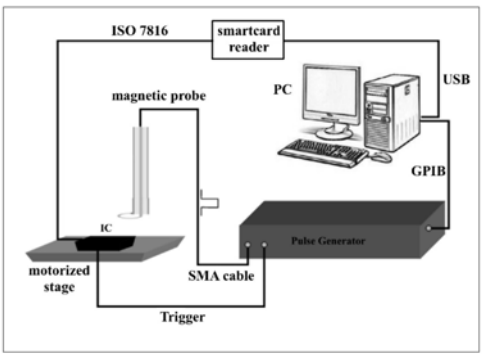
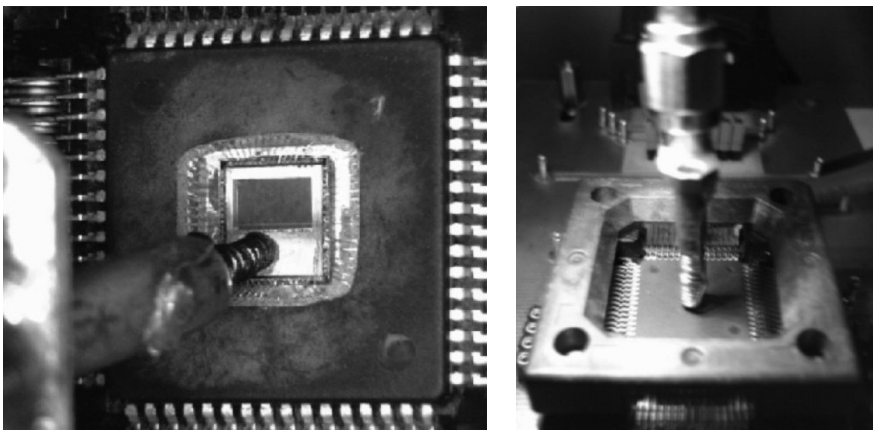


FIGURE 3.2 – Configuration d'attaque [39]



(a) Sur un dispositif ouvert (b) Sur un dispositif non altéré

FIGURE 3.3 – Sonde d'injection électromagnétique [39]

3.1.2 Injection électromagnétique

Une *injection électromagnétique* (EM), en plus de causer une violation de contraintes temporelles, peut permettre de modifier des valeurs dans le circuit. La faute engendrée est généralement temporaire et peut être locale. Le matériel nécessaire pour une telle perturbation, illustré à la figure 3.2, est principalement constitué : d'un générateur d'impulsion, d'une antenne d'injection, d'un microcontrôleur. Le générateur d'impulsion est utilisé pour envoyer des impulsions à travers l'antenne, illustré à la figure 3.3. L'antenne d'injection, généralement un solénoïde [1], peut fournir un champ électromagnétique fort.

1. « Un solénoïde est un dispositif constitué d'un fil électrique en métal enroulé régulièrement en hélice de façon à former une bobine longue. Parcouru par un courant, le solénoïde produit un champ magnétique dans son voisinage, et plus particulièrement à l'intérieur de l'hélice où ce champ est quasiment uniforme » (définition FuturaSciences)

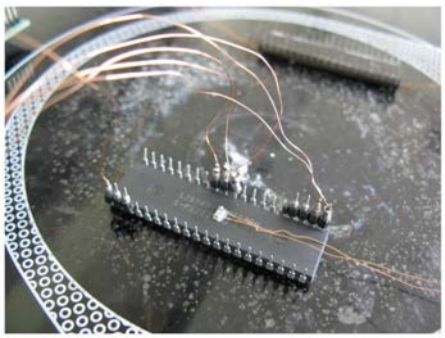


FIGURE 3.4 – Augmentation de la chaleur [65]

Cette méthode a été utilisée dans [39], sur un microcontrôleur 8 bits AVR, où un AES était implanté. Dans cette expérimentation, une faute a pu être effectuée sur un octet précis. De plus, cet octet a pu être sélectionné en choisissant le moment où la perturbation a été injectée.

Avec des injections EM, l'attaque peut améliorer sa précision. En effet, il est possible de ne fauter qu'un seul bit. C'est ce que démontre [60], sur un FPGA, où un AES-128 est implanté. L'expérimentation est réalisée sur 340 000 textes clairs aléatoires différents. Finalement, sur ces 340 000 fautes, les auteurs arrivent à générer 13 497 sorties faussées d'un seul bit.

3.1.3 Variation environnement

Un circuit fonctionne correctement sous certaines conditions prévues par le concepteur. La variation des paramètres externes du circuit peut entraîner des perturbations. Ces paramètres externes peuvent être la température, la tension d'alimentation ou encore le rayonnement électromagnétique. Comme expliqué dans la sous-section 3.1.1, une variation de la tension d'alimentation peut provoquer des violations de contraintes temporelles. Une baisse de la tension d'alimentation peut provoquer l'incapacité d'une bascule à mémoriser une nouvelle donnée. A *contrario*, une augmentation de la tension d'alimentation peut occasionner une accélération des éléments du circuit. Par conséquent, certaines instructions finissent plus tôt.

Lorsque la température du circuit varie en dehors des plages de fonctionnement prévues, une altération du circuit peut avoir lieu. Une modification aléatoire des bits de mémoire RAM, ou encore une impossibilité de la lecture de la mémoire non volatile sont des exemples d'altérations possibles. Cette méthode fut utilisée dans [65], sur un microcontrôleur ATmega162 où un RSA est implanté, pour réaliser des fautes temporaires et permanentes. Pour cela, une plaque chauffante est utilisée, comme illustré à la figure 3.4 où le circuit est placé sur la plaque. Lors de l'expérimentation, les auteurs ont constaté qu'au-dessus de 160 degrés, l'ATmega162 ne répond plus aux requêtes. De plus, ils ont également remarqué que le dispositif effectue plus de fautes lorsque la température est comprise entre 152 et 158 degrés.

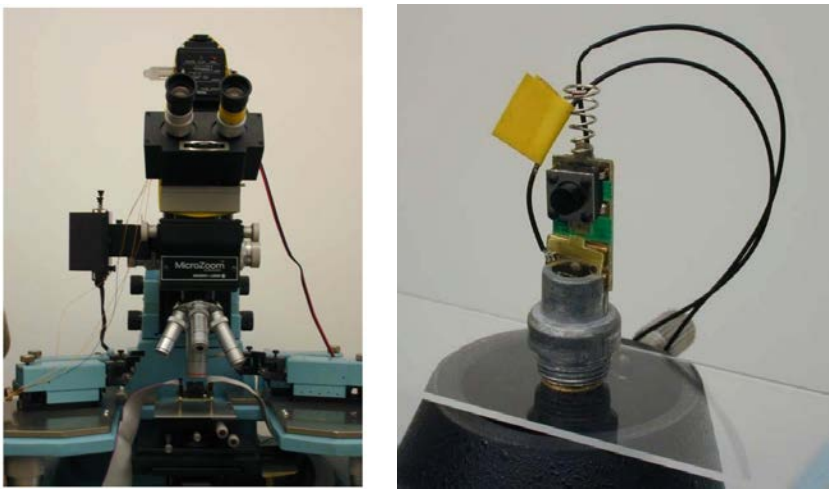


FIGURE 3.5 – Attaques par injection de lumière [104]

3.1.4 Injection laser et par lumière blanche

Quand un photon, avec une énergie plus grande que 1.1eV, frappe un électron d'une plaque métallique, il est absorbé par ce dernier. L'électron est alors éjecté de sa position initiale créant un trou. Une paire « électron - trou » se forme. Si un courant électrique est présent à cet instant, alors l'électron et le trou se séparent. De plus, si cela se produit plus d'une fois, le courant injecté peut devenir suffisant pour qu'un transistor change d'état, provoquant une faute par tir(s) laser. Ce type de perturbation permet d'avoir une faute local sur le circuit. Une perturbation par lumière blanche peut permettre d'obtenir une faute plus globale. Par exemple, dans l'article [104], un flash d'appareil photo est utilisé, par les auteurs, dans ce but, comme illustré à la figure 3.5.

3.2 Types d'attaques par perturbation

Lors d'une FA, la perturbation peut avoir lieu sur les paramètres d'entrée, le flux de contrôle, la mémoire ou encore les données durant l'exécution. Dans le cadre d'ECC, un attaquant peut injecter des fautes sur plusieurs types de données : les paramètres de la courbe, le scalaire, la représentation de corps [26], le point de base [11] et les points intermédiaires [14]. L'un des objectifs d'une attaque par perturbation est d'éviter des instructions précises, telles que des tests de sécurité. Dans ce cas, l'exploitation de la faute est dite *directe*. A *contrario*, des méthodes d'analyse sont employées afin d'obtenir la clé secrète. Dans cette section, certaines de ces méthodes sont décrites.

3.2.1 Analyse différentielle de faute (DFA)

L'*analyse différentielle de faute* (DFA) [12] consiste à exploiter la différence entre des messages chiffrés correctement et des messages chiffrés incorrectement suite à l'injection de faute(s). Pour cela, une ou des fautes du type *bit flip* sont généralement nécessaires. Ainsi, pour une même entrée (clairs), l'adversaire obtient deux sorties (chiffrés) différentes.

Exemple 3. *Dans [4], les auteurs décrivent une attaque d'un RSA qui peut facilement s'appliquer à ECC. Dans cette attaque, la faute est un bit flip sur le $j-$ème bit de la clé, où j est aléatoire. Soit $\widetilde{Q}$ le résultat fauté de la SM, alors Q et $\widetilde{Q}$ peuvent s'écrire :*

$$\begin{aligned} \widetilde{Q} = [\widetilde{k}]P &= \sum_{i=0}^{j-1} k_i 2^i P + \widetilde{k}_j 2^j P + \sum_{i=j+1}^{m-1} k_i 2^i P \\ Q = [k]P &= \sum_{i=0}^{m-1} k_i 2^i P \end{aligned} \tag{3.1}$$

Connaissant Q et $\widetilde{Q}$, l'adversaire peut calculer $Q - \widetilde{Q}$ ce qui lui permet de déduire le bit de clé k_j. En effet, si $Q - \widetilde{Q} = -2^j P$ alors $k_j = 0$ et si $Q - \widetilde{Q} = 2^j P$ alors $k_j = 1$. Finalement, l'attaquant peut retrouver la totalité de la clé en réitérant l'opération précédente avec d'autres valeurs de j.

3.2.2 Analyse *safe-error* (SEA)

L'attaque par analyse *safe-error* (SEA) [115] est une méthode d'exploitation extrêmement simple contre certaines protections. De plus, L'injection de la faute n'a pas besoin d'être très précise. Elle consiste à observer l'impact de la faute sur le calcul final. Si la faute n'a pas d'impact sur le résultat de l'opération cryptographique, alors l'adversaire peut en déduire par exemple que la valeur corrompue n'a pas été utilisée lors du calcul. Ainsi, il peut en déduire des informations sensibles. Inversement, si la faute à un impact sur le résultat alors l'adversaire peut également en déduire des informations sensibles, comme cela est illustré dans l'exemple qui suit.

Exemple 4. *Lors d'une SM réalisée avec l'algorithme du doublement et addition toujours, des `ADD` factices sont faites si le bit de clé est égal à 0. Comme les SEA n'ont pas besoin d'être précises, l'attaquant peut « viser » l'une des sous-opérations durant une opération factice. Par conséquent, il peut choisir le moment où la faute sera injectée et peut essayer de cibler les `ADD` en particulier. Dans ce cas, si le résultat de la SM est correct, alors la faute injectée durant `ADD` n'a pas eu d'impact sur le résultat final. Cette `ADD` est donc une opération factice et alors le bit de clé est égal à 0. Au contraire, si le résultat de la SM est erroné, alors l'opération `ADD` était utile au calcul de la SM et donc le bit de clé est égal à 1. Finalement, il suffit de répéter l'opération le nombre de fois nécessaire pour retrouver la clé en totalité.*

3.2.3 Autres méthodes

De nombreuses autres méthodes, pas forcement applicables à ECC, d'exploitation de faute existent, tels que l'analyse par collision (CFA) ou encore la réduction de ronde (RR). L'objectif de la CFA [13] est de produire une collision entre deux sorties de l'algorithme. Pour cela, l'attaquant choisit un message m_0 arbitraire qu'il chiffre en provoquant une erreur lors de l'opération. Ainsi, il obtient $\widetilde{c} = \widetilde{E}(m_0)$. Ensuite, il cherche un message M tel que $E(M) = c = \widetilde{c}$ qu'il compare avec $\widetilde{c} = \widetilde{E}(m_0)$ pour retrouver des informations sur la clé.

Le principe de la RR est de réduire le nombre de tours d'un algorithme. En effet, certains algorithmes sont basés sur la répétition de séquence identique. La sécurité, de ces algorithmes (p. ex. AES), dépend du nombre de tours effectué. Si celui-ci est réduit alors la sécurité diminue.

3.3 Attaques hybrides

Les *attaques hybrides* combinent les attaques par canaux cachés et par injection de fautes. Le principe est de rechercher des conditions pour lesquelles la faute apparaît. La réaction du circuit à l'injection de faute est considérée comme une fuite.

3.3.1 Analyse de la sensibilité à la faute (FSA)

Le stress d'un circuit est la sensibilité de celui-ci vis-à-vis d'une injection de faute. Plus le stress augmente, plus le circuit risque de faire une faute. Le but de l'*analyse de la sensibilité à la faute* (FSA) [83, 82] est de repérer le stress limite à partir duquel une faute est injectée. L'adversaire applique un stress au circuit. Puis il l'augmente progressivement jusqu'à ce qu'une valeur critique soit détectée. Une corrélation entre cette valeur du stress critique et les données sensibles, manipulées par le circuit, existe. Le principe de cette attaque est similaire à une CPA mais ne nécessite pas la valeur des sorties faussées.

Trois étapes articulent cette attaque. La première est la récupération des valeurs du temps critique Tc pour différentes entrées m. L'attaquant obtient des couples (message m_n, temps critique Tc_n). Ensuite, des résultats prédictibles $r_p(k)$ sont calculés en fonction des hypothèses sur la clé secrète k. L'adversaire obtient les triplets $(m_n, k, r_p(k))$. La corrélation, des n-uplets (m_n, Tc_n) et $(m_n, k, F(k))$, est ensuite calculée avec le coefficient de Pearson.

Cette attaque, initialement proposée pour attaquer un AES, fut adaptée à ECC dans [103]. L'attaque, réalisée sur une carte SASEBO-R avec un FPGA virtex II, cible une SM implantée avec l'algorithme de López-Dahab, sur le corps $\mathbb{F}_{2^{63}}$ avec une clé de 63 bits.

3.3.2 Analyse comportementale différentielle (DBA)

L'*attaque par analyse comportementale différentielle* (DBA) [101] combine les attaques SEA et DPA. Elle se base sur les changements en sortie lorsqu'une faute de collage a lieu. Grâce à cette méthode, qui s'articule en 4 étapes, la clé d'un AES-128 a été complètement retrouvée. Pour être efficace, l'attaquant doit pouvoir faire la distinction entre le comportement normal ou anormal du circuit.

Étape 1 : Choix des paramètres

Avant de réaliser l'attaque en elle-même, un adversaire doit définir certains paramètres : le nombre de bits ciblés N ; les hypothèses de clé k_p ; le nombre de bits fautés M ; l'ensemble des textes clairs T ; et la valeur de la faute. Pour la suite de la description de l'attaque, il est supposé que $N = M = 1$.

Plusieurs hypothèses pour l'attaque sont posées dans [101]. La faute de collage provoquée n'a pas besoin d'être connue mais doit toujours avoir la même valeur pour tous les bits affectés. De plus, la faute doit s'effectuer sur des valeurs intermédiaires particulières et n'affecter qu'un petit nombre de bits (typiquement pas plus de 8).

Étape 2 : Expérimentation

Pour tous les textes clairs t_i, un chiffrement, sans et avec perturbation, est effectué. L'adversaire stocke le comportement du circuit pour chaque chiffrement. Le comportement normal est noté $C_0(t_i)$ et le comportement anormal est noté $C'(t_i)$. Ensuite, il associe la valeur $c(t_i)$ à chaque texte clair t_i, grâce à l'équation 3.2 qui retourne 1 si la faute injectée produit une erreur.

$$c(t_i) = \begin{cases} 0 & si \quad C'(t_i) = C_0(t_i) \\ 1 & sinon \end{cases} \tag{3.2}$$

Étape 3 : Calcul de la corrélation

Pour chaque hypothèse de clé k_p, la mesure de corrélation Δ se calcule avec la formule suivante, où r_{t_i,k_p} est un résultat prédictible dépendant de t_i et de k_p :

$$\Delta = \frac{\sum_{t_i \in T} [r_{t_i,k_p} \times c(t_i) + (1 - r_{t_i,k_p}) \times (1 - c(t_i))]}{|T|} \tag{3.3}$$

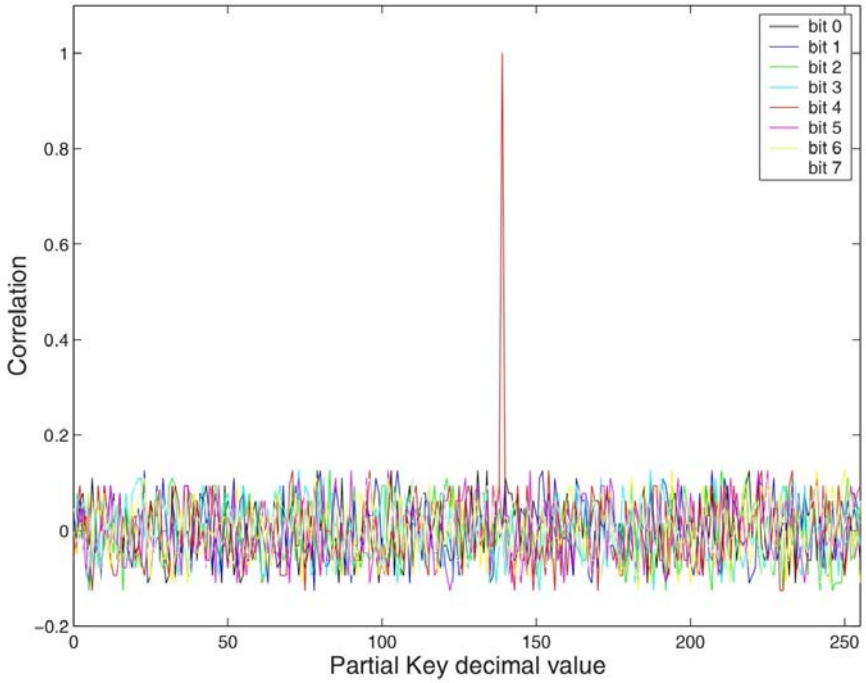


FIGURE 3.6 – Résultat d'une attaque DBA sur un AES [101]

Étape 4 : Interprétation

L'attaquant possède alors des couples (k_p, Δ). Grâce à ces points, il trace une courbe nommée DBA. Ensuite, les courbes DBA, associées à toutes les combinaisons possibles de bits collés, sont superposées. Un pic s'observe pour la valeur qui a été fautée, comme illustré à la figure 3.6.

3.4 Attaques spécifiques aux courbes elliptiques

La plupart des attaques [11, 26, 14], décrites prochainement, reposent sur le fait que le paramètre a_6 de la courbe E (cf équation 1.1, chap. 1) n'est pas utilisé pendant la SM. Elles consistent à transférer le problème du logarithme discret (DLP) sur une courbe moins robuste. Dans cette partie, quelques attaques sont décrites.

3.4.1 Faute sur le point d'entrée

Le point de base P est l'une des cibles potentielles d'une attaque. L'attaque, décrite dans [11], perturbe P afin qu'il n'appartienne plus à la courbe E. Ce point devient $\widetilde{P}$ tel que

$\widetilde{P} \notin E$ et $P \in \widetilde{E}$, où $\widetilde{E}$ est une courbe définie par $(a_1, a_2, a_3, a_4, \widetilde{a}_6)$ tel que :

$$\widetilde{a}_6 = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x \tag{3.4}$$

Le calcul de $[k] \cdot \widetilde{P}$ sur la courbe $\widetilde{E}$ remplace celui de $[k] \cdot P$ sur la courbe E. Comme $[k] \cdot P = [k] \cdot \widetilde{P}$ sur $\widetilde{E}$, le DLP est transféré dans un groupe d'ordre r où r est l'ordre de $\widetilde{P}$ et où il est plus aisé de le résoudre. Ainsi, la valeur de $k \bmod r$ peut être retrouvée. Le théorème chinois des restes (CRT) permet de retrouver la valeur de k en réitérant ce procédé avec différentes valeurs de $\widetilde{P}$. Cette attaque n'est possible que sur des implantations qui ne vérifient pas si le point de base P se trouve sur la courbe au début de la SM.

3.4.2 Faute sur le point de base après vérification

Lorsque le dispositif vérifie les entrées de l'algorithme avant la SM, l'attaque précédente ne fonctionne plus. Cependant, si les sorties de la SM ne sont pas vérifiées alors, une attaque sur un point intermédiaire [11] est faisable.

Une faute est supposée générée sur un bit d'une des coordonnées du point de base P, au début de la SM mais après la vérification de l'appartenance de P à la courbe. Le dispositif calcule alors $[k] \cdot \widetilde{P}$ sur $\widetilde{E}$ avec $\widetilde{P}$. Comme la faute est effectuée sur un bit de P, le point $\widetilde{P}$ est tel qu'il ne diffère que d'un bit de P et que $\widetilde{P} \in \widetilde{E}$. Le DLP sur E se réduit alors à un DLP sur $\widetilde{E}$ et se résoudre pour les points $\left[\frac{r}{s}\right] \widetilde{P}$ où r est l'ordre de $\widetilde{E}$ et s un petit diviseur de r. Par conséquent, la valeur de $k \bmod s$ peut être retrouvée. Comme précédemment, le CRT permet de retrouver la valeur de k en répétant cette opération pour différentes valeurs de s.

3.4.3 Faute sur la définition du corps

Dans un circuit, le corps $\mathbb{F}_p$ est représenté par p, stocké en binaire dans la mémoire. Ce p est une cible potentielle [26]. Une faute permanente sur p fausse la représentation de $\mathbb{F}_p$ lors de l'exécution. Les calculs sont alors faits modulo $\widetilde{p}$ et la SM devient $\widetilde{Q} = [k]\widetilde{P} = (\widetilde{x}, \widetilde{y})$. La courbe E devient $\widetilde{E}$, et $\widetilde{a}_6$ peut s'écrire :

$$\widetilde{a}_6 = y^2 - x^3 - a_4x = \widetilde{y}^2 - \widetilde{x}^3 - a_4\widetilde{x} \bmod \widetilde{p} \tag{3.5}$$

Soient R_0, R_1 et S définies par les équations suivantes :

$$\begin{cases} R_0 &= y^2 - x^3 - a_4x \\ R_1 &= \widetilde{y}^2 - \widetilde{x}^3 - a_4\widetilde{x} \\ S &= R_0 - \tilde{R}_1 \end{cases} \tag{3.6}$$

Alors $\widetilde{p}$ divise S et $\widetilde{p}$ est une combinaison de facteurs de S qui a le plus de bits en commun avec p. Ainsi, la factorisation de S permet de retrouver $\widetilde{p}$ dont la factorisation en facteurs premiers s'écrit $\widetilde{p} = \prod_{i=1}^{l} q_i^{e_i}$. Comme le point $\widetilde{Q} \in \{\widetilde{Q}_1, \ldots, \widetilde{Q}_l\}$ avec $\widetilde{Q}_i = \widetilde{Q} \bmod q_i^{e_i}$ alors, le DLP est transféré dans un groupe bien plus petit que $\mathbb{F}_p$. Finalement, comme pour les attaques précédentes, la valeur de la clé peut se retrouver grâce au CRT.

3.4.4 Faute sur les paramètres de la courbe

Une courbe de Weierstrass générale s'écrit $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ où les a_i sont les paramètres de la courbe qui sont des cibles potentielles de perturbations [26]. Comme le paramètre a_6 n'est pas utilisé dans les formules `ADD` et `DBL`, il peut être supposé comme corrompu. De plus et sans perte de généralité, le paramètre a_4 est aussi supposée ciblé par l'attaque. Une faute sur les autres paramètres serait similaire. L'attaquant possède le résultat de la SM faussée $\widetilde{Q} = [k]\widetilde{P} = (\widetilde{x}, \widetilde{y})$ ainsi que le système d'équations 3.7 suivant :

$$\begin{cases} \widetilde{a}_4x + \widetilde{a}_6 &= y^2 + a_1xy + a_3y - x^3 - a_2x^2 \\ \widetilde{a}_4\widetilde{x} + \widetilde{a}_6 &= \widetilde{y}^2 + a_1\widetilde{x}\widetilde{y} + a_3\widetilde{y} - \widetilde{x}^3 - a_2\widetilde{x}^2 \end{cases} \tag{3.7}$$

La résolution de ce système permet de trouver $\widetilde{a}_4$ et $\widetilde{a}_6$. Ensuite, l'attaquant calcule le logarithme de $\widetilde{Q}$ sur la courbe faussée $\widetilde{E}$. Comme pour les attaques précédentes, la valeur de la clé peut se retrouver grâce au CRT.

3.4.5 Conclusion

Les systèmes embarqués sont particulièrement sensibles aux attaques physiques en raison de leur proximité avec des usagers potentiellement malveillants. Les FA sont nombreuses et peuvent se réaliser plus ou moins facilement en pratique. Certaines nécessitent parfois une préparation du circuit (p. ex. décapsulation). Il est donc primordial d'avoir des protections contre les FA.

3.5 Protections contre les attaques par perturbation

Les attaques en faute perturbent le système durant l'exécution pour induire un comportement non spécifié du circuit. Des contre-mesures de ces attaques sont principalement basées sur la *détection* ou la *correction* de la faute. Grâce à la détection de l'attaque, plusieurs politiques d'action peuvent s'envisager, comme l'interruption de l'activité du système par un « suicide » de la puce (p. ex. en effaçant toutes les informations secrètes). Des capteurs, déterminant des modifications de l'environnement comme des capteurs de température, peuvent

permettre de détecter si une intrusion s'est produite. La détection est également possible grâce à des calculs redondants ou à des vérifications d'intégrité de données ou du flot de contrôle. La correction de la faute permet de ne pas laisser filtrer d'information à partir du résultat. Pour cela, la faute provoquée doit pouvoir être corrigée lors de l'exécution grâce par exemple à un vote par majorité. Dans cette partie, des protections contre les FA sont décrites.

3.5.1 Calculs redondants

La *redondance* de calcul constitue une part importante des protections contre les perturbations (volontaires ou non) du système. Elle consiste *à répéter* des séquences d'opérations lors de l'exécution. Trois types de redondance existent : *spatiale, temporelle* et d'*information*. Cette méthode permet, non seulement, de détecter des fautes, mais aussi de les corriger (sous certaines contraintes).

Redondance spatiale

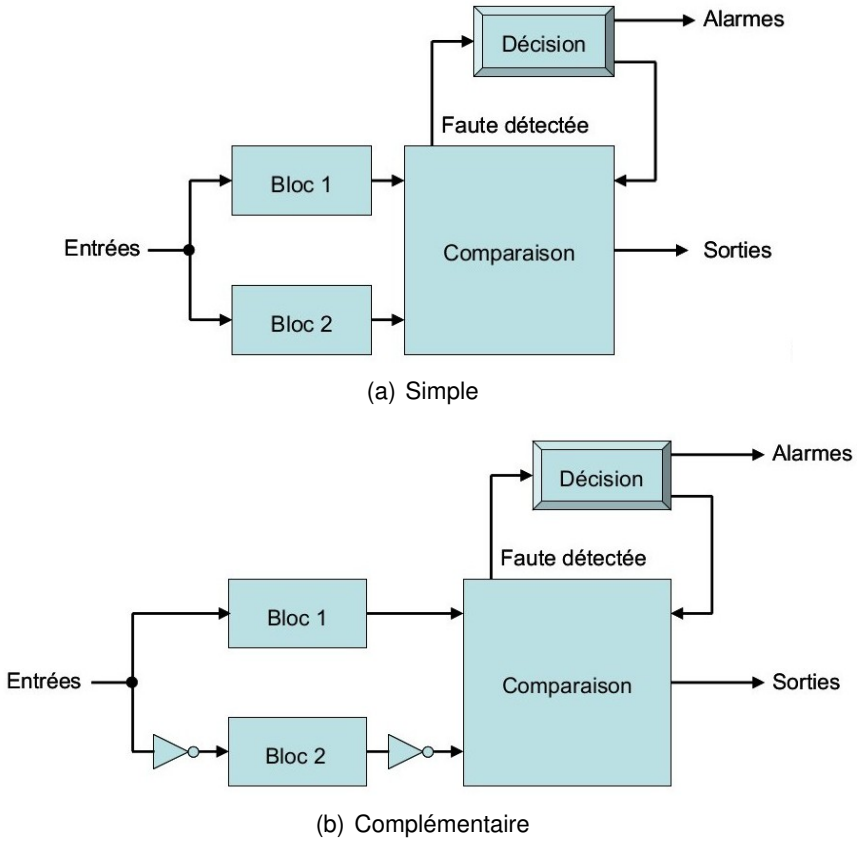
La *redondance spatiale* duplique les blocs d'opérateurs qui exécutent la séquence d'opérations. Plusieurs méthodes de duplication existent : simple, simple complémentaire, multiple,*etc*.

La duplication simple, illustrée à la figure 3.7(a), permet de détecter des erreurs simples (c.-à-d. une seule erreur par exécution). La même opération est effectuée sur plusieurs blocs qui font le même calcul. Ces blocs correspondent aux blocs 1 et 2 de la figure 3.7(a). Les résultats obtenus par les blocs 1 et 2 sont comparés grâce à un bloc de comparaison. Le résultat de la comparaison est ensuite envoyé à un bloc décisionnel qui détermine si une faute s'est produite ou non. Néanmoins, si l'adversaire arrive à reproduire la même faute au même endroit sur le bloc dupliqué, alors l'erreur ne sera pas détectée.

La duplication avec redondance complémentaire, illustrée à la figure 3.7(b), permet de rendre cette double faute plus compliquée. En effet, le bloc dupliqué est différent du premier bloc. Cette différence peut se traduire par un échange des opérandes par exemple. À la fin des blocs de calculs, les résultats sont comparés. Puis le bloc décisionnel détermine si une faute s'est produite ou non.

La duplication multiple, illustrée à la figure 3.8, permet de détecter et de corriger une erreur. Contrairement à une redondance simple, les blocs sont dupliqués n fois ($n \geq 3$). La triplication ($n = 3$), permettant de corriger une erreur, est la redondance multiple la plus utilisée. La correction de l'erreur se réalise grâce à un vote majoritaire où le résultat ayant obtenu le plus de voix est renvoyé. Bien sûr, l'élément effectuant le vote doit être protégé.

La redondance dynamique est une amélioration de la redondance multiple. Dans ce contexte, lorsqu'une erreur est détectée, le bloc ayant fait l'erreur est déconnecté. Il ne sera plus utilisé jusqu'à la ré-initialisation du système. Une autre amélioration de la duplication mul-



(a) Simple

(b) Complémentaire

FIGURE 3.7 – Redondance matérielle [21]

tiple est la redondance hybride. Il s'agit d'une duplication multiple dont certains blocs sont complémentaires.

La capacité de détection et de correction de ces schémas dépend du nombre de blocs dupliqués [5]. Si un schéma possède $2n + 1$ blocs, alors il sera capable de détecter $2n$ erreurs et d'en corriger n. Cette forme de redondance a un coût en surface et en consommation énergétique non négligeable.

Redondance temporelle

La *redondance temporelle* effectue le même calcul avec le même bloc matériel, mais à des instants différents. Elle repose sur le même principe que la redondance matérielle et est efficace contre les fautes temporaires. Les mêmes catégories que précédemment sont présentes pour une duplication temporelle. La redondance simple, permettant la détection de fautes, est illustrée à la figure 3.9(a). Le calcul est répété après un certain délai, puis les résultats sont comparés. Comme précédemment, un attaquant capable d'injecter la même faute au bon moment pourra contourner la protection. La redondance simple avec une rotation des opérandes

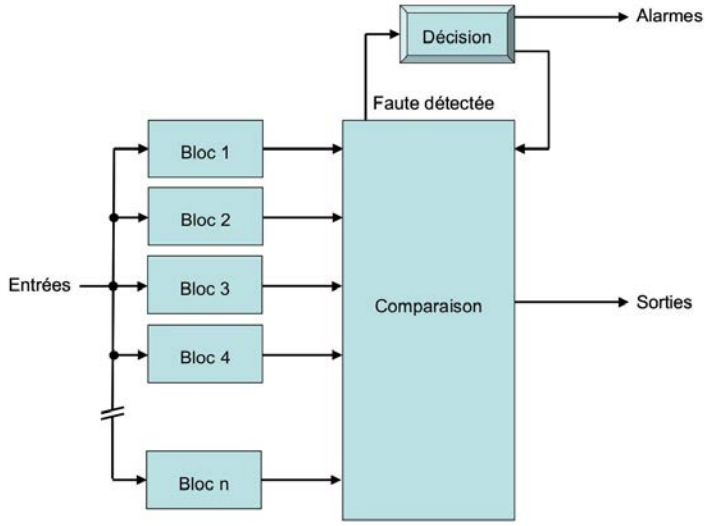


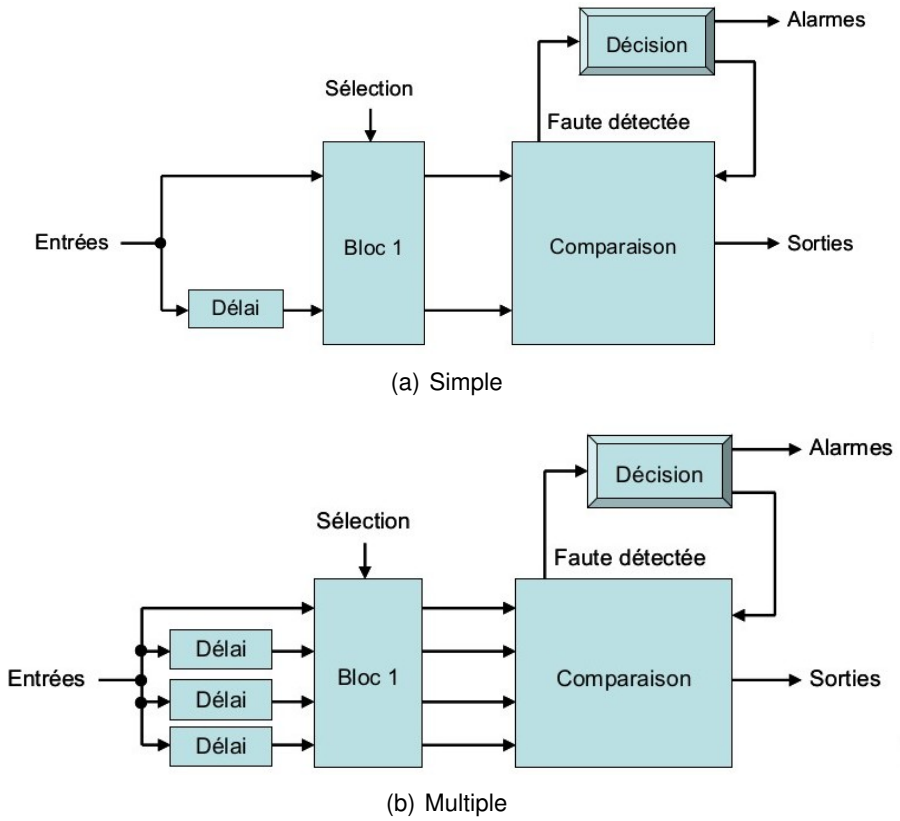
FIGURE 3.8 – Duplication multiple [21]

peut rendre ce type d'attaque plus difficile. Les opérandes sont échangés avant et après le deuxième calcul. Une redondance multiple, illustrée à la figure 3.9(b), permet la détection et la correction d'erreurs. Une fois tous les calculs effectués, un vote majoritaire s'effectue. Le résultat ayant obtenu le plus de voix est retourné.

Redondance d'information

Le troisième type de redondance est la *redondance d'information*. Elle se base sur l'utilisation de code détecteur ou correcteur d'erreurs. Un code associé à la sortie du bloc de calcul est comparé avec un code de référence. Ce code de référence s'obtient grâce à un bloc de prédiction de code. Si ce bloc de prédiction est moins coûteux que le bloc de calcul, alors le coût en surface de cette redondance est moindre qu'une redondance matérielle. De plus, même si ce type de redondance est plus complexe à réaliser (en particulier pour l'arithmétique dans $\mathbb{F}_p$), elle est aussi plus complexe à attaquer. Plusieurs types de codes sont utilisables pour effectuer de la redondance d'information : *checksum*, parité, linéaire, non linéaire, *etc*.

Les codes *checksum* et de parité, permettant de détecter une faute, consistent à ajouter une donnée au message. Cette donnée est dépendante du message en lui-même. Les codes linéaires, tels que les codes de Hamming [58] ou de Hsiao [64], permettent de détecter deux fautes (si le nombre de fautes n'est pas un multiple de trois) et d'en corriger une. Ces codes sont plus adaptés aux systèmes où les fautes multiples sont difficiles à réaliser. Les codes non linéaires, tels que les codes de Berger [7], de Dong [43] ou les codes arithmétiques [16], se



(a) Simple

(b) Multiple

FIGURE 3.9 – Redondance temporelle [21]

basent sur des opérations non linéaires. Ces opérations peuvent être des juxtapositions du poids de Hamming de la donnée en binaire.

3.5.2 Vérification par l'inverse

Les méthodes de vérification par l'inverse consistent à effectuer l'inverse de l'opération cryptographique $f(x)$, noté $f^{-1}(x)$. Il suffit, ensuite, de vérifier que $f^{-1}(f(x)) = x$. Cette étape de vérification doit se réaliser de manière sécurisée. Une attaque en faute devient alors plus compliquée à effectuer. En effet, si l'attaquant veut réussir une attaque, il doit être capable de fauter la comparaison (qui est sécurisée), en plus de l'opération $f(x)$. Ce type de vérification ne s'applique pas dans le contexte d'ECC.

3.5.3 Détection d'une perturbation sur ECC

En plus des calculs redondants, des mécanismes spécifiques à ECC, permettant de déterminer si une faute s'est produite, existent. Ces méthodes peuvent varier en fonction de la cible visée par l'adversaire (le point, la courbe ou le scalaire).

Attaque des points et paramètres de la courbe

Lors d'une attaque par perturbation sur ECC, l'adversaire peut cibler les paramètres de la courbe et/ou les coordonnées du point base ou d'un point intermédiaire. La vérification du point de base ou du point intermédiaire [11] permet d'éviter ce type d'attaque. Pour cela, les coordonnées du point sont injectées dans l'équation de la courbe. Si l'équation est vérifiée alors aucune faute n'est détectée. Cette vérification peut se faire à différents moments : au début et à la toute fin de la SM ou toutes les d itérations. Cependant, cette solution ne protège pas des attaques visant le scalaire. En effet, si le scalaire est modifié, les points intermédiaires et le point final restent sur la courbe et la faute n'est pas détectée.

Attaque sur le scalaire

Lorsque le scalaire est la cible de l'attaque, la vérification de l'équation de la courbe ne permet pas de détecter la faute. Des méthodes de redondance spécifiques à ECC [14] permettent de détecter des injections de faute. Les méthodes de randomisation et de fission additive du scalaire (voir sec. 2.3.2) peuvent également permettre de contrer une FA sur le scalaire, si plusieurs exécutions sont nécessaires.

3.5.4 Politique d'action en cas de détection de fautes

La détection de fautes permet d'envisager différentes politiques d'action qui dépendent de l'application et des menaces associées. Une première politique d'action est de continuer l'exécution du système en retournant le résultat erroné. À cause de cette non-réaction, l'attaquant est capable de réaliser des attaques DFA ou SEA. Une deuxième politique d'action est de stopper, puis de recommencer le calcul pour ne pas renvoyer un résultat faux. Par conséquent, le résultat est retourné avec un délai supplémentaire lorsqu'une faute est détectée par le système. Avec cette politique d'action, le système peut être vulnérable aux attaques du type SEA et DBA. L'arrêt de l'exécution est une autre réaction possible ayant les mêmes inconvénients que la politique d'action précédente. Toutes ces réactions sont susceptibles de fragiliser le cryptosystème.

Une autre réaction consiste à continuer le calcul sans retourner le résultat erroné et à rendre le cryptosystème inutilisable temporairement ou définitivement. Cette inhibition du système peut permettre d'éviter ou de compliquer les attaques DFA et DBA. Puisqu'une attaque DFA a besoin du résultat corrompu, si le résultat n'est pas renvoyé ces attaques sont évitées. Une attaque DBA nécessite plusieurs traces d'acquisition. Ainsi, si le système est inutilisable temporairement ou définitivement, l'adversaire ne pourra pas obtenir ces traces. L'effacement des données secrètes, aura le même impact que précédemment. La poursuite des calculs avec une clé aléatoire est également une possibilité envisageable.

3.5.5 Conclusion

Les protections contre les FA sont basées sur la détection et sur la correction de l'erreur provoquée. Le calcul redondant est l'une des méthodes principales pour détecter ou corriger une (ou des) faute(s). Dans le cadre d'ECC, la vérification de l'appartenance du point à la courbe est une solution efficace, et peu coûteuse, contre les attaques sur les points intermédiaires et sur les paramètres de la courbe. Cependant, l'ajout de calculs qui n'existeraient pas dans un système régulier peut compromettre sa résistance aux SCA par une dissymétrie n'existant pas sans la protection ajoutée.

CHAPITRE 4

ARITHMÉTIQUE DES ORDINATEURS POUR MICROCONTRÔLEURS

Les systèmes embarqués sont omniprésents dans notre société. Ils doivent être performants tout en ayant une forte contrainte de consommation énergétique, de taille et de coût. Ils sont souvent construits autour d'un microcontrôleur, schématisé à la figure 4.1. Un *microcontrôleur* est un circuit intégré composé principalement d'un *petit processeur* (typiquement de 8 à 32 bits de données), de *petites mémoires* ROM et RAM (internes ou externes), de *périphériques* d'entrée/sortie et d'une *horloge interne*.

Le processeur est l'élément central d'un microcontrôleur. Il interprète le programme contenu dans la mémoire ROM et traite les données contenues dans la mémoire RAM. Des unités arithmétiques et logiques (UAL) sont utilisées par le processeur afin de faire les calculs demandés par le programme. Les données manipulées sont placées dans des registres dont la taille dépend du type d'architecture du microcontrôleur (p. ex. 8, 16 ou 32 bits). De nombreuses familles de microcontrôleur existent, comme les microcontrôleurs PIC de Microchip, les microcontrôleurs AVR et XMEGA d'Atmel ou encore les microcontrôleurs Cortex ARM de ARM Ltd.

Lors de calculs cryptographiques, les nombres manipulés sont très grands (plus d'une centaine de bits). Par conséquent, les UAL ne peuvent pas les réaliser directement. En effet, ces

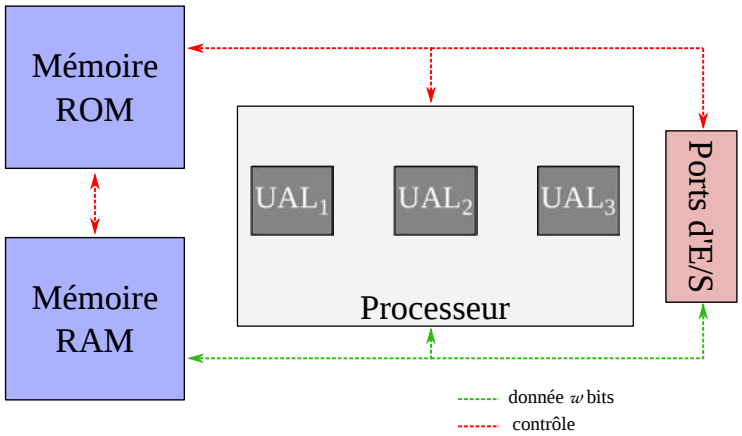


FIGURE 4.1 – Schéma de l'architecture simplifiée d'un microcontrôleur

cellules ne traitent que des nombres de 8, 16 ou 32 bits et que pour des entiers assez simples (et des flottants si une unité flottante est présente).

Les *grands nombres* sont donc *découpés* en plusieurs *mots machines* ou *chiffres* (d'une représentation en très grande base). Ainsi, un nombre A s'écrit : $A = \sum_{i=0}^{d-1} x_i \cdot 2^{i \cdot w}$ où d est le nombre de mots x_i de taille w bits. Ces mots sont ensuite placés dans des registres, dont la taille dépend du type d'architecture. Les UAL sont alors appliquées sur ces mots. Les opérations, réalisées sur des mots machines, sont notées `wadd` et `wmul` dans ce manuscrit (w est la taille des mots).

Les opérations sur les grands nombres sont réalisées grâce à des algorithmes qui appliquent le principe de *diviser pour régner*. Ces algorithmes permettent d'exploiter les UAL, puisqu'elles ne savent réaliser que des opérations de mots de taille w bits.

Dans ce chapitre, différents algorithmes pour effectuer les opérations de corps tels que la réduction modulaire, la multiplication et l'addition sont décrits.

4.1 La réduction modulaire

Les calculs lors de la SM s'effectuent dans un corps fini ($\mathbb{F}_p$ dans cette thèse). Ce corps est défini par un nombre premier p de taille n bits. Si deux entiers de n bits sont multipliés, alors le résultat sera de taille $2n$ bits et sera, très souvent, plus grand que p. Par conséquent, une réduction modulaire est primordiale afin de garder les nombres dans le corps.

La réduction modulaire de a par p, notée $a \bmod p$, s'effectue en calculant le reste r de la division euclidienne (DE) de a par p, décrit par le calcul de l'équation 4.1. Dans cette équation, q est le quotient et r le reste de la division, tel que $r < p$ et donc $r = a \bmod p$.

$$a = p \times q + r \tag{4.1}$$

Différents algorithmes de réduction modulaire existent. Certains sont spécifiques à des formes particulières de nombres premiers, d'autres fonctionnent pour des nombres premiers génériques.

4.1.1 Réduction de Barrett

La *réduction modulaire de Barrett* [6] consiste à faire une approximation du calcul du quotient q lors de la DE de a par p, décrite à l'équation 4.2. Dans cette équation, n est la taille de p, $\beta = 2^w$ est la base d'écriture des mots et $\left\lfloor \frac{\beta^{2n}}{p} \right\rfloor$ est précalculé.

Algorithme 10 : Réduction de Barrett

Entrées : a, p avec $0 \leq a < p^2$ et $v \leftarrow \left\lfloor \frac{\beta^{2n}}{p} \right\rfloor$ précalculé

Résultat : $s \equiv a \bmod p$

1 $m \leftarrow \left\lfloor \frac{a}{\beta^{n-1}} \right\rfloor$
2 $u \leftarrow m \times v$
3 $q \leftarrow \left\lfloor \frac{u}{\beta^{n+1}} \right\rfloor$
4 $s \leftarrow a - q \times p \bmod \beta^{n+1}$
5 **si** $s \geq p$ **alors**
6 $\quad s \leftarrow s - p$
7 **si** $s \geq p$ **alors**
8 $\quad s \leftarrow s - p$
9 **retourner** s

$$\begin{aligned} q &= \left\lfloor \frac{a}{p} \right\rfloor = \left\lfloor \frac{\frac{a}{\beta^{n-1}} \times \frac{\beta^{2n}}{p}}{\beta^{n+1}} \right\rfloor \\ q \simeq \tilde{q} &= \left\lfloor \frac{\left\lfloor \frac{a}{\beta^{n-1}} \right\rfloor \times \left\lfloor \frac{\beta^{2n}}{p} \right\rfloor}{\beta^{n+1}} \right\rfloor \end{aligned} \tag{4.2}$$

Comme les mots sont représentés dans la base β, les divisions par β^{n+1} et par β^{n-1} ne sont que de simples décalages. L'erreur de l'approximation de q se corrige avec au plus deux soustractions par p et est estimée par $q - 2 \leq \tilde{q} \leq q$. L'algorithme 10 décrit la procédure de la réduction modulaire de Barrett.

4.1.2 Réduction de Montgomery

La *réduction de Montgomery* [89], décrite à l'algorithme 11, nécessite une représentation spécifique des nombres, appelée le domaine de Montgomery. Un nombre a du domaine de Montgomery s'écrit $a = aR \bmod p$ où $R = \beta^d$ avec d le nombre de chiffre de p et $\beta = 2^{32}$ la base d'écriture des mots. Grâce à cette représentation, les divisions sont évitées pour être remplacées par de simples décalages, comme pour la réduction de Barrett.

Le résultat de l'algorithme 11 n'est pas la valeur exacte de $a \bmod p$ mais correspond à la valeur de $aR^{-1} \bmod p$. Une multiplication du résultat final par R permet d'obtenir la valeur de $a \bmod p$ en représentation classique. La conversion vers ou depuis le domaine de Montgomery

Algorithme 11 : Réduction de Montgomery

Entrées : a, p avec $a < pR$ et $-p^{-1} \bmod \beta$ précalculé où $\beta = 2^w$
Résultat : $s \equiv aR^{-1} \bmod p$ avec $s < 2p$

1 $r \leftarrow a \ (= \sum_{i=0}^{2n-1} a_i \beta^i)$
2 **pour** $i = 0$ **à** $n - 1$ **faire**
3 $\quad q_i \leftarrow -r_i p^{-1} \bmod \beta$ /* q_i est un mot machine de w bits */
4 $\quad r \leftarrow r + q_i p \beta^i$
5 $s \leftarrow \frac{r}{R}$
6 **retourner** s

n'est nécessaire qu'en tout début et en toute fin des calculs. Son surcoût est négligeable pour ECC.

4.1.3 Réduction avec un module spécifique

Les nombres premiers de Mersenne ou pseudo-Mersenne possèdent des propriétés qui leur sont spécifiques. L'utilisation de ces propriétés permet de réduire le coût des algorithmes de réduction modulaire. Cependant, les algorithmes de réduction, obtenus alors, ne fonctionnent pas avec tous les nombres premiers (à chaque forme de premier correspond un algorithme de réduction particulier).

Les nombres de Mersenne et pseudo-Mersenne

Les nombres de Mersenne s'écrivent sous la forme $p = 2^l - 1$. Un nombre a quelconque peux se décomposer en deux parties a_0 et a_1, tel que $a = a_1 \times 2^l + a_0$. Ainsi, la réduction modulaire de a par p est simplement $a \equiv a_1 + a_0 \bmod p$ puisque $2^l = 1 \bmod p$. L'algorithme 12 récapitule la réduction modulaire avec un nombre de Mersenne [76].

Algorithme 12 : Réduction modulaire avec un nombre de Mersenne

Entrées : a et p avec $0 \leq a < p^2$ et $p = 2^l - 1$
Résultat : $s \equiv a \bmod p$

1 $a_1 \leftarrow a >> l$
2 $a_0 \leftarrow a - (a_1 << l)$
3 $s \leftarrow a_0 + a_1$
4 **si** $s \geq p$ **alors**
5 $\quad s \leftarrow s - p$
6 **retourner** s

Algorithme 13 : Réduction modulaire avec un nombre de Solinas

Entrées : a et p où $p = P(2^k)$ avec $P(X) = X^d - C(X)$ et A la représentation polynomiale de a.
Résultat : s avec $s \equiv a \bmod p$

1 $L \leftarrow A \bmod X^{\frac{d}{2}}$
2 $U \leftarrow \dfrac{A}{X^{\frac{d}{2}}}$
3 $T \leftarrow U * C * X^{\frac{d}{2}} + L$
4 $L_p \leftarrow T \bmod X^d$
5 $U_p \leftarrow \dfrac{A}{X^d}$
6 $W \leftarrow U_p * C + L_p$
7 $s \leftarrow W(2^k)$
8 **si** $s \geq p$ **alors**
9 $\quad s \leftarrow s - p$
10 **retourner** s

S'ils permettent une réduction modulaire très peu coûteuse (juste une addition), les nombres de Mersenne utilisables en cryptographie sont très rares. En effet, seulement quatre existent avec des tailles comprises entre 100 et 700 bits (où $l = 107, 127, 521$ et 607). La rareté de ces nombres premiers limite leur utilisation en cryptographie.

Les nombres dits pseudo Mersenne [35] sont des nombres qui s'écrivent sous la forme $p = 2^l - c$ avec $c < 2^{\frac{l}{2}}$. Tout comme avec les nombres de Mersenne, un nombre a quelconque se décompose en deux parties a_0 et a_1, tel que $a = a_1 \times 2^l + a_0$. La réduction modulaire de a par un nombre pseudo Mersenne est alors similaire à une réduction par un nombre de Mersenne. En effet, comme $2^l \equiv c \bmod p$, la réduction modulaire de a par p correspond à $a \equiv a_1 \times c + a_0 \bmod p$. La réduction modulaire pour de tels premiers se résume à une petite séquence d'additions/soustractions et de décalages.

Les nombres de Solinas

Les nombres de Solinas ou de Mersenne généralisés [105] s'écrivent sous la forme $p = P(2^k)$. La variable P est un polynôme unitaire de degré d dont les coefficients sont dans l'ensemble $\{-1, 0, 1\}$. De plus, le second plus grand degré de P est inférieur ou égal à $\dfrac{d}{2}$. Par conséquent, le polynôme P s'écrit $P = X^d - C(X)$ où C est un polynôme de degré d'au plus $\dfrac{d}{2}$. La réduction modulaire décrite dans l'algorithme 13, utilise la représentation polynomiale des nombres.

Algorithme 14 : Réduction modulaire avec $p = 2^{192} - 2^{64} - 1$

Entrées : a et p avec $p = 2^{192} - 2^{64} - 1$.
Résultat : s avec $s \equiv a \bmod p$

1 $A \leftarrow (a_2, a_1, a_0)$ /* Les a_i sont des mots de 64 bits de a */
2 $B \leftarrow (0, a_3, a_3)$
3 $C \leftarrow (a_4, a_4, 0)$
4 $D \leftarrow (a_5, a_5, a_5)$
5 $s \leftarrow A + B + C + D$
6 **retourner** s

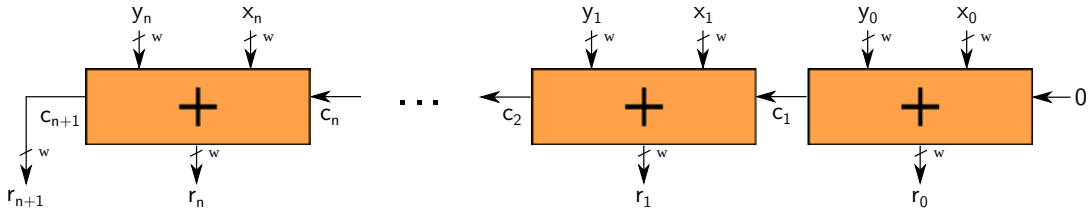


FIGURE 4.2 – Schéma d'une addition d'entiers

Certains de ces nombres sont utilisés dans des standards, comme le nombre premier nommé p_{192} présent dans le NIST. Le polynôme représentant ce nombre est $P = X^3 - X - 2$ et $p_{192} = P(2^{64}) = 2^{192} - 2^{64} - 1$.

L'algorithme 13 peut grandement se simplifier lorsque le nombre p est connu au préalable. Par exemple, l'algorithme 14, spécifique au premier p_{192} se réduit à 3 additions.

4.2 L'addition

Comme les UAL ne savent faire que des opérations entières sur des mots de w bits, les additions de deux grands nombres sont décomposées en de plus petites additions de mots, notées `wadd`. Ces `wadd` sont réalisables grâce aux UAL.

La figure 4.2 représente un schéma simplifié d'un additionneur séquentiel. Les x_i et y_i sont des mots machines de w bits des nombres X et Y. Les c_i correspondent aux retenues de l'addition avec une UAL, et les r_i sont les mots machines du résultat de $X + Y$. Avec cet additionneur, utilisant une représentation classique des nombres, l'addition ne peut être parallélisée à cause de la propagation de la retenue. Par exemple, pour calculer r_1 et c_2, la valeur de c_1 est nécessaire.

D'autres types d'additionneur, non décrits par la suite, existent. Les additionneurs à sélection de retenue, à retenue bondissante, ou encore à retenues anticipées en sont des exemples. L'objectif de ces additionneurs est de propager ou de calculer au plus vite les retenues, permettant ainsi un gain de temps.

	PPM	FA
s	$a \otimes b \otimes c$	$a \otimes b \otimes c$
r	$ab + a\bar{c} + b\bar{c}$	$ab + ac + bc$

TABLE 4.1 – Cellules d'additionneur

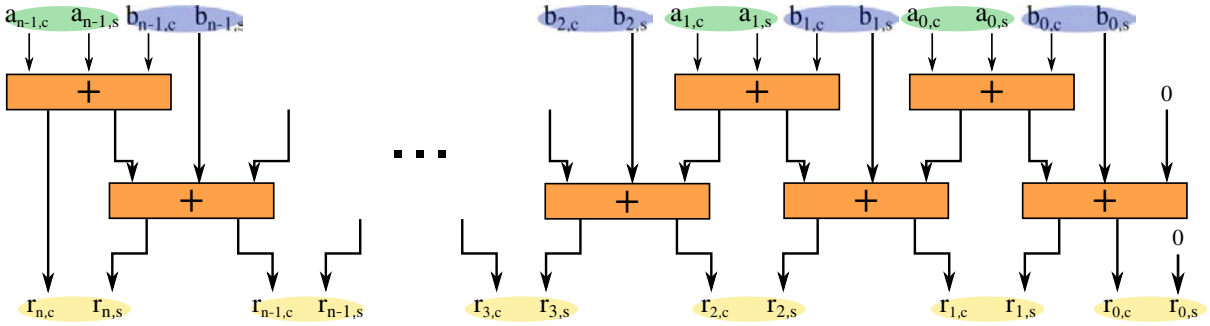


FIGURE 4.3 – Additionneur *carry-save*

L'utilisation de représentations redondantes des nombres permet de paralléliser les calculs sans avoir de propagation de retenue [45] (sur des largeurs plus grandes qu'un rang). C'est le cas, par exemple, des représentations dîtes *carry-save* (CS) ou *borrow-save* (BS).

Un chiffre, défini dans la représentation CS, s'écrit $b_i = b_{i,c} + b_{i,s}$ avec $b_i \in \{0, 1, 2\}$, $b_{i,c}$ et $b_{i,s} \in \{0, 1\}$. Ainsi, le nombre B se décompose comme $B = \sum_{i=0}^{n-1} (b_{i,c} + b_{i,s}) 2^i$.

En représentation BS, un chiffre est défini par $b_i = b_i^+ - b_i^-$ avec $b_i \in \{-1, 0, 1\}$, b_i^+ et $b_i^- \in \{0, 1\}$. Ainsi, un nombre B s'écrit $B = \sum_{i=0}^{n-1} (b_i^+ - b_i^-) 2^i$.

L'addition, avec ces représentations, se fait en deux étapes où chaque `wadd` d'une étape s'effectue en parallèle. La figure 4.3 schématise une addition avec la représentation CS. Chaque cellule correspond à une addition sur w bits qui prend trois opérandes et retourne deux sorties. Le résultat final de l'addition reste dans la même représentation.

Un additionneur BS est similaire à un additionneur CS, à la seule différence que ce n'est pas les mêmes cellules d'addition (les boîtes de la figure 4.3). Pour un additionneur BS et CS, les cellules sont respectivement des cellules *PPM* et *FA*[1] . Les calculs réalisés par ces cellules sont décrits dans le tableau 4.1. Dans ce tableau, a, b et c sont les opérandes des cellules et s et r sont les sorties.

1. À ne pas confondre avec les attaques en faute.

4.3 La multiplication

Tout comme l'addition de deux grands nombres, la multiplication de deux grands nombres n'est pas réalisable directement sur une UAL. Par conséquent, des algorithmes, permettant de décomposer et d'effectuer des multiplications sur des mots, sont nécessaires.

La méthode naïve est la technique enseignée dans les écoles pour multiplier a par b et correspond à l'équation 4.3, où $c_i = \sum_{k+l=i} a_k b_l$ et β est la base.

$$a \cdot b = \sum_{i=0}^{n-1} a_i \beta^i \cdot \sum_{i=0}^{n-1} b_i \beta^i = \sum_{i=0}^{2n-2} c_i \beta^i \tag{4.3}$$

Cette multiplication est simple à mettre en œuvre, mais elle est coûteuse (avec une complexité en $O(n^2)$). De plus, comme les c_i peuvent dépasser β, une propagation de retenue est possible. Pour toutes ces raisons, la méthode naïve est peu utilisée en pratique pour les très grands nombres. L'algorithme naïf de multiplication est très souvent utilisé pour de très petits nombres de mots en logiciel car il est très régulier et propice à des optimisations de boucles du compilateur.

D'autres méthodes de multiplication existent, la plupart reposent sur le principe « diviser pour régner ». C'est le cas, par exemple, des méthodes de Karatsuba et de Toom-Cook.

4.3.1 Karatsuba

La méthode de Karatsuba [95] applique le principe « diviser pour régner » et est décrite à l'algorithme 15. Les opérandes sont scindés en deux sous-mots afin de faire des multiplications élémentaires plus petites. Par exemple, la multiplication de deux nombres composés de deux mots machines s'effectue grâce à trois `wmul` et plusieurs `wadd`.

Une multiplication de deux nombres, a et b, composé de deux mots de 8 bits est schématisée à la figure 4.4. Les flèches bleues correspondent à des additions et les rouges à des multiplications de mots. Enfin, les flèches verts concernent les résultats intermédiaires nécessaire au résultat final de la multiplication. Comme les opérandes a et b sont composés de 16 bits, la taille du résultat sera de 32 bits.

Pour des nombres contenant plus de deux mots machines, les multiplications internes sont faites récursivement. En effet, les différentes parties obtenues, après le découpage des opérandes, seront encore trop grandes pour être réalisables directement par les UAL.

Cette méthode est plus efficace que la méthode naïve, avec une complexité en $O(n^{1.585})$ et est couramment utilisée lorsque les opérandes ont entre 8 et 100 mots machines. Bien sur, d'autres variantes de l'algorithme 15, gardant le même principe, existent.

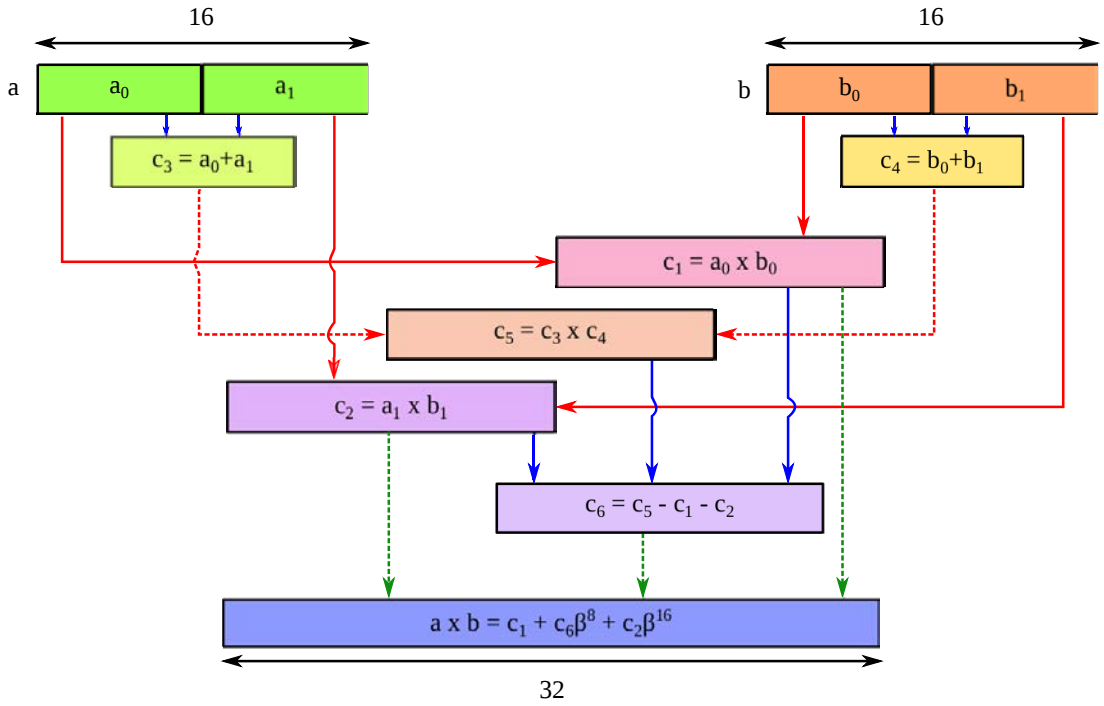


FIGURE 4.4 – Multiplication de deux nombres avec la méthode de Karatsuba

Exemple 5. *Multiplication de 38 par 62 en base 10 avec la méthode de Karatsuba.*

$$\begin{aligned}
38 \times 62 &= (3 \cdot 10 + 8) \cdot (6 \cdot 10 - 2) \\
&= 3 \cdot 6 \cdot 10^2 + ((3+8) \cdot (6+2) - 3 \cdot 6 - 8 \cdot 2) \cdot 10 + 8 \cdot 2 \\
&= 18 \cdot 10^2 + 54 \cdot 10 + 16 \\
&= 1 \cdot 10 \cdot 10^2 + 8 \cdot 10^2 + 5 \cdot 10 \cdot 10 + 4 \cdot 10 + 1 \cdot 10 + 6 \\
&= 10^3 + 8 \cdot 10^2 + 5 \cdot 10^2 + 4 \cdot 10 + 1 \cdot 10 + 6 \\
&= 10^3 + 13 \cdot 10^2 + 5 \cdot 10 + 6 \\
&= 10^3 + 1 \cdot 10 \cdot 10^2 + 3 \cdot 10^2 + 5 \cdot 10 + 6 \\
&= 2 \cdot 10^3 + 3 \cdot 10^2 + 5 \cdot 10 + 6 \\
&= 2356
\end{aligned} \tag{4.4}$$

4.3.2 Toom-Cook

Tout comme la méthode de Karatsuba, décrite précédemment, la méthode Toom-Cook [33, 108] applique le principe de « diviser pour régner ». Elle est basée sur une technique d'interpolation.

Le principe de base, d'une multiplication par interpolation, consiste à regarder les nombres A et B comme des polynômes $P(X)$ de degré n. La variable X est la base de représentation des nombres β, $(X = \beta)$. Ces polynômes sont évalués en $2n + 1$ valeurs afin de déduire la

Algorithme 15 : Multiplication de Karatsuba

Entrées : a et b de taille n et $k = \left\lceil \frac{n}{2} \right\rceil$
Résultat : $s = a \times b$

si $k = 1$ *ou* $k = 0$ **alors**
  retourner $a \times b$
$(a_0, a_1) \leftarrow$ (quotient de $\frac{a}{\beta^k}$, reste de $\frac{a}{\beta^k}$)
$(b_0, b_1) \leftarrow$ (quotient de $\frac{b}{\beta^k}$, reste de $\frac{b}{\beta^k}$)
$c_1 \leftarrow \texttt{Karatsuba}(a_0, b_0)$
$c_2 \leftarrow \texttt{Karatsuba}(a_1, b_1)$
$c_3 \leftarrow a_0 + a_1$
$c_4 \leftarrow b_0 + b_1$
$c_5 \leftarrow \texttt{Karatsuba}(c_3, c_4)$
$c_6 \leftarrow c_5 - c_1 - c_2$
$s \leftarrow c_1 + c_6\beta^k + c_2\beta^{2k}$
retourner s

valeur du produit AB en ces $2n+1$ valeurs. La valeur de $A \times B$ est ensuite retrouvée grâce à ces évaluations et à une interpolation. L'exemple suivant illustre une multiplication par interpolation de deux nombres.

Exemple 6. *Multiplication par interpolation de A par B :*

- ⋆ *Les nombres A et B sont vu comme des polynômes : $A = A_0 + A_1X$ et $B = B_0 + B_1X$ où X est la base de représentation des nombres A et B.*
- ⋆ *Interpolation aux points* $0, 1, \infty$.
  - $A(0) = A_0$, $A(1) = A_0 + A_1$, $A(\infty) = A_1$
  - $B(0) = B_0$, $B(1) = B_0 + B_1$, $B(\infty) = B_1$
- ⋆ *Déduction de $A \cdot B$ en ces points.*
  - $(A \cdot B)(0) = A_0 \cdot B_0$, $(A \cdot B)(1) = (A_0 + A_1)(B_0 + B_1)$, $(A \cdot B)(\infty) = A_1 \cdot B_1$
- ⋆ *Interpolation de* $r(X) = (A_0 + A_1X)(B_0 + B_1X) = r_0 + r_1X + r_2X^2$
  - $r(0) = r_0$, $r(1) = r_0 + r_1 + r_2$, $r(\infty) = r_2$
  - *Résolution d'un système linéaire*
    - $r_0 = A_0B_0$
    - $r_0 + r_1 + r_2 = (A_0 + A_1)(B_0 + B_1) \Rightarrow r_1 = (A_0 + A_1)(B_0 + B_1) - r_0 - r_2$
    - $r_2 = A_1 \cdot B_1$

La méthode Toom-Cook, fondée sur le principe d'interpolation, consiste à représenter les nombres comme des polynômes A et B de degré 2. Leur évaluation en cinq points

$(\infty, 2, -1, 1, 0)$ permet de réaliser la multiplication de A par B. Des variantes gardant le même principe existent mais avec des degrés de polynômes différents de 2.

Les méthodes de Karatsuba et de Toom-Cook ne garantissent pas la réduction modulaire. Par conséquent, lorsque deux nombres a et b doivent être multipliés modulo un nombre p, il est nécessaire d'appliquer un algorithme de réduction modulaire en plus de l'algorithme de multiplication.

Remarques :

- ⋆ L'algorithme de Karatsuba est un cas particulier d'une méthode d'interpolation.
- ⋆ Une autre cas particulier de méthode d'interpolation est la méthode de la transformée de Fourier rapide (FFT). Elle s'obtient en choisissant les racines primitives $2n^{\text{ième}}$ de l'unité comme valeur d'évaluation. La FFT est particulièrement rapide (complexité logarithmique) pour de très très grandes tailles. Cependant, elle n'est pas utilisable pour ECC car les opérandes ne sont pas encore assez grands.

4.3.3 Multiplication de Taylor

Les méthodes de multiplication vues précédemment ne retournent pas un résultat réduit modulo p. Cependant, des algorithmes effectuant la multiplication modulo p existent, comme la multiplication de Taylor [107, 97]. Cette multiplication utilise la décomposition en carré de l'équation 4.5 pour effectuer le calcul $a \times b$ modulo p où p est impair (4 est alors inversible).

$$a \times b = \frac{(a+b)^2}{4} - \frac{(a-b)^2}{4} \tag{4.5}$$

Algorithme 16 : Multiplication de Taylor

Entrées : a, b, p avec $0 \leq a, b, < p$ et $\texttt{mem}$ avec $\texttt{mem}(x) = \dfrac{x^2}{4}$
Résultat : $s = a \times b \bmod p$

1 $u \leftarrow \mid a + b - p \mid$
2 $v \leftarrow \mid a - b \mid$
3 $s \leftarrow \texttt{mem}(u) - \texttt{mem}(v)$
4 **si** $s < 0$ **alors**
5 $\quad s \leftarrow s + p$
6 **retourner** s

L'algorithme 16 récapitule cette méthode. Il a pour particularité de ne pas faire de multiplication pour calculer $a \times b$. En effet, des unités de mémoires (ou encore tables de mémoires),

nommées `mem` et précalculées au préalable, sont utilisées pour effectuer les carrées. Ainsi, pour une valeur u, la table `mem` retournera $\texttt{mem}(u) = \frac{u^2}{4}$.

Malgré une multiplication rapide, cette méthode demande un espace mémoire important puisque mem reçoit p entiers. Ce coût mémoire la rend difficilement exploitable sur des circuits embarqués pour les calculs cryptographiques.

4.4 Conclusion

Les opérations cryptographiques manipulent de très grands nombres. Cependant, les UAL dans les systèmes embarqués ne peuvent effectuer directement des opérations que sur des mots de w. Cette contrainte rend nécessaire l'utilisation d'algorithmes de calculs permettant de décomposer les calculs. Différents algorithmes de calculs existent afin de réaliser par exemple des additions, multiplication ou encore des réductions modulaires. Certains sont plus adaptés que d'autres dans le contexte de ECC. Dans la suite de ce manuscrit, les algorithmes de réduction de Montgomery et de multiplication de Karatsuba sont sélectionnés. En effet, pour des nombres avec quelques centaines de bits, ils sont efficaces.

DEUXIÈME PARTIE

Contributions

CHAPITRE 5

PROTECTIONS COMBINÉES CONTRE SCA ET FA

La mise en œuvre de protections contre un type d'attaque peuvent affaiblir le cryptosystème contre un autre type d'attaque. La plupart des publications de la littérature considèrent les SCA et les FA indépendamment, alors qu'il est primordial de les traiter simultanément dans bon nombre de cas d'usage. Par exemple, un attaquant capable d'effectuer des observations du rayonnement électromagnétique en champ proche, doit pouvoir aussi, très probablement, injecter des fautes du fait de la proximité avec le circuit.

Autre exemple, les schémas d'uniformisation utilisant des opérations factices pour la protection contre certaines SCA peuvent être vulnérables par rapport à la SEA. Dans le cas de protections contre les FA, l'ajout d'opérations de contrôle à base de redondances ne doit pas réduire la robustesse par rapport aux SCA (p. ex. en rompant le comportement uniforme ou en ajoutant des dépendances entre le comportement du circuit et des données sensibles).

Dans ce chapitre, deux contre-mesures combinées sont proposées afin de protéger la SM pour ECC simultanément contre les principales attaques de type SPA et celles de type FA. La première contre-mesure proposée est une extension de la *vérification de point* de la courbe. En plus d'assurer la protection contre les FA, cette nouvelle contre-mesure est conçue pour assurer l'uniformisation des calculs afin de se protéger contre les SPA. La seconde contre-mesure proposée, appelée *compteur d'itérations*, protège les bits du scalaire contre les FA avec un comportement uniforme pour la protection contre les SPA. En plus de ces contre-mesures, les protections standards contre les attaques du type DPA (cf. section 2.3) peuvent être utilisées en plus. Ces contre-mesures seront décrites pour les courbes de Weierstrass (en coordonnées jacobiennes et en coordonnées projectives) et pour les courbes de Montgomery (en coordonnées XZ).

Le coût, de ces protections, est d'abord évalué en fonction du nombre d'opérations dans $\mathbb{F}_p$: multiplication `M`, carré `S` et addition / soustraction `A`. L'hypothèse que le comportement d'une addition et d'une soustraction est supposé être le même sur $\mathbb{F}_p$ est faite pour ce document. Puis, des comparaisons détaillées sont rapportées pour l'implantation sur un microcontrôleur.

5.1 Vérification de point (PV)

La vérification de point (PV) (pour *point verification*), introduite dans [12], consiste à injecter les coordonnées du point dans l'équation de la courbe. Cela permet de vérifier que le point appartient effectivement à la courbe sélectionnée (qui est considérée comme sûre).

Le calcul de la PV peut s'intégrer dans la SM à différents moments, ce qui permet divers compromis entre sécurité et performance. Pour une protection à faible coût, la PV peut s'effectuer au début et à la fin de la SM. Par conséquent, une FA sur le point intermédiaire sera détectée tardivement, ce qui laisse le système vulnérable à certaines attaques dont notamment les FSA (voir sec. 3.3.1). Pour une détection plus précoce, mais avec un surcoût plus élevé, la PV peut s'effectuer toutes les d itérations ou de manière aléatoire pendant la SM. La protection la plus forte, contre les attaques FA, est obtenue lorsque $d = 1$ où la PV est effectuée à chaque itération de la SM. Dans ce chapitre, le nombre total de PV effectuées durant une SM sera noté ℓ où $0 \leq \ell \leq m$ où m est la taille du scalaire.

L'application de la PV ne doit pas affaiblir la mise en œuvre contre les SCA. Par exemple, l'utilisation de la PV après les opérations `DBL` est plus sûre. En effet, les `DBL` ne dépendent pas de la valeur des bits de clé. Par conséquent, l'ajout de la PV, à ce moment, ne fragilisera pas la SM vis à vis des observations.

Le premier objectif est d'utiliser la PV pour compenser les différences entre `ADD` et `DBL`, comme illustré à la figure 5.1. Cette figure représente une séquence d'opérations de courbe possibles lors d'une SM. Les boîtes notées `DBL` et `ADD` correspondent aux opérations de courbe et la boîte notée `V` correspond à la PV. Ainsi, si `V` est utilisée en compensation, une `ADD` aura le même coût total que `DBL+V` (donc la même durée mais pas nécessairement le même ordonnancement des opérations internes).

Le second objectif est d'assurer le même comportement (c.-à-d. la même séquence d'opérations dans $\mathbb{F}_p$), à la fois pour `ADD` et `DBL+V`. Pour cette première contre-mesure, la PV originale est explorée et modifiée afin d'assurer un comportement uniforme contre les attaques du type SPA.

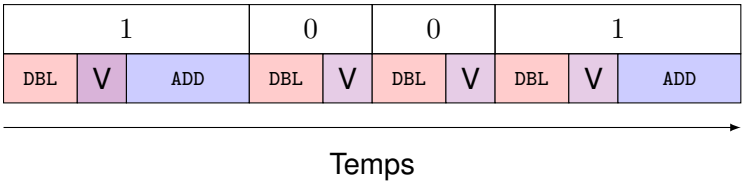


FIGURE 5.1 – Vérification de point dans DA

5.1.1 Uniformisation grâce à la PV sur les courbes de Weierstrass

Dans cette partie, l'uniformisation de la SM est décrite pour les courbes de Weierstrass, en coordonnées projectives et jacobiennes. Toutefois, le même principe est applicable aux autres types de coordonnées.

La PV est incluse dans DBL, et notée DBL+V, pour assurer un comportement uniforme de la SM. L'inclusion de la PV dans ADD accentuerait la « dissymétrie » entre les opérations réalisées lorsque le bit de clé est égal à 0 et quand il vaut 1. En effet, lorsque $k_i = 0$, seulement un DBL serait effectué, contre une ADD et une V lorsque $k = 1$, rendant la SM plus vulnérable aux SCA. De plus, comme le calcul de ADD est plus complexe et coûteux que DBL, l'ajout de la vérification peut permettre de compenser la différence de coût et de comportement entre ADD et DBL.

Coordonnées projectives

Les courbes de Weierstrass sur $\mathbb{F}_p$, en coordonnées projectives, sont rappelées à l'équation 5.1 ci-dessous, avec a et b les paramètres de la courbe :

$$E_{WP} : y^2z = x^3 + axz^2 + bz^3. \tag{5.1}$$

La multiplication par b, notée $\mathtt{M}_b$, peut être implantée avec une multiplication générique ou des additions dépendant de la valeur de b (p. ex. décomposition creuse). Comme en général $a = -3$, la multiplication par a se fait grâce à des additions.

Pour réaliser la PV, les coordonnées du point à vérifier sont injectées dans l'équation de courbe 5.1. Cette vérification coûte $4\mathtt{M} + 3\mathtt{S} + 1\mathtt{M}_b + 5\mathtt{A}$. Le tableau 5.1 récapitule les différents coûts pour les opérations ADD, DBL, DBL+V, et V. La colonne « PV basique » correspond à la PV originale, où les coordonnées sont simplement injectées dans l'équation de la courbe. La seconde colonne correspond à la PV transformée et modifiée, permettant d'uniformiser la SM.

La première colonne du tableau 5.1 permet d'observer que le coût de la PV basique ne permet pas de compenser la différence de coût entre ADD et DBL. En effet, le coût de ADD reste plus important que celui de DBL+V. Par conséquent, la vérification doit être modifiée.

	PV basique	PV modifiée
ADD	$11\mathtt{M} + 6\mathtt{S} + 18\mathtt{A}$	$11\mathtt{M} + 6\mathtt{S} + 18\mathtt{A} + 1\mathtt{M}_b$
DBL	$5\mathtt{M} + 6\mathtt{S} + 14\mathtt{A}$	/
DBL+V	$9\mathtt{M} + 9\mathtt{S} + 19\mathtt{A} + 1\mathtt{M}_b$	$11\mathtt{M} + 6\mathtt{S} + 18\mathtt{A} + 1\mathtt{M}_b$
V	$4\mathtt{M} + 3\mathtt{S} + 1\mathtt{M}_b + 5\mathtt{A}$	$6\mathtt{M} + 4\mathtt{A} + 1\mathtt{M}_b$

TABLE 5.1 – Coût en nombre d'opérations pour les courbes de Weierstrass en coordonnées projectives

Le coût de la vérification n'étant pas assez important, l'équation de départ doit être transformée. La nouvelle équation de vérification, notée V et décrite à l'équation 5.2, correspond à l'équation de E_{WP} multipliée par y.

$$\mathtt{V} : y^3z = x^3y + axyz^2 + byz^3. \tag{5.2}$$

Le coût de cette vérification est de $6\mathtt{M} + 3\mathtt{S} + 1\mathtt{M}_b + 5\mathtt{A}$, ce qui est trop important pour l'utiliser directement dans DBL. En effet, le coût de DBL+V serait $11\mathtt{M} + 9\mathtt{S} + 19\mathtt{A} + 1\mathtt{M}_b$, avec cette équation. Cependant, après factorisation de certaines opérations, entre DBL et V et après l'ajout de l'opération $\mathtt{M}_b$ à ADD, les coûts de ADD et DBL+V sont identiques.

Néanmoins, avoir le même coût pour les opérations sur la courbe n'assure pas qu'elles aient un comportement uniforme. Par conséquent, les calculs (au niveau corps) de ADD et DBL+V sont ré-ordonnancés afin d'avoir le même comportement pour ces deux opérations de courbe. La figure 5.2 récapitule les formules de ADD et de DBL+V après les étapes de factorisation et de ré-ordonnancement des calculs. Les opérations avec les encadrés arrondis bleus correspondent aux opérations de la vérification du point intermédiaire $Q_1 = (X_1, Y_1, Z_1)$, insérées au calcul du DBL. Une comparaison détermine si le point Q_1 appartient à la courbe.

L'ajout d'opérations fictives rendrait la SM vulnérable aux SEA. Par conséquent, chaque opération ajoutée à ADD et à DBL est nécessaire au renvoi du résultat de la SM. C'est notamment le cas pour l'opération de $\mathtt{M}_b$ ajoutée dans ADD. Cette opération, encadrée en rouge à la figure 5.2 ($t_f = b \times Z_2$), n'est pas fictive. En effet, comme Z_2 (correspondant à la coordonnée Z du point de base) n'est pas modifiée lors de la SM, la valeur de t_f est pré-calculée au début de la SM, notée C_0. Ainsi, à la fin de ADD, C_0 est comparé avec t_f. Si les deux valeurs sont égales, alors le résultat de ADD est retourné. Sinon, une faute est détectée.

Le surcoût de la vérification dans DBL est de $6\mathtt{M} + 4\mathtt{A} + 1\mathtt{M}_b$. Dans ADD, le surcoût est de $1\mathtt{M}_b$. Dans une SM, un DBL est effectué pour chaque bit de clé et une ADD lorsque le bit de clé vaut 1 (environ 50% des bits de clé). Par conséquent, le surcoût de cette SM uniforme, protégée contre les attaques en faute, est de $6\ell\mathtt{M} + 4\ell\mathtt{A} + \frac{3}{2}\ell\mathtt{M}_b$ où ℓ est de nombre de vérifications effectuées durant l'exécution.

Coordonnées jacobiennes

Les courbes de Weierstrass, en coordonnées jacobiennes, sont définies par l'équation suivante, avec a et b les paramètres de la courbe :

$$E_{WJ} : Y^2 = X^3 + aXZ^4 + bZ^6 \tag{5.3}$$

Avec cette équation de courbe, la vérification coûte $3\mathtt{M} + 4\mathtt{S} + 5\mathtt{A} + 1\mathtt{M}_b$. Comme illustré dans la première colonne du tableau 5.2, cette vérification ne permet pas de compenser la différence

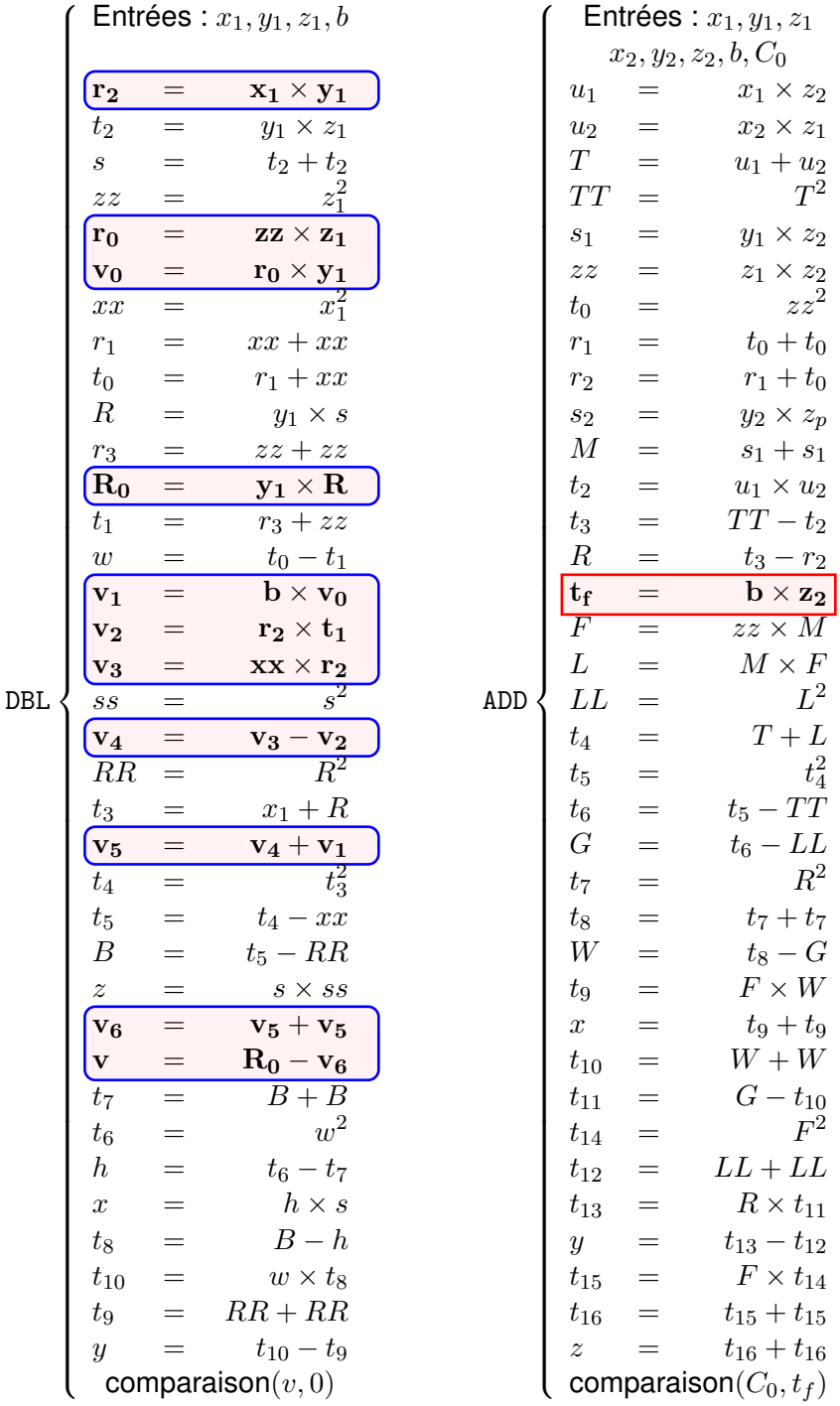


FIGURE 5.2 – Séquence d'opérations des ADD et DBL, pour les courbes de Weierstrass en coordonnées projectives (après l'ajout de V, factorisation et ré-ordonnancement)

de coût entre ADD et DBL. Comme pour les coordonnées projectives, l'équation de la courbe est transformée pour obtenir une nouvelle vérification. Grâce à une multiplication de l'équation 5.3 par la coordonnée Z, l'équation 5.4 est obtenue.

$$\mathtt{V} : Y^2Z = X^3Z + aXZ^5 + bZ^7 \tag{5.4}$$

	PV basique	PV
ADD	$11\mathtt{M} + 5\mathtt{S} + 13\mathtt{A}$	$11\mathtt{M} + 5\mathtt{S} + 17\mathtt{A} + 1\mathtt{M}_b$
DBL	$3\mathtt{M} + 6\mathtt{S} + 15\mathtt{A}$	/
DBL+V	$6\mathtt{M} + 10\mathtt{S} + 20\mathtt{A} + 1\mathtt{M}_b$	$11\mathtt{M} + 5\mathtt{S} + 17\mathtt{A} + 1\mathtt{M}_b$
V	$3\mathtt{M} + 4\mathtt{S} + 5\mathtt{A} + 1\mathtt{M}_b$	$8\mathtt{M} - 1\mathtt{S} + 5\mathtt{A} + 1\mathtt{M}_b$

TABLE 5.2 – Coût en nombre d'opérations pour les courbes de Weierstrass en coordonnées jacobiennes

Avec l'équation 5.4, V coûte $7\mathtt{M}+4\mathtt{S}+5\mathtt{A}+1\mathtt{M}_b$ (en faisant les calculs naïvement). Le nombre de multiplications est presque suffisant (il n'en manque qu'une seule !), mais le nombre de carrés et d'additions reste encore trop important. La formule de DBL est modifiée comme décrit à l'équation 5.5. Cette transformation permet d'écrire différemment s et t_3, qui sont respectivement égaux à $s = 4x_1 \times y_1^2$ et $t_3 = 3x_1^2$. Grâce à cette nouvelle formule, un carré et une addition sont supprimés au profit d'une multiplication.

Modification réalisée dans DBL

Calcul initial de DBL				Après modification de DBL		
xx	$=$	x_1^2				
yy	$=$	y_1^2		yy	$=$	y_1^2
t_0	$=$	$x_1 \cdot yy$		t_0	$=$	$x_1 + x_1$
t_1	$=$	$t_0 + t_0$	$\Rightarrow$	t_1	$=$	$t_0 + x_1$
s	$=$	$t_1 + t_1$		t_2	$=$	$t_0 + t_0$
t_2	$=$	$xx + xx$		s	$=$	$t_2 \cdot yy$
t_3	$=$	$t_2 + xx$		t_3	$=$	$t_1 \cdot x_1$

(5.5)

Tout comme dans le cas des courbes de Weierstrass en coordonnées projectives, la multiplication $\mathtt{M}_b$ par le paramètre b est ajoutée à ADD. Cependant, afin d'avoir le même coût et des séquences d'opérations uniformes pour ADD et DBL+V, $4\mathtt{A}$ sont également rajoutées à ADD. Une fois le même coût obtenu pour ADD et DBL+V, les opérations sont ordonnancées dans le but d'avoir les mêmes séquences d'opérations.

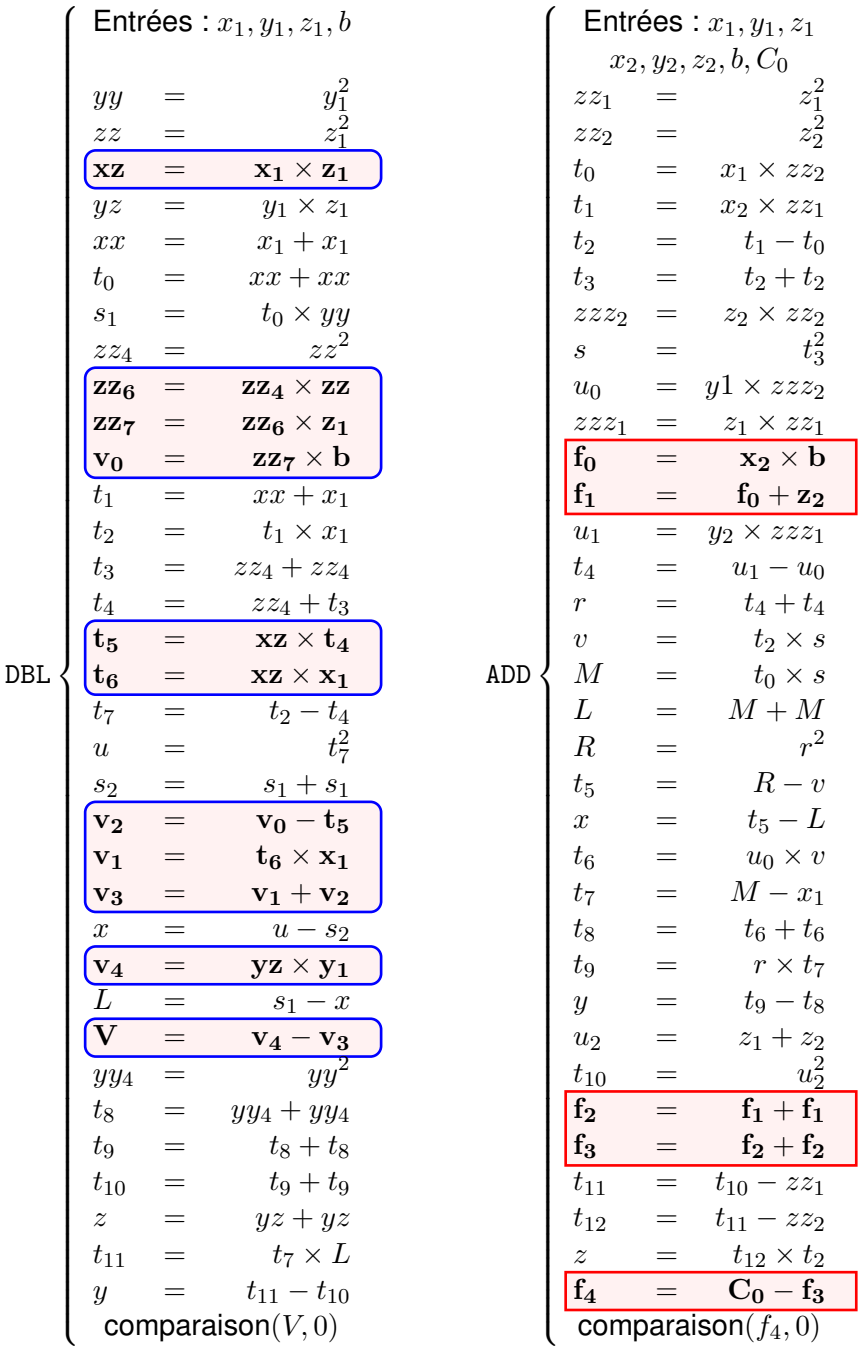


FIGURE 5.3 – Séquence d'opérations des ADD et DBL+V, pour les courbes de Weierstrass en coordonnées jacobiennes (après l'ajout de V, factorisation et ré-ordonnancement)

La figure 5.3 récapitule les nouvelles formules de ADD et DBL+V. Les opérations entourées d'un encadré arrondi bleu correspondent aux opérations nécessaires pour la vérification du point intermédiaire $Q_1 = (X_1, Y_1, Z_1)$ durant le DBL. Les opérations encadrées en rouge correspondent aux opérations ajoutées à ADD pour avoir le même coût et la même séquence d'opérations entre ADD et DBL+V. Comme précédemment, ces opérations ne sont pas fictives afin d'éviter des attaques du type SEA. Elles sont nécessaires pour récupérer le résultat de ADD (comme pour le cas des courbes de Weierstrass en coordonnées projectives). La valeur de C_0 est calculée au début de la SM et $C_0 = 4(b \times x_2 + z_2)$. La comparaison durant ADD est possible puisque les valeurs de x_2 et z_2 restent inchangées (car ce sont des coordonnées du point de base).

Avec ces formules, le coût de DBL+V est de $11\mathtt{M} + 5\mathtt{S} + 17\mathtt{A} + 1\mathtt{M}_b$, comme récapitulé dans la seconde colonne du tableau 5.2. Le surcoût de la SM uniforme est de $7\ell\mathtt{M} + 7\ell\mathtt{A} + \frac{3}{2}\ell\mathtt{M}_b$ (avec ℓ le nombre de vérifications effectuées). Cette SM est résistante aux attaques en faute avec un comportement uniforme sans utiliser d'opérations fictives.

	Projective	Jacobienne
PV	$\simeq 6\ell\mathtt{M} + 4\ell\mathtt{A} + \frac{3}{2}\ell\mathtt{M}_b$	$\simeq 7\ell\mathtt{M} + 7\ell\mathtt{A} + \frac{3}{2}\ell\mathtt{M}_b$
DAA	$\simeq 6.5m\mathtt{M} + 3m\mathtt{S} + 9m\mathtt{A}$	$\simeq 6.5m\mathtt{M} + 2.5m\mathtt{S} + 7.5m\mathtt{A}$

TABLE 5.3 – Surcoût de la SM (avec ℓ le nombre de vérifications effectuées, m la taille du scalaire et $0 \leq \ell \leq m$)

L'algorithme `doublement et addition toujours` (DAA) est une protection basique contre les SCA (voir sec. 2.3). Cet algorithme ajoute des opérations factices afin d'avoir la même séquence d'opérations quelle que soit la valeur du bit de scalaire traité. L'utilisation d'opérations fictives rend DAA vulnérable aux attaques du type SEA.

Les surcoûts pour la SM uniformisée grâce à PV et pour DAA sont récapitulés au tableau 5.3, pour les courbes de Weiertrass en coordonnées projectives et jacobiennes. La SM uniformisée avec PV est moins coûteuse que DAA. De plus, elle permet de protéger la SM contre les FA sur les points et les paramètres de la courbe.

5.1.2 Vérification de point uniforme pour les courbes de Montgomery

Les courbes de Montgomery, rappelées à l'équation 5.6, en représentation XZ ont des opérations de courbes moins coûteuses que les courbes de Weierstrass. De plus, grâce à l'algorithme de `l'échelle de Montgomery` (MLD), la SM est régulière et naturellement résistante aux attaques du type *safe-error*. Par conséquent, avoir une méthode permettant de protéger la SM des FA (autres que *safe-error*) est intéressant.

Différence entre les points intermédiaires T_1 et T_2

$$\text{Si } k_i = 0 : \begin{cases} T_1 \leftarrow 2T_1 \\ T_2 \leftarrow T_1 + T_2 \end{cases} \Rightarrow \begin{cases} T_1 = 2\alpha P \\ T_2 = (2\alpha+1)P \end{cases} \Rightarrow T_2 - T_1 = P$$

$$\text{Si } k_i = 1 : \begin{cases} T_1 \leftarrow T_1 + T_2 \\ T_2 \leftarrow 2T_2 \end{cases} \Rightarrow \begin{cases} T_1 = (2\alpha+1)P \\ T_2 = (2\alpha+2)P \end{cases} \Rightarrow T_2 - T_1 = P$$

FIGURE 5.4 – Différence des points intermédiaires lors d'un MLD

$$E_M : by^2 = x^3 + ax^2 + x. \tag{5.6}$$

La vérification de point (PV) directe, consistant à vérifier qu'un point appartient à la courbe, est trop chère pour les courbes de Montgomery utilisant les coordonnées XZ. Les coordonnées XZ correspondent à des coordonnées projectives dont la coordonnée y est ignorée. Par conséquent, réaliser une PV équivaut à vérifier que $x^3 + ax^2 + x$ est un carré grâce au symbole de Legendre. La valeur du symbole de Legendre est déterminée par l'équation 5.7, où $C = x^3 + ax^2 + x$ et p la caractéristique du corps $\mathbb{F}_p$. Comme la vérification de l'appartenance du point à la courbe avec le symbole de Legendre est coûteuse, une autre méthode doit être appliquée.

$$C^{\frac{p-1}{2}} = \begin{cases} 0 & \text{si } C \equiv 0 \bmod p \\ 1 & \text{si } C \text{ est un carré modulo } p \\ -1 & \text{sinon} \end{cases} \tag{5.7}$$

Algorithme : SM - `échelle de Montgomery`
Entrées : P et $k = (k_{m-1}, \ldots, k_0)_2$
Résultat : $[k] \cdot P$

```
T1 ← O, T2 ← P
pour i = m − 1 à 0 faire
    si k_i = 1 alors
        T1 ← T1 + T2        /* ADD */
        T2 ← 2 · T2         /* DBL */
    sinon
        T2 ← T1 + T2        /* ADD */
        T1 ← 2 · T1         /* DBL */
retourner Q1
```

Formule DBL : $2T_1$

$$\begin{aligned} A &= (x_1 + z_1)^2 \\ B &= (x_1 - z_1)^2 \\ u_0 &= A - B \\ x_3 &= AB \\ z_3 &= u_0(B + a_{24}u_0) \end{aligned}$$

Formule ADD : $T_1 + T_2$

$$\begin{aligned} u_0 &= (x_2 - z_2)(x_1 + z_1) \\ u_1 &= (x_2 + z_2)(x_1 - z_1) \\ x_3 &= (u_0 + u_1)^2 \\ z_3 &= x_P(u_0 - u_1)^2 \end{aligned}$$

FIGURE 5.5 – Formules ADD et DBL pour les courbes de Montgomery avec $T_1 = (x_1, z_1)$, $T_2 = (x_2, z_2)$ et x_P la coordonnée x de $T_2 - T_1$

Première tentative de détection de faute : $P = T_1 - T_2$

Lors de l'exécution de l'algorithme 2, la différence des points intermédiaires $T_2 - T_1$ est égale à P à chaque itération, comme rappelé à la figure 5.4. Cette constante interne peut s'exploiter afin de réaliser une détection de faute. En effet, si le point T_1 est fauté (resp. T_2), il devient $\widetilde{T_1}$ (resp. $\widetilde{T_2}$) et $T_2 - \widetilde{T_1} \neq P$ (resp. $\widetilde{T_2} - T_1 \neq P$). Ainsi, une faute sur les points intermédiaires est détectée si la différence des points T_1 et T_2 n'est pas égale à P.

Dans la formule de ADD, la coordonnée x de $T_2 - T_1$ est nécessaire afin de calculer $T_1 + T_2$, comme rappelé à la figure 5.5. La valeur de $T_2 - T_1$ est calculée avec les formules de ADD. Cette différence est calculable puisque la somme $T_1 + T_2$ est également calculée à chaque itération. Ainsi, deux opérations de courbe ont lieu lors d'une itération de MLD. La première est une ADD effectuant $T_1 + T_2 = (x_3, z_3)$, réalisée grâce à la coordonnée x du point de base P (notée x_P). La seconde est une ADD calculant $T_2 - T_1 = (x_4, z_4)$ effectuée avec x_3 (la coordonnée x du résultat précédent). L'équation 5.8 détaille les formules afin de calculer (x_3, z_3) et (x_4, z_4). Le résultat de $T_2 - T_1$ est ensuite comparé à P. Si $T_2 - T_1 \neq P$ alors une faute est détectée.

$$\begin{aligned} u_0 &= (x_2 - z_2)(x_1 + z_1) & u_1 &= (x_2 + z_2)(x_1 - z_1) \\ x_3 &= (u_0 + u_1)^2 & z_3 &= x_P(u_0 - u_1)^2 \\ x_4 &= x_3 & z_4 &= x_3(u_0 - u_1)^2 \end{aligned} \tag{5.8}$$

Malheureusement, comme les points sont en coordonnées XZ, la comparaison entre $T_2 - T_1$ et P n'est possible qu'après avoir normalisé la valeur de x_3 par z_3 et celle de x_4 par z_4. Le calcul de z_4 ne s'effectue donc pas exactement comme décrit à l'équation 5.8. L'équation 5.10 décrit le calcul de z_4 avec la normalisation de x_3 par z_3. Ainsi, la valeur de z_4 vaut $z_4 = \dfrac{x_3}{x_P}$.

La comparaison des points P et $T_2 - T_1$ est détaillée à l'équation 5.9. Lorsque z_4 est remplacée par sa valeur, la comparaison des deux points est équivalente à résoudre l'équation $x_3(1 - z_P) = 0$. Comme en général $z_p = 1$, cette égalité est toujours vraie. Si un point intermé-

diaire est fauté alors le test restera vrai. Par conséquent, les fautes ne sont pas détectées et une correction de la méthode de vérification est nécessaire.

Comparaison des points P et $T_1 - T_2$

$$\begin{aligned} \frac{x_P}{z_P} &= \frac{x_4}{z_4} \\ x_P z_4 - x_3 z_P &= 0 \\ x_P \frac{x_3}{x_P} - x_3 z_P &= 0 \\ x_3 - x_3 z_P &= 0 \\ x_3(1 - z_P) &= 0 \end{aligned} \tag{5.9}$$

$$\begin{aligned} z_4 &= \frac{x_3}{z_3}(u_0 - u_1)^2 \\ z_4 &= \frac{x_3(u_0 - u_1)^2}{x_P(u_0 - u_1)^2} \\ z_4 &= \frac{x_3}{x_P} \end{aligned} \tag{5.10}$$

Deuxième tentative de détection : $2P = 2(T_1 - T_2)$

La comparaison entre P et $T_1 - T_2$ ne permet pas de réaliser une détection de faute. Par conséquent, une autre méthode est nécessaire. Si la comparaison entre P et $T_1 - T_2$ est toujours vraie, peut-être que la comparaison entre $2P$ et $2(T_1 - T_2)$ ne l'est pas ?

Les valeurs de $2P$ et de $V = 2(T_1 - T_2) = (x_5, z_5)$ sont calculées avec les formules de `DBL` de la figure 5.5. Les résultats de ces `DBL` sont décrits à la figure 5.6. Une fois les points $2P$ et V calculés, une comparaison est réalisée. Cette comparaison est détaillée à la figure 5.7 où les calculs de $x_{2P}z_5$ et de $x_5 z_{2P}$ sont effectués. En remplaçant les valeurs dans l'équation 5.11, la comparaison s'avère toujours vraie même en présence de faute sur l'un ou l'autre des points intermédiaires (T_1 ou T_2).

Calcul de x_{2P} et z_{2P}

$$\begin{aligned} x_{2P} &= (x_P + 1)^2(x_P - 1)^2 \\ &= x_P^4 - 2x_P^2 + 1 \\ u_P &= 4x_P \\ z_{2P} &= u_P((x_P - 1)^2 + a_{24}u_P) \\ &= 4x_P(x_P^2 + ax_P + 1) \\ &= 4x_P^3 + 4ax_P^2 + 4x_P \end{aligned}$$

Calcul de x_5 et z_5

$$\begin{aligned} x_5 &= (x_4 + z_4)^2(x_4 - z_4)^2 \\ &= (x_3 + \frac{x_3}{x_P})^3((x_3 - \frac{x_3}{x_P})^2 \\ &= x_3^4 - 2\frac{x_3^4}{x_P^2} + x_3^4 x_P^4 \\ z_5 &= 4\frac{x_3^4}{x_P} + 4a\frac{x_3^4}{x_P^2} + 4\frac{x_3^4}{x_P^3} \end{aligned}$$

FIGURE 5.6 – Calcul $2(T_1 - T_2)$ et $2P$

$$\frac{x_{2P}}{z_{2P}} = \frac{x_5}{z_5} \quad \Leftrightarrow \quad x_{2P}z_5 = x_5 z_{2P} \tag{5.11}$$

- Calcul de $x_{2P}z_5$:

$$\begin{aligned} x_{2P}z_5 &= (x_P^4 - 2x_P^2 + 1)(4\frac{x_3^4}{x_P} + 4A\frac{x_3^4}{x_P^2} + 4\frac{x_3^4}{x_P^3}) \\ &= 4x_3^4x_P^3 + Ax_3^4x_P^2 - 4x_3^4x_P - 2Ax_3^4 - 4\frac{x_3^4}{x_P} + A\frac{x_3^4}{x_P^2} + 4\frac{x_3^4}{x_P^3} \end{aligned}$$

- Calcul de x_5z_{2P} :

$$\begin{aligned} x_5z_{2P} &= (x_3^4 - 2\frac{x_3^4}{x_P^2} + x_3^4 * x_P^4)(4x_P^3 + 4Ax_P^2 + 4x_P) \\ &= 4x_3^4x_P^3 + Ax_3^4x_P^2 - 4x_3^4x_P - 2Ax_3^4 - 4\frac{x_3^4}{x_P} + A\frac{x_3^4}{x_P^2} + 4\frac{x_3^4}{x_P^3} \end{aligned}$$

FIGURE 5.7 – Comparaison des points $2P$ et V

Tentative de détection uniforme

Une modification du test de la PV est nécessaire afin d'obtenir une détection de faute efficace. Les tests $T_1 - T_2 = P$ et $2(T_1 - T_2) = 2P$ ne permettent pas de détecter la présence de fautes. Cependant, en remarquant que $2(T_1 - T_2) = 2T_1 - 2T_2$, un nouveau processus de test est envisageable. En effet, peut-être qu'en calculant $2T_1 - 2T_2$ différemment (c.-à-d. en ne faisant pas un `DBL` de $T_1 - T_2$), une comparaison avec $2P$ serait possible.

Au lieu de doubler le résultat de $T_1 - T_2$, les points T_1 et T_2 sont tour à tour doublés. Une fois les doublements de T_1 et de T_2 effectués, les résultats sont soustraits en utilisant la coordonnée x adéquate (celle de $2T_1 + 2T_2$). Ainsi, le calcul $2T_1 - 2T_2$ est réalisé à la place de $2(T_1 - T_2)$. Les opérations, lors d'une itération, pour permettre cette PV sont : $2T_1$, $2T_2$, $2(T_1 + T_2)$ et $2T_1 - 2T_2$, ce qui est trop conséquent pour être avantageux.

Néanmoins, en constatant que certaines opérations de courbe sont déjà réalisées durant la SM, il est possible de réduire le surcoût de la PV. En effet, à l'itération i de la SM, $2T_1$ est réalisé, puis à l'itération $i+1$ et lorsque $k_i \neq k_{i+1}$, le calcul de $2(T_1 + T_2)$ est également effectué. Ces opérations sont illustrées par les encadrés de la figure 5.8. Cette figure décrit deux étapes d'un MDL. La première, représentée par la première boîte, correspond à un bit de clé égal à 0. Deux opérations de courbe sont alors faites (une `ADD` et un `DBL`). Le deuxième étage de boîte correspond à l'étape suivante de la MDL. Le bit de clé est alors respectivement supposé égal à 0 et à 1 pour les boîtes de gauche et de droite. Par conséquent, traiter simultanément deux bits de clé successifs peut réduire le coût de la PV. Ainsi, une étape de vérification ne

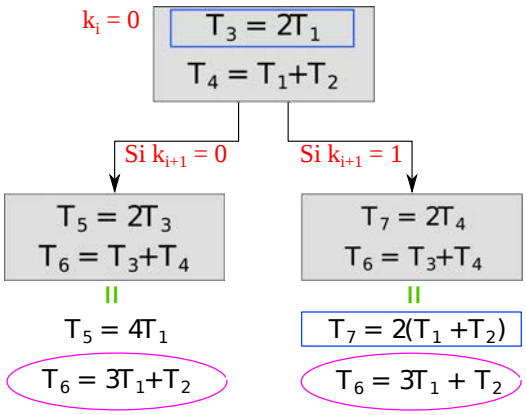


FIGURE 5.8 – Déroulement d'une itération de MLD

s'effectuera pas pour chaque bit de clé, mais pour deux bits consécutifs, réduisant le nombre de vérifications de moitié. De plus, la vérification utilisera des opérations déjà présentes.

Une nouvelle étape de la MDL est réalisée et elle est notée `ladderStep_V`. À ce stade, la nouvelle SM, notée `MLD_V` et traitant deux bits de clé à la fois, est protégée contre les FA. Cependant, comme les opérations en commun, entre la PV et les calculs nécessaires à la SM, dépendent de la valeur du bit de clé, `ladderStep_V` ne sera pas uniforme quelle que soit la valeur de k_i. C'est pourquoi, la PV doit encore être modifiée.

Détection uniforme de faute

En observant la figure 5.8, une opération est commune quelle que soit la valeur du bit de clé. Cette opération, entourée à la figure 5.8, correspond à $T_6 = 3T_1 + T_2$. L'étude de T_6 permet d'établir une relation avec P. En effet, en ajoutant (resp. en soustrayant) P à T_6, le résultat intermédiaire T_7 (resp. T_5) est retrouvé. Cette relation est décrite à l'équation 5.12.

$$\begin{cases} T_6 + P &= 3T_1 + T_2 + T_2 - T_1 &= T_7 \\ T_6 - P &= 3T_1 + T_2 - T_2 + T_1 &= T_5 \end{cases} \tag{5.12}$$

Grâce à cette constatation, une séquence d'opérations de courbe pour `ladderStep_V` se dégage et est décrite à l'algorithme 17. Cet algorithme prend en entrée deux points intermédiaires T_1 et T_2, ainsi qu'une variable nommée xor. La variable xor est le xor des deux bits du scalaire traités successivement durant la `ladderStep_V`. Durant cette étape, l'équivalent de deux étapes classiques de MDL et un test de détection de faute sont réalisés. Si une faute est commise pendant la SM, l'égalité entre T_8 et T_7 ($xor = 1$) et entre T_8 et T_5 ($xor = 0$) est fausse, permettant de déceler l'attaque.

Algorithme 17 : Séquence d'opérations de courbe pour `ladderStep_V`

Entrées : T_1, T_2, $xor \leftarrow k_i \oplus k_{i+1}$

$T_3 \leftarrow 2T_1$
$T_4 \leftarrow T_1 + T_2$
$T_5 \leftarrow 2T_4$ /*nouveau T_1, si $xor = 1$*/
$T_6 \leftarrow T_3 + T_4$ /*nouveau T_2*/
$T_7 \leftarrow 2T_3$ /*nouveau T_1, si $xor = 0$*/
$T_8 \leftarrow T6 + P$ /*utilise la coordonnée x du nouveau T_1*/
si $xor = 1$ **alors**
  $T_8 = T_7$
sinon
  $T_8 = T_5$

Le coût de l'étape de Montgomery originale est de $5\mathtt{M}+4\mathtt{S}+8\mathtt{A}+1\mathtt{M}_a$. Comme `ladderStep_V` traite deux bits de clé au lieu d'un, son surcoût est égal à $8\mathtt{M}+4\mathtt{S}+7\mathtt{A}+\mathtt{M}_a$. Ainsi, pour une SM réalisée avec l'algorithme `MLD_V`, le surcoût est de $4\ell\mathtt{M}+2\ell\mathtt{S}+4.5\ell\mathtt{A}+\frac{\ell}{2}\mathtt{M}_a$, où ℓ est le nombre de vérifications réalisées.

Le test de détection de `MLD_V` est plus rapide que l'utilisation du symbole de Legendre pour s'assurer que x^3+ax^2+x est un carré. Néanmoins, cette PV ne garantit pas que le point de base P appartient à la courbe. Cela permet de réaliser une attaque en donnant un mauvais point de base (voir la première attaque de la sec. 3.4.1). Pour éviter cela, la coordonnée y est conservée afin d'injecter les coordonnées du point dans l'équation de la courbe au début de la SM pour réaliser une PV basique. Lors des prochaines étapes de la SM, la coordonnée y est supprimée sans réduction de sécurité. Le coût du calcul de l'équation de la courbe est de $1M+2S+3A+\mathtt{M}_a$.

Finalement, notre PV uniforme protège simultanément les paramètres intermédiaires des points et des courbes contre les principales attaques FA avec un comportement uniforme, contrant ainsi les SPA. De plus, comme le MLD est uniforme, un compromis entre la sécurité et la performance est facilement envisageable. En effet, la vérification peut s'effectuer toutes les d itérations (régulièrement ou bien aléatoirement) ou encore au début et à la fin de la SM, réduisant considérablement le surcoût de la protection (mais aussi la réactivité de la détection).

5.2 Compteur d'itération

Pendant la SM, les manipulations de bits de clé sont très courtes et ont un comportement différent par rapport aux opérations arithmétiques sur le corps fini. Cependant, il est possible d'injecter des fautes durant les moments où les bits de clé sont manipulés (voir chap. 3). Un attaquant peut alors « forger » sa propre clé utilisée durant la multiplication scalaire $[k]P$. Ceci rend la protection du scalaire nécessaire. De plus, la protection PV, présentée précédemment,

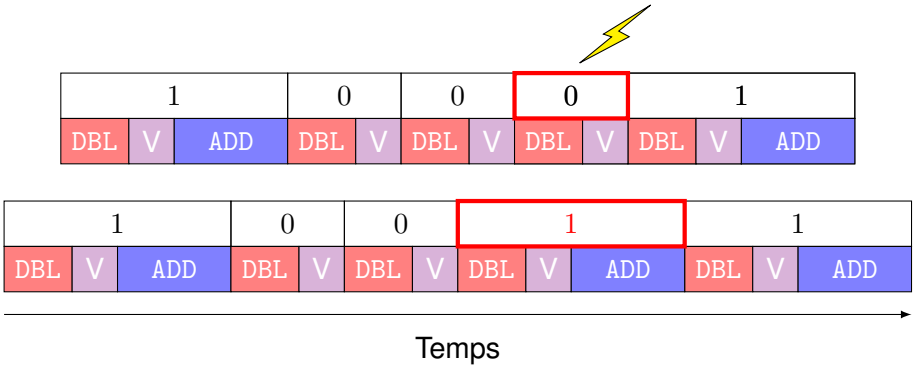


FIGURE 5.9 – Faute sur le scalaire

contrarie les attaques FA lorsque les points et les paramètres de la courbe sont ciblés. Cependant, la PV ne détecte pas les fautes lorsque la cible est le scalaire k. En effet, si un adversaire perturbe le scalaire (via p. ex. un *bit-flip*, voir chap. 3), alors un bit est modifié. La figure 5.9 illustre cette modification. La première séquence d'opérations correspond à une SM non perturbée lors de l'exécution. Une faute est effectuée sur le quatrième bit du scalaire. Cette perturbation entraîne un changement de bit du scalaire provoquant la deuxième séquence de la figure. Ainsi, une faute provoquera un changement de la séquence d'opérations de courbe. Néanmoins, malgré la présence manifeste de la faute, les points intermédiaires restent sur la courbe. Par conséquent, la PV ne détectera pas la faute bien que le résultat retourné ne sera pas celui attendu.

Une méthode est donc nécessaire afin de s'assurer que la séquence d'opérations, de la SM exécutée, correspond effectivement à la clé réelle k, même en présence d'une FA. Le scalaire sera alors protégé contre certaines attaques FA. Dans l'optique de traiter simultanément les SCA et les FA, cette protection doit aussi avoir un comportement uniforme afin de ne pas rendre le système vulnérable à des SPA.

5.2.1 Tentatives de protection du scalaire

Une solution naïve pour protéger des FA est d'effectuer une somme de contrôle qui calcule le poids de Hamming de k. Durant la SM, un registre est incrémenté de un à chaque `ADD`. Il est ensuite comparé à une valeur de référence (ici, le poids de Hamming) à la fin de la SM. Si les deux valeurs sont différentes alors une faute est repérée.

Malheureusement, cette méthode ne permet de détecter l'attaque que lorsqu'un nombre impair de bits de clé est corrompu. Par exemple, si un attaquant est capable de changer la valeur de deux bits de clé à différents indices ($i \neq i'$), alors le poids de Hamming du nouveau scalaire est le même que celui du scalaire initial. En effet, avec des itérations i et i' très distantes, un attaquant peut avoir le temps d'injecter deux fautes distinctes. Comme le poids de

Hamming est le même, la somme de contrôle ne remarque pas le changement des valeurs. Cette méthode est donc insuffisante.

Une autre solution consiste à compter le nombre de ADD en utilisant un poids dépendant de l'indice d'itération i. Lorsque $k_i = 1$, l'indice i est ajouté à un registre Reg (où i est petit et, au plus, le nombre de bits de k). Pour qu'une attaque en faute sur le scalaire ne soit pas détectée, un attaquant doit être capable de réaliser plusieurs *bit-flips* en fonction des valeurs intéressantes de i, ce qui est très improbable (c'est une hypothèse classique dans la littérature du domaine). Ainsi, l'écrasante majorité des fautes sur k sont détectées. Malheureusement, lorsque $k_i = 0$, le registre Reg n'est pas modifié, ce qui entraîne une baisse d'activité faible mais mesurable pouvant rendre le système vulnérable aux SCA. Cette deuxième solution est bonne contre les FA mais pas suffisante contre des SPA.

5.2.2 Protection du scalaire avec un registre

La baisse d'activité de la solution précédente est une faiblesse qu'il est nécessaire de corriger. Dans ce but, le registre Reg est scindé en deux parties, comme illustré à la figure 5.10. La première partie (à gauche) est celle qui contient l'information, alors que la seconde partie (à droite) ne sert qu'à camoufler l'activité du registre. Ces deux parties n'ont pas forcément la même taille. Cependant, il faut s'assurer que la partie dédiée au compteur (la partie gauche) sois suffisante pour le contenir. La partie de droit est initialisée avec une valeur aléatoire, nommée Θ. La variable i est additionnée au compteur (à gauche) ou à Θ (à droite) à chaque nouvelle itération. À la fin de la SM, la partie de gauche est comparée à une valeur de référence, qui est précalculée à partir de la clé avant le début de la SM. Cette protection est, par la suite, notée le compteur d'itérations IC. Le coût de cette protection est de ℓ petites additions (où la valeur des opérandes sont au plus le nombre de bits de k) et d'une génération d'un nombre aléatoire Θ.

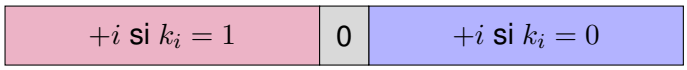


FIGURE 5.10 – Compteur d'itérations avec un registre.

L'algorithme 18 décrit une SM uniformisée avec PV (DBL+V et ADD ont le même coût et la même séquence d'opérations, voir sec. 5.1) et où IC a été ajouté. Une variable e est fixée à 0 à chaque début de la boucle de la SM. Ensuite, lorsqu'une ADD est réalisée, cette variable est modifiée et devient $e = 1$. À la fin de l'itération i (à la ligne 9 de l'algorithme 18), le registre Reg est actualisé. Si $e = 0$ alors la valeur de i est ajouté dans les bits de poids faible de Reg. Sinon, i est ajouté dans les bits de poids forts.

Cette méthode est adaptée aux algorithmes utilisant une instruction `if` pour réaliser une `ADD`. Cependant, pour les algorithmes qui n'ont pas d'instruction conditionnelle, tel que le MLD, `IC` nécessite d'en ajouter une, ce qui peut être vulnérable vis à vis des SCA.

Algorithme 18 : SM uniformisée avec `IC` et PV

Entrées : P et $k = (k_{m-1}, \ldots, k_0)_2$
Résultat : $[k] \cdot P$

1 $Q \leftarrow \mathcal{O}$
2 Reg $\leftarrow$ nombre aléatoire /* Correspond à Θ */
3 **pour** $i = m - 1$ **à** 0 **faire**
4 $\quad e \leftarrow 0$
5 $\quad Q \leftarrow$ DBL+V(Q)
6 $\quad$ **si** $k_i = 1$ **alors**
7 $\quad\quad Q \leftarrow$ ADD(Q, P)
8 $\quad\quad e \leftarrow 1$
9 $\quad$ Reg $= 2^{16} \times e \times i + (e + 1 \bmod 2) \times i$
10 **retourner** *Le résultat de la SM si égalité avec la valeur de référence.*

5.2.3 Protection du scalaire avec plusieurs registres

La fonction `cswap` est utilisée, dans le MDL, afin d'éviter une instruction conditionnelle, dépendant des bits du scalaire, dans l'algorithme 10. Cette fonction est appliquée dans MLD pour intervertir les coordonnées des points intermédiaires en fonction des bits de clé à chaque itération. L'utilisation de `cswap` peut permettre d'éviter l'instruction conditionnelle de l'algorithme 18 et ainsi de rendre `IC` encore plus uniforme.

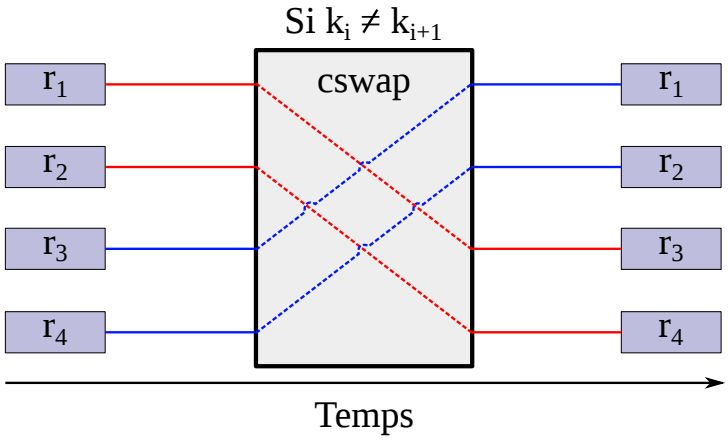


FIGURE 5.11 – Utilisation de `cswap` sur les registres

La variable `Reg` est scindée en 4 registres $r_1, \ldots, r_4$ afin d'appliquer la fonction `cswap`. Durant une étape de la SM, si $k_i \neq k_{i-1}$, `cswap` intervertie (r_1,r_2) avec (r_3,r_4), comme illustré à la

figure 5.11. Les bits de poids forts des registres r_1 et r_2 contiennent l'information nécessaire au compteur, alors que les bits de poids faibles contiennent du bruit. À chaque itération de la SM, les registres sont actualisés avec les valeurs adéquates. La figure 5.12 illustre cette modification. Si la variable i est paire, alors sa valeur est ajoutée à la partie de gauche de r_1 et à la partie de droite de r_2 . Sinon, la valeur de i est ajoutée à la partie de gauche de r_2 et à la partie de droite de r_1. De plus, des valeurs aléatoires sont ajoutées dans les bits de poids faibles de r_3 et r_4 indépendamment des bits de clé.

À la fin de la SM, les parties gauches de r_1 et de r_2 sont récupérées puis ajoutées entre elles. Ce résultat est comparé à une valeur de référence, calculée au préalable. Si une inégalité est observée, alors une faute est détectée.

Cette nouvelle version, notée `ICC`, coûte $\frac{m}{2}$ `swap`, $4m$ petites additions entières, 2 décalages et $2m$ de petites générations de nombres aléatoires.

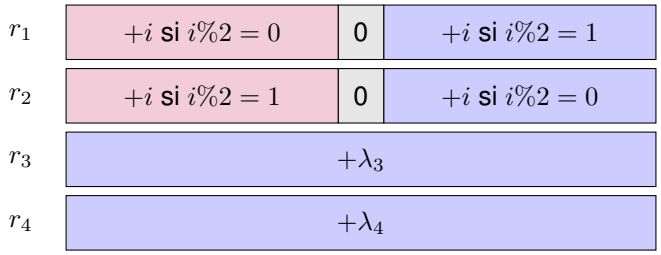


FIGURE 5.12 – Actualisation des registres

`ICC` détecte les attaques par *bit-flip* et résiste au SPA. Néanmoins, il ne détecte pas les fautes du type de *bit-set* ou *bit-reset* (les bits sont forcés respectivement à 1 et à 0), utilisés dans certaines SEA. En effet, si le i-ème bit est mis à 1 (resp. à 0) et que $k_i = 1$ (resp. $k_i = 0$), le résultat de la SM est donc celui attendu par le système, puisque le bit à la même valeur. Comme la séquence d'opérations de courbe est celle attendue, cette « non-modification » n'est pas détectée. Par conséquent, si le résultat de la SM est bien celui attendu, et ce malgré la faute, alors l'attaquant retrouve k_i. Les SEA peuvent s'éviter en utilisant des schémas de masquage tels que dans [34].

5.3 Politique de détection de faute

Les protections aux FA sont nombreuses. Elles ont pour objectif de détecter ou de corriger la ou les faute(s) survenue(s) lors de l'attaque. Différentes politiques de réaction à la détection de faute peuvent s'appliquer au moment de l'exécution : arrêt de l'exécution, effacement des données secrètes, re-calcul avec le même algorithme ou un autre algorithme, *etc*. Le choix de

la stratégie dépend de l'application et des menaces associées. Cependant, certaines politiques rendent le système vulnérable à certaines attaques (voir sec. 3.5.4).

Dans ces travaux, le résultat erroné n'est pas retourné afin de contrer les attaques du type DFA. Toutefois, l'exécution de la SM n'est pas stoppée, et un résultat est retourné afin de ne pas rendre le système vulnérable aux attaques du type SEA. De plus, lorsqu'une attaque est détectée, une clé aléatoire est utilisée à la place de la vraie clé. Ainsi, les attaques du type DBA peuvent devenir plus complexes.

La figure 5.13 illustre le principe de la stratégie appliquée. Un scalaire aléatoire k_r est généré avant la SM. Les deux clés sont manipulées de manière similaire, par exemple si un bit de k est demandé, alors le bit se trouvant au même emplacement de la clé k_r est aussi demandé. Si une attaque est détectée, k_r est utilisée, dès que possible, au lieu de k. Le coût de cette réaction aux attaques est le coût de la génération aléatoire de m bits (c.-à-d. la taille de k).

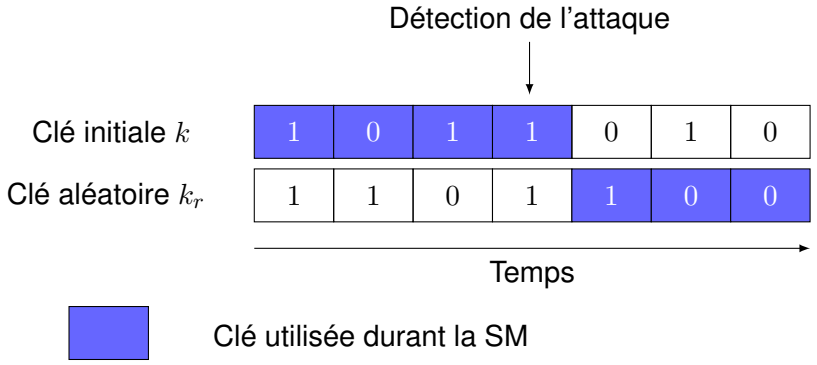


FIGURE 5.13 – Remplacement de la clé si une attaque est détectée

5.4 Implantations et résultats expérimentaux

Précédemment, deux types de protection ont été décrites. La PV, décrite en section 5.1, qui consiste à vérifier que le point courant appartient bien à la courbe prévue (et pas une courbe bien plus faible), permet de détecter des fautes sur les points et sur les paramètres de la courbe. Dans les cas des courbes de Weierstrass, cette méthode permet d'uniformiser la SM. Pour les courbes de Montgomery, cette méthode laisse la SM régulière. La seconde protection `ICC`, décrite dans la section 5.2, permet de protéger le scalaire avec un comportement uniforme. Ces deux protections ont été implantées, avec un niveau de sécurité de 128 bits, sur un microcontrôleur Cortex-M0 32 bits (STM32F0 Discovery board) avec la bibliothèque μNaCl [67]. L'implantation de PV et de ICC, ainsi que résultats expérimentaux de ces implantations sont décrits dans cette section.

La bibliothèque μNaCl est une bibliothèque cryptographique développée initialement pour des microcontrôleurs AVR [66]. Elle est basée sur la bibliothèque NaCl [37], puis a été étendue à des architectures de 16 et 32 bits, sur des microcontrôleurs SMP430 et ARM Cortex-M0. Dans la bibliothèque μNaCl, ECC est implanté avec les courbes de Bernstein en coordonnées XZ. Les courbes de Bernstein [8] sont des courbes de Montgomery sur $\mathbb{F}_p$ où $p = 2^{255} - 19$, avec comme paramètres $a = 48662$ et $b = 1$. De plus, l'algorithme 10, décrivant le MDL, est implanté pour réaliser la SM.

La principale variable utilisée dans la SM est la variable `state` qui se compose :

— des coordonnées des points T_1 et T_2 ;
— de la coordonnée x_P ;
— du scalaire ;
— du dernier bit de clé traité ;
— d'un compteur qui est décrémenté.

Lors d'une itération de la SM, la fonction `cswap` permet d'échanger T_1 et T_2 lorsque le bit de clé actuel est différent du précédent. Ensuite, les calculs sur les points intermédiaires sont réalisés grâce à la fonction `LadderStep` qui est exécutée sur la variable `state`.

5.4.1 Implantation des courbes de Montgomery

Dans le cas des courbes de Montgomery, les paramètres et les variables définis dans μNacl sont conservés dans `state`. Cependant, des modifications du code source de μNacl ont été nécessaires afin d'appliquer les protections proposées.

L'algorithme de MDL implanté dans μNaCl traite un à un les bits de clé. La SM avec PV, vue précédemment, traite quant à elle deux bits successifs à la fois afin de réduire le coût de la SM. Par conséquent, l'algorithme de SM à été modifié pour considérer deux bits.

La fonction `LadderStep` permet de calculer les points intermédiaires $T_1 + T_2$ et $2T_1$ pour une itération de la SM. Cette fonction n'effectue pas de vérification détectant des attaques en fautes. Par conséquent, la modification de `LadderStep` a été un autre changement conséquent. La fonction, nommée `ladderStep_V` et décrite à l'algorithme 17 dans la section 5.1.2, remplace la précédente.

L'ajout de la protection `ICC` fut une autre modification réalisée. Pour cela, une nouvelle variable du même type que `state` à été créée. Cette variable, nommée `regs_ICC` contient quatre registres qui composent le compteur d'itérations. Ces registres sont initialisés tels que décrit à la figure 5.12. Les nombres aléatoires nécessaires à `ICC` sont générés avec le générateur présent dans μNaCl. L'application de la fonction `cswap` à `regs_ICC` est également ajoutée à la SM. La valeur de référence est calculée avant la SM et placée en argument d'entrée.

La dernière modification correspond à l'ajout de la politique de réaction à la détection de faute. Pour cela, une clé aléatoire est générée avec le code de μNaCl au début de la SM. Cette

Dans μNaCl	:	bit $\leftarrow 1 \,\&\, \mathsf{state.k}[i]$
Après modification	:	bit $\leftarrow (1 \,\&\, \mathsf{state.k}[i] \,\&\, \mathsf{pv}) \,||\, (1 \,\&\, \mathsf{state.kr}[i] \,\&\, !\mathsf{pv})$

FIGURE 5.14 – Récupération du bit du scalaire

clé est incluse dans la variable `state`. Lors d'une itération, la manipulation de la clé est adaptée afin de traiter les deux clés (la clé de base k et la clé aléatoire k_r) de la même manière. De plus, cette manipulation se fait avec une seule instruction comme illustré à la figure 5.14. Les clés, contenues dans la variable `state`, sont récupérées respectivement en faisant `state`.k et `state`.kr. Après la modification de la manipulation de k, si une faute est repérée ($pv = 0$) alors le bit utilisé sera un bit du scalaire k_r.

Dans le but d'évaluer un compromis entre la performance et le niveau de protection, la PV peut s'effectuer toutes les d itérations. Par conséquent, les formules de `ADD` et de `DBL` d'origine sont conservées.

5.4.2 Implantation des courbes de Weierstrass

Dans le cas des courbes de Weierstrass, le même corps de base présent dans μNaCl, est utilisé (c.-à-d. $\mathbb{F}_p$ avec $p = 2^{255} - 19$). Néanmoins, les paramètres a et b définissant la courbe ont été générés avec SageMath puis ajoutés à `state`. Comme généralement le paramètre a est égal à -3, cette valeur est gardée pour l'implantation. Le paramètre b fut, quant à lui, généré aléatoirement en respectant certaines conditions spécifiques [59, 32], afin d'avoir une courbe sur le corps $\mathbb{F}_p$ avec $p = 2^{255} - 19$. En aucun cas cette courbe est certifiée sûre. Elle ne sert que d'exemple pour l'implantation afin d'évaluer le coût pratique des protections proposées. En plus des paramètres de la courbe, la coordonnée y a été ajoutée à la variable `state`.

L'algorithme de MDL utilisé pour la SM est remplacé par l'algorithme de la SM unifiée grâce à la PV (voir sec. 5.1). Bien évidemment, les formules avec la PV de `ADD` et de `DBL` pour les courbes de Weierstrass ont également remplacé celles pour les courbes de Montgomery. Tout comme pour les courbes de Montgomery, la formule de `DBL` est conservée afin d'évaluer le coût de la SM en fonction du nombre de PV.

La première version de `IC` fut ajoutée au code, comme d'écrit à l'algorithme 18. En effet, comme une instruction conditionnelle est présente dans l'algorithme, cette version est adaptée. Pour cela, un registre `Reg` fut créé et initialisé avec un nombre aléatoire Θ généré avec μNaCl. La valeur de référence est calculée avant la SM et placée en argument d'entrée. Enfin, tout comme pour les courbes de Montgomery, la réaction à la détection d'une attaque est implantée grâce à la modification de la figure 5.14.

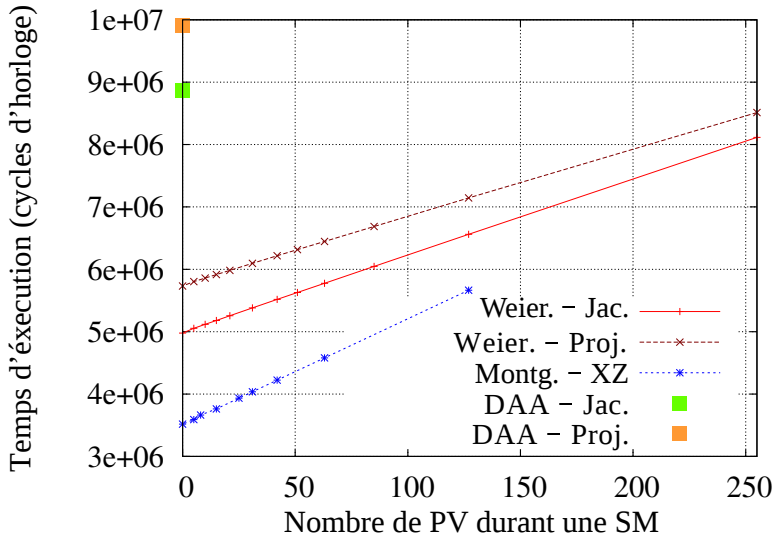


FIGURE 5.15 – Cycles d'horloge en fonction du nombre de PV

5.4.3 Résultats expérimentaux

Les différentes protections proposées dans ce chapitre ont été implantées sur un microcontrôleur Cortex-M0 32 bits (STM32F0 Discovery board) pour les courbes de Montgomery en coordonnées XZ et les courbes de Weierstrass en coordonnées jacobiennes et projectives. En plus des protections, l'algorithme `doublement et addition toujours` (DAA) est également implanté afin de comparer son surcoût pratique avec celui de la SM avec PV. Le scalaire, de 256 bits, et le point de base sont produits aléatoirement avant la SM. Les résultats de ces implantations sont décrits dans cette sous-section.

Le surcoût, en pratique, des protections est plus important que le surcoût théorique puisque la bibliothèque d'origine μNaCl a été optimisée pour remplir efficacement les registres des processeurs. Par conséquent, lorsque la PV ou `ICC` sont ajoutés, une pression de mémoire plus importante ralentit l'exécution.

La figure 5.15 illustre le temps d'exécution de la SM en fonction du nombre de PV réalisées pour les différentes courbes implantées. Le temps d'exécution de l'algorithme DAA est également illustré pour les courbes de Weierstrass. L'augmentation du nombre de PV implique un accroissement du temps d'exécution. Le coût pour chaque courbe augmente de façon linéaire (ce qui est prévisible). Malgré la croissance du temps d'exécution, la SM uniformisée grâce à la PV reste plus performante que DAA même dans le pire des cas (c.-à-d. PV à chaque itération).

Le surcoût de la SM en fonction du nombre de vérifications est récapitulé à la figure 5.16 pour les différentes courbes. La variable d, à l'axe des abscisses correspond à la fréquence à laquelle les PV sont faites. Par exemple, si $d = 5$ alors la PV est effectuée toutes les 5 itérations.

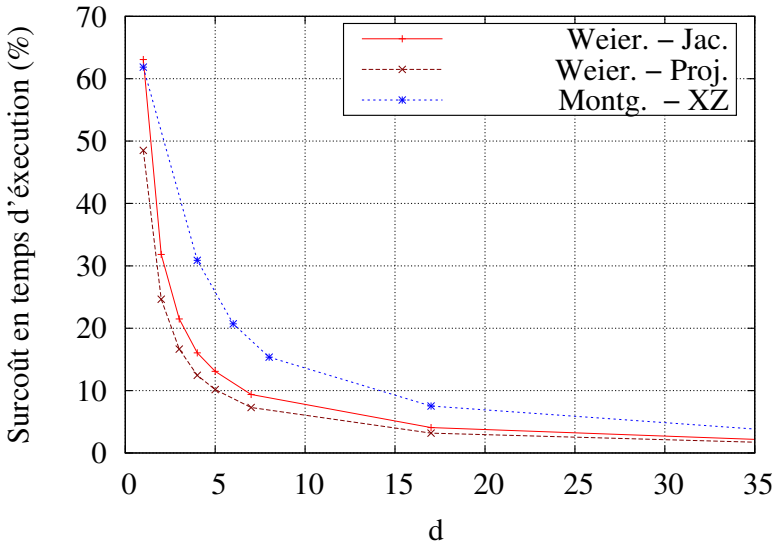


FIGURE 5.16 – Surcoût en fonction du nombre de PV

Le surcoût des différentes combinaisons de protection sur le Cortex-M0 est reporté à la figure 5.17. L'utilisation de la PV en début et en fin de la SM et l'utilisation seule de `ICC` sont respectivement représentées dans le graphique par les barres violettes et vertes. Clairement, ces deux méthodes sont peu coûteuses, quel que soit le type de courbes. Cependant, avec ces méthodes peu coûteuses, la faute n'est détectée qu'à la fin de la SM. Par conséquent, les bits de clé peuvent être exposés à des vulnérabilités durant l'exécution, impliquant une protection limitée contre les FA. La figure 5.17 permet également d'observer que l'ajout de `ICC` et de la réaction à l'attaque est négligeable lorsque la PV est effectuée à chaque itération ($d = 1$).

Le surcoût de la PV pour les courbes de Montgomery est de 62%, dans le pire des cas ($d = 1$), contre seulement 2.3% lorsque la PV n'est utilisée qu'au début et à la fin de la SM. Bien qu'un surcoût de 62% soit important, cette solution reste plus avantageuse que la redondance de calcul classique. En effet, avec ce type de redondance, la surface du circuit ou le temps d'exécution doublerait.

De plus, un compromis entre le niveau de protection et la performance du système est envisageable. Par exemple, si un concepteur de circuit ne peut se permettre que 20% de surcoût, alors, d'après la figure 5.16, il peut faire des PV toutes les 6 itérations (provoquant un surcoût de 20.7%). Ainsi, le nombre de bits exposés est limité.

Le surcoût de la SM avec la PV, pour les courbes de Weierstrass dans le pire des cas, est respectivement égal à 66% et à 50.3% pour les coordonnées jacobiennes et projectives. Le surcoût est moindre qu'une SM réalisée avec un algorithme DAA et permet en plus de protéger la SM de FA. En effet, pour les coordonnées jacobiennes, le surcoût est de 79.3% et pour les coordonnées projectives, le surcoût est de 73.4% avec DAA.

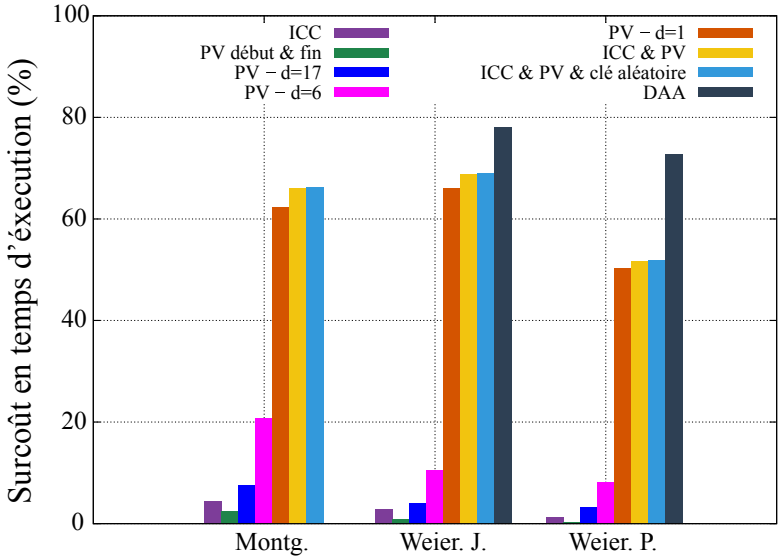


FIGURE 5.17 – Surcoût des différentes combinaisons de protection

Un compromis entre le niveau de sécurité et la performance est possible pour les courbes de Montgomery. Dans le cas des courbes de Weierstrass, cette constatation n'est pas forcément vraie. En effet, si la PV n'est pas faite à chaque itération dans le `DBL+V` alors l'opération de courbe sera un simple `DBL`. Le coût et la séquence d'opérations d'un `DBL` sont différents de celui d'une `ADD`. Ne pas réaliser un `DBL+V` à chaque itération supprimerait l'uniformisation de la SM. Par conséquent, l'absence de la PV, et donc de cet aspect uniforme, rend le système vulnérable aux SCA.

En plus du nombre de cycles d'horloge, le surcoût en taille de code et en RAM sont reportés au tableau 5.4, pour chacune des protections. L'ajout de la protection `ICC` accroît peu la taille du code. La taille du code lorsque la PV est réalisée à chaque itération n'est pas beaucoup (au plus 2%) plus conséquente que lorsque la PV est réalisée seulement en début et en fin de SM. Enfin, l'ajout de la réaction à la détection est aussi peu coûteuse, comme le montre la différence entre les ligne « PV+`ICC` » et « PV+`ICC`+réaction » du tableau 5.4.

5.5 Conclusion

Les attaques physiques du type SCA et FA doivent être considérées simultanément. En effet, certaines protections, contre l'un des types, peuvent laisser ou rendre le système vulnérable à l'autre type d'attaque. Dans ce chapitre, deux protections complémentaires ont été développées et implantées, pour les courbes de Weierstrass courtes et les courbes de Montgomery sur un microcontrôleur 32 bits. Ces protections ont pour but de traiter simultanément les SCA et les FA.

	protection	taille du code	taille de la RAM
Montg. XZ	ICC	2.7%	2.8%
	PV début et fin	10.6%	11%
	PV	12.6%	13.2%
	PV+ICC	16.2%	16.5%
	PV+ICC+réaction	17.3%	16.9%
Weier. Jac.	ICC	2.5%	2.6%
	PV début et fin	3.9%	4%
	PV	2.9%	2.6%
	PV + ICC	5.4%	5.6%
	PV+ICC+réponse	5.6%	5.8%
	DAA	0.4%	0.4%
Weier. Proj.	ICC	2.3%	2.4%
	PV début et fin	2.1%	2.2%
	PV	2.1%	2.2%
	PV + ICC	4.6%	4.8%
	PV+ICC+réponse	4.8%	5%
	DAA	0.4%	0.2%

TABLE 5.4 – Surcoûts expérimentaux.

La PV protège les paramètres et les coordonnées du point contre les FA tout en évitant une vulnérabilité aux SPA. Pour les courbes de Weiertrass, la PV permet une uniformisation moins coûteuse que l'algorithme DAA. De plus, contrairement à cette dernière, elle évite les attaques du type SEA, puisque aucune opération fictive n'est faite. Pour les courbes de Montgomery, la PV permet de protéger la SM contre des FA tout en gardant un comportement uniforme. Par ailleurs, un compromis entre le niveau de protection voulu et la performance permet d'avoir un bon niveau de protection avec un coût raisonnable.

La méthode ICC protège le scalaire des FA du type *bit-flip* avec un comportement uniforme et avec un faible coût. De plus, lorsqu'une attaque est détectée, la SM applique une clé aléatoire à la place de la véritable clé, compliquant des attaques telles que DFA ou DBA. Grâce à cela, ECC est protégée simultanément contre les FA et les SPA.

Même si la SM est préservée des attaques SPA classiques grâce à une séquence d'opérations uniforme, cela n'assure pas qu'elle soit sûre vis-à-vis d'autres SCA. Des attaques, comme par exemple des attaques dites horizontales [30, 112, 36] (voir sec 2), utilisent des différences dans les opérations de courbe mêmes. Par exemple, une attaque *big mac* exploite la possibilité de savoir si deux multiplications ont un opérande en commun. De plus, dans [3], les auteurs parviennent à discerner si une multiplication est un carré ou non. Un simulateur arithmétique d'activité à été développé, dans le but de valider les contre-mesures proposées dans ce chapitre, en considérant ces aspects. Il sera décrit dans le chapitre suivant.

CHAPITRE 6

SIMULATEUR D'ACTIVITÉ AU NIVEAU ARITHMÉTIQUE

Nous avons vu que, lors de la conception et l'implantation d'un système cryptographique, les protections contre les attaques par observation et celles par perturbation devraient être considérées simultanément. En effet, certaines protections contre les SCA peuvent fragiliser le système vis-à-vis des FA et réciproquement. Par exemple, l'uniformisation de la multiplication scalaire par l'ajout d'opérations factices contre des SCA rend cette opération vulnérable aux SEA. Dans l'optique de protéger la SM simultanément des SCA et des FA, deux protections ont été développées et décrites dans le chapitre précédent. La vérification de points (PV) et le compteur d'itérations (ICC) protègent la SM des FA, sans opérations factices, tout en ayant un comportement uniforme, contrant les attaques du type SPA. Néanmoins, une question se pose : des séquences d'opérations uniformes assurent-elles un comportement uniforme du système à un niveau plus bas lors du fonctionnement dans le circuit intégré ?

Les opérandes sont aussi une source de fuite d'information lors d'attaques par observation. En effet, leurs variations peuvent être assez facilement mesurables à certains endroits du circuit (en particulier lorsqu'elle se composent de plusieurs centaines de bits). Par exemple, dans un processeur, la sortie du banc de registres fournit les valeurs des registres lus aux différentes unités. Ainsi, les communications entre le banc de registres et certaines unités peuvent être espionnées avec une sonde EM très localisée. La même chose se produit sur les adresses et les données en entrée et en sortie de mémoires intégrées dans la puce. Que se passe-t-il si la séquence d'opérations sur $\mathbb{F}_p$ d'une `ADD` et d'un `DBL` sont parfaitement uniformes et similaires mais que l'algorithme à plus haut niveau utilise des opérandes au comportement assez différents selon le type des opérations considéré au niveau courbe (`ADD` *vs.* `DBL`) ?

L'article [3] montrait, en 2008, que dans une implantation uniformisée de RSA avec seulement une fonction de multiplication (pas de carré implanté) sur processeur ARM7, il est quand même possible de distinguer les multiplications des carrés lors de l'exponentiation modulaire du fait des différences entre $A \times B$ et de $A \times A$ dans la même fonction de multiplication. Il y a des corrélations supplémentaires entre les bits des opérandes, qui est deux fois le même, dans le cas du carré.

De plus, un article récent [54] montre bien que, même dans des technologies avancées, 28 nm dans ce cas, les cellules mémoires d'un circuit ont des comportements assez différents selon que le bit stocké soit un 1 ou un 0 en fonction de la valeur stockée précédemment dans la cellule.

Enfin, la logique de contrôle du processeur a aussi un impact sur l'activité électrique. Par exemple, dans un processeur pipeliné, certains signaux de contrôle ont des impacts sur les instructions aux différents étages du pipeline et potentiellement pendant plusieurs cycles d'horloges. Ceci peut « amplifier » certaines corrélations dans le temps. Il ne faut pas oublier que le chargement et le décodage des instructions consomme aussi de l'énergie et dans une proportion non négligeable par rapport aux opérateurs arithmétiques. Ainsi, l'activité de la logique de contrôle est également présente et significative dans les traces de consommation. Cette activité s'ajoute à celles engendrées par les opérations et les opérandes. Elle peut être un bruit parasite lors d'analyses statistiques ou bien un signal révélateur pour l'attaquant (en particulier quand l'activité liée aux opérations et aux opérandes à un comportement très uniforme, ce qui est le cas pour les opérations arithmétiques dans $\mathbb{F}_p$).

La variation du poids de Hamming (HW) des données manipulées est un modèle théorique, mais assez représentatif dans les petits processeurs, de la consommation dynamique d'un circuit (voir p. ex. [85]). Ainsi, en ne considérant que les unités arithmétiques et le stockage des opérandes, il est possible d'obtenir des traces de l'activité du circuit au niveau arithmétique sans considérer le contrôle du circuit.

Dans ce chapitre, un simulateur d'activité théorique, mais précis algorithmiquement, au niveau arithmétique est proposé. Ce simulateur est utilisé pour reproduire l'architecture d'un microcontrôleur 32 bits simple et classiqu. Il permet, entre autres, d'exécuter des SM complètes très rapidement. Au cours de l'exécution, les variations de HW des principaux éléments arithmétiques du processeur sont enregistrées afin d'évaluer l'impact des opérandes et des opérations. Nous ne considérons pas la logique de contrôle dans cette thèse. Toutefois, ceci constituerait une perspective intéressante pour un travail plus proche de l'électronique.

Le simulateur est développé en Python et avec le logiciel mathématique SageMath. SageMath est une interface en Python qui combine plusieurs bibliothèques mathématiques libres (`http://www.sagemath.org/`). Grâce à cela, des traces théoriques de l'activité sont obtenues plus facilement et beaucoup plus rapidement qu'avec un simulateur VHDL. En effet, comme dans ECC les nombres manipulés ont une taille supérieure à 200 bits et comme aucun support d'arithmétique dans les corps finis n'est disponible en VHDL, cela ralenti lourdement les simulations VHDL. Une simulation en SystemC serait probablement envisageable, mais nous ne connaissons pas cet environnement et nous avons préféré développer notre propre simulateur simplifié. De plus, SageMath apporte de nombreuses ressources mathématiques non disponibles en SystemC, pour lequel un interfaçage important serait nécessaire.

Grâce à notre simulateur, des fuites potentielles au niveau arithmétique peuvent être identifiées, permettant l'optimisation de SCA (p. ex. des DPA ou des attaques *template* plus ciblées). En effet, une fois une fuite localisée sur les traces arithmétiques, l'adversaire sait où concentrer son attaque. Notre simulateur peut aussi aider le concepteur pour la protection contre des SCA. En effet, le simulateur permet d'évaluer des contre-mesures à un niveau arithmétique afin de protéger simultanément la SM contre des SCA et des FA sans devoir implanter totalement chaque solution. Si des fuites arithmétiques sont détectées dans l'opération cryptographique, alors une correction du code initial peut être facilement testée.

Dans ce chapitre, le fonctionnement et l'utilisation du simulateur sont décrits. Enfin, des analyses et des attaques réalisées avec cet outil sont présentées.

6.1 Construction du simulateur

Un microcontrôleur est principalement constitué d'un cœur de processeur, d'une mémoire ROM, d'une mémoire RAM, de périphériques d'entrée/sortie et d'une horloge interne intégrés dans un circuit unique (pour des raisons de taille, de coût et de basse consommation d'énergie). Les calculs sont réalisés par le processeur grâce à des unités arithmétiques (UA). Les données nécessaires aux calculs et aux résultats transitent par des « bus » qui ont une largeur ou taille dépendant du type d'architecture du processeur. Par exemple, pour une architecture de 32 bits, les bus pourront transporter des mots de 32 bits à chaque cycle d'horloge. Afin d'obtenir l'activité théorique du circuit, des unités arithmétiques sont modélisées et leurs entrées et sorties sont enregistrées au niveau cycle et bit [1].

Le simulateur a pour objectif de reproduire le comportement d'un circuit au niveau arithmétique lors d'une SM. Par conséquent, des SM sont implantées pour quelques types de courbes elliptiques et pour quelques types de coordonnées. Les types de courbes et de coordonnées implantés sont les courbes de Weiertrass en coordonnées projectives et jacobiennes, les courbes d'Edwards en coordonnées projectives et les courbes de Montgomery en coordonnées XZ.

Plusieurs étapes, illustrées à la figure 6.1, composent le fonctionnement du simulateur : l'intégration des formules de calcul, leur ordonnancement, la création et la génération de fichiers de données et de paramètres, l'exécution et la génération des traces ainsi que des résultats. Lorsqu'un utilisateur souhaite employer des formules déjà présentes dans le simulateur, alors toutes ces phases sont transparentes pour lui. Au contraire, s'il souhaite appliquer des nouvelles formules, il devra les intégrer lui-même au simulateur et suivre les différentes étapes décrites par la suite.

1. Souvent appelée simulation CABA pour *cycle accurate and bit accurate*.

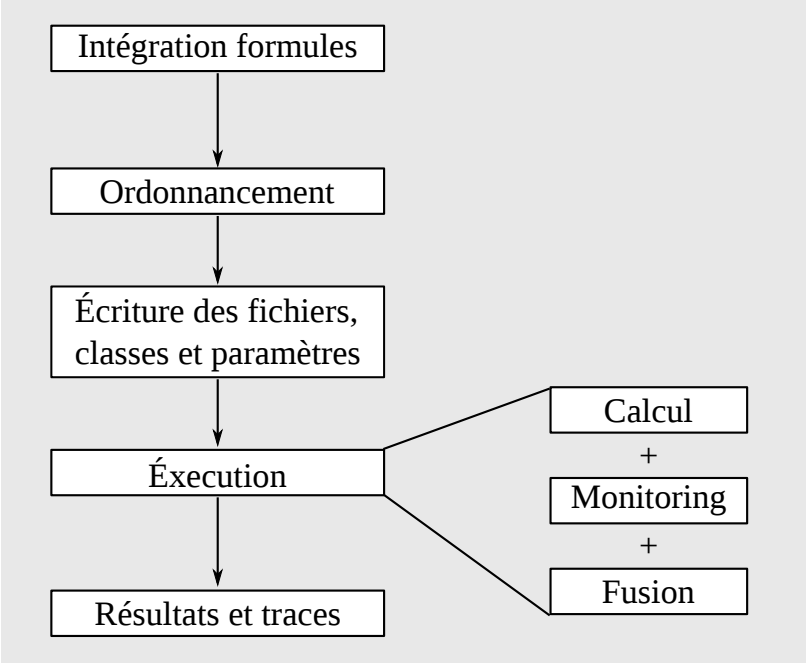


FIGURE 6.1 – Étapes du simulateur

Dans cette section, les algorithmes et les différentes structures nécessaires pour modéliser les unités arithmétiques sont détaillées. En outre, les différentes étapes du fonctionnement interne du simulateur sont également décrites dans cette section.

6.1.1 Unités arithmétiques dans le simulateur

Les nombres manipulés lors de calculs cryptographiques ont plusieurs centaines de bits. Les unités arithmétiques ne peuvent pas les traiter directement pour réaliser des opérations de corps tels que les additions (`fadd`) et les multiplications (`fmul`). Par conséquent, les nombres sont découpés en mots machine puis placés dans des registres afin de pouvoir appliquer différents algorithmes de calcul (voir chap. 4). Deux classes, nommées `register` et `fe_reg`, sont créées afin de représenter ces registres. Ces classes permettent de stocker les mots machine et ainsi reproduire un comportement le plus réaliste possible.

Classes `register` et `fe_reg`

Les classes `register` et `fe_reg` sont deux structures qui représentent les mots machine. Elles correspondent respectivement à un mot machine de 32 bits et à un élément du corps (une suite de mots machine).

```
1. sage :  a = ZZ.random_element(2^256)            /* Génération d'un a aléatoire */
2. sage :  b = ZZ.random_element(2^256)            /* Génération d'un b aléatoire */
3. sage :  fa = fe_reg('a',a); fb = fe_reg('b',b)  /* Affectation de a et b à fa et fb */
4. sage :  fa.fe_getValue()                        /* fa en mot de 32 bits */
  [4196364204, 4267878292, 1417697047, 166754131, 1277453640, 2830927515, 114941018,
  2033337000]
5. sage :  fb.fe_getValue()
  [1611638308, 3727385536, 3277129058, 3526737279, 2555186537, 1804422008, 1079871098,
  491466816]
6. sage :  fb[0].hw                                /* HW du 1er mot de fb */
  12
7. sage :  fb[0].bin
  [0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  1, 1, 0]
8. sage :  fc = fe_reg('c',0)
9. sage :  fc.fe_change(fb)                        /* Modification de fc et renvoie de la
                                                   distance de la HW */
  [12, 16, 13, 21, 14, 17, 14, 11]
10. sage :  fc.equal(fb)                           /* Test d'égalité */
  True
```

FIGURE 6.2 – Exemples de manipulations et d'opérations de `fe_reg`

Pour initialiser une variable de type `register`, le nom et la valeur de cette variable sont nécessaires. Une fois initialisée, cette variable a pour attributs : son nom, sa valeur, son poids de Hamming (HW) et sa représentation binaire. Cette classe possède plusieurs fonctions, comme par exemple la fonction `change` qui permet de modifier la valeur d'un registre en comptant la variation du HW entre la valeur précédente et la nouvelle. Les autres fonctions de cette classe ne seront pas plus détaillées dans cette section. En effet, comme les nombres de grandes tailles nécessitent plusieurs mots et puisque la classe `register` ne représente qu'un mot, cette structure ne sera pas directement utilisée par l'utilisateur dans le simulateur.

La structure `fe_reg` représente un nombre complet (élément du corps) constitué de plusieurs mots machine en regroupant plusieurs variables de type `register`. Tous les nombres, utilisés dans le simulateur et ayant plus de 32 bits, seront placés dans des variables de type `fe_reg`. Ainsi, toutes les opérandes et toutes les variables intermédiaires seront elles aussi de ce type. La figure 6.2 récapitule des exemples simples, avec SageMath, de manipulation cette classe. Dans cette figure, trois variables de type `fe_reg` sont créées : `fa`, `fa` et `fc`. Les deux premières sont initialisées avec des nombres a et b générés aléatoirement avec SageMath. La troisième est quant à elle initialisée à 0. La classe `fe_reg` possède plusieurs fonctions, telle que la méthode `fe_change`, utilisée à la ligne 9 de la figure 6.2. Cette fonction copie la valeur de `fb` dans `fc` en retournant la variation du HW (comme ici $fc = 0$ cela retourne le HW de fb). Plusieurs autres fonctions sont implantées dans la classe `fe_reg`, telles que l'ajout d'un registre

avec `add_reg` ou encore un test d'égalité avec `equal`. Elles ne seront pas plus détaillées dans cette section.

Arithmétique

Les calculs lors d'une SM sont réalisés dans le corps fini $\mathbb{F}_p$ avec p un grand nombre premier. Le simulateur peut être aisément étendu à d'autres types de corps finis. Le choix de ce corps fini est capital afin de définir le type d'algorithme utilisé pour les opérations de corps (voir chap. 4). En effet, certains corps permettent des réductions modulaires très peu coûteuses (p. ex. lorsque p est un premier de forme pseudo-Mersenne). Cependant, ces réductions simples fonctionnent rarement pour des premiers quelconques.

Le corps de base choisi initialement pour le simulateur est le corps fini $\mathbb{F}_p$ pour p un premier générique. Cela permet l'utilisation du simulateur pour plusieurs courbes différentes sur des corps ne nécessitant pas de propriétés particulières. Par conséquent, toutes les opérations sont implantées avec des algorithmes fonctionnant pour un corps quelconque. Toutefois, le corps $\mathbb{F}_p$ avec $p = 2^{255} - 19$ est couramment utilisé pour les courbes de Bernstein. Ainsi, des opérations spécifiques à ce corps sont également implantées.

L'addition et la soustraction de deux grands nombres sont réalisées avec l'addition et la soustraction présentes dans le code source de μNaCl. Ces opérations sont initialement écrites en C pour un microcontrôleur 32 bits. Comme le simulateur ne manipule que des variables de type `fe_reg` ou `register`, elles furent modifiées afin de les adapter.

L'algorithme de Karatsuba (voir alg. 15) est implanté pour effectuer la multiplication entre deux grands nombres. Le résultat de la multiplication est ensuite réduit avec un algorithme de réduction modulaire. En fonction du corps de base choisi pour la simulation, cette réduction sera effectuée par l'algorithme qui correspond au premier utilisé.

Lorsqu'un corps générique est choisi, l'algorithme de réduction modulaire de Montgomery est appliqué. Cela implique que les nombres se trouvent dans le domaine de Montgomery et s'écrivent $aR \bmod p$ avec $R = \beta^d$ où d le nombre de chiffres de p en base β et $\beta = 2^{32}$ dans notre cas d'étude. La conversion depuis et vers le domaine de Montgomery se fait seulement au début et à la fin des calculs, ce que est négligeable dans une SM complète. L'algorithme 19 décrit la multiplication modulaire, mot par mot, avec les méthodes de Karatsuba et de Montgomery.

Habituellement, lorsque le corps est défini par un p premier générique, la réduction n'est pas faite après la multiplication, comme c'est le cas ici. En effet, afin de ne pas faire doubler la taille du nombre, la multiplication et la réduction modulaire sont entrelacées dans le même temps avec la multiplication modulaire de Montgomery [89]. Pour simplifier, notre simulateur réalise, pour le moment, la multiplication puis la réduction modulaire dans le but de pouvoir utiliser des p premiers génériques mais aussi des p spécifiques. En effet, des corps spécifiques sont

Algorithme 19 : Multiplication et réduction modulaire

Entrées : a, b, p avec $-p^{-1} \bmod \beta$ précalculé, $R = \beta^d$ et $\beta = 2^{32}$
Résultat : $s \equiv a \times bR^{-1} \bmod p$ avec $s < 2p$

1 $r \leftarrow$ `Karatsuba`(a,b)
2 **pour** $i = 0$ **à** $d - 1$ **faire**
3 $\quad q_i \leftarrow -r_i p^{-1} \bmod \beta$
4 $\quad r \leftarrow r + q_i p \beta^i$
5 $s \leftarrow \dfrac{r}{R}$ **retourner** s

utilisés pour plusieurs courbes courantes et/ou standardisées (p. ex. les courbes de Bernstein ou du NIST).

Lorsque le corps de base est $\mathbb{F}_p$ avec $p = 2^{255} - 19$, une réduction pour ce pseudo-Mersenne est appliquée. Cette réduction modulaire ne fonctionne que pour cette valeur. Toutefois, elle est très peu coûteuse (quelques additions) et ne nécessite pas de représentation particulière des nombres.

6.1.2 Monitoring des opérations

L'objectif du simulateur est d'obtenir rapidement une modélisation précise de l'activité dans un processeur. Pour cela, chaque opération de corps réalisée durant la SM est monitorée afin d'obtenir des traces d'activité pour chacune d'entre-elles. Toutes les opérations de corps sont décomposées en de plus petites opérations qui se réalisent sur les mots machine grâce aux cellules d'addition(s) et/ou de multiplication(s) notées respectivement `wmul` et `wadd`. C'est sur ces cellules que le monitoring est effectué. La figure 6.3 illustre le monitoring de cellule d'addition durant une `fadd`. Les valeurs des « bus » sont enregistrées à chaque entrée et sortie d'une `wadd`, comme illustré par les cercles verts [2].

2. Vert clair pour les entrées et vert foncé pour les sorties.

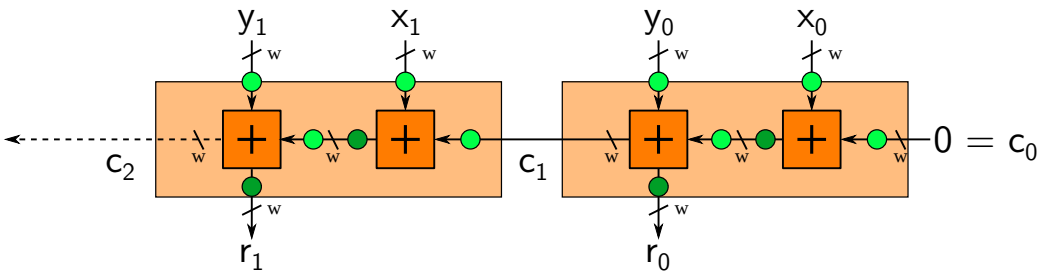


FIGURE 6.3 – Addition de nombres avec monitoring

```
11. sage :  rk = [0,0]
12. sage :  hw_mul = []; hw_add = []          /* Trace de la multiplication et de
                                              l'addition */
13. sage :  fc = rmul(fa,fb,hw_mul,rk)        /* Multiplication modulaire */
14. sage :  fd = radd(fa,fb,hw_add,rk)        /* Addition modulaire */
15. sage :  fc.fe_getValue()
  [4290043868, 1029128941, 2489042440, 2499571460, 196043332, 2816538375, 1596253781,
  2387411796]
16. sage :  fd.fe_getValue()
  [1513035216, 3700296533, 399858810, 3693491411, 3832640177, 340382227, 1194812117,
  2524803816]
```

FIGURE 6.4 – Exemples d'opérations de `fe_reg`

Toutes les opérations de corps (p. ex. `fadd`, `fsub`) sont implantées de telle sorte qu'elles utilisent des `fe_reg` comme opérandes. La figure 6.4 contient des exemples de `fmul` et `fadd` avec les variables `fa` et `fb` créées dans les exemples de la figure 6.2. Chacune des opérations prend en arguments `fa` et `fb`, qui sont les opérandes. Elles prennent respectivement en argument la liste `hw_mul` pour la `fmul` et la liste `hw_add` pour la `fadd`. Les variables `hw_mul` et `hw_add` sont nécessaires au stockage du monitoring des opérations. En effet, pour chaque opération, des traces sont obtenues puis ajoutées à ces listes. Par exemple, durant la `fadd`, la trace de `fa` + `fb` est ajoutée à la liste `hw_add`. Ainsi, les variables `hw_mul` et `hw_add` contiennent les traces des opérations antérieures. Dans l'exemple précédent, les listes sont toutes les deux vides puisque aucune opération n'a été réalisée auparavant. Le dernier argument des opérations `rmul` et `radd` est le rang attribué à l'opération lors de l'étape d'ordonnancement. La notion du *rang* est détaillée dans la section 6.1.4.

Des traces de valeurs de mots machine sont obtenues pour chaque opération de corps lors d'un calcul. La variation du HW, entre les différentes valeurs, n'est pas directement calculée. En effet, l'objectif du simulateur est de modéliser l'activité d'une SM et donc d'avoir une trace de la variation du HW pour toutes les opérations réalisées. De plus, comme en général, les opérations sont parallélisées dans un circuit, les traces d'opérations devront être fusionnées afin de reproduire cet aspect.

6.1.3 Intégration des formules

La SM est une succession d'opérations de courbe qui sont elles-mêmes des successions d'opérations de corps. Les formules, permettant de réaliser les opérations de courbe, doivent donc être intégrées au simulateur. Pour cela, la forme ou représentation statique à affectation unique (ou *static single assignment form*) (SSA) est utilisée. La représentation SSA impose que chaque variable n'est affectée qu'une seule fois, permettant, entre autres, de détecter plus

facilement les variables non utilisées lors du calcul. Deux structures, nommées `line_SSA` et `SSA`, sont créées afin d'adapter les formules, présentes dans [9] pour le simulateur.

La classe `line_SSA` représente une ligne d'une table SSA pour une variable donnée. Pour cette variable, les arguments d'entrée nécessaires, afin de créer une `line_SSA`, sont :

— `name` : le nom de la variable, par exemple c ;
— `pred` : les deux variables nécessaires pour calculer c, nommées *prédécesseurs* ;
— `tag` : le type d'opération nécessaire pour calculer c (`fadd`, `fmul`, `var`) ;
— `rk` : le moment où le calcul s'effectue, nommé $rang$ [3] ($[0, 0]$ par défaut) ;
— `value` : la valeur de c (0 par défaut) ;
— `out` : si la valeur est une variable de sortie de la formule ou non (faux par défaut).

Les prédécesseurs de la ligne sont stockés dans `pred` sous forme de liste (p. ex. `[a,b]`). Si la ligne correspond à un opérande du calcul, alors elle n'a pas de prédécesseur. Par conséquent `pred` s'écrira `[ , ]` et la valeur de `tag` sera `var`. Enfin, la valeur des variables (la variable correspondante à la ligne et ses prédécesseurs) n'a pas besoin d'être connue lors de la création d'une `line_SSA`.

Une ligne SSA est générée avec la commande `line_SSA(name, pred, tag, rk, value, out)`. Comme certains paramètres sont définis avec des valeurs par défaut, la génération d'une ligne SSA peut se faire plus simplement avec la commande `line_SSA(name, pred, tag)`. La figure 6.5 illustre des exemples d'utilisation de la classe `line_SSA`. Dans ces exemples, les variables t_0 et t_1 sont créées (aux lignes 1 et 4). Comme t_0 et t_1 ne possèdent pas de prédécesseur, la valeur de `pred` et de `tag` sont respectivement `pred = [,]` et `tag = var`. La ligne SSA, correspondante à la variable $X = t_0 + t_1$, est ensuite créée à la cinquième ligne. Comme les prédécesseurs de x sont t_0 et t_1, `pred = [`t_0`,`t_1`]` et `tag = fadd`.

Lorsqu'une ligne SSA est créée, elle peut s'intégrer dans une structure nommée `SSA` qui regroupe plusieurs `line_SSA`. La classe `SSA`, qui hérite de la classe `line_SSA`, représente un

3. Le notion de *rang* est décrite à la section 6.1.4.

```
1. sage : t_0 = line_SSA('t_0', [ , ], var)        /* Création d'une ligne SSA
                                                   pour la variable t_0 */
2. sage : t_0.pred
  [, ]
3. sage : t_0.out
  False
4. sage : t_1 = line_SSA('t_1', [ , ], var)
5. sage : X = line_SSA('X',[t_0,t_1], 'fadd', [0,0], 0, True)   /* Création d'une ligne SSA
                                                   pour la variable X = t_0 + t_1 */
6. sage : X.out
  True
```

FIGURE 6.5 – Exemples de créations et de manipulations de `line_SSA`

	Préd.	Opé.
X_1	,	var
Y_1	,	var
T_0	X_1, Y_1	`fadd`
T_1	T_0, T_0	`fadd`
M	T_1, T_0	`fadd`
A	M, M	`fmul`
xx	X_1, X_1	`fmul`
T_2	xx, T_1	`fadd`
B	T_2, T_0	`fadd`
X	B, B	`fadd`
⋮	⋮	⋮

TABLE 6.1 – Exemple de table SSA

tableau contenant les opérations de corps nécessaires pour réaliser une opération de courbe (`ADD` ou `DBL`). Ce tableau sera nommé *table SSA* dans le reste de ce document. Le tableau 6.1 est un exemple simplifié de table SSA. Chaque variable n'est affectée qu'une seule fois et n'a que deux prédécesseurs (au plus) dans le modèle de processeur utilisé. La structure `SSA` possède trois arguments : `tab` qui contient la table SSA ; `alg` qui correspond au nom de la table et `size` qui est le nombre de lignes de la table. Lors de l'initialisation de cette structure, ces paramètres sont optionnels. Ils sont alors par défaut égaux à `tab = [ ]`, `alg = ' '` et `size = 0`.

```
tab   =  SSA([],"dbl")                    /* Création de la table SSA tab pour le DBL */
 X1   =  line_SSA('X1',[ , ],var,[0,0])   /* Création des lignes SSA pour les variables
 Y1   =  line_SSA('Y1',[ , ],var,[0,0])   X1, Y1 et Z1 */
 Z1   =  line_SSA('Z1',[ , ],var,[0,0])
tab.add_lineSSA(X1)                       /* Ajout des lignes correspondant à X1, Y1 et Z1
tab.add_lineSSA(Y1)                       à tab */
tab.add_lineSSA(Z1)
 XX   =  tab.SSA_mul(X1,X1,"XX")
 ZZ   =  tab.SSA_mul(Z1,Z1,"ZZ")          /* Ajout des lignes correspondant la variable
 t2   =  tab.SSA_add(XX,ZZ,"t2")          créée avec l'opération adéquate (mul, add, sub)
 t3   =  tab.SSA_add(XX,XX,"t3")          et les prédécesseurs indiqués */
 M    =  tab.SSA_sub(t3,t2,"M")
 t5   =  tab.SSA_add(t1,M,"t5")
 X3   =  tab.SSA_sub(M,t5,"X3",True)      /* La variable X3 est la variable de sortie de
                                          ce calcul */
Retourner tab
```

FIGURE 6.6 – Exemples de créations et de manipulations de `line_SSA`

Nom de la fonction	Utilisation
`tab.get_index(var1)`	Renvoie la ligne correspondant à la variable `var1`
`tab.get_var(i)`	Renvoie la variable correspondant à la ligne `i`
`tab.get_pred(i)`	Renvoie les prédécesseurs correspondant à la ligne `i`
`tab.in_pred(i,var)`	Retourne vrai si `var` est un des prédécesseurs de la ligne `i`
`tab.clearSSA()`	Vide la table SSA `tab`
`tab.get_rank_full(rk)`	Renvoie la liste des lignes correspondant à des éléments de rang `rk`
`tab.archi_para(na,nm)`	Ordonnance les opérations de `tab` en fonction des dépendances de calcul et du nombre d'additionneurs `na` et de multiplieurs `nm`

TABLE 6.2 – Exemples de fonctions permettant la récupération de données présentes dans la classe `SSA`

Nom de la fonction	Utilisation
`tab.add_lineSSA(line_ssa)`	Ajoute la linge `line_ssa` à `tab`
`tab.SSA_mul(a,b,name)`	Renvoie une ligne nommée `name`, correspondant à $a \times b$, après l'avoir ajoutée à `tab`
`tab.SSA_square(a,name)`	Renvoie une ligne nommée `name`, correspondant à a^2, après l'avoir ajoutée à `tab`
`tab.SSA_add(a,b,name)`	Renvoie une ligne nommée `name`, correspondant à $a+b$, après l'avoir ajoutée à `tab`
`tab.SSA_sub(a,b,name)`	Renvoie une ligne nommée `name`, correspondant à $a-b$, après l'avoir ajoutée à `tab`

TABLE 6.3 – Exemples de fonctions permettant l'ajout de lignes dans une table SSA

Diverses fonctions sont incluses dans la classe `SSA`. Certaines d'entre elles sont axées sur la récupération d'éléments de la table comme par exemple les fonctions décrites au tableau 6.2. Dans ces exemples, la variable `tab` est une table SSA quelconque. Ces fonctions sont principalement utilisées dans des méthodes plus complexes qui sont décrites par la suite.

D'autres fonctions permettent l'ajout de lignes manuellement ou en fonction de l'opération caractérisant la ligne en question. Le tableau 6.3 décrit des exemples d'ajout de lignes à une table SSA nommée `tab`. Toutes ces fonctions, mises à part la première (`tab.add_lineSSA(line_SSA)`), renvoie une ligne SSA après l'avoir ajoutée à `tab`. Ces fonctions sont principalement utilisées lors de la phase d'intégration des formules de `ADD` et de `DBL`. Elles permettent de compléter simplement une table SSA et d'intégrer facilement les formules de `ADD` et de `DBL` au simulateur. La figure 6.6 décrit un exemple simplifié de création et de remplissage d'une table SSA. La table SSA, nommée `tab`, est créée avec l'étiquette `dbl`. Ensuite, les lignes qui correspondent aux opérandes de `dbl` (c.-à-d. X_1,Y_1 et Z_1) sont initialisées puis ajoutées à `tab`. Une fois les variables principales instanciées, la table est complétée grâce aux

méthodes d'ajout décrites au tableau 6.3. La table obtenue possédera 10 lignes et coïncide avec le calcul d'une variable nommée $X3$, qui sera la sortie de la formule `dbl`. Les valeurs des variables n'ont pas besoin d'être connues lors de la génération de la table. En effet, aucun calcul n'est encore réalisé puisqu'il s'agit d'une table générique pour une opération de courbe donnée.

6.1.4 Ordonnancement

Dans un processeur, les opérations sont en général réalisées en parallèle dès que possible quand les unités sont libres. Les plus petits microcontrôleurs n'offrent presque pas de parallélisme, mais certains plus performants offrent plusieurs unités parallèles. Afin de modéliser cette parallélisation, les opérations de corps, incluses dans la table SSA précédemment, sont organisées. Pour cela, un premier ordonnancement de ces opérations s'effectue en ne considérant que les relations de dépendance entre elles. Cette première étape permet de mettre en évidence les opérations parallélisables, comme illustré par les encadrés du tableau 6.5 où toutes les opérations encadrées sont indépendantes les unes des autres.

Comme une `fmul` est plus longue et plus complexe qu'une `fadd`, plusieurs `fadd` sont réalisables durant une `fmul`. De plus, ces `fadd` peuvent avoir, entres elles, des relations de dépendance, comme illustré à la figure 6.7. En effet, si les opérandes de ces `fadd` ne sont pas dépendantes du résultat de la `fmul`, elles peuvent s'effectuer en parallèle de cette même `fmul`. Par exemple, dans le tableau 6.5, la variable T_1, qui dépend de la variable T_0, peut être calculé à la suite de T_0 et ce pendant le calcul de xx. Grâce à cela, un deuxième ordonnancement est effectué et la notion de bloc est définie.

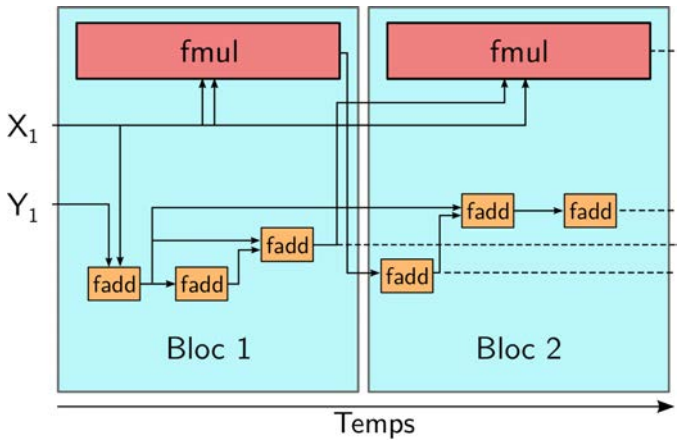


FIGURE 6.7 – Blocs de parallélisation

La notion de bloc a pu être aperçue à la figure 6.7 et est représentée par les boîtes bleues. Chaque bloc d'opération ne contient qu'une `fmul` mais peut contenir plusieurs `fadd`. Par consé-

quent, un bloc comporte, non seulement, des opérations parallélisables, mais aussi des `fadd` dépendantes les unes par rapport aux autres qui peuvent s'effectuer durant la `fmul` du même bloc.

	Préd.	Opé.	Rang	Bloc
xx	X_1,X_1	`fmul`	(1,0)	1
T_0	X_1,Y_1	`fadd`	(1,0)	
T_1	T_0,T_0	`fadd`	(1,1)	
M	T_1,T_0	`fadd`	(1,2)	
A	M,M	`fmul`	(2,0)	2
T_2	xx,T_1	`fadd`	(2,0)	
B	T_2,T_0	`fadd`	(2,1)	
X	B,B	`fadd`	(2,2)	
⋮				

TABLE 6.4 – Table SSA avec bloc

	Préd.	Opé.
xx	X_1, X_1	`fmul`
T_0	X_1, Y_1	`fadd`
T_1	T_0, T_0	`fadd`
M	T_1, T_0	`fadd`
A	M, M	`fmul`
T_2	xx, T_1	`fadd`
B	T_2, T_0	`fadd`
X	B, B	`fadd`
⋮	⋮	⋮

TABLE 6.5 – Table SSA

Cette notion de bloc et de second ordonnancement correspondent au rang qui fut introduit lors de la description des opérations de corps dans la sous-section 6.1.1. Elle est également présente dans la quatrième colonne du tableau 6.4, dont les opérations sont celles du tableau 6.5. Le rang d'une opération est constitué de deux éléments : le numéro du bloc de l'opération et son niveau de dépendance aux `fadd` du même bloc. Par exemple, une `fadd` avec un rang $[1, 0]$, comme pour T_0, est réalisée dans le bloc 1 et n'a pas de dépendance avec d'autres `fadd` du bloc. Au contraire, une opération avec un rang $[1, 1]$, comme T_1, à une relation de dépendance avec une autre `fadd` dont le rang sera $[1, 0]$. Cette autre `fadd`, en l'occurrence T_0, devra être faite préalablement.

La classe `SSA` possède une fonction, nommée `archi_para`, qui permet de réaliser les ordonnancements décrits précédemment. Cette fonction s'applique sur une table SSA et prend en argument le nombre d'additionneurs et le nombre de multiplieurs (par défaut tous les deux égaux à 1), comme illustré à la figure 6.8. Sur cette figure, une table SSA est générée, comme décrit à la sous-section 6.1.3, pour une formule de `DBL` des courbes de Weierstrass en coordonnées projectives. Cette table est ensuite ordonnancée avec la fonction précédente. Cette fonction va parcourir la table SSA afin d'attribuer un rang à chaque opération en considérant leurs dépendances et le nombre d'additionneurs et de multiplieurs. La figure 6.9 illustre les deux ordonnancements réalisés sur la table de la figure 6.8. Le premier ordonnancement en fonction des relations de dépendance des opérations permet d'obtenir la table de gauche. Ensuite, cette table est réorganisée comme décrit précédemment, ce qui permet d'obtenir la table de droite. C'est cette table qui sera utilisée lors de la SM.

```
1. sage : nb_add,nb_mul=1,1
2. sage : td = tab_doubleWP()
3. sage : td
  table SSA de l'algorithme dbl contient 28 variables
4. sage : g1 = td.archi_para(nb_add,nb_mul)
```

FIGURE 6.8 – Génération d'une table SSA et utilisation de la fonction `archi_para`

La fonction `archi_verif` est une autre méthode d'ordonnancement de la classe `SSA`. Elle diffère de la méthode précédente car elle attribue un rang aux opérations sans réaliser d'étape d'ordonnancement. Par conséquent, le rang des opérations dépendra de l'ordre chronologique d'insertion des opérations dans la table SSA. Cette fonction est principalement utilisée dans le cadre de la protection PV (proposée dans le chapitre 5.1) où les opérations sont ré-organisées spécifiquement afin d'assurer un comportement uniforme entre `ADD` et `DBL`. En effet, si le simulateur ré-ordonnance les opérations alors la SM uniformisée avec la PV ne sera plus uniforme.

Actuellement, plusieurs formules pour réaliser `ADD` et `DBL` sont déjà incluses dans le simulateur. C'est, notamment, le cas des formules pour les courbes de Weierstrass en coordonnées projectives et jacobiennes, pour les courbes d'Edwards en coordonnées projectives et pour les courbes de Montgomery en coordonnées XZ. Par conséquent, l'utilisation de ces formules est directe pour un usager.

6.1.5 Génération des fichiers et données

Une fois les étapes précédentes terminées, les paramètres et les données nécessaires à la simulation d'une SM sont initialisés. Deux types de paramètres sont différenciés : ceux essentiels pour la simulation de l'architecture et ceux utilisés pour la SM.

Le simulateur nécessite plusieurs paramètres qui sont sauvegardés dans la classe `simu_para`. Les premiers sont le nombre d'additionneurs et de multiplieurs. Ils sont respectivement égaux à 1 par défaut. La taille des mots machine correspond au deuxième paramètre et est notée `radix` dans le simulateur. Enfin, les derniers paramètres sont le nombre premier définissant le corps de base des calculs et les précalculs utilisés dans certains algorithmes. Les précalculs sont principalement utilisés pour la réduction de Montgomery qui nécessite le calcul du $R = radix^d$, où d est le nombre de chiffre de p et de $p^{-1} \bmod 2^{32}$ Les valeurs par défaut de ces paramètres sont sauvegardées dans la classe `simu_para`. La figure 6.10 décrit la manière de récupérer ces différents paramètres, où les valeurs par défaut sont précisées en commentaire.

Opérandes X1,Y1,Z1			
XX	[X1 , X1]	mul	[1, 0]
ZZ	[Z1 , Z1]	mul	[1, 0]
t2	[Y1 , Z1]	mul	[1, 0]
XX2	[XX , XX]	add	[2, 0]
s	[t2 , t2]	add	[2, 0]
tt	[ZZ , ZZ]	add	[2, 0]
t1	[ZZ , tt]	add	[3, 0]
t0	[XX , XX2]	add	[3, 0]
ss	[s , s]	mul	[3, 0]
R	[Y1 , s]	mul	[3, 0]
w	[t0 , t1]	sub	[4, 0]
t3	[X1 , R]	add	[4, 0]
Z3	[s , ss]	mul	[4, 0]
RR	[R , R]	mul	[4, 0]
t9	[RR , RR]	add	[5, 0]
t4	[t3 , t3]	mul	[5, 0]
t6	[w , w]	mul	[5, 0]
t5	[t4 , XX]	sub	[6, 0]
B	[t5 , RR]	sub	[7, 0]
t7	[B , B]	add	[8, 0]
h	[t6 , t7]	sub	[9, 0]
t8	[B , h]	sub	[10, 0]
X3	[h , s]	mul	[10, 0]
t10	[w , t8]	mul	[11, 0]
Y3	[t10 , t9]	sub	[12, 0]

Table `td` avec relation de dépendance

Opérandes X1,Y1,Z1			
XX	[X1 , X1]	mul	[1, 0]
ZZ	[Z1 , Z1]	mul	[2, 0]
XX2	[XX , XX]	add	[2, 0]
t0	[XX , XX2]	add	[2, 1]
t2	[Y1 , Z1]	mul	[3, 0]
tt	[ZZ , ZZ]	add	[3, 0]
t1	[ZZ , tt]	add	[3, 1]
w	[t0 , t1]	sub	[3, 2]
s	[t2 , t2]	add	[4, 0]
ss	[s , s]	mul	[5, 0]
R	[Y1 , s]	mul	[6, 0]
t3	[X1 , R]	add	[7, 0]
Z3	[s , ss]	mul	[7, 0]
RR	[R , R]	mul	[8, 0]
t9	[RR , RR]	add	[9, 0]
t4	[t3 , t3]	mul	[9, 0]
t6	[w , w]	mul	[10, 0]
t5	[t4 , XX]	sub	[10, 0]
B	[t5 , RR]	sub	[10, 1]
t7	[B , B]	add	[10, 2]
h	[t6 , t7]	sub	[11, 0]
t8	[B , h]	sub	[11, 1]
X3	[h , s]	mul	[12, 0]
t10	[w , t8]	mul	[13, 0]
Y3	[t10 , t9]	sub	[14, 0]

Table `td` après `archi_para`

FIGURE 6.9 – Résultat de la fonction `archi_para`

Les paramètres de la SM, enregistrés dans la classe `data_ms`, dépendent de l'utilisateur et correspondent à la courbe, au scalaire, au(x) point(s) de base et au nombre de SM souhaitées. La figure 6.11 récapitule ces différents paramètres présents dans la classe `data_ms`.

Un utilisateur peut choisir d'appliquer des courbes déjà incluses, celles qu'il désire ou encore d'en générer aléatoirement. Dans le cas de la génération aléatoire de la courbe, les paramètres la définissant sont produits aléatoirement tout en respectant certaines conditions [59, 32]. Bien sûr, ces courbes ne sont pas certifiées sûres, mais elles peuvent servir d'exemple. Les mêmes options sont possibles pour la génération du (ou des) point(s) de base. Le corps fini peut également être choisi par l'utilisateur ou être généré aléatoirement. S'il est généré aléatoirement, l'utilisateur devra toutefois indiquer la taille du nombre premier voulu (bien sûr, si peu de nombres premiers ont la taille souhaitée, alors le choix de ce nombre ne sera pas vraiment aléatoire mais n'est pas un problème pour les tailles de corps considérées habituellement dans ECC).

Une fois les différents paramètres décrits précédemment produits, les différents fichiers et classes (`simu_para` et `datas_ms`) les contenant sont créés automatiquement par le simulateur.

```
parametres  =  simu_para()          /* Les paramètres sont dans la classe
                                    simu_para() */
_prime      =  parametres.prime     /* défini par l'utilisateur */
_prime_p    =  parametres.prime_p   /* pré-calcul nécessaire pour la
                                    réduction de Montgomery */
_prime_reg  =  parametres.prime_reg /* la variable _prime dans le type
                                    fe_reg */
_radix      =  parametres.radix     /* 2^32 par défaut */
_na         =  parametres.na        /* 1 par défaut */
_nm         =  parametres.nm        /* 1 par défaut */
```

FIGURE 6.10 – Récupération des paramètres du simulateur

```
datas    =  datas_sm()    /* Les données sont dans la classe
                          datas_sm() */
_scal    =  datas.scal    /* Liste des scalaires pour les SM */
_points  =  datas.points  /* Liste des points pour les SM */
nb       =  datas.nb_sm   /* Nombre de SM */
```

FIGURE 6.11 – Récupération des paramètres de la SM

Le fichier exécutant la SM est également généré automatiquement. Il contient l'algorithme de la SM ainsi que les formules de `ADD` et `DBL`. Lorsque tous les fichiers sont créés, la simulation de la SM peut s'effectuer.

6.1.6 Exécution et fusion des traces

Lors de l'exécution d'une SM avec le simulateur, les opérations de corps sont monitorées par l'enregistrement des valeurs des différentes entrées et sorties des UA. Cependant, garder toutes les traces de toutes les opérations de corps, durant toute la SM, demande une grande capacité de mémoire RAM. Par conséquent, le traitement des traces est fait régulièrement au cours de la SM et non à la fin. Pour cela, la notion de *bloc*, introduite précédemment, est utilisée.

Pour chaque bloc, deux variables `hw_femul` et `hw_feadd` contiennent les traces des opérations. La première deviendra la trace de l'opération `fmul`, alors que la seconde contiendra une liste des traces de toutes les `fadd` du bloc, puisque plusieurs `fadd` peuvent se faire durant une `fmul`. À chaque début de bloc, ces deux variables sont réinitialisées comme des listes vides puis elles sont re-remplies au cours des calculs. À la fin de chaque bloc, une fusion et une

synchronisation des traces, contenues dans ces listes, s'effectuent afin d'obtenir une seule et même trace pour le bloc en question.

L'étape de fusion s'effectue grâce à la fonction `syncro_conso`. La figure 6.12 illustre une portion de calcul de deux blocs de calcul. Les listes `hw_femul` et `hw_feadd` sont vidées à chaque début de bloc. Elles sont ensuite remplies au fur et à mesure des opérations effectuées dans le bloc. Une fois les opérations terminées, les traces sont fusionnées.

Les entrées de la fonction `syncro_conso` sont : les traces `hw_femul` et `hw_feadd`, le nombre d'additionneurs et de multiplieurs (ici resp. 1 et 1) et la trace globale de l'opération de courbe (`ADD` ou `DBL`), nommée `conso`. La trace globale d'un bloc sera nommée T_g dans le reste de ce document.

La trace du bloc de calcul se base sur la trace `hw_femul` (puisqu'il n'y a qu'une seule `fmul` par bloc). Cette trace est parcourue afin de déterminer les instants où l'additionneur est inutilisé. Lorsqu'une inactivité est constatée, une `wadd`, des traces de `fadd`, est insérée à cet instant dans la trace, comme illustré à la figure 6.13. L'insertion des `wadd` respecte les contraintes de dépendance entre les différentes `fadd` du bloc. La trace T_g, obtenue par ce procédé, est ensuite assimilée par la trace `conso`. Cette étape de fusion est répétée pour chaque bloc de l'opération de courbe.

```
 1. hw_feadd = [ ]; hw_femul = [ ]              /* bloc 1 */
 2. E = radd(A, D, hw_feadd, [5, 0])
 3. H = rmul(Z1, Z1, hw_femul, [5, 0])
 4. t5 = rsub(A, D, hw_feadd, [5, 0])
 5. t3 = rsub(B, E, hw_feadd, [5, 1])
 6. syncro_conso(hw_feadd,hw_femul,1,1,conso)
 7. hw_feadd = [ ]; hw_femul = [ ]              /* bloc 2 */
 8. t2 = radd(H, H, hw_feadd, [6, 0])
 9. J = rsub(E, t2, hw_feadd, [6, 1])
10. syncro_conso(hw_feadd,hw_femul,1,1,conso)
```

FIGURE 6.12 – Blocs 5 et 6 de `DBL` pour les courbes d'Edwards

À la fin de la SM, la trace obtenue ne contient que les valeurs des mots en entrée des unités arithmétiques, et non les variations de HW. Par conséquent, cette trace doit subir une dernière transformation. Cette modification consiste à calculer les variations de HW entre les différentes valeurs consécutives, présentes dans la trace, et permet d'obtenir une trace de l'ac-

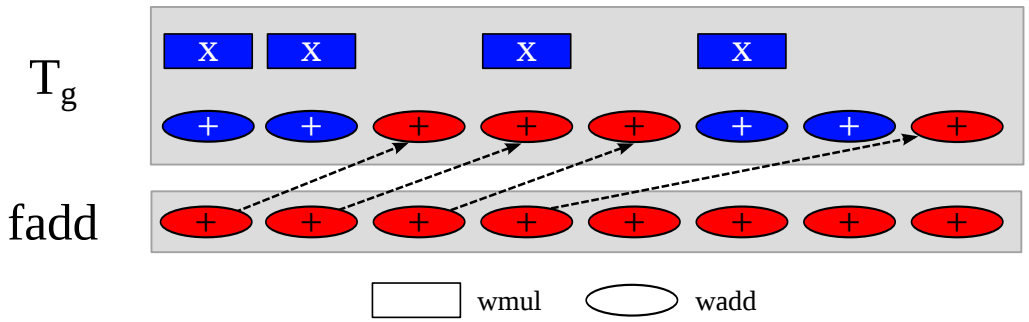


FIGURE 6.13 – Fusion de traces

tivité durant la SM. Finalement, l'étape de fusion permet de concrétiser l'aspect parallélisation des opérations d'un circuit dans le simulateur.

La trace de la SM est ensuite écrite dans un fichier. Si plusieurs SM sont effectuées, alors les traces s'ajoutent à ce même fichier. De plus, chaque ligne correspondra à une SM. Une fois des traces obtenues, des analyses et des attaques SCA sont possibles.

6.2 Génération et exploitation de traces

La section précédente décrit les différentes étapes de développement qui compose le simulateur. Une fois le simulateur au point, des traces d'opérations et de SM peuvent être recueillies. Dans cette section, le simulateur est utilisé afin d'analyser l'impact des opérandes sur des opérations de corps, ainsi que pour attaquer des SM. Réaliser des analyses sur ces traces ont un sens puisque dans la littérature du domaine, le modèle de la variation de Hamming est régulièrement utiliser dans des attaques.

6.2.1 Impact des opérandes sur les opérations de corps

Les opérations de corps sont utilisées tout au long de la SM. Par conséquent, si ces opérations fuient arithmétiquement, alors cela peut rendre la SM vulnérable. De plus, examiner l'impact des opérandes des opérations peut s'avérer intéressant pour valider ou invalider des protections contre les SCA basées sur l'uniformisation.

Analyse de l'addition sur $\mathbb{F}_p$

L'addition de corps `fadd` est réalisée par des additions de mots `wadd`. Lorsque les deux opérandes sont des nombres aléatoires, alors la variation de HW, des opérandes des `wadd`, est proche de 50%. Cela est vérifié par l'analyse des traces obtenues par le simulateur. Pour cela,

trois types d'acquisition de traces ont été réalisées pour une addition de corps. La première acquisition, qualifiée de *normale*, fut réalisée pour des opérandes x et y aléatoires. La deuxième, qualifiée de *double*, fut réalisée pour le même opérande x et correspond au calcul de $x+x$ avec x aléatoire. Enfin, la troisième acquisition, qualifiée de *triple*, correspond au calcul de $x + 2x$ avec x aléatoire. Les types d'opérandes sont récapitulés au tableau 6.6.

Type	Opérande
Normal	x et y aléatoires
Double	x avec x aléatoire
Triple	x et $2x$ avec x aléatoire

TABLE 6.6 – Type d'acquisition

Durant ces différentes campagnes, les opérations sont monitorées durant leurs exécutions, permettant d'obtenir des traces pour chacune d'entre-elles. Pour chacune des campagnes, les traces sont ensuite moyennées. Puis leur moyenne est comparée avec celle des autres campagnes, grâce à une différence des moyennes. De forte différences sont facilement observables pour les différents types d'opérandes choisis. En outre, ces différences sont directement observables sur les traces.

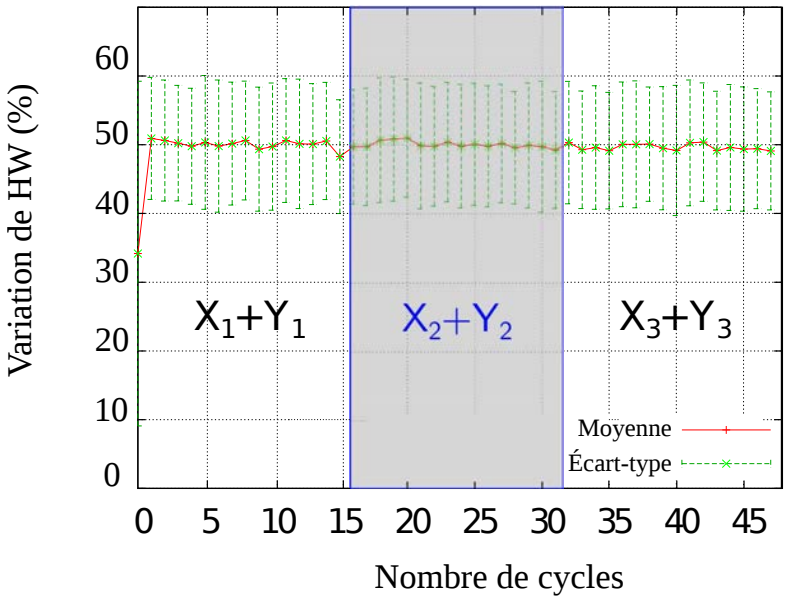


FIGURE 6.14 – Variation de WH de trois `fadd` consécutives avec en opérandes (x, y) aléatoires

La figure 6.14 représente la variation de HW durant trois `fadd` consécutives. Les opérandes de ces `fadd` sont (x, y) aléatoires, tels que $x \neq y$. Comme attendu, la variation de HW reste toujours très proche de 50%. La seule exception est au tout premier cycle. En effet, le premier registre de la première `fadd` est nul, puisqu'il correspond à la retenue sortante de la `wadd`

précédente. De plus, le troisième registre est également nul, puisque le sortie du `wadd` ne lui est pas encore parvenu. Par conséquent, lorsque la première `fadd` débute, seul un registre sur trois est non nul.

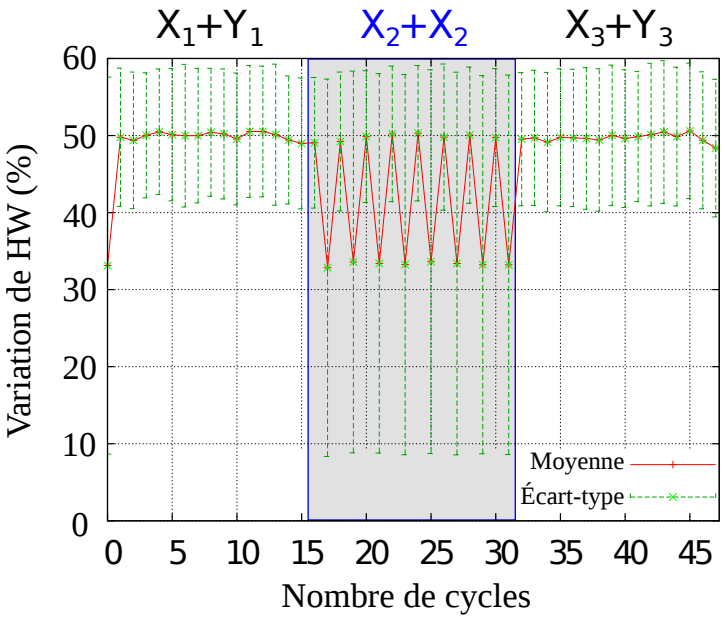


FIGURE 6.15 – Variation de WH de trois `fadd` consécutives, où la deuxième à pour opérandes (x, x) aléatoires

La figure 6.15 représente également trois `fadd` consécutives. Néanmoins, les opérandes de la deuxième `fadd` ne sont plus tels que $x \neq y$, mais tels que $x = y$. Ainsi, la figure 6.15 illustre une séquence de `fadd` avec des opérandes $x_1 + y_1$, $x_2 + x_2$ et $x_3 + y_3$. La variation du HW pour la première et la troisième `fadd` est, ici aussi, proche de 50%, alors que pour la deuxième `fadd` un motif en dent de scie se discerne. Par conséquent, les opérandes des `fadd` ont un impact significatif sur la variation du HW d'une `fadd` et donc sur le comportement de l'activité d'un circuit. Toutefois, l'impact sera-t-il suffisant pour être exploité ?

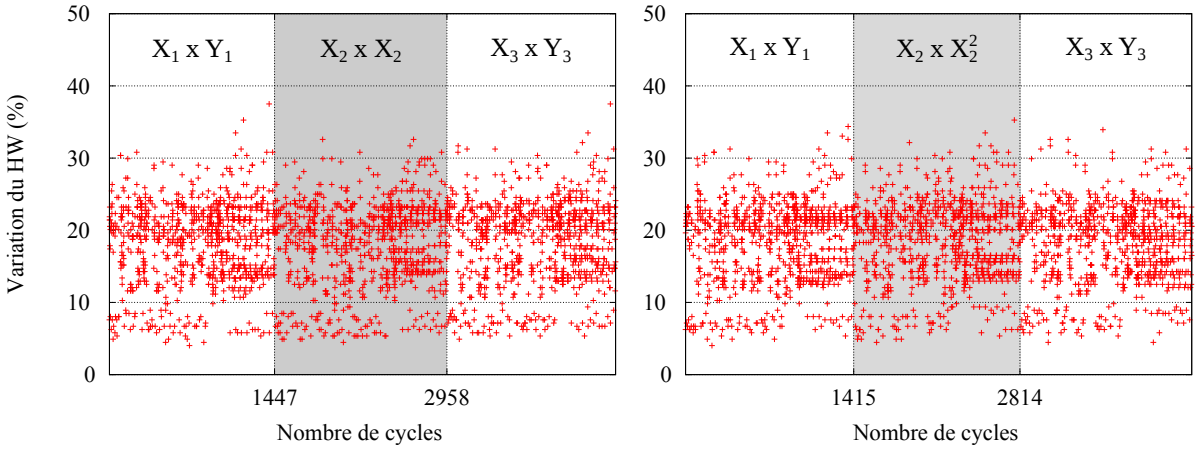
Analyse des traces pour la multiplication sur $\mathbb{F}_p$

L'étude de l'impact des opérandes de la `fmul` est, dans un premier temps, réalisée comme décrit précédemment pour la `fadd`. Trois campagnes d'acquisition, permettant d'obtenir des traces pour chacune des opérations, sont réalisées avec les mêmes conditions sur les opérandes décrit au tableau 6.6. La trace moyenne de chaque campagne est notée M_i avec $i = 1, 2, 3$. Le tableau 6.7 illustre le types des opérandes en fonction de la valeur de i.

L'impact des opérandes de la `fmul` n'est pas aussi flagrante que pour la `fadd` (qui s'observait à l'œil nu), comme l'illustre la figure 6.16 qui représente trois `fmul` consécutives avec des

i	Opérandes
1	x et y
2	x
3	x et x^2

TABLE 6.7 – Type d'opérande en fonction de i



(a) La deuxième `fmul` est un carré

(b) La deuxième `fmul` est $X_2 \times X_2^2$

FIGURE 6.16 – Trois `fmul` avec différents types d'opérandes

opérandes différents. Toutefois, si les opérandes ont un impact sur les traces d'activités, alors, pour une `fmul` quelconque, le type d'opérande devrait pouvoir être retrouvé. Par conséquent, une nouvelle campagne d'acquisition est lancée pour des `fmul` dont le type d'opérande est inconnu. L'objectif est de pouvoir déterminer si ces `fmul` ont des opérandes (x, y), (x, x) ou (x, x^2).

La moyenne des traces, des `fmul` inconnues, est calculée et notée M_O. Cette moyenne est ensuite soustraite aux traces moyennes M_i de chaque campagne. Puis, la plus grande valeur d de chaque différence de moyenne (en valeur absolue) est stockée. La différence des moyennes, entre les traces inconnues et les traces d'un même type, sera moins importante qu'avec les autres types. Le type d'opérande du plus petit d est alors retourné. Avec cette méthode, décrite à l'algorithme 20 le type d'opérande des multiplications est retrouvé avec un taux de réussite allant jusqu'à 67%.

Même si l'impact des opérandes sur la `fmul` est moins flagrante que pour la `fadd`, il est suffisamment important pour permettre de distinguer le type d'opérande. De plus, lorsque les opérandes ont une valeur particulière, alors cela à également un impact sur la trace d'activité. La figure 6.17 représente la variation de HW pour trois `fmul` consécutives. La première et la troisième de ces `fmul` ont des opérandes aléatoires de 256 bits. La deuxième de ces `fmul`

Algorithme 20 : Analyse de l'impact des opérandes sur la multiplication

Entrées : C_i les traces des trois campagnes et T des traces de multiplication
Résultat : Le type d'opérandes utilisé pour les multiplications de T

$M_i \leftarrow$ moyennes des C_i pour chaque i
$N \leftarrow$ moyenne de T
pour *les trois campagnes* i **faire**
    $d_i \leftarrow (M_i - N)^2$
$m \leftarrow$ minimum des d_i
retourner *Le type d'opérande associé à* m

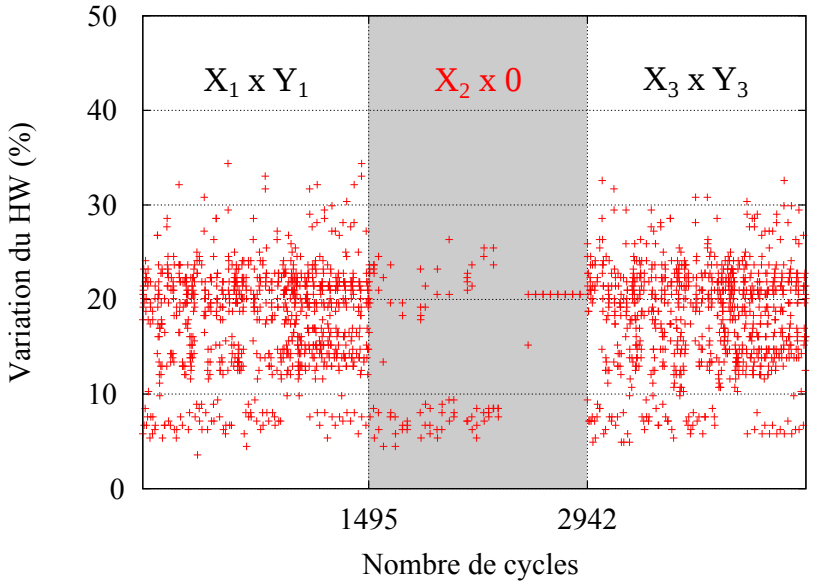


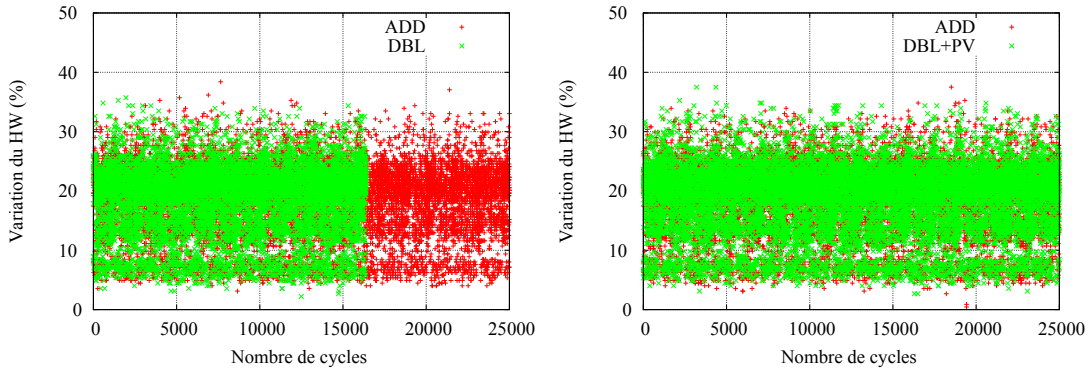
FIGURE 6.17 – Trois `fmul` consécutives, où la deuxième à pour opérandes $(x, 0)$

à un opérande aléatoire et l'autre opérande égal à zéro. Seules les variations non nulles de HW sont reportées sur le graphique afin d'alléger sa lecture. Clairement, la deuxième `fmul` est distinguable des deux autres. Ainsi, l'impact des opérandes sur la `fmul` est également discernable sur les traces d'activité du simulateur. Le principal objectif du simulateur est de s'assurer que les protections proposées dans le chapitre 5 sont bien uniformes. Des analyses sur les opérations de courbes sont donc réalisées.

6.2.2 Attaques SPA

L'uniformisation des calculs dans la SM a pour but d'éviter des SCA comme par exemple des SPA. Ces attaques exploitent la différence de comportement des opérations telles que `ADD` et `DBL`. La figure 6.18(a) représente la variation du HW durant une `ADD` et d'un `DBL` pour des courbes de Weierstrass en coordonnées projectives. Comme attendu, le comportement de ces

deux opérations se distingue. En effet, comme un doublement de point nécessite moins d'opérations de corps qu'une `ADD`, la durée d'un `DBL` sera plus courte qu'une `ADD`. Cette distinction est réduite lorsque la PV est ajoutée à `ADD`, comme illustrée à la figure 6.18(b). La robustesse de l'uniformité de la SM avec la PV est étudié en réalisant des SPA sur cette SM.



(a) `ADD` et `DBL` (b) `ADD` et `DBL+V`

FIGURE 6.18 – Traces d'opérations pour les courbes de Weierstrass en coordonnées projectives

Grâce au simulateur, des SPA furent réalisées. Elles ciblèrent des SM implantées, respectivement, avec les algorithmes de `doublement et addition` (DA), `doublement et toujours addition` (DAA) et avec la SM uniformisé grâce à la PV. Comme le point de base P de la SM est un élément public de ECC, il est supposé connu pour ces attaques. Ainsi, pour retrouver le scalaire k, nous possédons le couple (trace T, point P) pour une SM.

La SPA est effectuée étape par étape. En effet, pour un bit du scalaire, des traces des opérations intermédiaires (`ADD` ou `DBL`) sont réalisées grâce au simulateur. Ces traces sont ensuite comparées à la trace T. L'opération qui correspond le plus à T est retenue comme étant celle réalisée par la SM, permettant de déterminer le bit du scalaire, comme illustré au tableau 6.8. Les portions de trace correspondant à l'opération déterminée précédemment sont ensuite tronquées de la trace T. Cette étape est répétée jusqu'à ce que toute la trace T soit traitée.

Opé. i	Opé. $i+1$	Bit i du scalaire
`DBL`	`DBL`	0
`DBL`	`ADD`	1
`ADD`	`DBL`	Ne se prononce pas
`ADD`	`ADD`	L'une des `ADD` n'est pas une `ADD`

TABLE 6.8 – Bit de la clé à l'itération i

	DA	DAA	PV
Projective	71.72%	64%	50%
Jacobienne	71.48%	49.2%	35.8%

TABLE 6.9 – Taux de réussite de SPA sur des courbes de Weierstrass

Le tableau 6.9 récapitule le taux de réussite des SPA effectuées pour les courbes de Weierstrass en coordonnées projectives et jacobiennes. Le taux de réussite est le pourcentage de bits retrouvés sur la totalité du scalaire. La première constatation qui peut être faite est que la SM classique est plus vulnérable que celle avec des protections. Cependant, nous pouvons remarquer que le taux de réussite des SPA sur les deux autres SM n'est pas négligeable. En effet, même s'il n'est pas aussi important que pour une SM non protégée, le nombre de bits retrouvés est suffisant pour obtenir le scalaire dans sa totalité. L'observation des résultats des SPA pour les SM réalisées avec DAA permet de s'assurer que la SM uniformisée avec la PV n'est pas plus vulnérable à la SPA que cette dernière. Mieux, elle semble même être plus uniforme.

Le fort taux de réussite des SPA sur les SM uniformes peut s'expliquer par l'absence de la logique de contrôle du processeur, du chargement et du décodage des instructions, dans les traces attaquées. En effet, seules les unités arithmétiques sont présentes sur ces traces. Comme nous l'avons observé dans les parties précédentes, les opérandes ont un impact significatif sur l'activité d'un circuit. Ces impacts peuvent expliquer que, malgré un comportement apparemment uniforme, les SPA aboutissent.

6.2.3 Attaques DPA grâce au simulateur

Les protections proposées dans le chapitre 5, en plus d'être uniformes, ne doivent pas induire de nouvelles vulnérabilités. Pour cela, des DPA sont réalisées sur des traces obtenues grâce au simulateur. Toutes les acquisitions et les attaques furent réalisées sur *IGRIDA*. *IGRIDA* est une grille de calcul, de 1700 cœurs, mise à disposition pour les équipes IRISA/INRIA, et localisée à Rennes.

Acquisition de traces

Plusieurs campagnes d'acquisitions sont réalisées grâce au simulateur. Pour chacune des campagnes d'acquisition, le scalaire et tous les points de base sont générés aléatoirement, avec SageMath, par le simulateur lors de son étape de paramétrisation. Ces données sont alors stockées dans la classe `datas_SM`. Les simulations de SM sont ensuite effectuées en Python, sur la grille de calcul *IGRIDA*. Les traces obtenues sont stockées dans plusieurs fichiers puisqu'elles sont volumineuses. De plus, cela permet de les extraire plus efficacement par la suite. Les simulations de SM furent réalisées pour les courbes de Weiertrass en coordonnées

projectives et jacobiennes, de Montgomery en coordonnées XZ et d'Edwards en coordonnées projectives.

Dans le cas des courbes de Weiertrass, plusieurs cas de figure ont été simulés. Le premier correspond à des SM implantées avec l'algorithme de `doublement et addition`, et donc à des SM non protégées et vulnérables aux SCA. Le deuxième cas de figure est une SM avec une vérification de point incluse dans `ADD`. Le troisième cas de figure est similaire au deuxième, la vérification de point est alors incluse dans `DBL`. Enfin, le dernier cas de figure est une SM implantée avec les protections décrites dans le chapitre 5.

Dans le cas des courbes de Montgomery, la SM fut implantée avec l'algorithme de l'`échelle de Montgomery`, dans un premier temps. Dans un second temps, la SM fut implantée avec la méthode décrite dans le chapitre 5.

Enfin, les courbes d'Edwards sont implantées par un algorithme `doublement et addition` avec des formules de `ADD` et `DBL` uniformes.

Attaque DPA sur les traces du simulateur

Une fois des traces de SM récupérées, des SCA sont réalisables. Cette partie se focalisera, plus précisément, sur des DPA, réalisées sur les traces obtenues auparavant. Ces attaques se décomposent en plusieurs étapes, comme décrit plus précisément à la section 2.2.2. Dans un premier temps, des portions connues de l'opération cryptographique sont rejouées, pour chaque hypothèse de clé, afin d'obtenir un résultat prédictible r_p. Ensuite, les traces sont réparties, en plusieurs groupes, en fonction de la valeur de r_p. La différence des moyennes est alors appliquée sur ces groupes et l'hypothèse de clé ayant obtenue le plus grand pic est celle retenue.

Dans le cas d'ECC, les séquences d'opérations connues de la SM sont les opérations `ADD` et `DBL`. En effet, pour une hypothèse de clé donnée, la succession des `DBL` et des `ADD` est connue. Pour les attaques effectuées dans cette partie, le résultat prédictible r_p est la parité d'une des coordonnées du point intermédiaire à la fin d'une itération de la SM. La parité de la coordonnées x (resp. y) est celle utilisée lorsque les points sont en coordonnées jacobiennes (resp. projectives).

Par conséquent, `ADD` et `DBL` sont ré-implantées pour l'attaque, mais sans les variables de type `fe_reg`.

La figure 6.19 illustre le déroulement de la DPA réalisée dans cette partie. Les bits du scalaire sont attaqués un à un. Par conséquent, afin de ne pas devoir refaire toute la SM pour chacun des bits visés, une fonction, nommée `step_SM` et décrite à l'algorithme 21, est créée. À partir d'un point intermédiaire et du bit de scalaire courant, cette fonction renvoie le point intermédiaire suivant (après `DBL` et si nécessaire `ADD`). Pour cela, les points intermédiaires sont stockés après chaque calcul de la valeur de r_p et pour chaque hypothèse de bit. Ensuite,

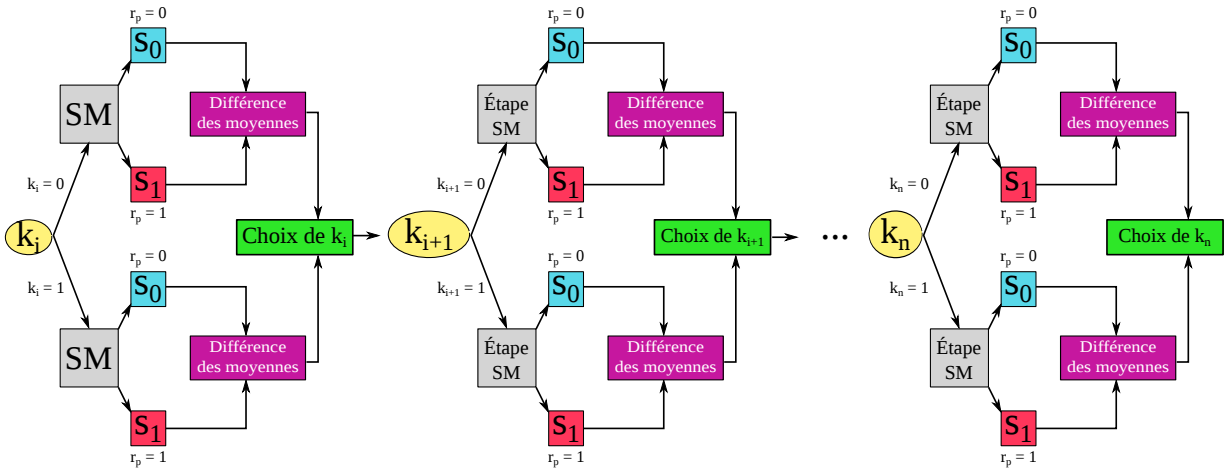


FIGURE 6.19 – Schéma de la DPA

lorsque le bit courant est déterminé, l'ensemble de points correspondant à ce bit est gardé. Ces points sont ensuite réutilisés à l'étape suivante de la DPA dans la fonction `step_MS`.

Algorithme 21 : `step_SM`

Entrées : k_i, P_i et P
Résultat : P_{i+1}

1 $P_{i+1} \leftarrow 2P_i$
2 **si** $k_i = 1$ **alors**
3 $\quad P_{i+1} \leftarrow P_{i+1} + P$
4 **retourner** P_{i+1}

Les calculs, pour les différentes hypothèses de k_i, sont faits successivement. De plus, comme les traces sont réparties dans plusieurs fichiers, une parallélisation des calculs peut s'effectuer très simplement. La figure 6.20 décrit cette parallélisation. La variable n_f correspond au nombre de fichiers de traces utilisés lors de l'attaque et la variable n_r correspond, quant à elle, au nombre de bits attaqués. La fonction `Accumulation` est exécutée en parallèle n_f fois. Les résultats de ces exécutions sont ensuite regroupés afin de continuer l'attaque sur le bit courant.

Plusieurs DPA furent lancées en faisant varier le nombre de traces utilisées. Ces attaques ont été lancées sur des SM avec différents niveaux de protection. La première SM, notée SM non protégée, est réalisée avec un algorithme de `doublement et addition` classique. La deuxième correspond à une SM réalisée avec une PV basique rajoutée dans la `ADD` du `doublement et addition`. La suivante correspond à une PV basique ajoutée dans le `DBL`. Enfin, la dernière SM correspond à une SM uniformisée avec la PV décrite dans le chapitre 5 avec `ICC`.

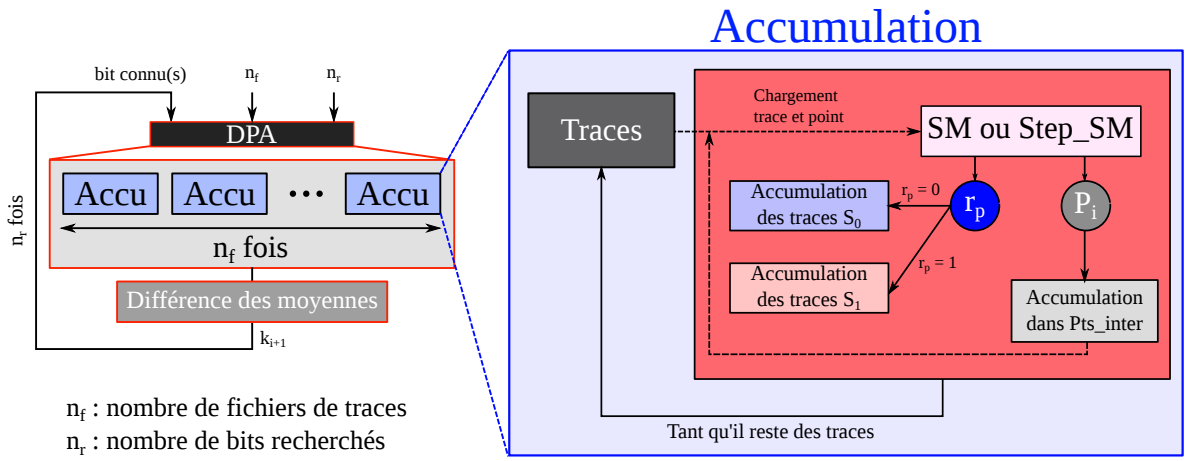


FIGURE 6.20 – Parallélisation de la DPA

Nombre de traces	Non protégée	PV basic	PV et IC
300	57%	50%	40%
400	70%	53%	43%

TABLE 6.10 – Taux de réussite de DPA sur des traces de SM en coordonnées jacobiennes

Les taux de réussite des différentes DPA effectuées, en fonction du nombre de traces, sont récapitulés dans les tableaux 6.11 et 6.10. Ces tableaux montrent que les attaques réalisées sur des SM non protégées ont un plus fort taux de réussite que des SM avec une PV (basique ou non) incluse dans le `DBL`. En outre, le taux de réussite est moindre pour les PV basiques (c.-à-d. lorsque les coordonnées du point sont simplement injectées dans l'équation de la courbe) que la PV décrite dans le chapitre 5.

Concernant les SM dont la PV fut incluse dans `ADD`, le taux de réussite des DPA (avec 200 traces) sont respectivement de 73% et de 63.3% pour resp. les coordonnées jacobiennes et projectives. Ces taux sont nettement plus élevés que pour les autres méthodes d'inclusion de la PV dans la SM. Cela est prévisible, puisque lorsque la PV est ajoutée dans le `ADD` alors la dissymétrie entre les opérations `ADD` et `DBL` est accentuée.

Nombre de traces	Non protégée	PV basique	PV et IC
200	56%	53%	50%
300	67%	60%	53%

TABLE 6.11 – Taux de réussite de DPA sur des traces de SM en coordonnées projective

Coordonnées	Taux de réussite
Jacobiennes	73%
Projectives	63.3%

TABLE 6.12 – Taux de réussite de la DPA sur des courbes de Weieirstrass

6.3 Conclusion

Un simulateur d'activité théorique au niveau arithmétique fut développé dans ce chapitre. Il se compose de différentes parties qui ont pour but de reproduire un comportement réaliste d'un circuit lors d'une SM. Grâce à ce simulateur des traces d'opérations de corps, de courbe et de SM ont été obtenues. Les traces d'opérations ont permis d'illustrer l'impact qu'ont les opérandes sur l'activité. En effet, avec des analyses statistiques simples, les opérandes (x, y) ont pu être distingués des opérandes (x, x) et $(x, 0)$ pour les additions et les multiplications de corps.

En plus de ces analyses sur les opérations de corps, des SPA et des DPA furent menées contre différentes méthodes d'implantation de la SM. Les résultats de ces attaques montre que la SM, uniformisée avec la PV et protégée contre les FA, a un comportement plus uniforme qu'une SM effectuée avec l'algorithme DAA. De plus, ces protections ne fragilisent pas la SM vis à vis des DPA. Toutefois, même si le modèle d'attaque est très favorable à l'attaquant, le taux de réussite des attaques montre qu'il reste des fuites exploitables. Par conséquent, des travaux sont encore nécessaires afin de se protéger d'attaques avancées.

CHAPITRE 7

CONCLUSION ET PERSPECTIVES

La cryptographie basée sur les courbes elliptiques (ECC) est un cryptosystème dont la sécurité repose sur le problème du logarithme discret. La principale opération d'ECC est la multiplication scalaire (SM). Cette opération, notée $[k]P$, consiste à multiplier un très grand entier k, la clé secrète ou publique selon les cas, avec un point P de la courbe E, un paramètre public. De nombreux algorithmes permettent de réaliser cette opération. Ils consistent à réaliser des successions d'opérations de courbe (p. ex. `ADD` et `DBL`) afin d'obtenir $[k]P$. Comme le scalaire k est la clé secrète dans certains protocoles, la SM doit être robuste vis à vis des attaques physiques.

Les attaques par observation (SCA) et les attaques par perturbation (FA) sont des attaques physiques qui exploitent les différences de comportement d'un circuit à l'exécution. Les premières tirent avantage des corrélations entre les informations sensibles et les paramètres physiques externes observables du circuit. Les secondes perturbent le circuit afin d'obtenir un comportement non spécifié. La SM est particulièrement vulnérable à ce type d'attaques. Il est donc capital de l'en protéger.

Les protections des SCA se basent sur l'uniformisation et/ou la randomisation du comportement du circuit. La détection et/ou la correction des fautes provoquées constituent le fondement des protections contre les FA. Les protections contre les SCA et les FA sont généralement considérées indépendamment les unes par rapport aux autres. Néanmoins, certaines protections, contre un type d'attaque, peuvent rendre le système vulnérable à l'autre type d'attaque. Par exemple, l'ajout d'opérations factices pour le calcul du résultat est une méthode pour contrer des attaques du type SPA. Cette protection rend le système vulnérable aux attaques du type SEA. Ainsi, protéger le système simultanément contre les SCA et les FA est primordial.

Les microcontrôleurs sont omniprésents dans notre société. Ils se retrouvent, par exemple, dans les appareils domestiques ; les imprimantes ; les appareils médicaux ; les distributeurs de billets ou encore dans de l'armement militaire. Par conséquent, il est vital de les protéger contre des attaques qui peuvent les cibler. Ils sont, en particulier dans des systèmes embarqués, vulnérables aux attaques physiques.

L'objectif de cette thèse était de concevoir, pour des (très) petits microcontrôleurs, une SM pouvant résister simultanément à certaines SCA et FA. Pour répondre à cette problématique, deux protections furent développées dans ce manuscrit et implantées sur un Cortex M0.

La première est la vérification de point (PV). Elle permet de protéger les paramètres de la courbe, ainsi que les coordonnées du point de base et courant contre la majorité des FA. De plus, cette protection assure un comportement uniforme vis-à-vis des SPA. Elle fut développée, en particulier, pour les courbes de Weierstrass en coordonnées projectives et en coordonnées jacobiennes et pour les courbes de Montgomery en coordonnées XZ.

Dans le cas des courbes de Weierstrass, la PV consiste à injecter les coordonnées d'un point dans l'équation de la courbe. Les calculs de la PV furent inclus dans l'algorithme `doublement et addition`. Cet algorithme basique n'est pas régulier et donc vulnérable aux SCA. Afin d'obtenir une SM régulière, les calculs de la PV ont été intégrés dans l'opération `DBL`, renommée `DBL+V`, de telle sorte qu'ils compensent les différences de coût entre les opérations de courbe `ADD` et `DBL`. De plus, les séquences d'opérations de `ADD` et `DBL+V` furent ordonnancées afin d'assurer les mêmes séquences d'opérations. Le coût de cette méthode, qui n'est pas vulnérable aux SEA, puisque sans opérations fictives, est moindre que celui d'un `doublement et addition toujours` (DAA).

Dans le cas des courbes de Montgomery, la PV est incluse dans l'algorithme de l'`échelle de Montgomery` (MLD) qui est un algorithme régulier naturellement résistant aux SPA. Elle consiste à s'assurer que la distance entre les deux points intermédiaires de MLD est toujours la même. L'ajout de la PV laisse la SM uniforme. Par conséquent, un compromis entre la performance et le niveau de sécurité est réalisable en n'effectuant pas la PV pour toutes les itérations. Le surcoût de cette méthode est au pire des cas de 62% (lorsque la PV est réalisée à chaque itération).

La deuxième protection, nommée le *compteur d'itérations* `ICC`, protège le scalaire contre des FA de type *bit-flip*. Elle consiste à s'assurer que la clé utilisée lors de la SM est bien celle attendue. Pour cela, les `ADD` sont associées à des *poids*. Ces poids correspondent à l'indice de l'itération où a lieu la `ADD` durant la SM. Les poids des `ADD` sont ensuite comptabilisés, dans un ou plusieurs registres, lorsque les `ADD` sont réalisées durant la SM. Le ou les registres sont modifiés quelque soit la valeur du bit courant. Par conséquent, cette méthode a un comportement uniforme durant la SM. Deux versions de cette protection existent. La première n'utilise qu'un registre mais nécessite une instruction conditionnelle. Elle est adaptée pour les algorithmes de SM ayant également une telle instruction. La seconde version de `ICC` n'implique pas l'emploi d'une instruction conditionnelle. Cependant, quatre registres sont utilisés avec une fonction `cswap` qui est généralement appliquée lors de l'algorithme de MDL. Cette protection contre les FA du type *bit-flip* a un faible coût (quelques pourcents) et elle est uniforme.

En plus de ces deux protections, une réaction à la détection d'une faute fut mise en place. Cette réaction consiste à remplacer le scalaire initial par un scalaire aléatoire. Ce scalaire est généré au préalable avant la SM. De plus, il est manipulé de la même manière que le scalaire

initial durant toute la SM. Cette réaction permet de compliquer les attaques du type DFA, SEA ou encore DBA, avec un surcoût négligeable.

Plusieurs axes de développement sont possibles pour ces travaux. Premièrement, même si la SM, obtenue dans ces travaux, est uniforme et donc théoriquement résistante aux SPA classiques, réaliser des SCA sur un vrai banc d'attaques pourrait permettre de valider plus solidement les protections proposées.

Un deuxième axe serait d'étendre la PV plus largement. En effet, la PV fut réalisée, dans ces travaux, pour les courbes de Weierstrass en coordonnées projectives et jacobiennes, ainsi que pour les courbes de Montgomery en coordonnées XZ. Toutefois, il est envisageable d'appliquer ces méthodes sur d'autres types de coordonnées, d'autres types de courbes ou encore d'autres formules de calcul de `ADD` et `DBL`.

L'uniformisation de la SM avec la PV n'ajoute pas d'aléa lors de la SM. Par conséquent, la PV et `ICC` n'ont aucune prétention à contrer les attaques du type DPA ou CPA par exemple. Par conséquent, il peut être judicieux, et même nécessaire, de coupler ces protections avec des contre-mesures usuelles décrites dans la section 2.3. Une implantation matérielle et des tests sur un banc d'attaques physique permettraient d'obtenir le surcoût de cette nouvelle SM protégée et d'éprouver sa robustesse à différentes attaques.

L'uniformisation de la SM fut réalisée, pour les courbes de Weierstrass, à partir d'un algorithme de `doublement et addition` basique. Un quatrième axe de recherche pourrait adapter la PV et `ICC` à un autre algorithme, comme par exemple w-NAF ou encore un algorithme utilisant un scalaire en représentation à base double. L'utilisation d'un algorithme plus performant comme base de travail pourrait permettre de réduire le coût de la SM avec PV.

Afin de valider les contre-mesures précédentes, un simulateur arithmétique d'activité fut également développé. En effet, même si les protections précédentes préservent la SM contre les SPA classiques, d'autres SCA peuvent peut-être exploiter ces dernières. Ce simulateur reproduit une architecture 32 bits d'un microcontrôleur très simple et réalise un monitoring des variations du poids de Hamming (HW) durant la SM. Grâce à ce simulateur, des traces d'activités furent recueillies pour des SM non protégées et des SM implantées avec les protections décrites antérieurement. Ces traces furent ensuite exploitées via des SPA et des DPA pour différents niveaux de protection de la SM. Les SPA ont montré que la SM avec PV est plus uniforme qu'une SM implanté avec un algorithme de `doublement et toujours addition`. De plus, cela a illustré que les opérandes peuvent avoir suffisamment d'impact sur l'activité pour causer des fuites d'information. Le taux de réussite des DPA fut moindre pour la SM uniformisée avec la PV avec une SM classique. Par conséquent, notre protection ne fragilise pas la SM vis à vis de la DPA.

Le simulateur arithmétique d'activité est implanté pour les courbes de Weierstrass et de Montgomery respectivement en coordonnées projectives et jacobiennes et en coordonnées

XZ. D'autres types de courbes et types de coordonnées pourraient être ajoutés au simulateur. De plus, une interface graphique permettrait de simplifier l'utilisation du simulateur pour un plus large emploi. Outre les aspects pragmatiques, il pourrait être profitable d'utiliser le simulateur afin d'optimiser et/ou faciliter des attaques sur des traces réelles.

BIBLIOGRAPHIE

[1] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 29–45, 2002.

[2] T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In *Proc. Information Security - ISC*, pages 218–233, Bristol, UK, 2003.

[3] Frédéric Amiel, Benoit Feix, Michael Tunstall, Claire Whelan, and William P. Marnane. Distinguishing Multiplications from Squaring Operations. In *Proc. Selected Areas in Cryptography - SAC*, pages 346–360, Sackville, New Brunswick, Canada, 2008.

[4] F. Bao, R. H. Deng, Y. Han, and more authors. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In *Proc. Security Protocols*, pages 115–124, Paris, France, 1997.

[5] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2) :370–382, February 2006.

[6] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO*, pages 311–323, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[7] Jay M. Berger. A Note on Burst Detecting Sum Codes. *Information and Control*, 4(2-3) :297–299, 1961.

[8] D. J. Bernstein. Curve25519 : New Diffie-Hellman Speed Records. In *Proc. Public Key Cryptography - PKC*, pages 207–228, 2006.

[9] D. J. Bernstein and T. Lange. Explicit-formulas database. `http://hyperelliptic.org/EFD/`.

[10] D. J. Bernstein and T. Lange. Safecurves. `https://safecurves.cr.yp.to/`.

[11] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Proc. Advances in Cryptology - CRYPTO*, pages 131–146, Santa Barbara, California, USA, 2000.

[12] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Proc. Advances in Cryptology*, pages 513–525, Santa Barbara, California, USA, 1997.

[13] J. Blömer and V. Krummel. Fault Based Collision Attacks on AES. In *Proc. Fault Diagnosis and Tolerance in Cryptography - FDTC*, pages 106–120, Yokohama, Japan, 2006.

[14] 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*, pages 36–52, Yokohama, Japan, 2006.

[15] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *Proc. Advances in Cryptology - EUROCRYPT*, pages 37–51, Konstanz, Germany, 1997.

[16] Luca Breveglieri, Israel Koren, Paolo Maistri, and M. Ravasio. Incorporating Error Detection in an RSA Architecture. In *Proc. Fault Diagnosis and Tolerance in Cryptography - FDTC*, pages 71–79, Yokohama, Japan, 2006.

[17] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 16–29, Cambridge, MA, USA, 2004.

[18] Eric Brier and Marc Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In *Proc. Public Key Cryptography - PKC*, pages 335–345, Paris, France, 2002.

[19] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. *Computer Networks*, 48(5) :701–716, 2005.

[20] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 45–68, Taipei, Taiwan, 2017.

[21] G. Canivet. *Analysis of faulted-based attack effects and secure design on a reconfigurable platform*. Theses, Institut National Polytechnique de Grenoble - INPG, 2009. ISBN : 978-2-84813-136-8.

[22] Julien Cathalo, François Koeune, and Jean-Jacques Quisquater. A New Type of Timing Attack : Application to GPS. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 291–303, Cologne, Germany, 2003.

[23] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES*, pages 13–28, Redwood Shores, CA, USA, 2002.

[24] Benoît Chevallier-Mames. Self-Randomized Exponentiation Algorithms. In *Proc. The Cryptographers' Track at the RSA - CT-RSA*, pages 236–249, San Francisco, CA, USA, 2004.

[25] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis : Side-Channel Atomicity. *IEEE Trans. Computers*, 53(6) :760–768, 2004.

[26] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Des. Codes Cryptography*, 36(1) :33–43, 2005.

[27] Mathieu Ciet and Marc Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In *Proc. Information and Communications Security - ICICS*, pages 348–359, Huhehaote, China, 2003.

[28] Christophe Clavier. *De la sécurité physique des crypto-systèmes embarqués*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2007.

[29] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. ROSETTA for Single Trace Analysis. In *Proc. Conference on Cryptology in India - INDOCRYPT*, pages 140–155, Kolkata, India, 2012.

[30] Christophe Clavier, Benoit Feix, and more authors. Horizontal Correlation Analysis on Exponentiation. In *Proc. Information and Communications Security -ICICS*, pages 46–61, Barcelona, Spain, 2010.

[31] Christophe Clavier and Marc Joye. Universal Exponentiation Algorithm. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 300–308, Paris, France, 2001.

[32] H. Cohen and G. Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Maths and Applications. Chapman & Hall/CRC, July 2005.

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

[34] Jean-Sébastien Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Proc. Cryptographic Hardware and Embedded Systems- CHES*, pages 292–302, Worcester, MA, USA, 1999.

[35] R. Crandall. Method and apparatus for public key exchange in a cryptographic system, 1992.

[36] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. Improving the Big Mac Attack on Elliptic Curve Cryptography. In Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater, editors, *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, volume 9100 of *Lecture Notes in Computer Science*, pages 374–386. pringer Berlin Heidelberg, 2016.

[37] Tanja Lange Daniel J. Bernstein and Peter Schwabe. μNaCl library. `https://nacl.cr.yp.to/`.

[38] Nicolas Debande, Youssef Souissi, Maxime Nassar, Sylvain Guilley, Thanh-Ha Le, and Jean-Luc Danger. "Re-synchronization by moments" : An efficient solution to align Side-Channel traces. In *IEEE International Workshop on Information Forensics and Security, WIFS*, pages 1–6, Iguacu Falls, Brazil, 2011.

[39] A. Dehbaoui and All. Injection of transient faults using electromagnetic pulses - Practical results on a cryptographic system-. *IACR Cryptology ePrint Archive*, page 123, 2012.

[40] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A Practical Implementation of the Timing Attack. In *Proc. Smart Card Research and Applications, CARDIS*, pages 167–182, Louvain-la-Neuve, Belgium, 1998.

[41] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, 1976.

[42] V.S. Dimitrov, G.A. Jullien, and W.C. Miller. An algorithm for modular exponentiation. *Information Processing Letters*, 66(3) :155 – 159, 1998.

[43] Hao Dong. Modified Berger Codes for Detection of Unidirectional Errors. *IEEE Trans. Computers*, 33(6) :572–575, 1984.

[44] Harold M. Edwards. A Normal form for elliptic curves. *Bulletin (New Series) of the American Mathematical Society*, 44(3) :393–422, 2007.

[45] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Burlington : Elsevier, 2003.

[46] 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*, pages 198–213, Washington, D.C., USA, 2008.

[47] P-A. Fouque and F. Valette. The Doubling Attack - *Why Upwards Is Better than Downwards*. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 269–280, Cologne, Germany, 2003.

[48] Pierre-Alain Fouque, Frédéric Muller, Guillaume Poupard, and Frédéric Valette. Defeating Countermeasures Based on Randomized BSD Representations. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 312–327, Cambridge, MA, USA, 2004.

[49] J. Francq. *Design and Securization of High-Performance Arithmetic Units for ECC*. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, 2009.

[50] Catherine H. Gebotys and Robert J. Gebotys. Secure Elliptic Curve Implementations : An Analysis of Resistance to Power-Attacks in a DSP Processor. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 114–128, Redwood Shores, CA, USA, 2002.

[51] Daniel Genkin, Adi Shamir, and Eran Tromer. Acoustic Cryptanalysis. *J. Cryptology*, 30(2) :392–443, 2017.

[52] B. Gierlichs, L. Batina, and P. Tuyls. Mutual Information Analysis - A Generic Side-Channel Distinguisher attack. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 426–442, 2008.

[53] Christophe Giraud and Vincent Verneuil. Atomicity Improvement for Elliptic Curve Scalar Multiplication. In *Proc. Smart Card Research and Advanced Application - CARDIS*, pages 80–101, Passau, Germany, 2010.

[54] Robert Giterman, Osnat Keren, and Alexander Fis. A 7T security oriented SRAM bitcell. *IEEE Transactions on Circuits and Systems—II : Express Briefs*, 66(8) :1396–1400, August 2019.

[55] L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Proc. Public Key Cryptography - PKC*, pages 199–210, Miami, FL, USA, 2003.

[56] Sylvain Guilley, Karim Khalfallah, Victor Lomné, and Jean-Luc Danger. Formal Framework for the Evaluation of Waveform Resynchronization Algorithms. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - WISTP*, pages 100–115, Heraklion, Crete, Greece, 2011.

[57] JaeCheol Ha and Sang-Jae Moon. Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 551–563, Redwood Shores, CA, USA, 2002.

[58] R. W. Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 29(2) :147–160, 1950.

[59] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[60] Y. Hayashi, N. Homma, T. Sugawara, and more authors. Non-invasive EMI-based fault injection attack against cryptographic modules. In *Proc. Electromagnetic Compatibility EMC*, pages 763–767, Long Beach, CA, USA, 2011.

[61] N. Homma, A. Miyamoto, T. Aoki, and more authors. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE Trans. Computers*, 59(6) :795–807, 2010.

[62] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Comparative Power Analysis of Modular Exponentiation Algorithms. *IEEE Trans. Computers*, 59(6) :795–807, 2010.

[63] Naofumi Homma, Sei Nagashima, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. High-Resolution Side-Channel Attack Using Phase-Based Waveform Matching. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 187–200, Yokohama, Japan, 2006.

[64] M. Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. *IBM Journal of Research and Development*, 14(4) :395–401, 1970.

[65] M. Hutter and J-M. Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *Proc. Smart Card Research and Advanced Applications*, pages 219–235, Berlin, Germany, 2013.

[66] Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR Microcontrollers. In *Proc. AFRICACRYPT*, pages 156–172, Cairo, Egypt, 2013.

[67] Michael Hutter, Peter Schwabe, Gesine Hinterwälderand Björn Haase, and Ana Helena Sánchez. μNaCl library. `http://munacl.cryptojedi.org/index.shtml`.

[68] Institut Mines-Télécom and TELECOM ParisTech. DPA contest. `http://www.dpacontest.org/tables.php`.

[69] K. Itoh, T. Izu, and M. Takenaka. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 129–143, Redwood Shores, CA, USA, 2002.

[70] Tetsuya Izu, Bodo Möller, and Tsuyoshi Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In *Proc. Progress in Cryptology - INDOCRYPT*, pages 296–313, Hyderabad, India, 2002.

[71] Tetsuya Izu and Tsuyoshi Takagi. Exceptional Procedure Attack on Elliptic Curve Cryptosystems. In *Proc. Public Key Cryptography - PKC*, pages 224–239, Miami, FL, USA, 2003.

[72] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1) :36–63, Aug 2001.

[73] Marc Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 135–147, Vienna, Austria, 2007.

[74] A. Karatsuba and Y. Ofman. MMultiplication of multi-digit numbers on automata (in Russian). *Doklady Akad. Nauk SSSR*, 145(2) :293–294, 1962.

[75] Chris Karlof and David A. Wagner. Hidden Markov Model Cryptanalysis. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 17–34, Cologne, Germany, 2003.

[76] Donald Ervin Knuth. *The art of computer programming, Volume II : Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[77] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177) :203–209, 1987.

[78] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. Advances in Cryptology - CRYPTO*, pages 104–113, 1996.

[79] P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Proc. Advances in Cryptology - CRYPTO*, pages 388–397, Santa Barbara, California, USA, 1999.

[80] P.C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1) :5–27, 2011.

[81] Langer EMV-Technik. Langer EMV-Technik. `https://www.langer-emv.de/en/index`.

[82] Y. Li, K. Ohta, and K. Sakiyama. New Fault-Based Side-Channel Attack Using Fault Sensitivity. *IEEE Trans. on Information Forensics and Security*, 7(1) :88–97, 2012.

[83] Y. Li, K. Sakiyama, S. Gomisawa, and more authors. Fault Sensitivity Analysis. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 320–334, Santa Barbara, CA, USA, 2010.

[84] Pierre-Yvan Liardet and Nigel P. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *Cryptographic Hardware and Embedded Systems - CHES*, pages 391–401, Paris, France, 2001.

[85] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks : Revealing the Secrets of Smart Cards*. Springer, 2007.

[86] T. S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 238–251, Worcester, MA, USA, 2000.

[87] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Proc. Advances in Cryptology - CRYPTO*, pages 417–426, Santa Barbara, California, USA, 1985.

[88] P. L. Montgomery. Speeding the Pollar and Elliptic Curves Methods of Factorisation. *Mathematics of Computation*, 48(177) :243–264, January 1987.

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

[90] Elke De Mulder, Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering*, 33(5-6) :367–382, 2007.

[91] Nicolas Méloni. *Arithmétique pour la Cryptographie bassée sur les Courbes Elliptiques*. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, 2007.

[92] NIST. Fips 186-4. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.

[93] Katsuyuki Okeya and Dong-Guk Han. Side Channel Attack on Ha-Moon's Countermeasure of Randomized Signed Scalar Multiplication. In *Proc. Progress in Cryptology - INDOCRYPT*, pages 334–348, New Delhi, India, 2003.

[94] Katsuyuki Okeya and Kouichi Sakurai. Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack. In *Proc. Progress in Cryptology - INDOCRYPT*, pages 178–190, Calcutta, India, 2000.

[95] Christof Paar. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields. *IEEE Trans. Computers*, 45(7) :856–861, 1996.

[96] G. Piret and J-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 77–88, Cologne, Germany, 2003.

[97] Thomas Plantard. *Arithmétique modulaire pour la cryptographie*. PhD thesis, Montpellier 2 University, France, 2005.

[98] Simon Pontie, Paolo Maistri, and Régis Leveugle. An Elliptic Curve Crypto-Processor Secured by Randomized Windows. In *Proc. Euromicro Conference on Digital System Design - DSD*, pages 535–542, Verona, Italy, 2014.

[99] C. Rechberger and E. Oswald. Practical Template Attacks. In *Proc. Information Security Applications - WISA*, pages 440–456, Jeju Island, Korea, 2004.

[100] Joost Renes, Craig Costello, and Lejla Batina. Complete Addition Formulas for Prime Order Elliptic Curves. In *Proc. Advances in Cryptology - EUROCRYPT*, pages 403–428, Vienna, Austria, 2016.

[101] B. Robisson and P. Manet. Differential Behavioral Analysis. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 413–426, Vienna, Austria, 2007.

[102] T. Roche, V. Lomné, and K. Khalfallah. Combined Fault and Side-Channel Attack on Protected Implementations of AES. In *Proc. Smart Card Research and Advanced Applications -CARDIS*, pages 65–83, 2011.

[103] Hikaru Sakamoto, Yang Li, Kazuo Ohta, and Kazuo Sakiyama. Fault Sensitivity Analysis Against Elliptic Curve Cryptosystems. In *Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC*, pages 11–20, Nara, Japon, 2011.

[104] S. Skorobogatov and R. Anderson. Optical Fault Induction Attacks. In *Cryptographic Hardware and Embedded Systems CHES*, pages 2–12, Redwood Shores, CA, USA, 2002.

[105] J. Solinas. Generalized Mersenne numbers. Technical report, Center for AppliedCryptographic Research, University of Waterloo, ON, Canada, 1999.

[106] Douglas Stebila and Nicolas Thériault. Unified Point Addition Formulæ and Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems - CHES*, pages 354–368, Yokohama, Japan, 2006.

[107] Fred J. Taylor. Large moduli multipliers. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal - ICASSP*, pages 792–795, Denver, Colorado, USA, 1980.

[108] A. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Mathematics-Doklady*, 7 :714–716, 1963.

[109] Elena Trichina and Antonio Bellezza. Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks. In *Proc. Cryptographic*

Hardware and Embedded Systems - CHES, pages 98–113, Redwood Shores, CA, USA, 2002.

[110] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving Differential Power Analysis by Elastic Alignment. In *Proc. Topics in Cryptology - CT-RSA - The Cryptographers' Track at the RSA*, pages 104–119, San Francisco, CA, USA, 2011.

[111] I. Verbauwhede, D. Karaklajic, and J-M. Schmidt. The Fault Attack Jungle - A Classification Model to Guide You. In *Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 3–8, Tokyo, Japan, 2011.

[112] Colin D. Walter. Sliding Windows Succumbs to Big Mac Attack. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 286–299, Paris, France, 2001.

[113] Colin D. Walter. Breaking the Liardet-Smart Randomized Exponentiation Algorithm. In *Proc. Smart Card Research and Advanced Application Conference, CARDIS*, pages 59–68, San Jose, CA, USA, 2002.

[114] Colin D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In *Proc. Cryptographic Hardware and Embedded Systems - CHES*, pages 191–204, Cambridge, MA, USA, 2004.

[115] S-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9) :967–970, 2000.

[116] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In *Proc. Information Security and Cryptology - ICISC*, pages 414–427, Seoul, Korea, 2001.

[117] Sung-Ming Yen, Lee-Chun Ko, Sang-Jae Moon, and JaeCheol Ha. Relative doubling attack against montgomery ladder. In *Proc. Information Security and Cryptology - ICISC*, pages 117–128, Seoul, Korea, 2005.

[118] E Zeckendorf. Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bull. Soc. Roy. Sci. Liège*, 41 :179–182, 1972.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Support logiciel robuste aux attaques passives et actives pour l'arithmétique de la cryptographie asymétrique sur des (très) petits cœurs de calcul.

Mot clés : Cryptographie basée sur les courbes elliptiques ; attaque par observation ; attaque par perturbation ; protections

Résumé :

Cette thèse porte sur le développement et l'évaluation de protections contrant simultanément des attaques par perturbation (FA) et des attaques par observation (SCA) dans le contexte de la cryptographie basée sur les courbes elliptiques (ECC). Deux protections ont été proposées pour la multiplication scalaire (SM), l'opération principale d'ECC. La première, nommée vérification de point (PV), permet une uniformisation de la SM grâce à une vérification de l'appartenance du point courant à la courbe. La SM ainsi obtenue est uniforme et donc résistante aux SPA mais aussi résistante à certaines FA. La seconde, nommée compteur d'itérations (IC), protège le scalaire contre certaines FA, tout en ayant un comportement uniforme et avec un très faible surcoût.

Ces deux protections ont été implantées sur un microcontrôleur Cortex M0 pour les courbes de Weierstrass et de Montgomery, et ce pour différents types de coordonnées. Le surcoût de ces méthodes varie entre 48 % et 62 % dans le pire des cas (lorsque la PV est réalisée à chaque itération de la SM). Cela est moindre que celui des protections de bases habituelles contre les SCA.

Un simulateur d'activité théorique au niveau arithmétique est également proposé. Il reproduit l'architecture d'un microcontrôleur 32 bits très simple. L'activité théorique est modélisée grâce à la variation du poids de Hamming des données manipulées lors de l'exécution. Grâce à ce simulateur, l'impact des opérandes sur l'activité des unités arithmétiques a pu être illustré. De plus, des attaques SPA et DPA furent réalisées pour évaluer les protections précédentes. Nos protections montrent une amélioration de la sécurité.

Titre : Robust software support against passive and active attacks for the arithmetic of asymmetric cryptography on (very) small computing cores.

Keywords: Elliptic curve cryptography ; Side chanel attack ; Fault attack ; Protections

Abstract: This thesis deals with protection development and evaluation against fault attacks (FA) and side channel attacks (SCA) simultaneously. These protections have been developed for elliptic curves cryptography (ECC) and its main operation, the scalar multiplication (MS).

Two protections have been proposed. The first is point verification (PV) checking that the current point is effectively on the curve, with a uniformization behavior. Thus, this new SM with PV is robust against some FAs and also SPA, since it is uniform. The second one is called counter iteration (IC). ICC protects the scalar against major FAs with a uniform behavior. Its overhead is very small.

Our protections have been implemented on Cortex M0 microcontroller for Weierstrass and Montgomery curves and for different types of coordinates. The overhead is between 48 % and 62 %, in the worst case (when the PV is made at each SM iteration). This overhead is smaller than overhead of usual basic protections against SPA.

A theorical activity simulator has also been developed. It reproduces the architecture of a simple 32-bit microcontroller. Theoric activity is modelled by the Hamming weigh variations of manipulated data during execution. Thanks to the simulator, the impact of operands is illustrated for arithmetic units. Moreover, SPA and DPA attacks were made for evaluating our protections. Our protections show some security improvements.