Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de
DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES
Spécialité : **électronique, électrotechnique, automatique et traitement du signal**
Arrêté ministériel : 7 août 2006

Présentée par
Simon Pontié

Thèse dirigée par **Régis Leveugle et** co-encadrée par **Paolo Maïstri**

préparée au sein du **Laboratoire TIMA** dans **l'École Doctorale EEATS**

Sécurisation matérielle pour la cryptographie à base de courbes elliptiques

Thèse soutenue publiquement le 21 novembre 2016, devant le jury composé de :

M. Viktor Fischer
Professeur des universités, Président
M. Arnaud Tisserand
Directeur de recherche CNRS, Rapporteur
M. Lionel Torres
Professeur, Rapporteur
M. Philippe Elbaz-Vincent
Professeur, Examinateur
M. Pierre-Yvan Liardet
Docteur, Examinateur
M. Régis Leveugle
Professeur, Directeur
M. Paolo Maistri
Chargé de recherche CNRS, Co-encadrant

Remerciements

Ce manuscrit est le résultat d'un travail de thèse que j'ai effectué au sein du laboratoire TIMA dans l'équipe AMfoRS. J'énumère ici un certain nombre des personnes envers lesquelles je suis redevable de m'avoir accompagné durant toute cette période.

Mes premiers remerciements vont à mes encadrants Régis Leveugle et Paolo Maïstri. Leurs conseils m'ont permis d'éviter de nombreux écueils. Je tiens particulièrement à les remercier pour la confiance qu'ils m'ont accordée, en stage de master comme en thèse, et pour la disponibilité sans faille qu'ils m'ont accordée.

Par ailleurs, je remercie chaleureusement Arnaud Tisserand, Liones Torres, Viktor Fischer, Pierre Yvan Liardet et Philippe Elbaz-Vincent qui m'ont fait l'honneur de faire partie du jury. Je souhaite en particulier exprimer toute ma gratitude à Arnaud Tisserand et Liones Torres qui ont rapporté cette thèse. Je les remercie vivement pour l'attention qu'ils y ont portée.

Une partie de mes travaux n'aurait pas été possible sans la collaboration de l'institut Fourier (Grenoble). Je suis reconnaissant pour cela à Philippe Elbaz-Vincent et à Marie-Angela Cornelie. Ce travail de thèse a benificié d'un financement partiel du LabEx PERSYVAL-Lab (ANR-11-LABX-0025).

J'ai pu élargir le spectre de mes recherches grâce à une fructueuse collaboration avec l'équipe SLS du laboratoire TIMA. Je suis reconnaissant envers toutes les personnes ayant participé à ces travaux: Alban Bourge, Adrien Prost-Boucle, Frédéric Rousseau et Olivier Muller.

Mon travail n'aurait pu être mené à bien sans la disponibilité et l'accueil chaleureux que m'ont témoignés Robin Rolland-Girod et tout le personnel du CIME nanotech de Grenoble.

Je veux remercier les enseignants de l'ENS Cachan antenne de Bretagne qui m'ont donné le gout de la recherche et tous mes collègues enseignants de l'école PHELMA, en particulier mon tuteur d'enseignement Mounir Benabdenbi.

J'adresse également mes remerciements à tous mes collègues du TIMA, administratifs, personnels du service informatique, doctorants, enseignants et chercheurs avec qui j'ai partagé les problèmes et les plaisirs quotidiens. Je m'adresse plus chaleureusement à certains d'entre eux qui ont marqué mes années au TIMA: Alban Bourge, Amine Chahed, Asma Mkhini, Kaïs Chibani, Salma et Mohammed Ben Jrad, Pierre Vanhauwaert, Jean-François Coulon.

Au terme de ce parcours, je remercie enfin celle qui m'est chère, Charlotte Perrin qui m'a soutenu et épaulé tout au long de ma thèse.

Table des matières

Introduction générale

La multiplication des objets connectés pousse les ingénieurs d'aujourd'hui à résoudre de nouveaux défis. Tout d'abord il est apparu un critère fort d'autonomie qui ne peut être atteint qu'avec un objectif de consommation énergétique faible dès le début du développement de tels produits. La notion de sécurité dans ces objets est aussi primordiale: ils doivent être capables d'assurer la confidentialité des données manipulées et ne répondre qu'à des ordres ayant été autorisés après une phase d'authentification.

Cette thèse s'inscrit dans ce contexte en proposant des solutions cryptographiques matérielles sécurisées. Je me focalise sur la cryptographie asymétrique, aussi nommée cryptographie à clé publique. Les accélérateurs matériels étudiés dans ce document utilisent la cryptographie basée sur les courbes elliptiques.

La sécurité ne peut être évaluée qu'après avoir fixé un modèle d'attaquant. Je considère dans ces travaux des attaquants avec accès physique au système cryptographique. Cela signifie que j'évalue la fuite d'informations sensibles par canaux auxiliaires même si leur collecte nécessite un accès physique. Je m'intéresse plus particulièrement aux fuites présentes dans les profils de puissance consommée. J'utilise des attaques réelles par analyse de puissance consommée pour évaluer la sécurité des protections proposées et démontrer la faisabilité de nouvelles attaques. Je me concentre sur des contre-mesures qui n'impactent pas ou peu les performances du système protégé.

Je commence dans le Chapitre I par aborder les outils mathématiques sous-jacents à la cryptographie basée sur les courbes elliptiques. Cette étude est orientée vers la conception matérielle, chaque concept étant illustré par des solutions d'électronique numérique permettant son implantation. Dans un deuxième temps, je décris les attaques par canaux auxiliaires existantes et leurs contre-mesures.

Je serai alors en mesure d'expliquer dans le Chapitre II les choix algorithmiques et architecturaux que j'ai faits pour concevoir et implanter plusieurs accélérateurs matériels cryptographiques. Ces accélérateurs peuvent supporter toutes les courbes de Weierstrass réduites et sont utilisés comme base des expérimentations réalisées dans les chapitres suivants. J'ai aussi étudié les capacités offertes par la synthèse de haut niveau appliquée aux courbes elliptiques. Cette étude sort un peu du cadre de cette thèse car les attaques avec accès physique n'ont pas été étudiées. Elle est présentée dans l'Annexe 3.

J’étudie la sécurité qu'apporte l'utilisation de formules unifiées dans le Chapitre III. La plupart des implémentations cryptographiques sur les courbes elliptiques sont basées sur des courbes de Weierstrass. Un certain nombre de ces courbes ont une courbe quartique de Jacobi équivalente. J’ai développé des accélérateurs matériels basés sur cette forme de courbe car elle permet de mettre en œuvre une protection par formule unifiée de manière efficace. Je montre expérimentalement qu'une nouvelle attaque non supervisée peut être réalisée, et que cette seule contre-mesure ne suffit donc pas à protéger un circuit. Mon attaque est basée sur la détection de la réutilisation d’un point en identifiant des ressemblances à l’intérieur de traces de puissance consommée. Cette attaque met en œuvre des synchronisations et des mesures de similarité par corrélation couplées à un algorithme de classification hiérarchique pour reconstruire la chaine d’opérations basiques sur les points et identifier le secret utilisé.

Dans le Chapitre IV je propose une contre-mesure alternative à un niveau hiérarchique supérieur et pouvant donc être utilisée sur n’importe quelle courbe elliptique. Il s’agit d’une utilisation astucieuse d'opérations fictives couplée avec une multiplication scalaire par fenêtrage aléatoire permettant la désynchronisation des attaques par analyse de puissance consommée. Le fenêtrage permet d’améliorer les performances du crypto-processeur en utilisant des points pré-calculés. Il est aussi une première source d’aléa. La seconde source d’aléa est une latence variable entre des opérations fictives introduites dans le calcul et la partie du secret manipulé à laquelle elles sont liées. Cela permet de minimiser la fuite d’information collectée par un attaquant même si celui-ci est capable de détecter une opération fictive par injection de faute. J’ai développé des attaques spécifiques pour évaluer le niveau de sécurité apporté par cette contre-mesure.

Chapitre I État de l'art

Ce chapitre dresse un portrait de l'état de l'art des implémentations cryptographiques matérielles basées sur les courbes elliptiques. L'objectif est double : analyser l'étendue des choix technologiques/algorithmiques offerts à un concepteur de circuits numériques, mais aussi comparer nos circuits, décrits dans les chapitres suivants, à ces implémentations de référence. Je rappelle que la finalité de cette thèse est de prendre en compte les fuites d'informations par canaux auxiliaires lors de la conception d'un circuit numérique. C'est pourquoi un état de l'art de ces attaques et des contre-mesures existantes est aussi dressé. Avant de me pencher sur ces aspects matériels, je commence par aborder les concepts sous-jacents à la cryptographie basée sur les courbes elliptiques. Je m'efforce de garder un point de vue concepteur de circuit numérique en illustrant ces concepts par leurs implémentations matérielles. Pour une approche plus mathématique sur les courbes elliptiques en cryptographie je renvoie le lecteur vers l'ouvrage [CoFr05].

1. Cryptographie asymétrique et applications

Il existe plusieurs solutions permettant d'assurer la confidentialité d'un message numérique. La première famille de solutions utilise la même clé pour chiffrer et déchiffrer un message, c'est la cryptographie symétrique. L'algorithme le plus utilisé est le chiffrement par bloc AES. La seconde famille permet de chiffrer et déchiffrer avec des clés différentes. Ce sont les systèmes à clé publique basés sur la cryptographie asymétrique. L'objet numérique que l'on nomme clé publique permet de chiffrer un message tandis que la clé privée associée permet de le déchiffrer. Le principal avantage de ces systèmes est la simplification de l'infrastructure de gestion des clés. La cryptographie asymétrique permet d'assurer la confidentialité des messages mais elle peut aussi servir à authentifier un message par signature numérique.

Les principaux algorithmes permettant la mise en œuvre de cryptographie asymétrique sont basés sur le problème de factorisation ou sur le problème du logarithme discret. Par exemple, le RSA est basé sur le problème de factorisation d'un entier en deux grands nombres premiers. Les protocoles de signature numérique (DSA/ECDSA) [Vans92] et l'algorithme d'échange de secret partagé Diffie-Hellman (DH/ECDH) [DiHe76; Mill86] reposent quant à eux sur le

problème du logarithme discret. Ce problème s'applique usuellement sur deux objets mathématiques différents : le groupe multiplicatif du corps fini d'ordre premier $GF(p)$ ou le groupe formé par les points d'une courbe elliptique. Dans cette thèse, j'utilise plus particulièrement la cryptographie basée sur les courbes elliptiques (ECC). Son principal avantage est que l'on ne connait pas d'algorithme sous-exponentiel pour résoudre le problème du logarithme discret sur ce groupe lorsque la courbe est bien choisie (je ne parlerai pas ici de cryptographie post-quantique). Ce n'est pas le cas pour le problème du logarithme discret sur le groupe $GF(p)$ ou pour le problème de factorisation dans le système RSA pour lesquels il existe des algorithmes sous-exponentiels (mais non polynomiaux). La principale conséquence est que pour le système ECC des clés plus petites permettent d'atteindre un niveau de sécurité donné (taille proportionnelle d'un facteur 2 à la taille d'une clé symétrique) [ANSS14]. Ceci est d'autant plus intéressant que le niveau de sécurité est élevé. L'utilisation de la cryptographie basée sur les courbes elliptiques a aussi d'autres différences majeures que j'identifie dans le paragraphe suivant.

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

Les opérations de chiffrement sont généralement réalisées à l'aide de cryptographie symétrique, plus rapide que la cryptographie asymétrique. Une clé secrète éphémère commune entre les deux interlocuteurs peut être obtenue par l'utilisation de la cryptographie asymétrique (voir section I-2-E page 23). La première solution de mise en œuvre est un interlocuteur chiffrant une clé secrète choisie au hasard avec la clé publique du second interlocuteur. Le second peut alors déchiffrer la clé secrète avec sa clé privée pour commencer la communication sécurisée. L'inconvénient est qu'un attaquant peut enregistrer l'échange et attendre des progrès significatifs en cryptanalyse pour casser la clé publique et retrouver ainsi le secret ayant servi à chiffrer la communication. Il sera dès lors capable de déchiffrer toute la communication. La seconde solution permet de résoudre ce problème en utilisant des clés éphémères. La mise en œuvre classique est l'échange de secret partagé Diffie-Hellman. Si les deux interlocuteurs sont en ligne ils peuvent utiliser des clés éphémères et cette attaque sera extrêmement ralentie car il faudra casser autant de clés que de conversations à déchiffrer : c'est ce que l'on appelle la propriété de confidentialité persistante. Même si un attaquant découvre dans le futur la clé privée d'un interlocuteur il sera incapable d'en déduire le secret partagé. Malheureusement, le temps de génération d'une paire de clés RSA est incompatible

avec les contraintes de latence généralement requises. En pratique la propriété de confidentialité persistante n'est utilisable que pour les systèmes basés sur le logarithme discret tels que ECC car la génération de clés est extrêmement rapide. L'utilisation du protocole Diffie-Hellman ne permet toutefois pas toujours de satisfaire la propriété de confidentialité persistante. Il est possible de chiffrer un message à l'aide de ce protocole (ex: PGP [Jivs12]) mais le destinataire de ce message peut être hors-ligne lorsque le message est émis. Dans cette situation de communication asynchrone l'émetteur du message ne peut pas utiliser de secret éphémère provenant du destinataire et doit utiliser la clé publique du destinataire pour forger le secret. La confidentialité persistante n'est donc plus assurée car le secret doit pouvoir être déchiffré par la clé privée du destinataire. Ce service ne peut donc pas bénéficier de la confidentialité persistante. La messagerie instantanée peut par contre bénéficier de cette propriété ; c'est le cas par exemple du protocole Off-the-Record Messaging (OTR) [DiGe05]. Une solution alternative (OTR modifié) permet de s'affranchir des limitations des communications asynchrones. Plusieurs applications d'échange de messages sur réseaux mobiles utilisent cette technique : c'est le cas de l'application Whatsapp [What16].

La dernière différence majeure qui caractérise ces systèmes cryptographiques est le coût énergétique de ceux-ci. La consommation énergétique des opérations de signature/vérification et d'échange de clés utilisant le RSA n'est pas équitablement répartie entre les deux interlocuteurs. En effet la vérification de signature est une opération très rapide à effectuer en comparaison de la signature [WaGu05; GuPa04]. Ceci n'est pas le cas pour ECC où le coût énergétique de ces opérations est sensiblement le même. Pour des niveaux de sécurité supérieurs à 80 bits de sécurité, les courbes elliptiques sont aussi moins consommatrices d'énergie que le RSA.

2. Cryptographie basée sur les courbes elliptiques

La cryptographie basée sur les courbes elliptiques fait appel à différentes opérations. La Figure I-1 montre que ces opérations peuvent être hiérarchisées et met en avant les liens de dépendance. Les protocoles cryptographiques sont basés sur l'exponentiation. Dans la suite, je nomme cette opération multiplication scalaire. Elle fait appel aux opérations de doublage et d'addition de points. Ces deux opérations basiques sur les points sont des calculs sur leurs coordonnées, qui sont des éléments d'un corps.

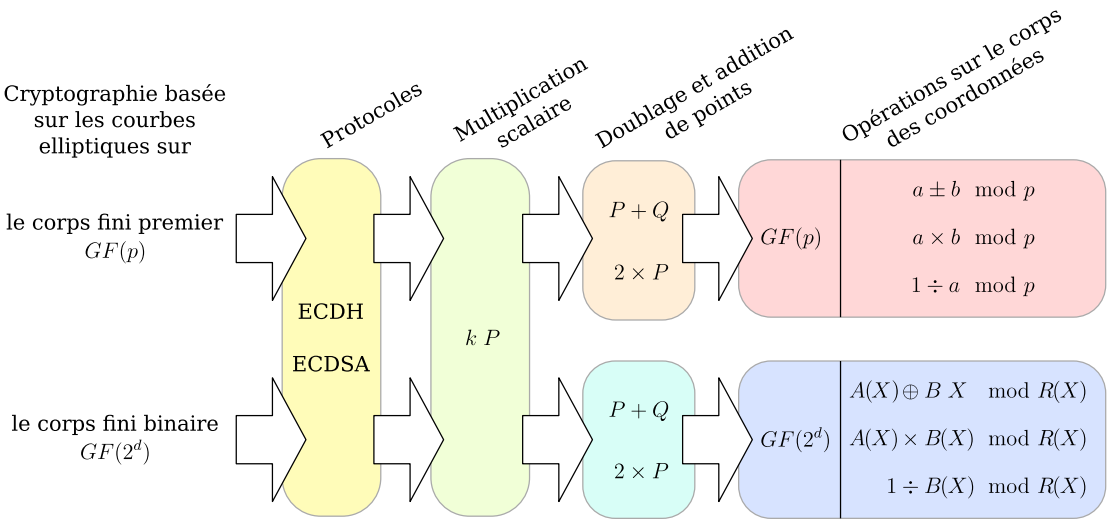


Figure I-1 Hiérarchie des opérations sur les courbes elliptiques en cryptographie

2-A. Représentation affine de courbes elliptiques en cryptographie

L'équation générale d'une courbe elliptique en représentation affine est donnée par l'équation de Weierstrass appliquée à un corps donné :

$$E_{générale}:\ y^2 + a_1 \cdot x \cdot y + a_3 \cdot y = x^3 + a_2 \cdot x^2 + a_4 \cdot x + a_6$$

Le logarithme discret est en général un problème compliqué sur une courbe elliptique, c'est-à-dire que le meilleur algorithme connu est exponentiel en la taille de p. Il existe des courbes où ce problème peut devenir sous-exponentiel. Le choix d'une courbe pour la cryptographie est donc un problème mathématique complexe car il faut éviter ces courbes faibles. Je ne traiterai pas ce problème. Pour favoriser l'interopérabilité entre les équipements, la majorité des courbes utilisées proviennent de standards ou de recommandations. Cela peut représenter un domaine de courbes elliptiques ou des jeux de paramètres dans ce domaine.

Les deux corps majoritairement utilisés sont le corps fini d'ordre premier $GF(p)$ (avec p un nombre premier plus grand que 3) et le corps binaire $GF(2^d)$. Sur $GF(p)$ et quand a_4 et a_6 sont non nuls, la courbe elliptique est isomorphe à une courbe avec une équation de Weierstrass réduite de la forme :

$$E\big(GF(p)\big):\ y^2 = x^3 + a_4 \cdot x + a_6 \mod p \qquad Si\ p \neq 2{,}3$$

De la même manière, toutes les courbes elliptiques ordinaires sur $GF(2^d)$ sont isomorphes à une courbe avec une équation de Weierstrass réduite de la forme :

$$E\left(GF(2^d)\right):\ y^2 + x \cdot y = x^3 + a_2 \cdot x^2 + a_6 \mod R(X)$$

Il existe donc un nombre fini de points à coordonnées dans le corps associé sur chaque courbe. Ceci est évident pour les courbes sur des corps finis. Le nombre de points sur $E\left(GF(p)\right)$ est proche de p (théorème de Hasse). Les courbes avec un nombre de points exactement égal à p sont évitées car c'est un cas où le logarithme discret peut être accéléré. On choisit en général un nombre de points premier. Cette hypothèse peut être relâchée avec un nombre de points non premier mais étant le produit entre un grand nombre premier et un petit cofacteur (par exemple 4). Ceci permet de travailler sur des courbes alternatives permettant des optimisations ou la mise en œuvre de contre-mesures. L'identification exacte du nombre de points à coordonnées dans $GF(p)$ ou $GF(2^d)$ sur une courbe elliptique est à nouveau un problème mathématique complexe. En cryptographie on désigne donc une courbe elliptique par son équation, le corps dans lequel elle est appliquée et un point générateur. On précise aussi l'ordre du point générateur et le cofacteur de la courbe. Le nombre de points sur la courbe est le produit de ces deux grandeurs (tous les points ne sont pas forcément dans la famille générée par le point générateur).

Les tailles classiques de p sont de 160 bits à 512 bits. Dans la suite, j'illustre les notions de base sur une courbe elliptique sur $GF(37)$ soit un p sur 6 bits qui est bien trop petit pour être utilisé en cryptographie mais facilitera la représentation de certains concepts.

2-B. Opérations de base sur les points

L'approche mathématique classique pour introduire l'utilisation des courbes elliptiques en cryptographie est d'énoncer les concepts communs à ces courbes sur différents corps puis de préciser ces concepts sur $GF(p)$ avec $p > 3$ et $GF(2^d)$ mais aussi des corps de caractéristique 3. Seules les courbes elliptiques sur $GF(p)$ et $GF(2^d)$ seront considérées dans cette thèse car ce sont les plus utilisées. On considérera qu'à chaque fois que l'on note $GF(p)$, il est supposé implicitement que p est un nombre premier supérieur à 3. Je commence par étudier le cas du corps premier $GF(p)$ car cela me permettra d'illustrer les concepts sur le corps des réels (de manière plus précise, sur le corps des rationnels). Dans un second temps j'explique les différences entre les courbes elliptiques sur $GF(2^d)$ et $GF(p)$.

Soit la courbe elliptique suivante :

$$E_{(-12,20,37)}: y^2 = x^2 - 12 \cdot x + 20 \bmod 37$$

Les points de cette courbe ont des coordonnées x et y dans $GF(37)$, c'est-à-dire des entiers entre 0 et 36. Pour mieux comprendre la cryptographie sur les courbes elliptiques je vais tout d'abord m'intéresser à cette même équation exprimée sur les rationnels. La courbe en noir (a) de la Figure I-2 représente une partie de l'infinité de points à coordonnées rationnelles de l'équation :

$$E_{(-12,20)}: y^2 = x^3 - 12 \cdot x + 20$$

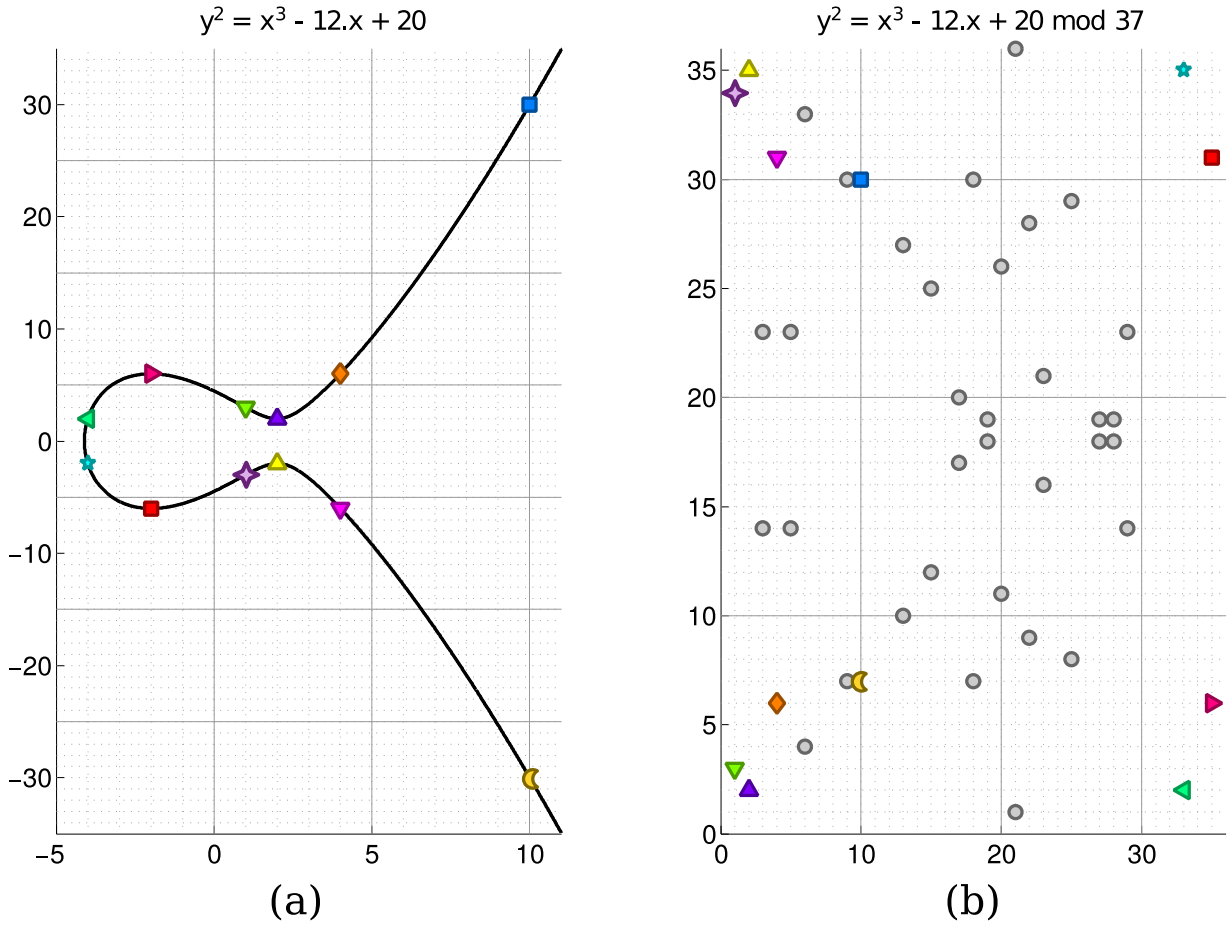


Figure I-2 Courbe de $E_{(-12,20)}$ à gauche et l'ensemble des points vérifiant $E_{(-12,20,37)}$ à droite

On peut observer sur cet extrait de courbe 12 points mis en évidence car ils ont des coordonnées entières. Par exemple, le point $(x = 10, y = -30)$ est sur la courbe car il

vérifie bien l'équation $E_{(-12,20)}$. Ces points sont aussi solutions de l'équation sur $GF(37)$: $E_{(-12,20,37)}$. On identifiera les points solutions de cette équation par leurs coordonnées entières entre 0 et 36. Le point ☾ devient donc $(x = 10, y = 7)$ dans $GF(37)^2$, en ayant ajouté 37 à la coordonnée y pour obtenir les coordonnées canoniques dans l'intervalle spécifié. L'ensemble des points solutions de $E_{(-12,20,37)}$ sont représentés dans la partie (b) de la Figure I-2. Cette courbe elliptique discrète est représentée par un nombre fini de points, ici 46 points.

Par définition, il existe une loi d'addition entre deux points sur une courbe elliptique. Le résultat de cette addition sera un point de la courbe. Cette loi peut-être illustrée par une construction géométrique. Les deux droites en pointillés sur la Figure I-3 décrivent l'addition de deux points distincts avec des abscisses différentes : ■ $(-2,-6)$ et ▼ $(1,3)$. Tout d'abord la droite passant par ces deux points opérandes est tracée. Il n'existe qu'un seul autre point étant à l'intersection de cette droite et de la courbe elliptique : ■ $(10,30)$. Le point symétrique par rapport à l'axe x de cette intersection est le point résultat de l'addition : ☾ $(10,-30)$.

Pour le cas particulier d'addition de deux points identiques, on définit donc une construction particulière pour définir cette opération qui est le doublage d'un point. Les deux lignes pleines sur la Figure I-3 décrivent la construction du doublage du point ♦ $(4,6)$. Elle est identique à l'addition de points sauf qu'elle utilise la droite tangente à la courbe au point opérande à doubler. On peut observer que l'on converge vers cette construction pour l'addition de deux points très proches. Le point d'intersection ✦ $(1,-3)$ est le symétrique par rapport à l'axe x du point résultat du doublage: ▼ $(1,3)$.

Il existe un second cas particulier : l'addition de deux points distincts qui ont la même abscisse x_1. Si le premier point a pour ordonnée y_1 alors le second ne peut avoir pour ordonnée que $-y_1$. Ces deux points sont les seuls points de la courbe à avoir cette abscisse. Il sera donc impossible de les additionner avec la construction géométrique classique car il n'existe pas de troisième point d'intersection entre la courbe elliptique et la droite verticale d'abscisse x_1. Le résultat de cette addition est un point à l'infini qui ne peut pas être représenté sur une courbe elliptique affine : P_∞. Il s'agit de l'élément neutre de la loi d'addition de point. Cela signifie que le point (x_1, y_1) est l'opposé du point $(x_1, -y_1)$. Sur la Figure I-3 on pourra par exemple identifier le point ■ $(10,30)$ qui est l'opposé du point ☾ $(10,-30)$. On a donc $-($■ $+$ ▼$) =$ ■.

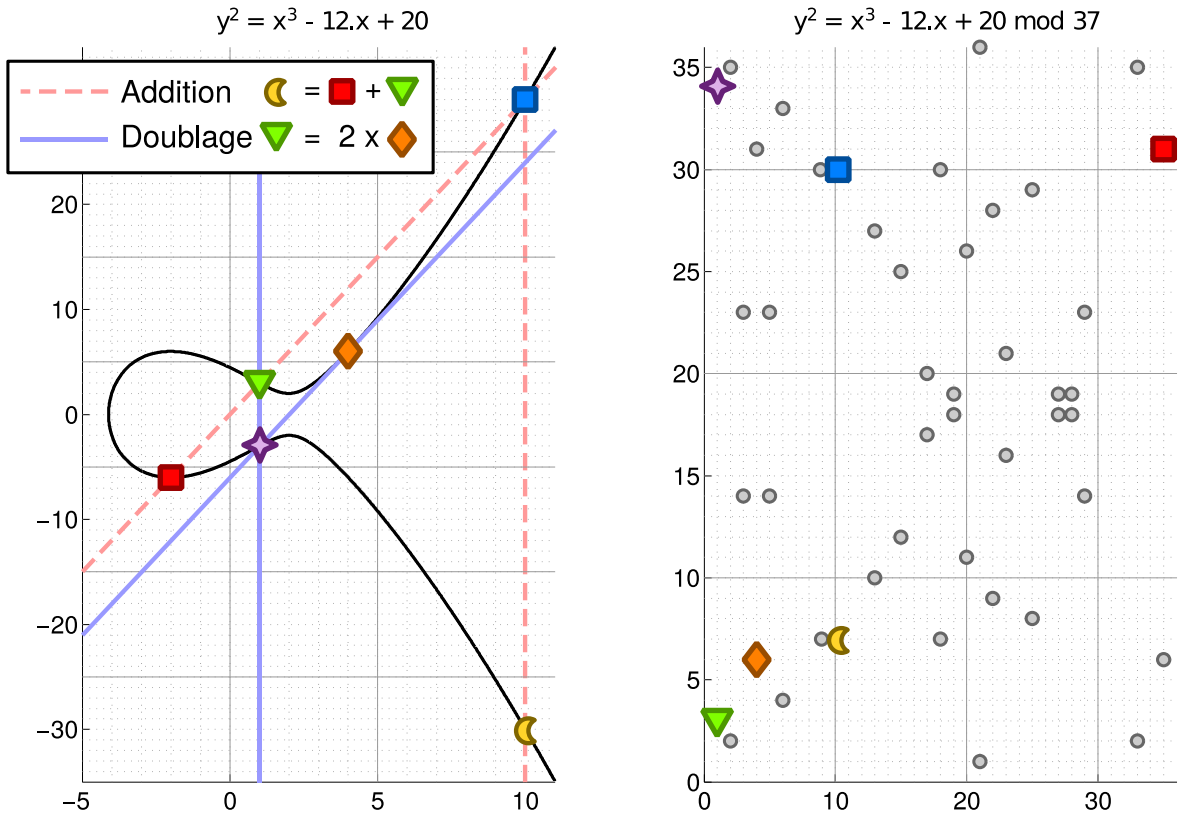


Figure I-3 Opérations d'addition et de doublage de point

En pratique ces deux opérations basiques sur les points (addition et doublage) sont réalisées à l'aide de formules :

$$R(x_3, y_3) = P(x_1, y_1) + Q(x_2, y_2) \text{ avec } Q \neq P \text{ et } Q \neq -P \text{ et } P \neq P_\infty \text{ et } Q \neq P_\infty$$

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2}$$

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

$$R(x_3, y_3) = 2 \times P(x_1, y_1) \text{ avec } Q \neq P_\infty$$

$$\lambda = \frac{3 \cdot x_1^2 + a_4}{2 \cdot y_1}$$

$$x_3 = \lambda^2 - 2 \cdot x_1$$

$$y_3 = \lambda \cdot (x_1 - x_3) - y_1$$

Un exemple d'ordonnancement est proposé dans la Figure I-4 pour l'addition et dans la Figure I-5 pour le doublage.

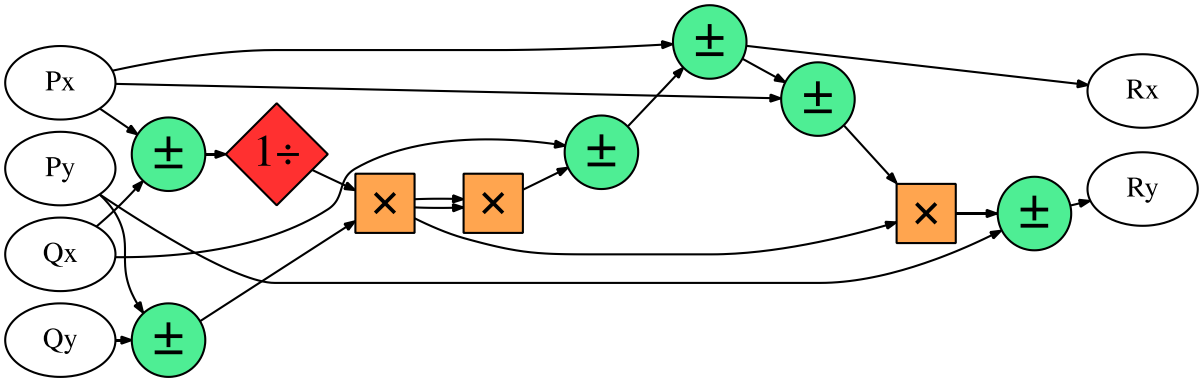


Figure I-4 Exemple d'ordonnancement pour l'addition deux points

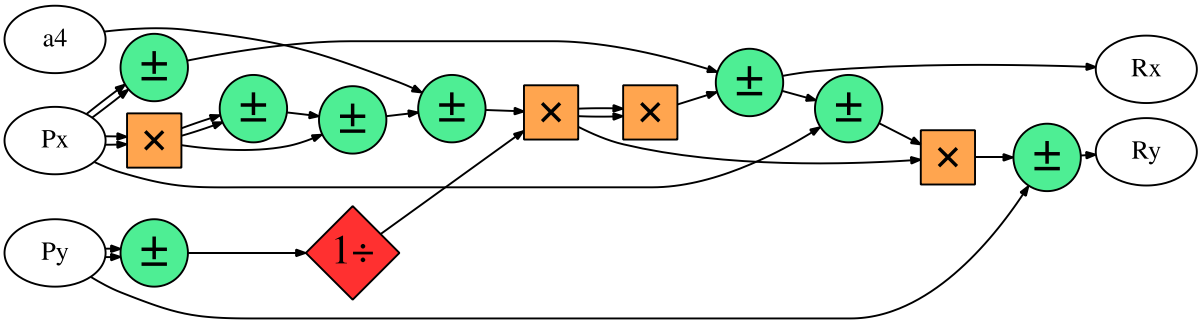


Figure I-5 Exemple d'ordonnancement pour le doublage d'un point

Tout d'abord nous pouvons remarquer que l'opération d'addition de points n'est pas une simple addition des coordonnées mais une opération plus complexe faisant intervenir plusieurs opérations classiques sur les entiers modulos un nombre premier : une inversion; des multiplications et des additions. On remarquera également que, comme pour les constructions géométriques, la formule permettant de calculer les coordonnées du point résultant d'une addition ne fonctionne que si les deux opérandes (points sur la courbe elliptique) sont distincts et ne sont pas l'opposé l'un de l'autre. Si les deux points opérandes sont égaux, il s'agit d'un doublage et la formule utilisée est différente. Ces formules ne dépendent pas du paramètre b de la courbe. En réalité, ce paramètre est utilisé uniquement pour s'assurer qu'un point est ou non sur la courbe en vérifiant si ses coordonnées sont solutions de l'équation de la courbe elliptique.

Nous pouvons vérifier que $(10,-30) = (-2,-6) + (1,3)$.

$$R(10,-30) = P(-2,-6) + Q(1,3)$$

$$\lambda = \frac{-6-3}{-2-1} = 3$$

$$x_3 = 3^2 + 2 - 1 = 10$$

$$y_3 = 3 \cdot (-2-10) + 6 = -30$$

L'analogie me permettant d'illustrer ces opérations sur le corps des entiers au lieu de $GF(p)$ a ses limites. Par exemple, elle se complique si nous tentons d'additionner les points $(1,3)$ et $(10,-30)$ alors $\lambda = \frac{3+30}{1-10} = \frac{-11}{3}$ et $x_3 = \frac{11^2}{3^2} - 11 \simeq -2{,}44$. Les coordonnées du point résultant ne sont pas entières mais rationnelles. Les formules présentées sont en réalité résolues dans le corps $GF(p)$. C'est-à-dire que toutes les additions, soustractions, multiplications et inversions sont réalisées modulo un nombre premier p. Par exemple λ devient 21 qui est congru à $\frac{-11}{3}$ modulo 37 car $21 \times 3 - 2 \times 37 = -11$. Le point résultant a donc les coordonnées suivantes dans $GF(37)^2$: $(23,16)$.

2-C. Nombre de points sur une courbe elliptique

Je vais maintenant m'intéresser au nombre de points d'une courbe elliptique sur $GF(p)$. Il existe une méthode naïve pour observer que le nombre de points d'une courbe elliptique a une borne maximale. Pour un x donné il existe au maximum 2 points avec des ordonnées opposées solutions de l'équation $E_{GF(p)}$. Nous pouvons donc affirmer qu'il y aura $2p + 1$ points au maximum sur la courbe. On connait depuis 1933 des bornes plus précises pour le nombre de points sur une courbe elliptique grâce au théorème de Hasse. On notera $\#E_{a_4,a_6,p}$ le nombre de points d'une courbe elliptique, aussi nommé ordre de la courbe elliptique. Le théorème de Hasse assure que :

$$p + 1 - 2\sqrt{p} \leq \#E_{a_4,a_6,p} \leq p + 1 + 2\sqrt{p} \qquad \#E_{a_4,a_6,p} = \text{Nombre de points}$$

L'ordre de la courbe $E_{-12,20,37}$ est donc compris entre 26 et 50 points. L'ordre exact de cette courbe est de 47 points : les 46 représentés dans la Figure I-2 et le point à l'infini P_∞. Il faut connaitre l'ordre d'une courbe elliptique pour deux raisons. Tout d'abord ce nombre intervient dans les protocoles cryptographiques et dans la procédure de génération de clé secrète. De plus, pour des nombres de points particuliers, la complexité du problème du logarithme discret peut être réduite (exemple : $\#E_{a_4,a_6,p} = p$); ces courbes doivent donc être évitées pour sélectionner uniquement des courbes plus sûres. Dans l'exemple de la Figure I-2, le nombre de points a été trouvé par force brute en vérifiant pour chacun des p^2 points s'ils étaient solutions de l'équation de la courbe elliptique. Cette opération n'est pas réalisable sur une courbe de taille réelle. Il existe des algorithmes permettant le comptage de points sur une courbe elliptique. L'estimation faite grâce au théorème de Hasse permet de réduire le champ des recherches. Cette opération reste couteuse et rend difficile la génération « à chaud » de courbes elliptiques (même si cela n'est pas impossible).

Le comptage de points est une étape indispensable dans la génération de courbe elliptique sûre. Mais ce n'est pas la seule étape : il faut aussi vérifier que la courbe générée n'est pas une courbe spéciale pour laquelle on connait un algorithme au plus sous-exponentiel pour résoudre le logarithme discret. En réalité, les outils cryptographiques comme les bibliothèques logicielles ou les accélérateurs matériels supportent un ensemble de courbes fixées. Les courbes utilisées en pratique sont définies par des standards/recommandations étatiques ou des consortiums industriels. Je discuterai plus en détail de la normalisation des courbes elliptiques dans la section I-3-A (particulièrement page 28). L'ordre des courbes elliptiques utilisées en cryptographie doit respecter certaines contraintes pour éviter des attaques connuss. Par exemple, cet ordre doit être un nombre premier de grande taille ou le produit d'un grand nombre premier et d'un petit entier ou cofacteur, qui vaut 1 dans le cas d'une courbe d'ordre premier. En se basant sur les courbes utilisées dans le logiciel `openssl` (version 1.0.1) nous pouvons observer la forme des courbes les plus fréquemment employées. Pour les 25 courbes elliptiques sur $GF(p)$, 23 ont un ordre premier. Seulement deux courbes (`secp112r2` et `secp128r2`) ont un ordre non premier avec un cofacteur de 4. Une nouvelle courbe (`curve25519`) a récemment été ajoutée dans la version 1.1 d'`openssl`. Cette courbe a un cofacteur de 8. L'utilisation de courbes avec un cofacteur plus grand que 1 peut sembler anecdotique au vu de ces chiffres. La réalité est toute autre, car dans les dernières années de nouvelles courbes elliptiques avec un cofacteur de 4 ou 8 ont émergé : les courbes quartiques de Jacobi et les courbes tordues (« twisted ») de Edwards. Ces courbes permettent d'unifier les opérations de doublage et d'addition de points en une seule opération permettant ainsi d'améliorer la sécurité vis-à-vis de certaines attaques physiques. Je reviendrai en détail sur ces courbes dans la section I-5-B (page 41). Le Chapitre III sera dédié à l'évaluation du niveau de sécurité apporté par cette unification. Les courbes avec un cofacteur supérieur à 1 peuvent aussi permettre d'améliorer les performances (ex: les courbes de Montgomery qui sont, par exemple, utilisées par Apple).

2-D. Multiplication scalaire sur une courbe elliptique

Soit P un point d'une courbe elliptique. Il est possible d'ajouter P à lui même en réalisant une opération de doublage pour obtenir le point $2 \times P$. On peut ensuite calculer $3 \times P$, $4 \times P$,... C'est la multiplication scalaire : une multiplication entre un scalaire et le point d'une courbe elliptique dont le résultat est un aussi un point.

En cryptographie, un point générateur est associé à chaque courbe elliptique. On l'appelle

ainsi car il peut générer une famille de points par la loi d'addition. Cette famille contient un nombre fini de points. Ce nombre est appelé l'ordre du générateur que je noterai n. Nous avons $(n-1) \times P = -P$ et donc $n \times P = P_\infty$. Si nous continuons à ajouter P nous retombons sur P : $(n+1) \times P = P$. Ce point générateur sert, entre autres, à dériver les clés publiques des clés privées. L'ordre du générateur est n, mais pour une courbe elliptique avec un cofacteur de 1 c'est aussi l'ordre de la courbe. Dans le cas d'un cofacteur supérieur à 1, la famille générée par le point générateur contiendra aussi un nombre premier de points : $\#E \div \text{Cofacteur}$.

Il est possible de réaliser une multiplication scalaire $k \times P$ en ajoutant P à lui même $k-1$ fois : $P + P + P + P \ldots$ mais cette méthode n'est pas optimal. Des algorithmes de multiplication scalaire plus performants seront explorés dans la section I-3-C (page 35).

2-E. Protocoles basés sur les courbes elliptiques

Le problème compliqué permettant la mise en œuvre de la cryptographie sur les courbes elliptiques est le logarithme discret. Il s'agit pour un attaquant d'identifier le scalaire k lorsque l'on connait les points P et Q tels que $Q = k \times P$. Les meilleurs algorithmes permettant de résoudre ce problème pour un scalaire de 256 bits requièrent de calculer en moyenne $2^{256 \div 2}$ multiplications scalaires. Il s'agit des algorithmes Rho de Pollard [Poll78] et Big-Step-Giant-Step [Shan71]. L'opération permettant de calculer Q à partir du scalaire k et du point P est beaucoup plus simple car elle nécessite seulement une multiplication scalaire.

Les clés secrètes sont des scalaires choisis au hasard dans l'intervalle d'entiers entre 1 et $n-1$, n étant l'ordre du point générateur G de la courbe elliptique. La clé publique associée à une clé secrète k_s est un point de la courbe elliptique U tel que $U = k_s \times P$. Les contraintes dans le choix de k_s permettent d'assurer que U ne sera pas le point à l'infini. Il est aussi possible d'ajouter des contraintes comme l'interdiction d'utiliser la clé secrète 1 pour éviter d'avoir une clé publique égale à G. Les clés secrètes et publiques sont associées à une courbe elliptique et ne peuvent pas être utilisées sur une autre courbe.

Les deux protocoles cryptographiques principaux basés sur les courbes elliptiques sont ECDSA et ECDH. Le protocole ECDSA permet d'émettre (Figure I-6) et vérifier (Figure I-7) des signatures tandis que ECDH permet d'échanger un secret partagé (Figure I-8).

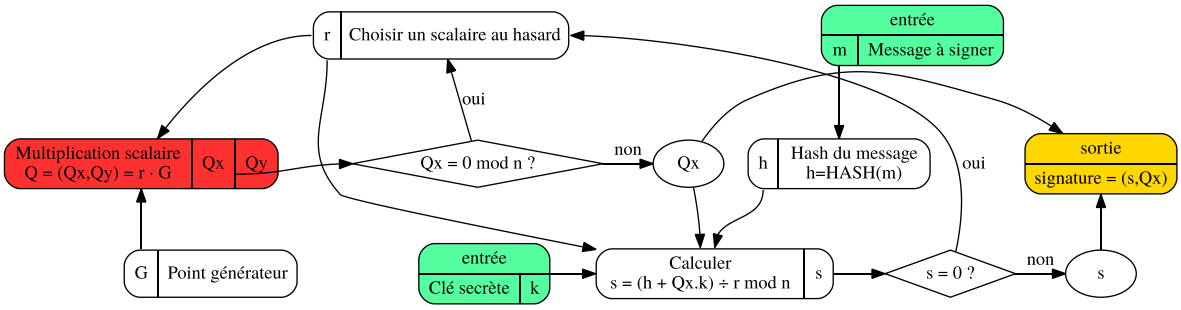


Figure I-6 ECDSA: signature d'un message *m* avec la clé privée *k*

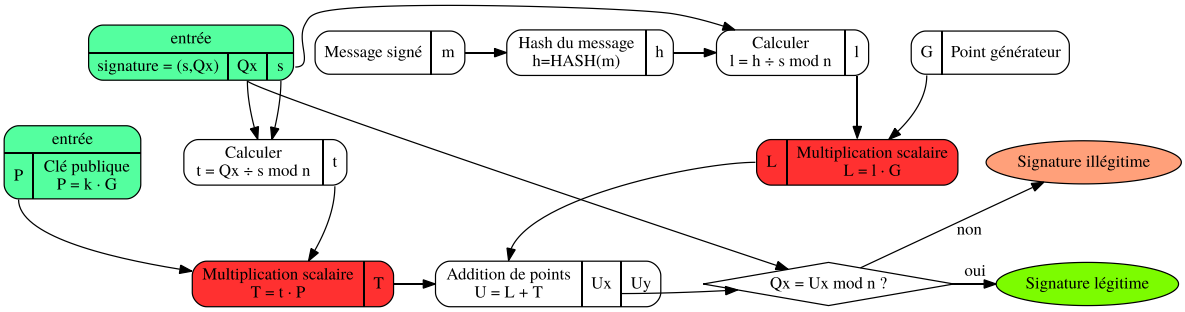


Figure I-7 ECDSA: vérification de la signature d'un message *m* avec la clé publique *P* associée à la clé privée *k*

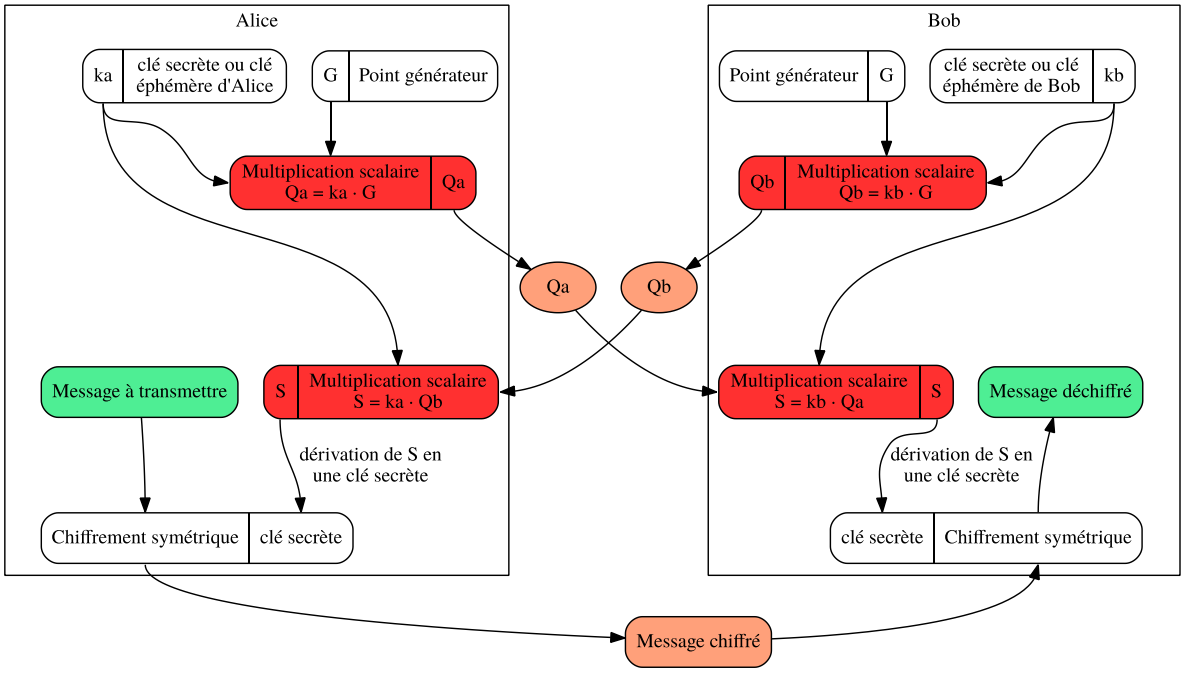


Figure I-8 ECDH: Échange d'un secret partagé pour transmettre un message chiffré

Tous ces protocoles sont basés sur une ou deux multiplications scalaires. Le temps de calcul d'une signature, d'une vérification de signature ou d'un échange de secret partagé est

principalement consommé par cette opération. Cette opération est critique pour les performances du système cryptographique mais aussi pour la sécurité de la clé secrète. La clé secrète est manipulée par une multiplication scalaire dans le protocole ECDH. Le scalaire utilisé dans une signature ECDSA est un nombre r choisi au hasard mais s'il est découvert par un attaquant qui connait aussi la signature (s, Qx) et le message m, alors la clé secrète est compromise car:

$$k = \frac{s \cdot r - \text{hash}(m)}{Qx} \mod n$$

La multiplication scalaire est donc critique, car si le scalaire utilisé est découvert un secret peut-être compromis, sauf dans le cas de la vérification d'une signature ECDSA car aucun élément secret n'est manipulé.

2-F. Le corps fini binaire

Comme le montre la Figure I-1 (page 15), la différence entre l'utilisation d'une courbe elliptique basée sur le corps fini binaire ou premier réside dans les opérations sur le corps et les opérations d'addition et de doublage de points. Soient $A = (a_{d-1}, a_{d-2}, \ldots, a_0)$ et $B = (b_{d-1}, b_{d-2}, \ldots, b_0)$ des éléments de $GF(2^d)$. Ils peuvent être respectivement représentés par les polynômes $A(X) = a_{d-1} \cdot x^{d-1} + a_{d-2} \cdot x^{d-2} + \ldots + a_1 \cdot x + a_0$ et $B(X) = b_{d-1} \cdot x^{d-1} + b_{d-2} \cdot x^{d-2} + \ldots + b_1 \cdot x + b_0$. Les coefficients a_i et b_i sont des éléments de $GF(2)$, c'est-à-dire qu'ils peuvent prendre les valeurs 0 ou 1. La loi d'addition entre coefficients est définie comme l'opération booléenne XOR. Contrairement aux additions d'entiers utilisées dans $GF(p)$ il n'y a jamais de retenue. Le degré du polynôme résultant est donc toujours inférieur ou égal au degré maximum des opérandes. Comme le montre la Figure I-9 et la Figure I-10, on peut voir les éléments de $GF(2^d)$ comme des entiers dont les coefficients représentent les bits ; l'addition est alors une addition classique entre entiers représentés sous forme binaire sans propagation de retenue. Les additions sur $GF(2^d)$ seront plus rapides.

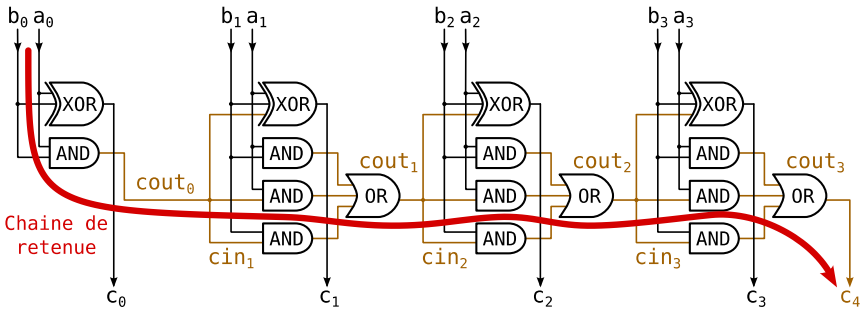


Figure I-9 Exemple d'additionneur d'entiers et sa chaîne de retenue

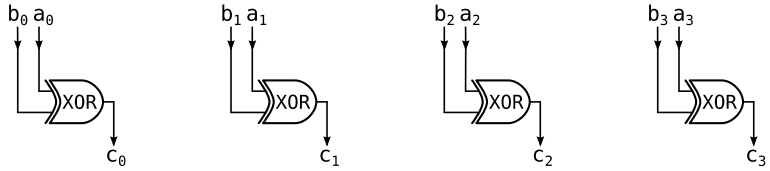


Figure I-10 Additionneur sur $GF(2^d)$ (pas de chaîne de retenue)

La multiplication entre deux éléments de $GF(2^d)$ est une multiplication classique entre polynômes modulo un polynôme irréductible $R(X)$. $R(X)$ étant de degré $d+1$, cela assure que le résultat d'une multiplication sera toujours un polynôme de degré maximal d.

L'équation courte de Weierstrass utilisée étant différente de celle utilisée sur $GF(p)$, les formules permettant de réaliser les opérations d'addition et de doublage de points sont aussi différentes :

$$R(x_3, y_3) = P(x_1, y_1) + Q(x_2, y_2) \text{ avec } Q \neq P \text{ et } Q \neq -P \text{ et } P \neq P_\infty \text{ et } Q \neq P_\infty$$

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$$

$$y_3 = \lambda \cdot (x_1 + x_3) + y_1 + x_3$$

$$R(x_3, y_3) = 2 \times P(x_1, y_1) \text{ avec } Q \neq P_\infty$$

$$\lambda = x_1 + \frac{y_1}{x_1}$$

$$x_3 = \lambda^2 + \lambda + a_2$$

$$y_3 = \lambda \cdot (x_1 + x_3) + y_1 + x_3$$

La multiplication scalaire et la couche protocolaire ne sont pas spécifiques au corps fini sur lequel la courbe elliptique est appliquée.

Des travaux récents [GaGe14] basés sur la sommation de polynômes [Sema04] suggèrent l'existence potentielle d'un algorithme sous-exponentiel pour résoudre le logarithme discret

sur des courbes elliptiques sur $GF(2^d)$. Il n'y a pas de consensus sur cette question et, même sous des hypothèses optimistes, le temps de calcul et la mémoire nécessaires au calcul du logarithme discret sur des clés de tailles réelles restent très grands [GaGa16].

3. Optimisation des performances

Plusieurs techniques et méthodes existent pour optimiser les calculs arithmétiques sur une courbe elliptique. Je présente ici celles qui sont adaptées à des cibles matérielles.

3-A. Multiplication sur GF(p)

Tous les calculs pour la cryptographie basée sur les courbes elliptiques reposent sur l'arithmétique du corps fini des coordonnées (voir Figure I-1). Il est donc indispensable d'utiliser un multiplieur efficace. Je m'intéresse ici à la multiplication sur le corps $GF(p)$. L'opération à réaliser est :

$$c = a \times b \mod p$$

Les éléments a, b, c et p sont des entiers de taille entre 200 et 512 bits. La constante p est le nombre premier associé à la courbe elliptique utilisée. Suivant la courbe, ce nombre premier peut avoir une forme spéciale ou avoir été choisi au hasard parmi les nombres premiers de la taille désirée. L'utilisation de nombres premiers spéciaux permet d'optimiser la coûteuse opération de réduction qui suit une multiplication.

3-A.i Nombres premiers spéciaux

Les courbes elliptiques les plus utilisées sont celles proposées par le NIST (*National Institute of Standards & Technology (USA)*) sur $GF(p)$ introduites dans le standard FIPS [FIPS00]. Ce standard décrit notamment trois courbes `P-256`, `P-384` et `P-521` qui sont conformes au référentiel général de sécurité émis par l'ANSSI [ANSS14]. Elles utilisent des nombres spéciaux. Par exemple, le p de la courbe `P-256` est un nombre premier de Mersenne généralisé:

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$$= (\texttt{0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF})_{16}$$

L'avantage de ce type de nombre est l'existence des méthodes rapides pour réaliser l'opération de réduction modulaire. Par exemple, pour `P-256` la réduction d'un nombre de 512

bits modulo p coûte seulement 4 additions et 4 soustractions sur des entiers de 256 bits. Les opérandes de cette opération sont des réorganisations des digits de 32 bits formant le mot à réduire. Cette optimisation est particulièrement efficace sur CPU.

Une autre technique de réduction modulaire est utilisable pour le p de la courbe `curve25519` [Bern06] :

$$p = 2^{255} - 19$$

$$= (\texttt{0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFED})_{16}$$

L'auteur de cette courbe propose de manipuler les nombres sous une forme particulière et d'utiliser l'unité de calcul flottant souvent présente dans les CPU. La réduction modulaire proposée est inspirée de la méthode de réduction des pseudo-nombres premiers de Mersenne, appliquée à la représentation des nombres proposée.

Toutes les courbes elliptiques sur $GF(p)$ utilisant des nombres premiers spéciaux ont été originellement conçues pour accélérer le logiciel effectuant la réduction modulo p. Il est quand même possible de tirer un bénéfice de cette optimisation sur des cibles matérielles [GüPa08; AlRa14; SaGü14]. Ceci nécessite d'implémenter autant de méthodes de réduction que de nombres premiers spéciaux à supporter. La multiplication sur $GF(p)$ est alors réalisée par un multiplieur d'entiers suivi d'une réduction.

La réduction modulo un nombre spécial est aussi utilisable sur une arithmétique utilisant une représentation modulaire des nombres (RNS) [BiTi15].

3-A.ii Nombres premiers aléatoires

La quatrième courbe conforme avec le référentiel général de sécurité émis par l'ANSSI [ANSS14] est la courbe FRP256V1, définie dans un avis relatif aux paramètres de courbes elliptiques définis par l'Etat français dans le journal officiel n°241 du 16/10/2011. La forme de ce nombre premier ne semble pas permettre d'optimiser la réduction modulo p :

$$p = (\texttt{OxF1FD178C0B3AD58F10126DE8CE42435B3961ADBCABC8CA6DE8FCF353D86E9C03})_{16}$$

Ce nombre premier est congru à 3 modulo 4. Cette propriété permet d'optimiser le calcul de racine carré modulo p. Cette opération est utilisée pour retrouver les coordonnées y potentielles d'un point à partir de sa coordonnée x (pour des courbes de Weierstrass). On peut ainsi économiser la bande-passante lors de la transmission d'un point en envoyant seulement

la coordonnée x et 1 bit : c'est la compression de point. En 2005, le consortium ECC Brainpool a proposé une nouvelle méthode de génération pseudo-aléatoire de courbes elliptiques vérifiables [Brai05]. Comme la courbe `frp256V1`, ces courbes ont pour seule spécificité d'être congrues à 3 modulo p ; par contre, la méthode de génération est publique. En 2016, une nouvelle courbe a été générée : `mdcurve201601` [BaDe16]. La méthode de génération et la manière d'obtenir la graine de la nouvelle courbe ont été publiées avant que la graine soit connue. La graine a été calculée à partir des résultats de loteries dans plusieurs pays en février 2016. La confiance que l'on peut accorder à cette courbe est importante. La vérifiabilité de la graine permet de s'assurer que les auteurs n'ont pas visé une classe de courbe elliptique dont ils connaîtraient les faiblesses. Le p de `mdcurve201601` n'a aucune forme spéciale.

De nouvelles courbes elliptiques sont disponibles chaque année. Pour assurer une compatibilité avec toutes les courbes existantes et les courbes futures, les accélérateurs matériels doivent être capables de supporter les opérations sur $GF(p)$ pour des nombres premiers p aléatoires, c'est-à-dire des p pour lesquels il n'existe pas forcément d'algorithme de réduction modulo p plus efficace que les algorithmes génériques.

3-A.iii Multiplieur de Montgomery

Les algorithmes génériques de réduction modulo p étant peu efficaces et coûteux, la manière la plus efficace de réaliser une multiplication modulo p consiste à mixer la réduction et la multiplication. Le multiplieur qui permet de combiner ces deux opérations est le multiplieur de Montgomery [Mont85]. La Figure I-11 illustre la différence majeure entre ce multiplieur et les multiplieurs modulo p classiques.

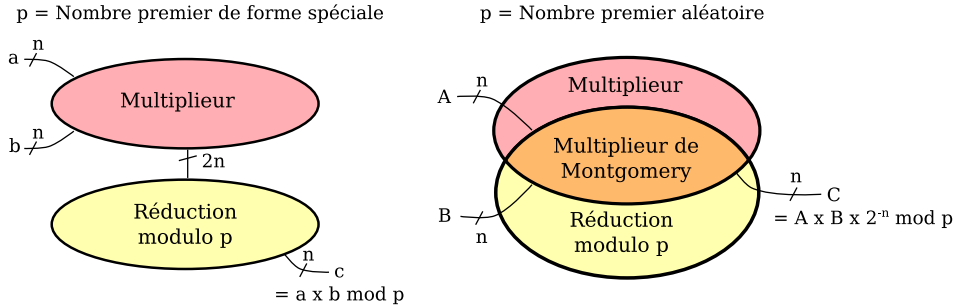


Figure I-11 Mixer la réduction modulaire et la multiplication: le multiplieur de Montgomery

On peut observer que le multiplieur de Montgomery ne calcule pas exactement le produit des opérandes en entrée. Le résultat est bien réduit mais il a été divisé par 2^n, n étant le

nombre de bits nécessaires pour coder p; on a donc $p < 2^n$. Pour s'affranchir de cette difficulté, les calculs sont réalisés dans un domaine où l'image d'un nombre a est le nombre A tels que $A = a \times 2^n$: c'est le domaine de Montgomery. La Figure I-12 montre sur l'exemple du calcul de $e = a \times b + c \mod p$ comment l'utilisation de cette représentation alternative permet de s'affranchir de la division inhérente au multiplieur de Montgomery. Le coût des transpositions de domaines dans des calculs cryptographiques est négligeable car elles sont effectuées seulement en début en en fin de calcul : les nombres sont représentés dans le domaine de Montgomery le plus longtemps possible. Sur la Figure I-12 on peut observer qu'il est inutile de transposer les variables intermédiaires comme d dans le domaine classique.

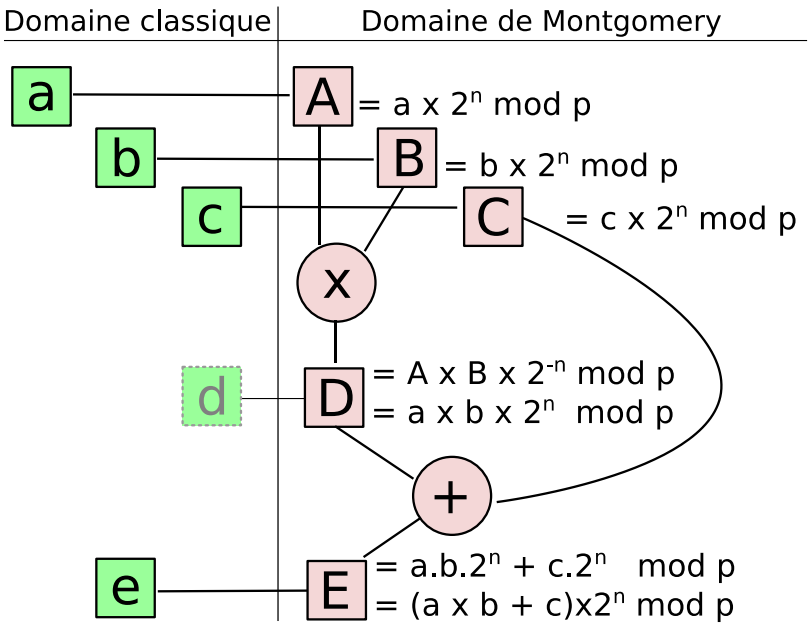


Figure I-12 Calculs dans le domaine de Montgomery

Les transpositions de domaine se font avec le multiplieur lui-même. Il faut avoir préalablement calculé la valeur réduite sur n bits de $2^{2n} \mod p$. En utilisant le multiplieur de Montgomery avec pour opérandes un nombre représenté dans un des deux domaines et la constante 2^{2n} réduite ou 1, les deux transpositions s'effectuent de la manière suivante :

$$A = (a \times 2^{2n}) \times 2^{-n}$$

$$a = (A \times 1) \times 2^{-n}$$

Il existe plusieurs variantes du multiplieur de Montgomery. Je vais commencer par présenter l'algorithme implémentant la version série-parallèle. La Figure I-13 donne un

exemple de ce multiplieur avec un nombre premier p de seulement 3 bits pour faciliter l'illustration. Je rappelle que dans la pratique des nombres d'environ 256 bits sont utilisés.

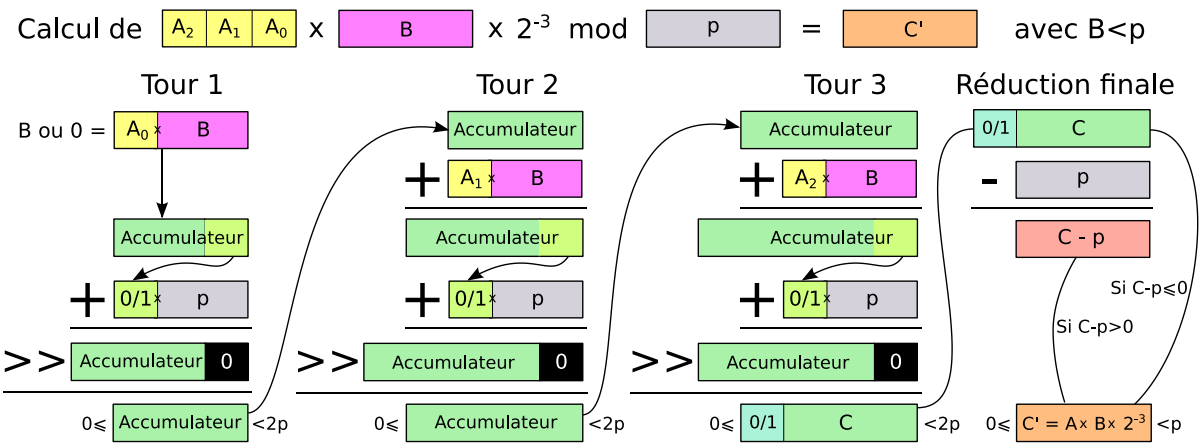


Figure I-13 Multiplieur de Montgomery série-parallèle: exemple avec un p sur 3 bits

Le nombre A est lu bit à bit : c'est l'opérande série. Par contre, dans les étapes où le nombre B est utilisé, tous ses bits sont lus simultanément. Pour un nombre premier p de n bits, il y aura n itérations (3 tours boucles dans l'exemple Figure I-12) et une réduction finale pour ramener le résultat dans l'intervalle $[0, p-1]$. La première itération (tour 1) consiste à initialiser un accumulateur à B ou 0 suivant la valeur du bit de poids faible de A : 1 ou 0. La seconde étape du tour est une étape de réduction, le bit de poids faible de l'accumulateur est forcé à 0 en lui ajoutant p si nécessaire. Je peux alors diviser l'accumulateur par deux sans perdre d'information. Dans le tour de boucle suivant, B ou 0 est ajouté à l'accumulateur suivant la valeur du bit A_1. Chaque fin d'itération se termine par une réduction de la taille occupée par l'accumulateur en effectuant une division par 2. En pratique il faudra que B respecte la précondition $B < p$. Sous l'hypothèse de départ $p < 2^n$ ceci permet d'assurer la post-condition $C < 2p$ [Guer02]. Une réduction finale de C est obligatoire pour assurer $C < p$ et pouvoir l'utiliser par exemple comme opérande d'une prochaine multiplication.

La multiplication de Montgomery est plus générale que celle présentée ci-dessus : p peut ne pas être premier tant qu'il est premier avec R lorsque l'on calcule $A \times B \times R \bmod p$. Nous travaillons sur $GF(p)$ avec p premier donc cette capacité du multiplieur de Montgomery ne nous intéresse pas. Ce multiplieur s'adapte parfaitement à des calculs sur $GF(2^d)$. Il est aussi possible de manipuler des nombres découpés en digits plus larges que 1 bit et d'appliquer des méthodes de pipeline. Ces deux points ne seront pas utilisés dans cette thèse, le lecteur intéressé pourra les trouver décrits dans l'Annexe 1.

3-B. Systèmes de coordonnées

Les formules d'addition et de doublage de points présentées dans les parties 2-B et 2-F ne sont pas les plus efficaces. En effet elles nécessitent d'inverser un élément de $GF(p)$ ou de $GF(2^d)$ à chaque opération élémentaire sur des points. Cette opération d'inversion est donc utilisée de nombreuses fois. Elle est un élément crucial pour les performances de la multiplication scalaire ; il faut donc un inverseur matériel coûteux en surface.

Pour éviter le surcoût d'un inverseur optimisé, il est usuel d'utiliser un système de coordonnées projectives : la courbe elliptique est représentée par une surface en 3 dimensions en ajoutant une coordonnée Z aux points. La courbe elliptique affine décrite dans la section I-2 (page 15) est l'intersection entre cette surface et le plan d'équation $Z = 1$. Il existe plusieurs systèmes de coordonnées qui utilisent des projections différentes et peuvent utiliser plus que 3 dimensions. Je ne m'intéresse qu'aux systèmes projectifs utilisant 3 coordonnées pour minimiser le coût mémoire. Le système projectif le plus efficace pour des courbes elliptiques sur $GF(p)$ utilise des coordonnées Jacobiennes [CoFr05]. On notera en minuscule les coordonnées affines (x, y) utilisées dans la section I-2 et en majuscule les coordonnées Jacobiennes (X, Y, Z). Le passage de coordonnées Jacobiennes aux coordonnées affines nécessite une inversion dans $GF(p)$: le calcul de Z^{-1}.

$$x = X \times \left(\frac{1}{Z}\right)^2 \qquad y = Y \times \left(\frac{1}{Z}\right)^3$$

si Z est différent de 0. On considère parfois les points à coordonnée Z nulle comme des points représentant le point à l'infini : P_∞. En coordonnées Jacobiennes, il existe $p - 1$ représentations de chaque point de la courbe elliptique. La conversion d'un point de coordonnées affines en coordonnées Jacobiennes est une opération gratuite puisqu'il suffit d'ajouter une coordonnée Z égale à 1 :

$$X = x \qquad Y = y \qquad Z = 1$$

La Figure I-14 et la Figure I-15 décrivant les opérations d'addition et de doublage de points en coordonnées Jacobiennes mettent en évidence l'avantage de ce type de coordonnées : l'inversion présente dans les opérations élémentaires sur les points en coordonnées affines a été remplacée par des multiplications faisant intervenir Z.

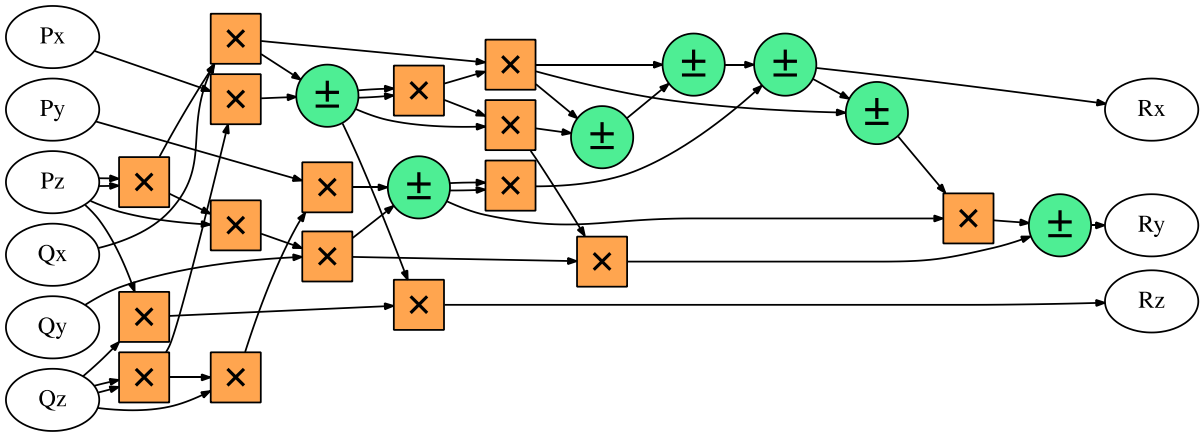


Figure I-14 Exemple d'ordonnancement pour l'addition de point $R = P + Q$ en coordonnées Jacobiennes ($GF(p)$)

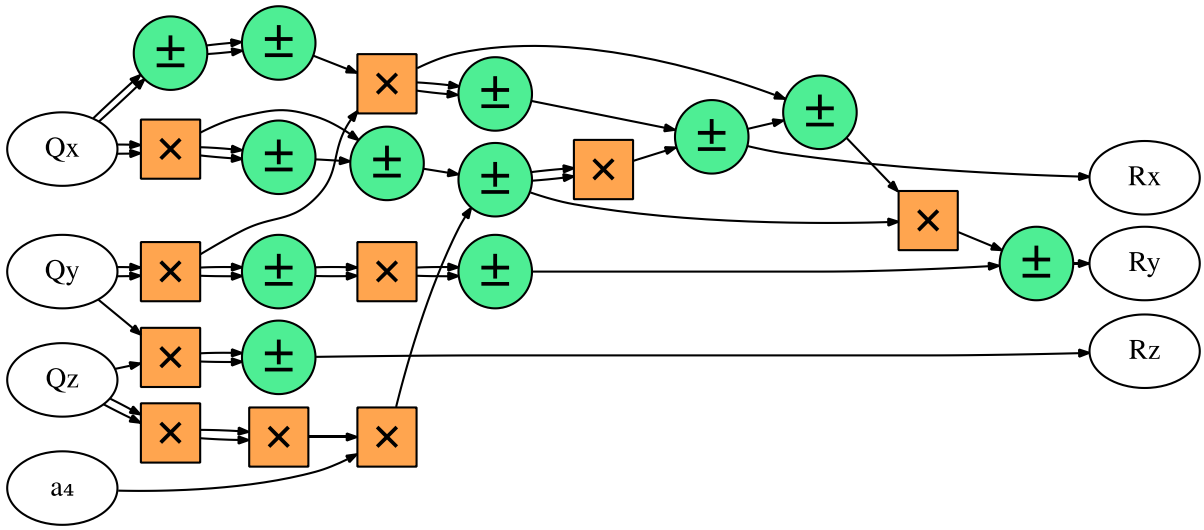


Figure I-15 Exemple d'ordonnancement pour le doublage de point $R = 2 \times Q$ en coordonnées Jacobiennes ($GF(p)$)

L'utilisation d'un tel système de coordonnées à un coût en multiplications mais il permet d'éviter l'utilisation d'un inverseur optimisé. Un inverseur sera toujours nécessaire pour transposer le point final en coordonnées affines, mais celui-ci ne sera utilisé qu'une seule fois par multiplication scalaire. On peut donc se permettre d'utiliser un inverseur moins optimisé pour économiser de la surface et ainsi réduire le coût du crypto-processeur. On utilise généralement un inverseur basé sur le petit théorème de Fermat. Il suffit donc de réaliser une exponentiation avec des multiplications itératives selon la formule :

$$\frac{1}{a} \mod p = a^{p-2}$$

Le coût d'une opération d'addition ou de doublage peut-être estimé en nombre de multiplications. En effet le temps de calcul d'une addition est négligeable par rapport au

temps de calcul d'une multiplication, surtout pour les grands nombres que nous manipulons. Souvent les opérations d'élévation au carré sont comptées distinctement des multiplications. Cela s'explique par le fait que l'on peut optimiser la multiplication pour le cas où les deux opérandes sont égaux. Ceci n'a de sens qu'en logiciel où le coût de la mémoire programme est négligeable. Implémenter un opérateur matériel spécifique à l'élévation au carré a un coût important en surface pour un gain d'environ 20% en terme de temps de calcul par rapport à l'utilisation d'un opérateur multiplieur. Comme je m'intéresse à une cible matérielle, je comptabilise les élévations au carré comme des multiplications. Le coût de l'addition de points décrite Figure I-14 est de 16 multiplications sur $GF(p)$ tandis que le doublage Figure I-15 coûte 10 multiplications. Je m'intéresse dans la suite à minimiser le coût du crypto-processeur développé en utilisant un seul multiplieur. Ces chiffres sont donc dans mon cas un bon indicateur des performances que l'on pourra attendre de ces opérations. Pour le cas des crypto-processeurs utilisant plusieurs multiplieurs, ces chiffres en nombre de multiplications ne suffisent plus à estimer les performances car il faut observer les dépendances de données pour connaître le taux d'activité des multiplieurs. On peut observer qu'une addition de points sera environ 60% plus lente que le doublage d'un point.

On peut utiliser le même système de coordonnées pour les courbes sur $GF(2^d)$ [CoFr05]. La Figure I-16 et la Figure I-17 décrivent les opérations d'addition et de doublage de points sur ces courbes avec les points en coordonnées Jacobiennes.

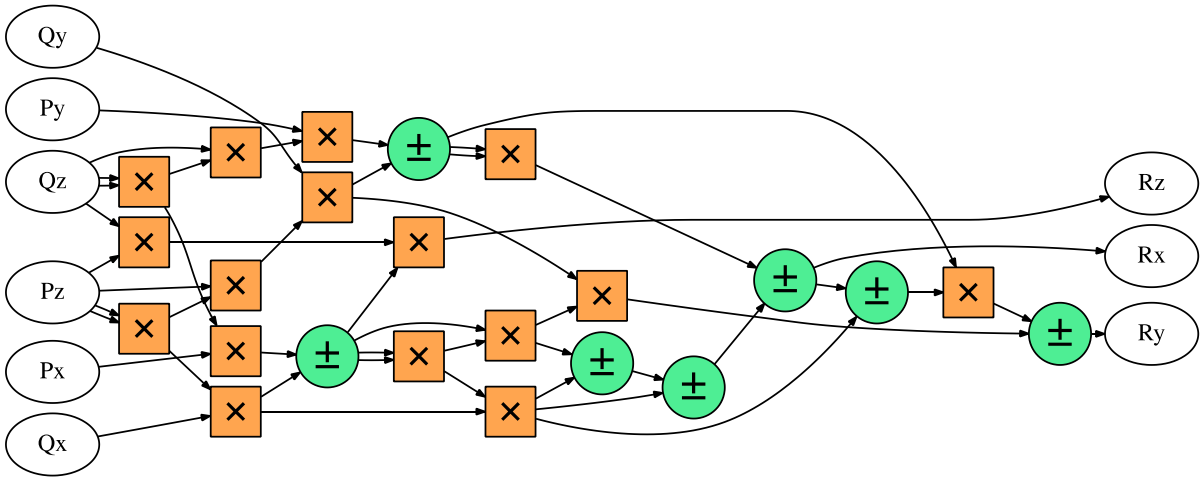


Figure I-16 Exemple d'ordonnancement pour l'addition de point $R = P + Q$ en coordonnées Jacobiennes ($GF(2^d)$)

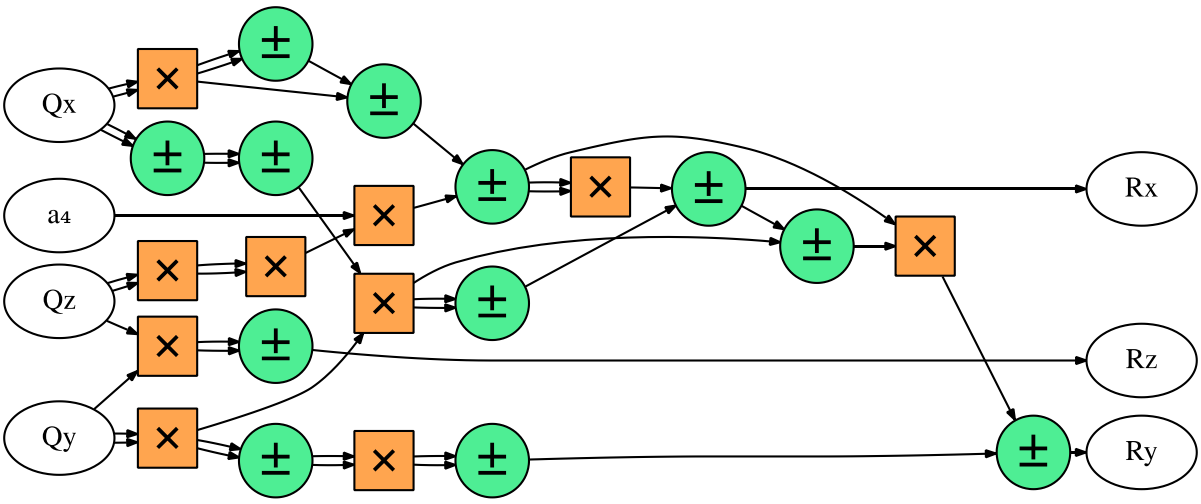


Figure I-17 Exemple d'ordonnancement pour le doublage de point $R = 2 \times Q$ en coordonnées Jacobiennes ($GF(2^d)$)

3-C. Multiplication scalaire

Comme indiqué précédemment, l'opération la plus complexe et la plus coûteuse en temps de calcul dans la cryptographie basée sur les courbes elliptiques est la multiplication scalaire. Elle a déjà été introduite dans la section I-2-D (page 22). Je vais étudier et comparer les algorithmes de multiplication scalaire les plus communs.

L'algorithme naïf consistant à additionner le point P à lui-même k fois pour obtenir le résultat de la multiplication scalaire $k \times P$ n'est pas envisageable : pour une courbe de 256 bits il nécessiterait dans le pire cas 2^{256} opérations sur les points. En utilisant l'opération de doublage de point, on peut mettre en œuvre l'algorithme « Doublages et Additions » qui ne nécessite que $q - 1$ doublages et en moyenne $\frac{q-1}{2}$ additions. On note q le nombre de bits nécessaire pour coder l'ordre du point générateur. C'est aussi la taille en bits des scalaires. La Figure I-18 illustre l'algorithme « Doublages et Additions ». Pour calculer $k \times P$, on parcourt le scalaire k bit à bit de gauche à droite, c'est-à-dire du poids fort au poids faible. Lorsque l'algorithme rencontre le premier bit non nul, on initialise le point accumulateur avec les coordonnées du point P. L'initialisation doit obligatoirement se réaliser comme cela, il n'est pas possible d'initialiser le point accumulateur avec l'élément neutre P_∞ car l'addition de points (comme le doublage) n'est pas définie pour un opérande étant P_∞. Après ce bit d'initialisation, on continue à parcourir le scalaire. Pour chaque bit, on double le point accumulateur. Si le bit est non nul, on additionne le point P à l'accumulateur.

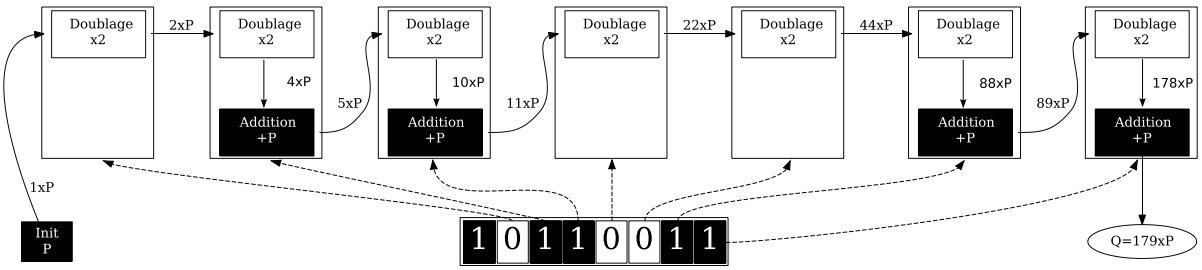


Figure I-18 Illustration de l'algorithme de multiplication scalaire « Doublages et Additions » : exemple sur un scalaire de taille réduite

L'algorithme « Doublages et Additions » utilise un seul point accumulateur. La Figure I-19 montre qu'en utilisant un second accumulateur on peut utiliser un algorithme qui cette fois va parcourir les bits du scalaire de droite à gauche (du bit de poids faible à celui de poids fort). Cette fois nous ajoutons $2^i \times P$ au premier accumulateur lorsque l'on rencontre un bit k_i non nul (première ligne de la Figure I-19). Le second accumulateur, initialisé à P, est doublé pour chaque bit parcouru : c'est lui qui contient le point $2^i \times P$ (seconde ligne de la Figure I-19). Les performances de cet algorithme sont exactement les mêmes que celles de l'algorithme « Doublages et Additions ».

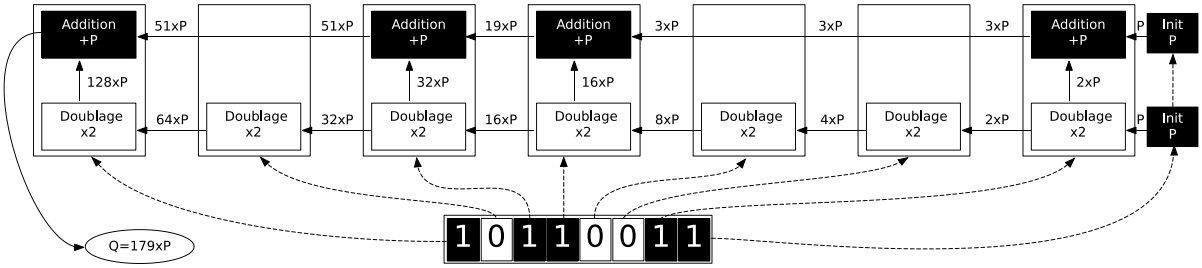


Figure I-19 Illustration d'un algorithme de multiplication scalaire de droite à gauche

Une autre variante de l'algorithme « Doublages et Additions » est l'algorithme « m-ary » qui consiste à utiliser une table de pré-calculs contenant $2^m - 1$ points : $1 \times P, 2 \times P, \ldots, (2^m - 1) \times P$. Pour le cas particulier où l'on calcule la multiplication entre un scalaire et le point générateur G, il est imaginable de pré-calculer cette table hors-ligne. La plupart des multiplications scalaires utilisent un point P non-prévisible, un pré-calcul de cette table hors-ligne est alors impossible. La Figure I-20 décrit l'algorithme de « m-ary » pour le cas $m = 3$ et un scalaire de taille réduite pour l'illustration. Comme pour l'algorithme « Doublages et Additions », le scalaire est parcouru de gauche à droite mais les performances sont améliorées par l'utilisation d'un nombre inférieur d'additions de points. En effet le scalaire est parcouru par fenêtres ($\frac{q}{m}$ fenêtres) contenant des digits de m bits et les additions n'ont lieu qu'une fois par fenêtre et seulement si la fenêtre contient un digit non nul, (probabilité d'occurrence :

$\frac{2^m-1}{2^m}$). La multiplication scalaire avec un scalaire de q bits coûte donc seulement $q-m$ doublages et en moyenne $\frac{q}{m}\frac{2^m-1}{2^m}$ additions. Il est intéressant de remarquer que pour le cas particulier $m=1$, cet algorithme est identique à l'algorithme « Doublages et Additions ».

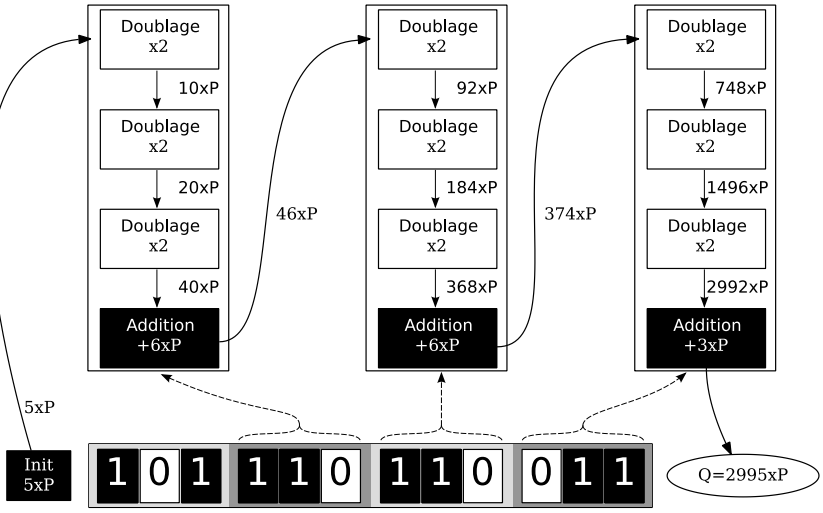


Figure I-20 Algorithme de multiplication scalaire 3-ary

Le Tableau I-1 permet d'estimer le gain en performance obtenu grâce à l'algorithme de multiplication scalaire par fenêtrage « m-ary ». Cette estimation a été faite en nombre de multiplications sur $GF(p)$ pour le système de coordonnées Jacobiennes (voir 3-B et un scalaire de 256 bits. Si on comptabilise les pré-calculs dans le coût total, on observe qu'il est inutile d'utiliser des fenêtres de plus de 5 bits. À partir de cette taille, le coût des pré-calculs devient plus grand que l'économie d'additions de points. Pour un scalaire de 256 bits, la taille de fenêtre utilisant le moins de multiplications sur $GF(p)$ (4839) est de 4 bits. Pour un scalaire de 512 bits, cette taille est de 5 bits et coûte 9502 multiplication sur $GF(p)$. Cette optimisation a un coût en mémoire, pour stocker 2^m-1 points. Par exemple, ceci représente 12 Kbits pour un scalaire de 256 bits et un fenêtrage de 4 bits.

Tableau I-1 Coût moyen d'une multiplication scalaire sur 256 bits avec l'algorithme m-ary en multiplication sur GF(p) avec le système de coordonnés Jacobiennes

m	1 Doublages et Additions	2	3	4	5	6
Additions	4080	4064	4048	4032	4016	4000
Doublages	2501	1250	834	625	500	417
Pré-calculs	0	26	78	182	390	806
Total	6581	5340	4960	**4839**	4906	5223
Total (sans pré-calcul)	6581	5314	4882	4657	4516	4417

Le calcul de l'opposé d'un point est très simple car il nécessite seulement une soustraction/addition sur $GF(p)$. Cette propriété est utilisée par des algorithmes recodant le scalaire. L'objectif est de réduire le nombre d'additions de points en diminuant la densité du scalaire (obtenir plus de bits ou digits nuls) par un recodage autorisant les bits ou les digits à prendre des valeurs négatives. Ceci peut s'appliquer sans fenêtrage (c'est l'algorithme NAF, forme non-adjacente) [MoOl90] ou avec fenêtrage (c'est l'algorithme wNAF) [OkSc04]. Il est aussi possible d'ajouter des opérateurs basiques sur les points comme le triplement ou le quintuplement. Un recodage du scalaire dans une base autre que 2 est alors possible pour améliorer les performances : par exemple le DBNS (système de nombres à base double) [DoIm06]. C'est une méthode alternative au stockage de points pré-calculés qui est basée sur l'ajout d'opérateurs basiques sur les points (triplement ou quintuplement de point) pour améliorer les performances. Ces deux recodages peuvent être combinés : par exemple avec le codage multi-base NAF (mbNAF) [LoMi08]. L'objectif de cette thèse est de protéger le scalaire des attaques par canaux auxiliaires. Ces optimisations ne seront pas utilisées pour minimiser la manipulation du secret en évitant le recodage.

Les techniques de fenêtrage présentées ne s'appliquent qu'à des multiplications scalaires de gauche à droite. La multiplication scalaire de droite à gauche présentée précédemment utilise un accumulateur contenant initialement P doublé itérativement (voir Figure I-19, page 36). Appliquer le fenêtrage à cette méthode est trop coûteux car il serait nécessaire de doubler tous les points pré-calculés à chaque itération.

4. Crypto-processeurs de référence

Pour positionner les crypto-processeurs présentés dans cette thèse, je les compare à d'autres circuits existants. Il existe une grande variété de crypto-processeurs basés sur les courbes elliptiques. J'ai choisi de me focaliser sur la plus grande classe de circuits récents comparables. Il s'agit des circuits supportant une courbe de Weierstrass de 256 bits sur $GF(p)$ et ciblant les familles de FPGA Xilinx : 5, 6 et 7. Les technologies de ces trois familles étant proches, les ressources matérielles utilisées par des crypto-processeurs sur ces FPGAs sont donc comparables. Elles sont exprimées en LUTs (table de correspondance à 1 bit de sortie et 6 bits d'entrée) et en FFs (Flip-Flop, élément mémorisant de 1 bit) mais aussi en SLICEs (qui contiennent 4 LUTs et 8 FFs, sauf pour la famille 5 où un SLICE ne contient que 4 FFs). Par contre les familles récentes permettent des fréquences d'horloge plus importantes.

J'ai identifié 9 circuits qui serviront de référence pour évaluer mes propres circuits. Un de ces circuits [MaLi13] a déjà été évoqué (voir section I-3-A page 146) pour son multiplieur de Montgomery sur $GF(p)$ extrêmement pipeliné. Une réduction modulaire efficace grâce à un nombre premier spécial est utilisée dans 5 circuits [SaGü14; RoMu14; AlRa14; KoDe16]. L'élimination du calcul de la coordonnée y sur une courbe de Montgomery est utilisée pour améliorer les performances des circuits décrits dans [SaGü14; KoDe16; Dond15] et un système modulaire de représentation (RNS) est utilisé par le circuit dans [BaMe14] pour accélérer les calculs sur $GF(p)$. Le circuit présenté dans [AlRa14] à la particularité de supporter les 5 courbes du NIST sans reconfigurations. Un de ces circuits [Dond15] manipule des scalaires de taille légèrement supérieure à 256 bits (288 bits) à cause d'une contre-mesure aux attaques par canaux auxiliaires. Le circuit le plus rapide [KoDe16] (calcul d'une multiplication scalaire en 0.12 ms) utilise 4 multiplieurs matériels sur $GF(p)$.

J'ai aussi conçu des circuits sortant un peu du cadre de cette thèse car non-protégés contre les attaques matérielles. Ils m'ont permis d'évaluer les possibilités de la synthèse de haut niveau appliquée à la cryptographie basée sur les courbes elliptiques. Ce travail est présenté dans l'Annexe 3. Les circuits obtenus supportant la courbe `P-256` seront aussi utilisés comme éléments de comparaison.

5. Attaques par canaux auxiliaires contre les implémentations ECC

Les courbes elliptiques ont été choisies pour éviter de pouvoir, dans un temps raisonnable, calculer la clé privée à partir de la clé publique, c'est-à-dire résoudre le problème du logarithme discret. Un attaquant peut toutefois accélérer la recherche de la clé en utilisant des canaux auxiliaires (aussi appelés canaux cachés). L'exemple le plus simple de canal auxiliaire est la mesure du temps mis par le coprocesseur pour effectuer un calcul : c'est une attaque passive car elle nécessite seulement de l'observation. Elle est aussi non-invasive car l'extraction de l'information n'affecte pas l'intégrité du matériel. Il existe d'autres attaques passives où l'attaquant doit avoir un accès physique (au moins proche) au crypto-processeur : attaques par analyse de la puissance consommée ou analyse du champ électromagnétique émis. Je me focaliserai sur les attaques par espionnage du temps de calcul et de la puissance consommée car elles sont les moins coûteuses à mettre en œuvre. J'évoquerai aussi des attaques actives (attaques par injection de fautes) mais elles ne seront pas au cœur de mon étude.

5-A. Analyse du temps de calcul

La première attaque par analyse du temps de calcul réellement implémentée date de 1996 [Koch96]. Nous pouvons facilement voir que l'algorithme « Doublages et Additions » (Figure I-18, page 36) est sensible à cette attaque. Le nombre de doublages effectués par cet algorithme est constant, il s'agit du nombre de bits de la clé privée, mais le nombre d'additions de points effectuées est le nombre de bits à 1 de la clé privée. Il est donc trivial de retrouver le poids de Hamming (nombre de bit non-nuls) de cette clé par une analyse du temps de calcul. La contre-mesure la plus simple à cette attaque est de réaliser la multiplication scalaire en temps constant. La Figure I-21 décrit l'algorithme « Doublages et Additions Systématiques » ; il est dérivé de l'algorithme « Doublages et Additions », mais une addition fictive est effectuée pour les bits nuls. Une addition est donc effectuée pour chaque bit.

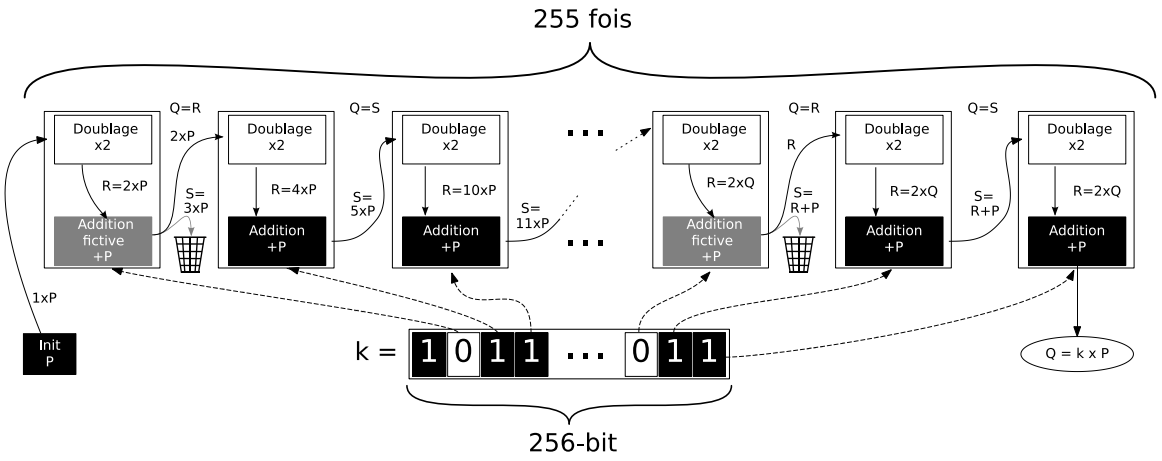


Figure I-21 Algorithme de multiplication scalaire « Doublages et Additions Systématiques »

Ici la position temporelle des opérations fictives est directement dépendante de la valeur du bit du scalaire. Une attaque active par injection de faute peut servir à tester l'utilité d'une addition de points : si, suite à une injection de faute pendant une addition, le point résultant est toujours sur la courbe alors l'attaquant peut en déduire que cette opération était fictive. Il peut alors retrouver tous les bits nuls du scalaire : c'est l'attaque par « C safe-error » [SuKi01]. Cela montre qu'une contre-mesure à une attaque peut ouvrir une brèche à une nouvelle attaque.

Il est préférable d'utiliser une contre-mesure basée sur un algorithme de multiplication scalaire naturellement régulier. C'est le cas de l'échelle de Montgomery [JoYe02]. Il ne faut cependant pas confondre l'échelle de Montgomery qui est un algorithme de multiplication scalaire opérant sur les points d'une courbe elliptique avec le multiplieur de Montgomery qui

permet de réaliser une multiplication entre entiers, modulo un nombre premier. Pour le calcul de $k \times P$, l'échelle de Montgomery est basée sur la manipulation de deux points Q_1 et Q_2 tels que $Q_2 - Q_1 = P$. Elle a trois avantages : elle est régulière naturellement, il est possible d'effectuer les opérations de doublage et d'addition en parallèle pour accélérer le calcul (ceci a bien sûr un coup important en matériel), et lorsque la courbe utilisée peut s'exprimer sous la forme d'une courbe de Montgomery (cofacteur divisible par 4) il est possible de réduire le nombre de multiplications sur le corps grâce à l'élimination du calcul de la coordonnée y (qui peut être reconstruite en fin de calcul) [LóDa99]. Par exemple, les circuits décrit dans [SaGü14; KoDe16; Dond15] implémentent cette optimisation. L'échelle de Montgomery parcourt les bits du scalaire de gauche à droite mais il existe aussi un algorithme naturellement régulier parcourant les bits de droite à gauche [Joye07].

5-B. Analyse simple de consommation (SPA)

La plus simple des attaques par espionnage de la puissance consommée est la SPA (Analyse Simple de Consommation). Elle permet de retrouver le scalaire en analysant une seule trace de puissance consommée d'une multiplication scalaire. Cette attaque a été décrite en 1999 contre l'algorithme de chiffrement symétrique DES [KoJa99] et en 2002 contre la multiplication scalaire sur une courbe elliptique [BrJo02]. L'attaque par analyse du temps de calcul contre l'algorithme « Doublages et Additions » permet seulement de retrouver le poids de Hamming du scalaire alors qu'une attaque SPA permet de retrouver tous les bits du scalaire.

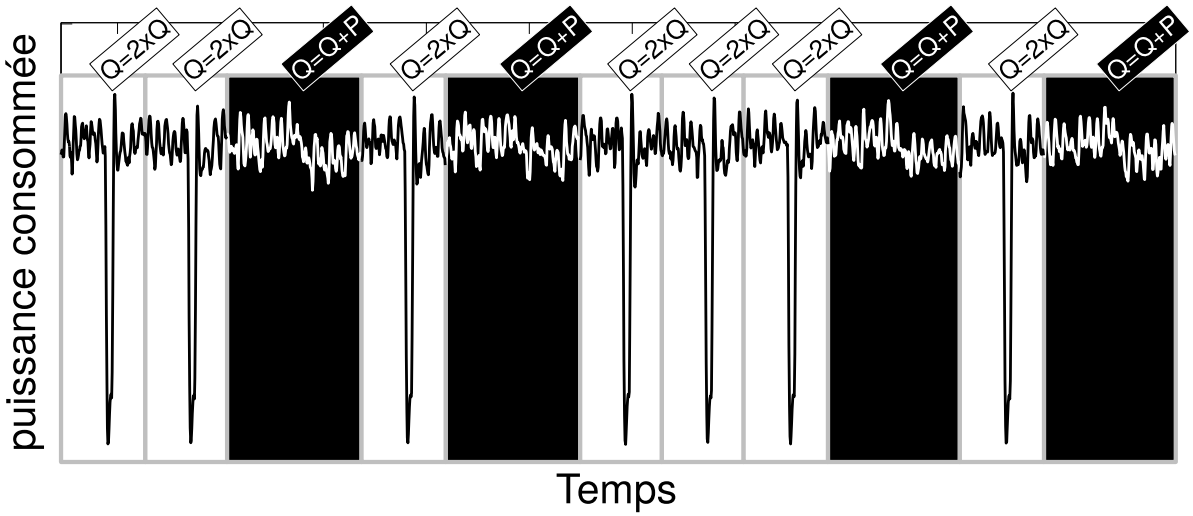


Figure I-22 Extrait d'une trace de puissance consommée d'une multiplication scalaire par l'algorithme « Doublages et Additions » sur cible FPGA Virtex 2

En effet le doublage et l'addition de points utilisant des formules différentes, ils génèrent des traces de puissance différentes. La Figure I-22 montre l'extrait d'une trace de puissance consommée par un crypto-processeur réalisant une multiplication scalaire avec l'algorithme « Doublages et Additions » sur la courbe `secp256k1` et implémenté dans une cible FPGA Virtex 2 (Fabricant Xilinx). Nous observons que l'œil humain est capable de reconnaître les opérations effectuées par détection de motifs sur la trace de puissance et de reconstruire la chaîne d'additions et de doublages qui a été utilisée. Connaissant cette chaîne, l'attaquant retrouve immédiatement le scalaire utilisé.

Les algorithmes réguliers permettant de se protéger contre les attaques par analyse du temps de calcul (voir 5-A sont aussi efficaces pour se protéger des attaques par SPA car la chaîne d'additions et de doublages est indépendante du scalaire utilisé. Par contre ce type de contre-mesure a un surcoût moyen de 100% en nombre d'additions par rapport à l'algorithme « Doublages et Additions ».

Les algorithmes réguliers ne protègent pas contre la détection de la chaîne d'additions et de doublages, ils ne font que supprimer la fuite d'information dans cette chaîne. Il est théoriquement possible de lutter directement contre la fuite exploitée par l'attaque SPA par une harmonisation des motifs de consommation des additions et des doublages. L'apparition de deux motifs différents dans la Figure I-22 est due à l'utilisation de formules différentes pour les opérations de doublage de point (voir Figure I-15, page 33) et d'addition de points (voir Figure I-14, page 33). Il est possible d'utiliser des opérations fictives sur le corps pour obtenir une addition de points et un doublage de points proches [ChCi04]. Comme indiqué précédemment, il existe aussi des formes alternatives de courbes elliptiques où une formule d'addition de points peut être valide même si l'on additionne un point à lui-même, ce qui signifie que le doublage d'un point peut être réalisé avec l'opérateur d'addition de points. C'est ce que l'on appelle des formules unifiées. Les courbes elliptiques avec un cofacteur divisible par 3 peuvent se mettre sous la forme d'une courbe Hessian [JoQu01]. Sous cette forme il existe une formule d'addition permettant aussi d'effectuer un doublage, mais la formule n'est pas complète car elle est invalide lorsque l'on manipule le point à l'infini. D'autres formes permettant d'avoir une formule d'addition complète (aussi nommée « formule fortement unifiée ») existent, mais elles nécessitent d'avoir un cofacteur divisible par 4 : les courbes de Edwards tordues (ou courbes de Edwards généralisées) [BeBi08] et les courbes quartiques de Jacobi [LiSm01]. Elles sont birationnellement équivalentes à une courbe elliptique. Il existe des formules unifiés plus efficaces que celles proposées initialement pour les courbes tordues d'Edwards [HiWo08] comme pour les courbes

quartiques de Jacobi [HiWo09; Plût12].

Dans le Chapitre III nous étudierons la résistance des formules unifiées contre les attaques par canaux auxiliaires, plus particulièrement les courbes quartiques de Jacobi. Une courbe quartique de Jacobi sur $GF(p)$ est de la forme :

$$JQ'_{a,d}: y^2 = d \cdot x^4 + 2 \cdot a \cdot x^2 + 1 \mod p$$

On s'intéressera plus particulièrement aux coordonnées homogènes projectives étendues (sous la forme d'une intersection entre deux quadriques) et au cas $d = 1$. L'équation s'écrit alors :

$$JQ_a: \begin{cases} Y^2 = T^2 + 2 \cdot a \cdot X^2 + Z^2 \mod p \\ X^2 = Z \cdot T \mod p \end{cases}$$

La transformation entre le système de coordonnées affines vers les coordonnées homogènes projectives est simple :

$$X = x \qquad Y = y \qquad Z = 1 \qquad T = x^2$$

$$x = \frac{X}{Z} \qquad y = \frac{Y}{Z}$$

Le point à l'infini est défini comme $P_\infty = (X = 0, Y = 1, Z = 0, T = 1)$. La Figure I-23 décrit l'addition complète et unifiée sur JQ_a. On peut donc additionner deux points P et Q même si $P = P_\infty$, $Q = P_\infty$, $P + Q = P_\infty$ (c'est-à-dire $P = -Q$), ou $P = Q$ (c'est-à-dire un doublage). Cette addition ne requière que 11 multiplications sur $GF(p)$ (10 pour les courbes de Edwards tordues avec 4 coordonnées par point).

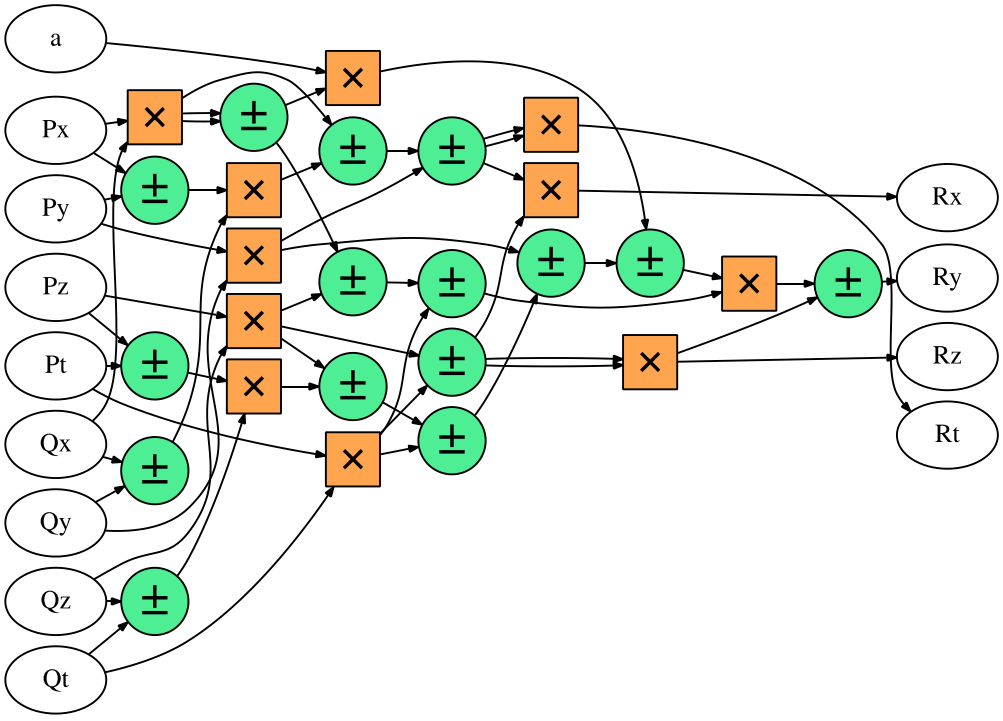


Figure I-23 Exemple d'ordonnancement pour l'addition complète et unifiée de points R=P+Q sur $JQ_{a,1}$

L'implémentation de cryptographie basée sur les courbes elliptiques nécessite de vérifier que le point en entrée est bien un point de la courbe elliptique utilisée [BiMe00]. Pour une courbe d'ordre premier, il suffit de vérifier que le point d'entrée vérifie bien l'équation de la courbe elliptique. Pour des courbes à cofacteur supérieur à 1, comme les courbes d'Edwards ou les courbes quartique de Jacobi, il faut vérifier que le point est aussi dans le bon groupe pour éviter une attaque exploitant les petits sous-groupes [AnBr03]. Je n'étudierai pas cette étape de vérification préliminaire à la multiplication scalaire. Je renvoie le lecteur intéressé vers une méthode permettant de fusionner les étapes de décompression et validation de point [Hamb15].

5-C. Analyse différentielle de consommation

Lorsque l'attaquant peut obtenir plusieurs traces d'une multiplication scalaire utilisant le même scalaire secret, il peut mettre en œuvre une attaque DPA (Differential Power Analysis) [KoJa99]. Elle a été généralisée sur les courbes elliptiques [Coro99]. Contrairement à la SPA, la DPA utilise plusieurs traces de puissance. Une analyse statistique de ces traces permet à l'attaquant de retrouver la clé privée. La Figure I-24 décrit une attaque par DPA contre l'algorithme « Doublages et Additions Systématiques ». Le principe est le suivant : l'attaquant observe passivement le circuit pendant une grande quantité de multiplications scalaires avec le même scalaire privé. Il sauvegarde pour chaque opération $k \times P_i$, la trace de puissance et le point opérande P_i. L'objectif est de retrouver la valeur de k. La découverte des bits du scalaire se fait de gauche à droite. La Figure I-24 décrit une attaque sur l'algorithme présenté dans la Figure I-21 (page 40) où l'attaquant connaît déjà 4 bits de poids fort du scalaire : $(1011)_2$. Il cherche à connaître la valeur du bit suivant : k_3. L'attaquant prévoit la valeur d'un bit dans l'algorithme de multiplication scalaire dépendant de P_i et de k_3 : ici nous avons choisi un bit du point accumulateur au tour de boucle numéro 5. Cette prévision nécessite de faire un pari sur la valeur de k_3 : c'est une hypothèse statistique. Chaque trace de puissance peut alors être classée en deux groupes, en fonction de la valeur du bit de l'accumulateur. La trace moyenne de chaque groupe est alors calculée. Si un pic apparaît dans la différence entre ces deux traces moyennes, cela confirme la prévision faite sur la valeur de k_3. Dans le cas inverse, la prédiction était surement mauvaise, ceci peut être vérifié en réalisant la même procédure avec un pari différent sur k_3. En réalisant cette opération sur chaque bit du scalaire il est possible de déterminer la clé privée.

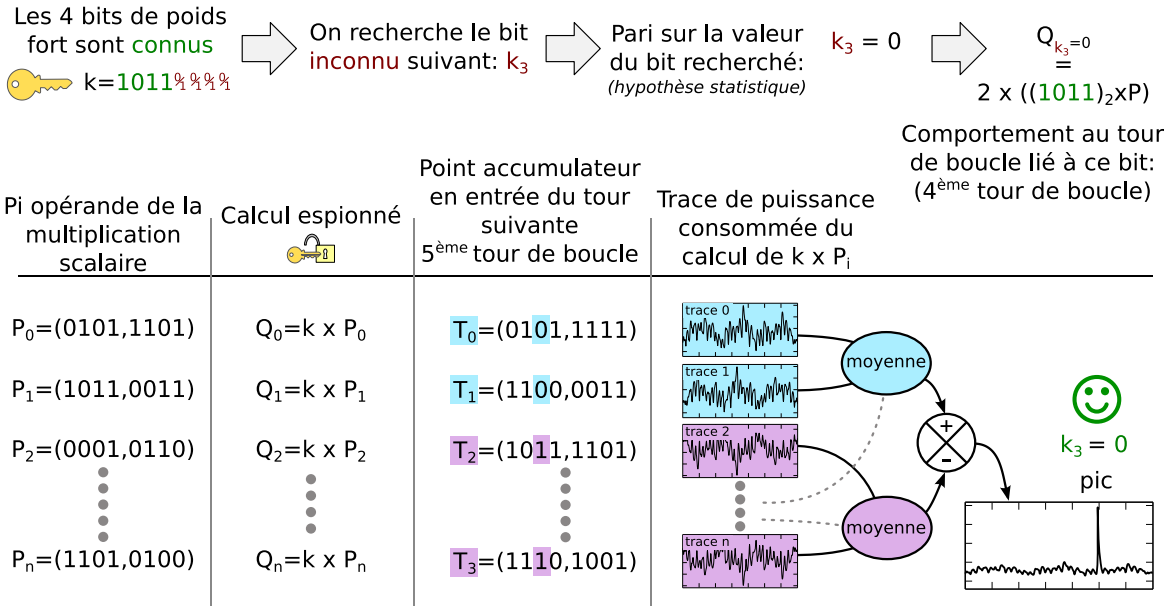


Figure I-24 Attaque DPA contre une multiplication scalaire avec l'algorithme « Doublages et Additions Systématiques »

Pour réaliser cette attaque il faut être capable d'obtenir plusieurs, voire un grand nombre de traces de multiplications scalaires utilisant le même scalaire secret et connaître l'algorithme utilisé pour pouvoir prédire une valeur interne. Il faut aussi que les traces soient bien synchronisées pour pouvoir les comparer. Pour se protéger d'une telle attaque il est possible de rendre la prédiction des variables internes à partir de P_i et k_i hasardeuse en utilisant un algorithme qui prend des choix de façon aléatoire. L'ajout de hasard dans la multiplication scalaire est masqué par l'opération de moyenne dans l'attaque, mais cela ralentit l'attaque car un plus grand nombre de traces est nécessaire. On peut aussi généraliser cette méthode en PPA (Partitioning Power Analysis) dans laquelle on divise les traces en plus de deux groupes en pondérant chacun des groupes. Pour lutter contre les contre-mesures par masquage des variables internes, on peut attaquer plusieurs instants : c'est la DPA d'ordre supérieur à 1. Une autre amélioration est la CPA (Correlation Power Analysis) qui utilise un critère d'évaluation basé sur la corrélation.

Les contre-mesures classiques à ce type d'attaque sont l'utilisation de représentations de points aléatoires (choix aléatoire de la coordonnée Z dans un système projectif) ou le masquage du scalaire en ajoutant à celui-ci le produit d'un petit nombre aléatoire et de l'ordre du générateur [Coro99]. Ces contre-mesures ralentissent le calcul de la multiplication scalaire car elles ajoutent des opérations. Je vais maintenant m'intéresser à des contre-mesures qui permettent de fusionner des optimisations de performance présentées dans la section I-3-C (page 36) avec une contre-mesure aux attaques par canaux auxiliaires.

Il est possible d'utiliser la possibilité de facilement effectuer une soustraction entre deux points d'une courbe elliptique. Ceci peut s'appliquer à un algorithme de multiplication scalaire de droite à gauche [OsAi01]. Mais la protection ne s'applique qu'aux zones du scalaire contenant des bits non-nuls consécutifs. Cette contre-mesure ne peut pas protéger de manière équitable toutes les zones du scalaire secret, notamment les zones contenant beaucoup de bits nuls. Une approche similaire a été proposée sur un algorithme parcourant les bits de gauche à droite [HaMo02] : c'est une multiplication scalaire NAF avec une forme non-adjacente relaxée. Cette protection nécessite une première étape de recodage (les bits sont parcourus de droite à gauche) suivie de la multiplication scalaire (où le scalaire recodé est parcouru de gauche à droite). La valeur secrète intervient dans deux opérations, le recodage et la multiplication scalaire. L'avantage de cette contre-mesure est qu'elle ne nécessite que quelques additions et doublages supplémentaires par rapport à l'algorithme « Doublages et Additions Systématiques ». Si l'attaquant peut choisir le point opérande, des attaques par collisions peuvent être menées [FoMu04]. Par contre si le recodage aléatoire du scalaire impacte les représentations projectives du point accumulateur (voir section I-3-B page 32) la détection de collisions semble compromise.

Il existe aussi des contre-mesures exploitant le gain en performance que peut apporter le fenêtrage. Nous avons vu dans la section I-3-C que cette méthode basée sur des points pré-calculés est difficilement applicable aux algorithmes de multiplication scalaire de droite à gauche. Une contre-mesure à l'attaque DPA utilisant une table de points et un calcul de droite à gauche a tout de même été proposée [Möll02]. Le hasard est introduit dans l'étape initiale qui consiste à remplir une table de 2^w points aléatoires de la courbe elliptique tels que leur somme donne le point à l'infini P_∞. À chaque tour de boucle, un digit de w bits est lu pour savoir à quel point de la table l'accumulateur sera ajouté. L'accumulateur qui contenait initialement P est alors doublé w fois. Une étape finale sur les points de la table permet de calculer le résultat final de la multiplication scalaire. L'étape initiale de remplissage de la table étant un problème difficile, elle ne peut pas être réalisée en temps réel. Il faut donc préalablement calculer hors ligne des tables de points initiaux, ce qui a un coût en espace mémoire et limite le nombre d'états initiaux possibles.

La technique de fenêtrage est plus facile à appliquer dans des algorithmes de multiplication scalaire de gauche à droite. Par exemple, il est possible d'ajouter du hasard dans la table de points pré-calculés : ceci peut être compensé par un recodage du scalaire, c'est la méthode RT-WM [ItYa02]. Il est aussi possible d'agir directement sur la multiplication scalaire en partitionnant le scalaire en fenêtres qui se chevauchent. Les bits du scalaire peuvent être

présents dans deux fenêtres ; cette redondance peut servir à ajouter du hasard en masquant un ou plusieurs bits dans une fenêtre et en les démasquant dans la fenêtre suivante : c'est la méthode O-WM. Il est aussi possible d'utiliser des fenêtres de taille aléatoire dans le découpage du scalaire en digit. La contre-mesure proposée dans [AhHa03] utilise des fenêtres de 1, 2 ou 3 bits. Pour assurer de bonnes performances, les tailles de fenêtres n'ont pas la même probabilité d'occurrence. Etant donné que des doublages fictifs sont utilisés pour masquer les fenêtres de petites tailles, les fenêtres de 1 ou 2 bits dégradent les performances de la multiplication scalaire : les auteurs ont donc choisi qu'elles seraient moins probables. Cette disparité dans les probabilités d'occurrence pourrait être utilisée par un attaquant pour accélérer une attaque par canaux auxiliaires. Le fenêtrage aléatoire peut aussi être combiné avec la soustraction de points pour obtenir un algorithme wNAF modifié [LiSm01].

Je proposerai une nouvelle protection par fenêtrage aléatoire dans le Chapitre IV. Je montrerai comment une utilisation astucieuse d'opérations fictives peut perturber une attaque par canaux auxiliaires tout en minimisant l'information qu'un attaquant peut obtenir avec une attaque par « C safe-error ».

5-D. Analyse horizontale de consommation

Pour lutter contre l'attaque SPA, il est possible d'utiliser des formules unifiées (voir section I-5-B page 43). Si cette contre-mesure est utilisée seule pour protéger un algorithme « Doublages et Additions » (voir Figure I-18, page 36), l'attaque par SPA simplement basée sur une détection de motifs ne permet plus de distinguer les opérations d'addition des opérations de doublage. Il est tout de même raisonnable de penser que des attaques plus évoluées peuvent être mises en œuvre. Je vais m'intéresser ici aux études existantes basées sur la comparaison de traces d'opérations basiques (comme l'addition de points) extraites d'une seule trace provenant d'un canal auxiliaire. On nomme ce type d'attaques, « attaques horizontales » car elles comparent entre eux des extraits d'une seule trace, les attaques verticales étant celles où plusieurs traces différentes sont comparées entre elles.

Le pendant de la multiplication scalaire dans le système RSA est une exponentiation par un exposant secret. Cette opération peut être réalisée par plusieurs algorithmes : le « Élévations au carré et Multiplications » ou le « Élévations au carré et Multiplications Systématiques » qui sont respectivement les pendants des algorithmes « Doublages et Additions » et « Doublages et Additions Systématiques ». Un multiplieur peut toujours être utilisé pour réaliser l'opération d'élévation au carré : contrairement aux courbes elliptiques, les deux opérations basiques sont naturellement unifiées. Il existe de nombreux autres travaux sur les attaques

horizontales contre le système RSA pour, par exemple, différentier les opérations de multiplication des opérations d'élévation au carré. Je présenterai ici un état de l'art des attaques horizontales sur les systèmes RSA et ECC, par espionnage de la puissance consommée ou du champ électromagnétique émis. Il existe aussi de nombreuses attaques où des collisions sont provoquées en choisissant un opérande du calcul (exemples [YeKo05; HoMi08; DuPa16]) mais ceci sort du cadre de mon étude.

Tableau I-2 Attaques horizontales contre implémentations ECC ou RSA

Réf.	Crypto.	Cible	Source de la fuite	Nombre de traces	Identification	Méthode
[AmFe09]	128-bit Montgomery ou mult. naïve par digits	ARMv7	Produit avec radix 2^{16}	3000 traces de puissance	Différentiation de l'élévation au carré et de la mult.	Comparaison de moyenne de sous-traces
[Walt04]	ECC P-192	Attaque théorique	Mult. De Montgomery $GF(p)$	1 trace de puissance niveau de sécurité de 96 à 17.6 bits	Identification de l'addition de points	Détection de la réduction finale
[StTh06]	ECC (formule unifiée)	Attaque théorique	soustraction $GF(p)$	Amélioration de l'attaque [Walt04]	Identification de l'addition de points	Détection dans la soustraction
[Walt01]	RSA m-ary (1024-bit)	Simulation d'un produit de digits	Produit avec radix 2^k (ex. $k = 32$)	1 trace de puissance taux d'erreur 0%	Différentiation de l'élévation au carré et de la mult.	Diminution de l'influence d'un opérande par moyenne astucieux
[ClFe12]	RSA (1024-bit)	Modèle de fuite	Mult. naïve par digits radix 2^{32} (ex. $k = 32$)	1 trace de puissance taux d'erreur 0% (sans bruit) taux d'erreur 10% (avec bruit)	Identification de l'élévation au carré	Comparaison de $\boldsymbol{x_i} \cdot y_j$ et $x_j \cdot \boldsymbol{y_i}$
[BaJa15]	ECC (formule unifiée)	Plusieurs modèles de fuite	Mult. naïve par digits radix 2^k	1 trace de puissance Etude de la Qualité en fonction du SNR	Identification de l'addition de points	Opérande Commune dans des mult. sur $GF(p)$
[Wiva11]	RSA-2048 Élévations au carré et Mult. Systématiques	Modèle de fuite et Carte à puce non identifiée	Mult. sur $GF(p)$ Architecture Inconnue	5000 traces électro-magnétiques	Identification de mult. fictives	Comparaison de mult. successives
[Helb14]	ECC Échelle de Montgomery	FPGA Spartan 3	Position mémoire des points	1 trace 9 positions de sonde électro-magnétique	Attaque sur l'échelle de Montgomery	Détection de la localisation des données
[Pelm14]	RSA avec base RNS	FPGA Spartan 3^E	Adresse Mémoire liée à un bit secret	1 trace électro-magnétique	Attaque sur l'échelle de Montgomery	Masquage Trace unique avec points d'intérêt
[VaDr15]	ECC (ECDSA)	FPGA Cyclone III produit de digits 8-bit	2 produits 8-bit	256 traces de puissance avec un point choisi	Identification de la clé secrète	Collision entre deux produits 8-bits

Le Tableau I-2 contient une liste non exhaustive d'attaques horizontales contre des implémentations ECC et RSA. L'origine de la fuite provient la plupart du temps du multiplieur sur $GF(p)$. Les deux types de multiplieurs attaqués sont des multiplieurs naïfs (« schoolbook-method ») par digits qui nécessitent une étape de réduction supplémentaire (voir section I-3-A.i page 27) ou des multiplieurs de Montgomery (voir section I-3-A.iii page 29).

La première attaque horizontale a été proposée par Colin D. Walter contre une implémentation RSA en 2001 [Walt01]. Le même auteur a aussi proposé une attaque contre la contre-mesure à formule unifiée pour ECC [Walt04], il a montré que la multiplication scalaire peut être attaquée si l'attaquant est capable de détecter la réduction finale dans un multiplieur de Montgomery. En effet, en utilisant la formule unifiée proposée dans [BrJo02], la probabilité de réductions finales dans l'addition de points est différente de celle dans le doublage de point. Cette attaque a été étendue à la soustraction sur $GF(p)$ [StTh06]. Dans les deux situations, identifier la réduction finale semble difficile pour des canaux auxiliaires à faible bande passante. De plus, il existe des contre-mesures simples : il est possible d'éliminer l'étape de réduction finale dans un multiplieur Montgomery (voir fin de section I-3-A.iii page 29); l'implémentation matérielle d'une réduction consiste à effectuer une soustraction par p et conserver le résultat ou non en fonction de la retenue sortante, donc dans la plupart des cas la soustraction sera toujours faite et l'attaquant doit tenter de détecter si le résultat a été sauvegardé ou non. Cette attaque semble impossible à mettre en œuvre en présence de la contre-mesure par représentation projective aléatoire du point opérande de la multiplication scalaire.

Un autre type d'attaque exploite des fuites à un plus haut niveau dans le multiplieur. Par exemple avec un multiplieur basé sur la méthode naïve par digits, un attaquant peut essayer de différencier les élévations au carré des multiplications, ces deux opérations étant réalisées avec le même multiplieur. Cette différenciation a été proposée en comparant les traces de puissance consommée de chaque multiplication/élévation au carré [ClFe12]. Cette méthode est aussi efficace contre un multiplieur de Montgomery [AmFe09]. Ce type d'attaque horizontale basé sur la comparaison d'extraits d'une trace se divise en trois étapes : un traitement initial sur la trace ou sur les sous-traces correspondant aux opérations comparées ; ensuite, le calcul d'un élément de distinction permet de comparer les sous-traces des opérations ; enfin, une dernière étape de classification identifie les opérations (élévations au carré et multiplications ou additions et doublages). Le Tableau I-3 liste les méthodes d'attaques et les sources de fuites exploitées par différentes attaques horizontales. Contre les

formules unifiées, deux types d'attaques sont possibles : détecter des multiplications avec des opérandes communes [BaJa15] ou différencier les élévations au carré et les multiplications [AmFe09].

Tableau I-3 Les 3 étapes d'une attaque horizontale

Réf.	Traitement	Distinction	Classification
[Walt01]	Moyenne astucieuse	Distance euclidienne	AHCA modifié
[ClFe12]	Concaténation astucieuse	Corrélation horizontale	Seuil
[Wiva11]	Compression	Corrélation verticale	Seuil
[Helb14]	Décimation	Distance euclidienne	« k-mean »
[Peim14]	Points d'intérêt	Distance euclidienne	« k-mean » flou

Toutes les attaques présentées peuvent être utilisées contre la contre-mesure par formule unifiée mais, comme elles attaquent le multiplieur sur $GF(p)$, l'attaquant doit connaître ou deviner quel type de multiplieur est utilisé et comment sont ordonnancées les multiplications dans la formule unifiée. Dans le Chapitre III, je me focaliserai sur un scenario où l'attaquant détecte que la contre-mesure par formule unifiée est utilisée (par simple observation de la puissance consommée par exemple) mais ignore comment la formule unifiée a été implémentée et ne connait pas l'algorithme de multiplication sur $GF(p)$ utilisé. Les attaques sur le multiplieur $GF(p)$ semblent difficiles à mettre en œuvre dans ce scenario mais ces attaques peuvent être modifiées pour attaquer directement la réutilisation d'opérande (le point P) dans la formule unifiée permettant de réaliser l'addition de points. L'attaque que je présente dans le Chapitre III est aussi basée sur les trois étapes présentées plus haut. Je vais donc décrire les différentes méthodes existantes pour chacune de ces étapes.

La première étape des attaques horizontales par comparaison d'extraits de trace nécessite de découper la trace de puissance consommée en sous-traces. Chaque sous-trace est associée à une opération basique (multiplication sur $GF(p)$ ou addition de points). Ce découpage peut être fait par détection de pics dans l'autocorrélation de la trace mais reste une opération délicate en présence de bruit. Ce partitionnement de la trace peut être suivi d'un traitement sur les sous-traces. Un premier exemple de traitement est une compression avec perte pour diminuer la quantité de données traitées par les étapes suivantes et accélérer l'attaque : l'attaque décrite dans [Wiva11] est un cas extrême de compression puisque chaque sous-trace est compressée en un point unique représentant l'énergie de l'opération basique. À cause de la perte d'information dans la compression, un nombre important de traces est nécessaire (5000) pour réussir cette attaque contre une implémentation RSA. Cette attaque n'est donc pas

purement horizontale. Une compression plus classique est proposée dans [HeIb14] : le champ électromagnétique est mesuré avec un échantillonnage rapide mais le résultat est compressé pour diminuer la quantité de données. Un traitement astucieux a été proposé dans [Walt01] : grâce à la connaissance de l'implémentation utilisée pour le multiplieur sur $GF(p)$ (multiplication naïve par digits), l'attaquant peut éliminer la contribution du second opérande dans les sous-traces. Cette méthode permet d'améliorer la qualité de la mesure permettant de distinguer les opérations avec un premier opérande commun, mais cela nécessite une grande connaissance des choix d'implémentations faits dans la cible. En effet, cette technique nécessite de découper la sous-trace en sous-sous-traces pour pouvoir effacer au moins partiellement l'influence des digits du second opérande par un procédé basé sur des moyennes. Une autre méthode exploitant ces sous-sous-traces associées à la multiplication élémentaire entre deux digits a été proposée dans [ClFe12]. Aucune compression n'est utilisée mais de nouvelles sous-traces sont construites par réorganisation des sous-sous-traces. Cela permet de détecter la symétrie des digits du premier et du second opérande dans une élévation au carré. Un autre traitement, difficile à mettre en œuvre mais très efficace est d'identifier des points d'intérêts dans les sous-traces [PeIm14].

Dans la seconde étape de l'attaque, ces sous-traces (améliorées ou non) sont utilisées pour faire des mesures de similarité ou de dissimilarité. La mesure la plus commune est la distance euclidienne entre deux sous-traces qui donne une mesure de dissimilarité entre les deux opérations associées à ces sous-traces [HeIb14; Walt01]. Une autre technique consiste à mesurer la similarité par calcul de la corrélation (sans déphasage) entre ces sous-traces. Deux attaques exploitent cette distinction et sont présentées dans [ClFe12] : les auteurs utilisent le facteur de corrélation de Pearson, qui dépend du poids de Hamming des mots manipulés sur un modèle de fuite donné. Cela permet de faciliter les études théoriques avant d'implémenter une attaque réelle. Hélas, il reste impossible de distinguer deux mots ayant le même poids de Hamming dans ce modèle alors que l'on sait que c'est possible en réalité [VaDr15] : il faut donc garder à l'esprit que ce type de modèle peut sous-estimer les fuites réelles d'information. La mesure de corrélation est aussi utilisée dans [Wiva11], mais cette fois sur la dimension verticale (sur plusieurs traces à un même instant) de l'attaque. Ceci est seulement possible car chaque sous-trace a été comprimée en un seul échantillon. Dans le Chapitre III, je montrerai qu'autoriser du déphasage dans la mesure par corrélation présente des avantages.

Je tiens à rappeler que les attaques horizontales sont très puissantes car elles peuvent potentiellement permettre de retrouver un secret en exploitant une seule trace. L'avantage est donc qu'elles peuvent être montées avec succès même dans certains cas où du hasard a été

ajouté pour contrer une attaque par DPA. Par exemple, si la représentation du point opérande est choisie aléatoire en début de multiplication scalaire, l'attaque horizontale est immune à cette contre-mesure. Pour le cas d'une signature par le protocole ECDSA (voir Figure I-6, page 24), le scalaire r est un nombre aléatoire. Ceci empêche l'utilisation de la verticalité (cas d'une attaque DPA) car le scalaire sera différent pour chaque trace. Si une attaque horizontale permet de retrouver le scalaire aléatoire r utilisé et que l'attaquant connaît la signature résultante et le message signé, alors il peut facilement retrouver la clé privée k utilisée :

$$k = \frac{r \cdot s - h}{Q_x}$$

Il faut donc maximiser l'exploitation de la mesure des critères de distinction obtenus pour utiliser le moins possible la dimension verticale avec pour objectif la réalisation d'une attaque en une seule trace. C'est la raison d'être de la troisième étape des attaques horizontales qui consiste à appliquer un algorithme de classification pour différencier les multiplications des élévations au carré (ou les additions de points des doublages de points) à partir des mesures obtenues dans l'étape numéro 2. Un algorithme de classification peut permettre de prendre des décisions sans utilisation de seuils manuels qui sont difficilement définissables car dépendants de la cible.

La première attaque horizontale publiée [Walt01] utilise un algorithme de classification basique pour identifier la réutilisation d'opérande dans un algorithme d'exponentiation utilisant l'algorithme « m-ary » pour un système RSA. Les élévations au carré sont détectées car elles ne sont proches d'aucune autre opération, mais cette estimation nécessite de définir un seuil. Toutes les opérations restantes sont des multiplications qui sont associées à des digits différents de l'exposant secret. Elles sont donc classées en $2^m - 1$ groupes, chacun associé à une valeur pré-calculée correspondant à une valeur possible de digit de l'exposant secret. Pour réaliser cette classification, l'auteur propose un algorithme de classification hiérarchique ascendant [John67] (AHCA) modifié ; contrairement aux algorithmes hiérarchiques classiques, le nombre de groupes est restreint artificiellement. Une alternative aux algorithmes hiérarchiques a été proposée, il s'agit de l'algorithme « k-mean » [Maot67]. Les auteurs de [HeIb14] utilisent cet algorithme itératif qui minimise une valeur de cohérence des groupes pour retrouver le secret utilisé dans des traces de multiplications scalaires sur cible FPGA. La multiplication scalaire implémentée utilise l'échelle de Montgomery (voir 3-C) et des fuites électromagnétiques localisées sont exploitées pour identifier les bits du scalaire. Les auteurs soulignent la difficulté de mettre en œuvre un algorithme de classification multi-variables.

Une version modifiée de l'algorithme « k-mean » nommé « k-mean flou » est proposé dans [PeIm14] : il autorise des sous-traces (ou opérations basiques) à être plus ou moins dans un ou plusieurs groupes. Une première classification infructueuse permet de raffiner l'attaque en identifiant des points d'intérêt. Une seconde classification permet alors d'obtenir de meilleurs résultats. Comme cet algorithme est flou, la sortie est une probabilité pour chaque opération d'être dans un groupe ou un autre, en lieu et place d'une classification booléenne dure dans un groupe. Grâce au raffinement par sélection de points d'intérêt, l'attaque permet de retrouver l'exposant secret d'une implémentation RSA par l'algorithme « Élévations au carré et Multiplications Systématiques » en exploitant une fuite dans le canal électromagnétique. Même si une classification échoue, le résultat de l'attaque peut servir à orienter une attaque par force brute sur le secret [Wiva11].

Le Tableau I-2 montre que les attaques réelles purement horizontales (en une seule trace) sont rares et exploitent toujours l'information de localisation disponible dans les fuites électromagnétiques [HeIb14; PeIm14]. Réussir une attaque purement horizontale par analyse de puissance semble difficile car ce canal est global à tout le circuit ; il n'y a donc pas d'information de localisation. La plupart des attaques visent un multiplieur par digit sur $GF(p)$: multiplieur naïf par digits ou multiplieur de Montgomery par digits. Je décrirai dans le Chapitre II un multiplieur de Montgomery série-parallèle. Cela signifie que le premier et le second opérande seront manipulés de manière complétement différente (en série et en parallèle). Cette asymétrie apporte une protection contre certaines attaques basées sur la détection des élévations au carré. Par exemple, les attaques comparant un produit partiel ($\boldsymbol{a_i} \times b_j$) avec son complémentaire ($a_j \times \boldsymbol{b_i}$) ne peuvent être mises en œuvre que contre un multiplieur par digits [ClFe12]. Il s'agit d'identifier les élévations au carré ($a = b$) en identifiant que ces deux produits partiels sont identiques. Un seul article mentionne une attaque contre la contre-mesure par formule unifiée sur les courbes quartiques de Jacobi [Walt04] mais l'attaque suppose l'utilisation d'un multiplieur par digits. Dans le Chapitre III, je montrerai qu'une attaque réelle non-supervisée par analyse de puissance consommée nécessitant très peu de connaissance de la cible est possible même avec un multiplieur de Montgomery série-parallèle.

6. Conclusion

Il existe une grande variété de courbes elliptiques utilisées en cryptographie. Comme je l'ai montré, des choix techniques visant à améliorer les performances sont souvent incompatibles

avec de nombreuses courbes. Le crypto-processeur au cœur de cette thèse, décrit dans le Chapitre II, privilégie l'interopérabilité en n'utilisant que les optimisations compatibles avec un maximum de courbes elliptiques.

Les contre-mesures existantes pour sécuriser un crypto-processeur basé sur les courbes elliptiques contre les attaques par canaux auxiliaire sont généralement néfastes pour les performances. Dans cette thèse, je m'intéresse aux contre-mesures à faible impact sur les performances. Les possibilités offertes par les courbes quartiques de Jacobi grâce à une formule d'addition unifiée efficace semblent prometteuses. C'est pourquoi j'étudie la sécurité qu'elles apportent réellement vis-à-vis d'attaques par analyse de puissance dans le Chapitre III. Les protections de circuit basées sur des opérations fictives peuvent ouvrir la porte à de nouvelles attaques ; je montre dans le Chapitre IV qu'en les utilisant de manière originale, il est possible de mettre en œuvre une contre-mesure avec un impact minimal ou même positif sur les performances.

Chapitre II Architecture d'un crypto-processeur basé sur les courbes elliptiques

Dans ce chapitre, je décris l'architecture du crypto-processeur développé pour valider l'attaque du Chapitre III et la contre-mesure du Chapitre IV. Les multiplieurs par digits ayant fait l'objet de beaucoup d'études, nous nous focaliserons ici sur un multiplieur série-parallèle. L'architecture que je présente est le fruit d'un travail où la minimisation du coût, la généricité et la rapidité de conception ont été favorisées par rapport aux performances du système pour obtenir un crypto-processeur de référence pour les chapitres suivants. L'architecture que j'ai conçue est divisée en plusieurs niveaux hiérarchiques (voir Figure I-1, page 15) et ce chapitre la décrit de bas en haut. Elle contient les opérateurs sur $GF(p)$ et/ou $GF(2^d)$; un bloc réalisant les opérations d'addition et de doublage de points, et un autre pour la multiplication scalaire par fenêtrage. Le résultat de mon travail de conception est un crypto-processeur simple ou double corps (supportant les courbes sur $GF(p)$ et sur $GF(2^d)$), pour une courbe elliptique fixée ou non, pouvant supporter plusieurs algorithmes de multiplication scalaire. Ces choix de configuration sont fixés avant synthèse pour passer d'une description générique à crypto-processeur spécifique. Cette implémentation servira de base dans les chapitres suivants pour implémenter des contre-mesures en vue de les évaluer. J'ai publié une première version de l'architecture de ce crypto-processeur [PoMa14d][PoMa14a]. Je n'aborde pas la méthode de vérification utilisée pour valider les circuits conçus. Pour plus détail sur ce point, je renvoi le lecteur vers l'Annexe 2. Dans l'Annexe 3, je démontre qu'une méthode de conception alternative avec une description haut-niveau permet d'accélérer le développement pour obtenir rapidement un crypto-processeur compétitif par rapport à l'état de l'art.

1. Arithmétique sur le corps des coordonnées

Mon travail de conception ayant démarré avant les progrès récents des algorithmes de calcul du logarithme discret sur les courbes elliptiques basées sur le corps $GF(2^d)$ [GaGe14], j'ai initialement choisi l'interopérabilité en développant un crypto-processeur pouvant supporter aussi bien les courbes elliptiques sur le corps $GF(p)$ que sur le corps $GF(2^d)$. Il est tout de même possible de fixer le support d'un seul corps avant la synthèse. La multiplication

sur le corps est réalisée avec un multiplieur de Montgomery série-parallèle, les opérandes sont donc lues de manière asymétrique. Pour minimiser la surface occupée, je n'utilise qu'un seul multiplieur. L'additionneur d'entiers utilisé par le multiplieur est partagé avec l'opérateur d'addition/soustraction sur modulo p ou $R[X]$ toujours dans le but de réduire le coût en minimisant la surface occupée. Comme l'additionneur/soustracteur et le multiplieur sur le corps partagent un opérateur, ils ne peuvent pas être utilisés en parallèle. C'est avec ces contraintes que les opérations de base sur les points (addition et doublage) sont réalisées dans la section suivante.

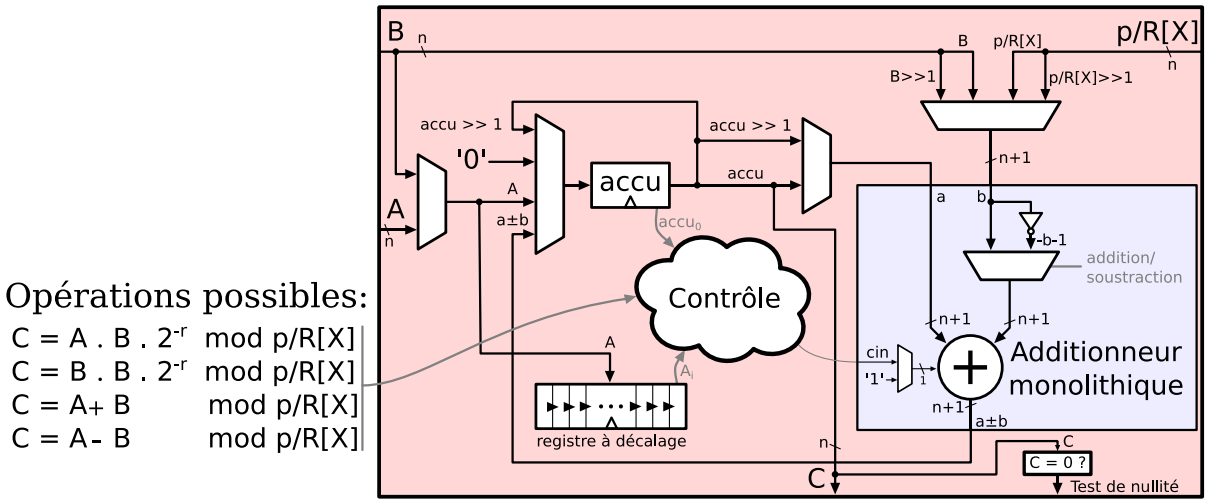


Figure II-1 Architecture de l'unité arithmétique sur le corps : Multiplieur de Montgomery série-parallèle et Additionneur/soustracteur modulo p ou modulo $R[X]$

L'unité arithmétique dont l'architecture est décrite dans la Figure II-1 permet d'effectuer quatre opérations : ajouter deux éléments du corps, soustraire deux éléments du corps, ou effectuer une multiplication ou une élévation au carré dans le domaine de Montgomery. Cette unité est basée sur un additionneur d'entiers monolithique combinatoire. Cet additionneur peut effectuer à chaque cycle l'addition de deux grands entiers avec retenue entrante et sortante. Grâce à la possibilité de choisir l'inverse bit-à-bit du second opérande et en utilisant la retenue entrante, il est possible d'utiliser la représentation en complément à deux pour effectuer une soustraction sur $GF(p)$ (l'addition et la soustraction sont la même opération sur (2^d)). En désactivant la propagation de retenue dans l'additionneur (voir Figure I-9 et Figure I-10, page 26) il est possible de transformer l'additionneur d'entiers en additionneur de polynômes sur $GF(2)$ pour supporter le corps $GF(2^d)$. La description de l'unité arithmétique permet de fixer cette transformation avant synthèse ou de garder la possibilité d'effectuer cette configuration en cours de fonctionnement pour un crypto-processeur supportant les deux corps (« dual fields »).

L'addition et la soustraction sur $GF(p)$ ou $GF(2^d)$ sont réalisées via la partie opérative

décrite Figure II-1. Le contrôle commande la réalisation d'une addition ou d'une soustraction classique suivie d'une soustraction par p dans le corps $GF(p)$. Si, et seulement si, le résultat de cette réduction est positif ou nul (détection par observation de la retenue sortante) alors le résultat est inscrit dans l'accumulateur. Le résultat de l'opération est alors disponible dans l'accumulateur.

Pour simplifier le routage des données et minimiser le nombre de multiplexeurs sur les entrées de l'unité arithmétique (décrites dans la section II-2 page 58), j'ai choisi de permettre une opération d'élévation au carré dans le domaine de Montgomery qui est une multiplication où l'opérande A est remplacé par B. L'opérande A de la multiplication de Montgomery est échantillonné en début de calcul tandis que l'opérande B doit être stable car consulté tout au long du calcul. La valeur de l'opérande A est ensuite lue bit à bit via un registre à décalage. Après une mise à zéro de l'accumulateur, le calcul peut commencer. L'opérande B est ajouté à l'accumulateur et le résultat est inscrit ou non dans l'accumulateur, avant un ajout de p avec ou sans sauvegarde du résultat. Le décalage à droite de 1 bit est effectué en amont de l'opération d'addition pour éviter de calculer le bit de poids faible que l'on sait être nul.

Il faut vérifier que certaines valeurs manipulées ne sont pas nulles pour la détection d'addition de points égaux ou opposés et pour la détection de coordonnées projectives nulles. L'unité arithmétique est donc équipée d'un comparateur indiquant si l'accumulateur et donc la valeur de sortie est nulle.

2. Opérations de base sur les points

L'unité arithmétique décrite dans la section précédente est utilisée par le bloc d'opérations de base sur les points dont l'architecture est décrite dans la section suivante.

2-A. Architecture

La Figure II-2 décrit l'architecture du bloc permettant de réaliser une addition ou un doublage de points. Il contient 3 registres liés à la courbe utilisée qui stockent la valeur du nombre premier p du corps $GF(p)$ ou du polynôme $R[X]$ du corps $GF(2^p)$, les valeurs des deux coefficients a_4 et a_6 pour une courbe de Weierstrass réduite sur $GF(p)$ ou a_2 et a_6 pour une courbe de Weierstrass réduite sur $GF(2^d)$, et la valeur $2^{2r} \mod p$ qui ne peut pas être calculée mais qui peut être vérifiée par deux multiplications de Montgomery par 1, et qui sert à transposer un élément du corps dans le domaine de Montgomery (voir section I-3-A.iii page 29). Ces trois registres apparaissent dans l'architecture du bloc décrite en Figure II-2.

Pour les courbes de Weierstrass, un banc de 9 registres est aussi utilisé. Les 13 registres de la Figure II-2 ont tous la même taille que le plus grand des nombres premiers supporté. Les trois premiers registres du banc notés r_x, r_y et r_z stockent les coordonnées du point servant d'accumulateur. Les 6 autres numérotés de 0 à 5 permettent de stocker les valeurs temporaires durant le calcul d'addition ou de doublage de points. Les derniers éléments de la partie opérative de ce bloc sont des multiplexeurs permettant à la partie contrôle de choisir l'origine des données envoyées à l'unité arithmétique. La partie contrôle a pour rôle d'utiliser cette partie opérative pour réaliser les types d'opérations que sont la vérification de certaines valeurs telles que 2^{2r} ; la transposition des coordonnées Jacobiennes du point accumulateur (r_x, r_y, r_z) entre le domaine classique et le domaine de Montgomery, et le calcul des formules d'addition et de doublage de points aussi bien pour les courbes de Weierstrass sur $GF(p)$ que sur $GF(2^d)$. Si avant synthèse le support d'un des deux groupes est désactivé, alors la partie contrôle est allégée de la branche permettant le calcul des formules sur ce corps et l'unité arithmétique est elle aussi spécialisée pour un seul corps.

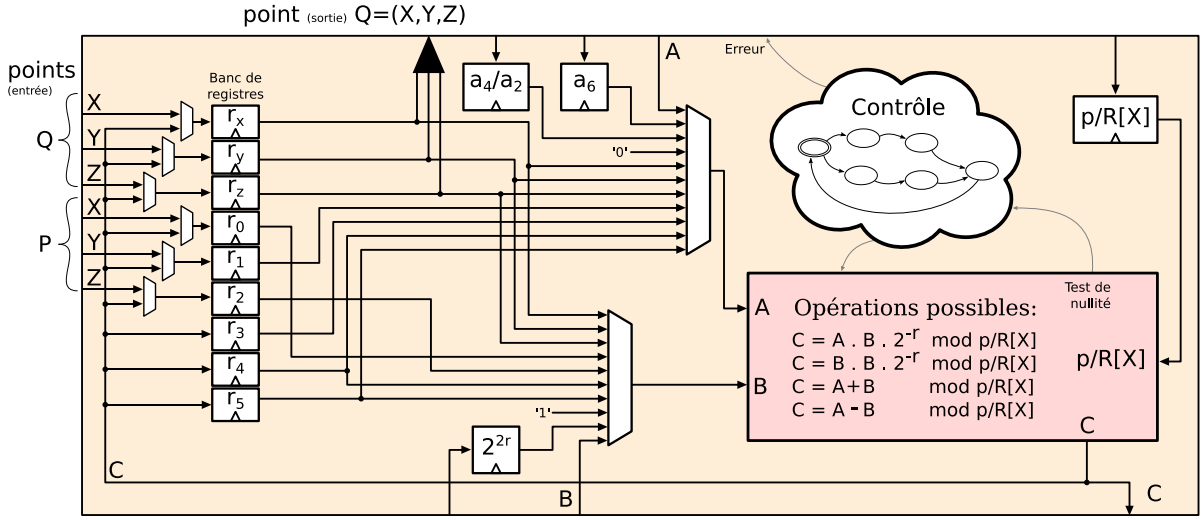


Figure II-2 Architecture de l'opérateur d'addition et de doublage de points sur une courbe de Weierstrass

2-B. Opérations fictives

Pour permettre l'implantation de contre-mesures comme l'algorithme « Doublages et Additions Systématiques » (voir section I-5-B page 40) ou la nouvelle contre-mesure proposée dans le Chapitre IV, le composant décrit dans la Figure II-2 est capable de réaliser des additions ou doublages fictifs. Une opération fictive se comporte comme une opération classique sauf que la désactivation de l'écriture du résultat annihile son effet. Cette désactivation semble détectable par analyse de canaux auxiliaires, c'est pourquoi j'ai choisi de toujours effectuer 9 écritures au lieu de 3. Dans le cas des opérations fictives, ces écritures

sont trois rotations des coordonnées du point accumulateur dans leurs registres. Comme le montre la partie droite de la Figure II-3, cela permet d'effectuer des écritures qui n'ont aucun effet à la fin de l'opération. La partie gauche de la Figure II-3 montre une opération réelle : la même méthode d'écriture dans les registres est utilisée mais à chaque rotation, une ancienne coordonnée du point accumulateur est écrasée par une nouvelle. J'assure ainsi que 9 écritures dans les mêmes registres auront lieu aux mêmes moments que ce soit pour une opération fictive ou pour une opération réelle. Pour une opération fictive, les trois registres sont donc toujours vivants (ils contiennent une valeur que l'on ne peut pas écraser). Bien qu'il soit possible pour une opération réelle que l'un de ces registres soit parfois utilisable pour stocker une valeur temporaire (après la dernière lecture d'une coordonnée du point opérande), je m'interdit cette possibilité pour maximiser la ressemblance entre opérations fictives et réelles.

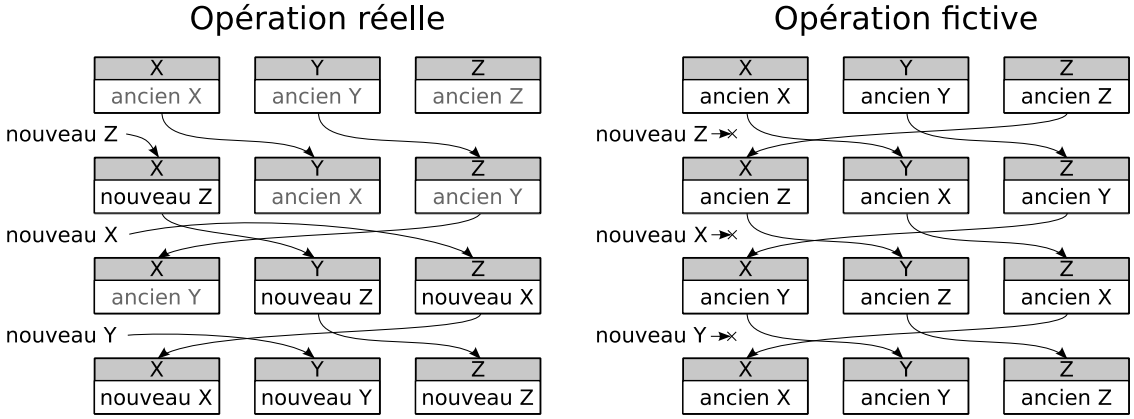


Figure II-3 Rotation des coordonnées de l'accumulateur et opérations fictives

2-C. Ordonnancement et allocation des registres

J'ai conçu ce circuit PCPO (Partie Opérative – Partie Contrôle) avec pour objectif de minimiser la surface occupée. Je n'utilise donc qu'une seule unité arithmétique (décrite dans la section II-1 page 56). Pour pouvoir bénéficier facilement de l'optimisation des performances qu'offrent les méthodes par fenêtrage (avec lecture du scalaire de gauche à droite), j'ai choisi de prendre en considération l'existence d'un seul point accumulateur $Q = (r_x, r_y, r_z)$. En conséquence, les opérations basiques sur les points auront toujours en opérande ce point Q, et le point résultant écrasera les valeurs des anciennes coordonnées de l'accumulateur : $Q = Q + P$ et $Q = 2 \times Q$. Le point P, second opérande de l'addition est inscrit dans les registres temporaires (r_0, r_1, r_2). Ces registres seront utilisés pendant les opérations de doublage et d'addition, le point P devra donc être inscrit dans ces registres avant chaque addition de points. Pour réaliser les opérations d'addition et de doublage de points, il faut au minimum 5 registres en plus des registres (r_x, r_y, r_z). Pour avoir plus de

degrés de liberté dans l'ordonnancement et minimiser la taille des multiplexeurs nous utiliserons un registre de plus que le nombre minimal, c'est-à-dire 6 registres.

Deux multiplexeurs permettent de sélectionner les opérandes de l'unité arithmétique qui peuvent être un registre, une constante liée à la courbe (coefficients de la courbe elliptique et $2^{2r} \bmod p$) ou une valeur constante tel que 0 ou 1. Les ordonnancements finaux et les allocations des registres associées pour l'addition et le doublage de points en coordonnées Jacobiennes sur $GF(p)$ sont décrits dans la Figure II-4 et la Figure II-5.

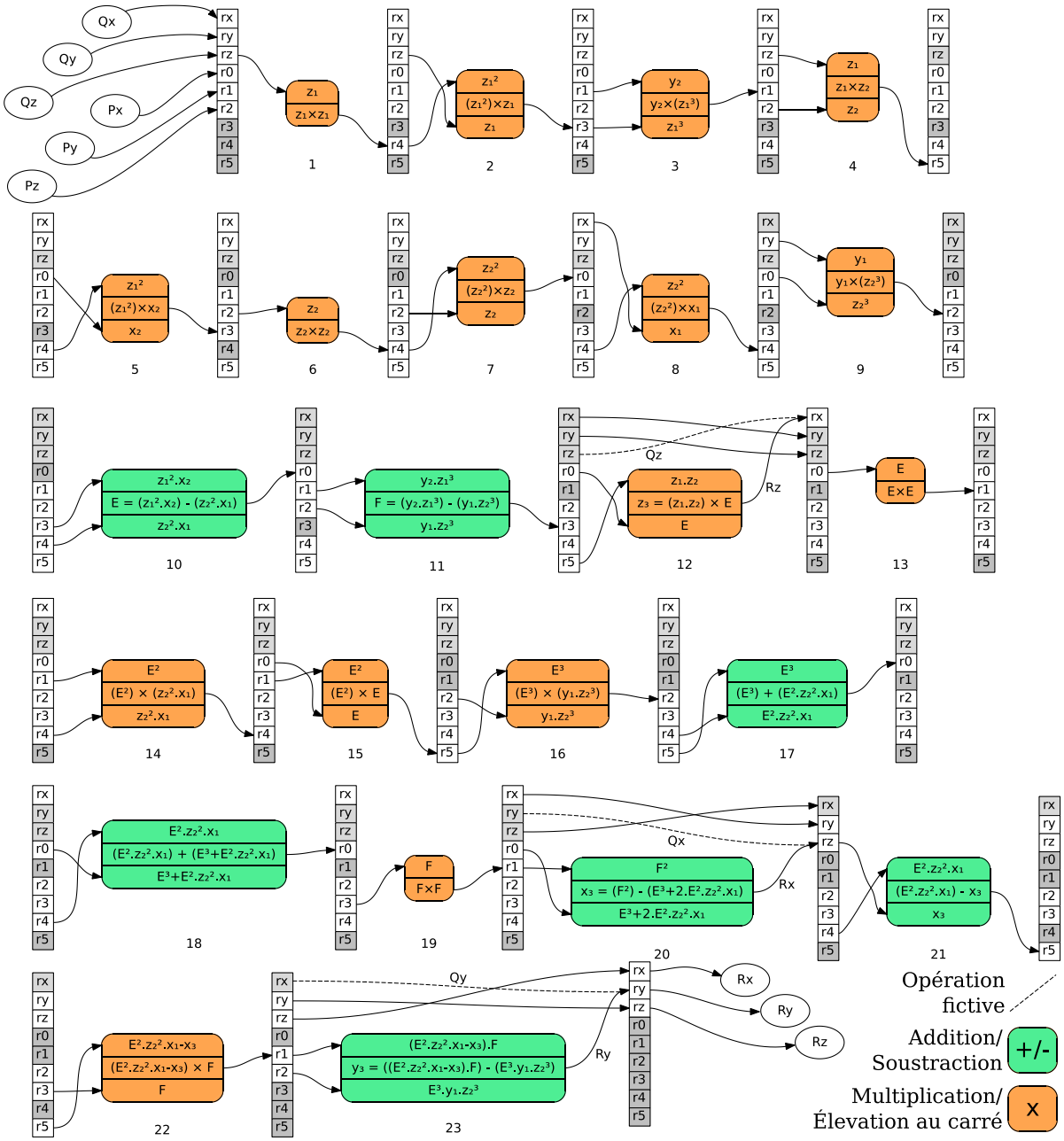


Figure II-4 Ordonnancement et allocation des registres pour l'addition de deux points en coordonnées Jacobiennes sur une courbe elliptique de Weierstrass sur $GF(p)$

Ils ont été obtenus après un premier ordonnancement effectuant les opérations arithmétiques au plus tôt, suivi d'itérations manuelles pour réduire la taille des multiplexeurs en amont de l'unité arithmétique. Comme le montre la Figure II-2 (page 58) décrivant l'architecture du bloc, le multiplexeur en amont de l'opérande A de l'unité arithmétique peut router 2 constantes liées à la courbe, le nombre 0, 7 registres sur les 9 disponibles ou une valeur externe. Le multiplexeur en amont de l'opérande B peut router 1 constante liée à la courbe, le nombre 1, 7 registres sur les 9 disponibles ou une valeur externe.

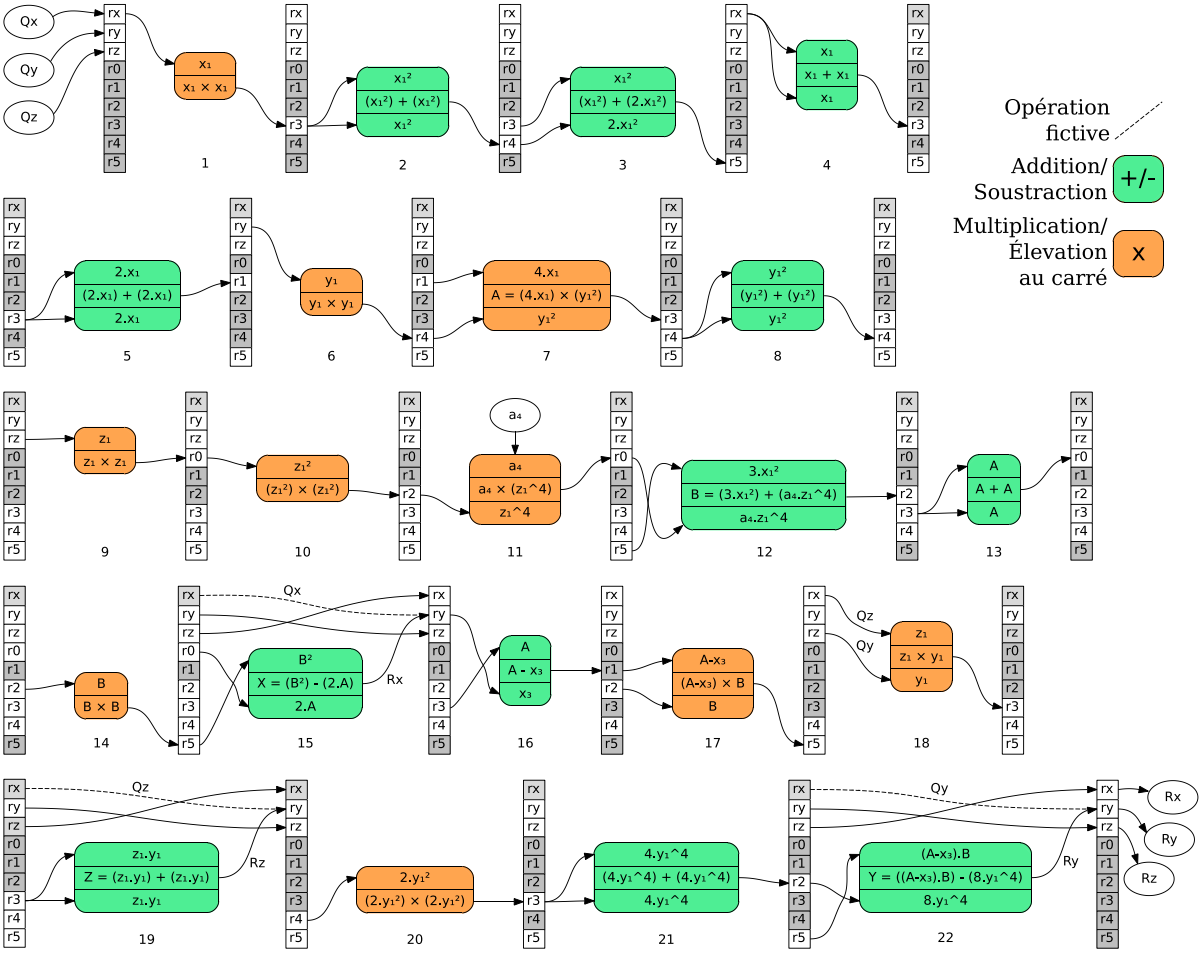


Figure II-5 Ordonnancement et allocation des registres pour le doublage du point accumulateur en coordonnées Jacobiennes sur une courbe elliptique de Weierstrass sur $GF(p)$

La même méthode a été utilisée pour produire un ordonnancement effectuant l'addition et le doublage de points sur $GF(2^d)$. Ce bloc peut être synthétisé pour supporter les deux corps mais il est possible de minimiser la surface occupée en indiquant avant synthèse qu'un seul corps doit être supporté.

2-D. Courbes quartiques de Jacobi

L'implémentation d'un crypto-processeur supportant les courbes quartiques de Jacobi nécessite quelques modifications, notamment des points à 4 coordonnées au lieu de 3 pour le

choix des coordonnées Jacobiennes sur des courbes de Weierstrass. Cette fois, seule l'addition a été implémentée puisqu'elle peut aussi servir à réaliser l'opération de doublage (voir section I-5-B page 43).

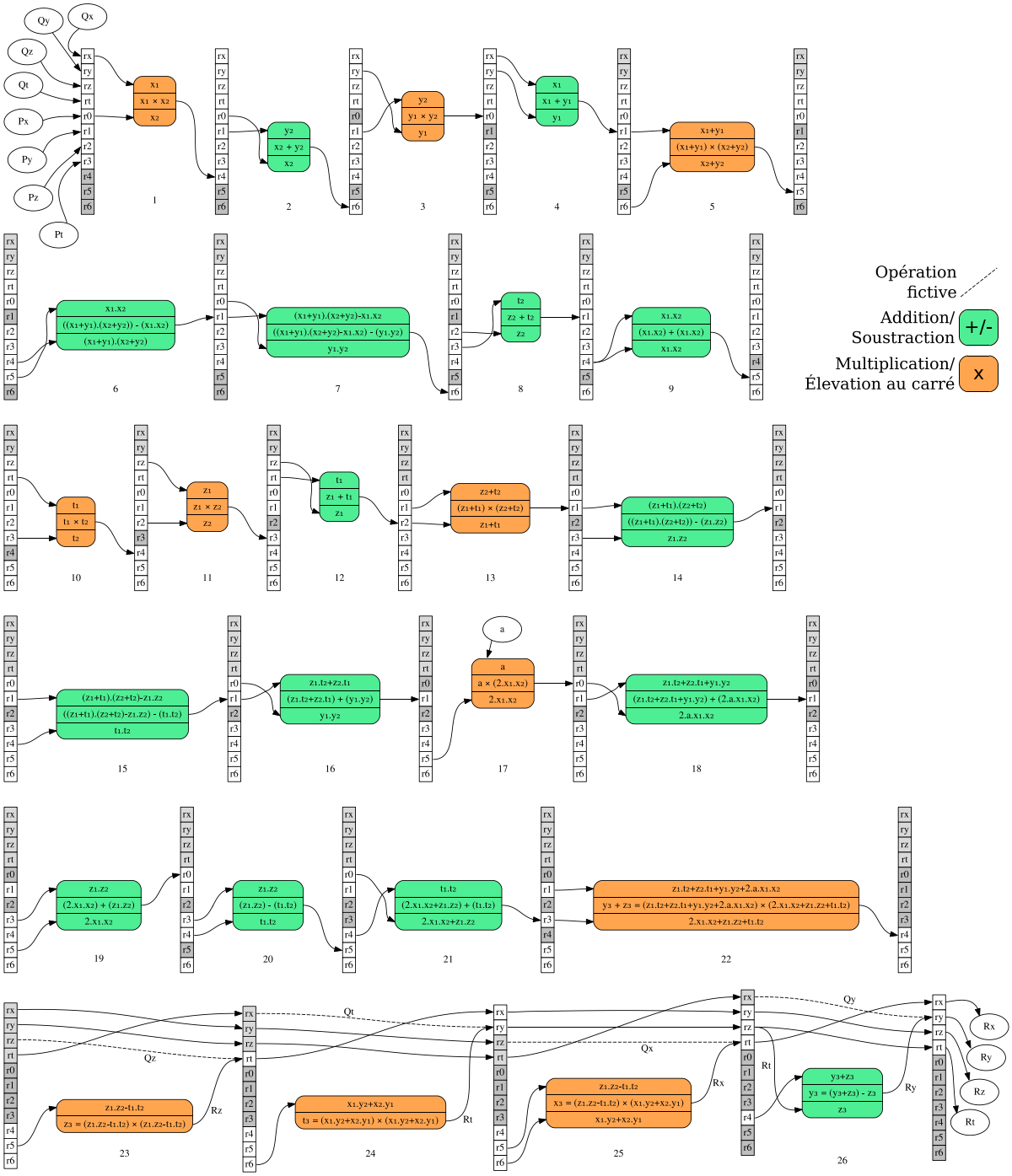


Figure II-6 Ordonnancement et allocation des registres pour l'addition de points sur une courbe quartique de Jacobi.

L'addition de points sur une courbe quartique de Jacobi requiert deux registres supplémentaires par rapport aux additions ou doublages sur les courbes de Weierstrass : c'est-à-dire que le bloc d'addition de points utilise 4 registres pour stocker les 4 coordonnées du point accumulateur et 7 registres pour stocker le second opérande et les valeurs temporaires

durant le calcul. La Figure II-6 fait apparaître un banc de registres entre les opérations 12 et 13. C'est entre ces deux opérations que tous les registres sont utilisés : r_0 à r_6 pour le calcul du point résultat et r_x à r_t en cas d'opération fictive. Les quatre dernières opérations sont accompagnées de rotation des coordonnées du point accumulateur (voir section II-2-B page 59).

3. Multiplication scalaire

L'unité d'addition et de doublage de points décrite dans la section précédente est utilisée par l'opérateur de multiplication scalaire. Cet opérateur a été décrit pour réaliser une multiplication scalaire par fenêtrage (voir section I-3-C page 36) utilisant l'algorithme « m-ary ». La taille de la fenêtre est fixée avant synthèse. Il est aussi possible d'insérer des opérations fictives pour, par exemple, effectuer une multiplication scalaire avec l'algorithme « Doublages et Additions Systématiques » (voir Figure I-21, page 40). Seules les multiplications scalaires de gauche à droite sont supportées.

La Figure II-7 décrit l'architecture de cet opérateur. La partie opérative est constituée de 3 blocs principaux : l'opérateur d'addition et de doublage de points, une mémoire permettant de stocker des points pré-calculés et un registre à décalage stockant et manipulant le scalaire secret. La partie contrôle permet d'effectuer plusieurs opérations. La première opération est l'enregistrement du point P opérande à l'adresse correspondant à $1 \times P$ de la mémoire. Après cette opération le point P est encore représenté dans le domaine classique, ses coordonnées sont donc chargées dans les registres du point accumulateur de l'opérateur d'addition et de doublage de points. C'est cet opérateur qui se charge de déplacer les coordonnées de P du domaine classique vers le domaine de Montgomery. Une fois cette transposition effectuée, la nouvelle représentation de P est inscrite à l'adresse correspondant à $1 \times P$ de la mémoire. Il s'en suit la phase de calcul préliminaire permettant de remplir la mémoire avec tous les points pré-calculés : $2 \times P$, $3 \times P$, …, $(2^m - 1) \times P$. L'unité de multiplication scalaire est alors prête à effectuer une multiplication scalaire avec le point P pour opérande. Le scalaire k est lu et inscrit dans son registre à décalage et la multiplication scalaire par fenêtrage commence (voir Figure I-20, page 37). La lecture itérative des bits de k se fait par décalage de 1 bit du scalaire. Pour chaque décalage une opération de doublage est réalisée sur le point accumulateur. Les bits débordant de ce décalage sont transmis à un registre à décalage de la taille de la fenêtre (m bits). Après m décalages et donc m doublages, le registre de fenêtre est utilisé en adresse de la mémoire pour fournir le point $i \times P$ correspondant et l'ajouter au point

accumulateur. Une fois que tous les bits du scalaire ont été parcourus, l'accumulateur est transposé dans le domaine classique et la lecture de celui-ci permet de récupérer le résultat $k \times P$.

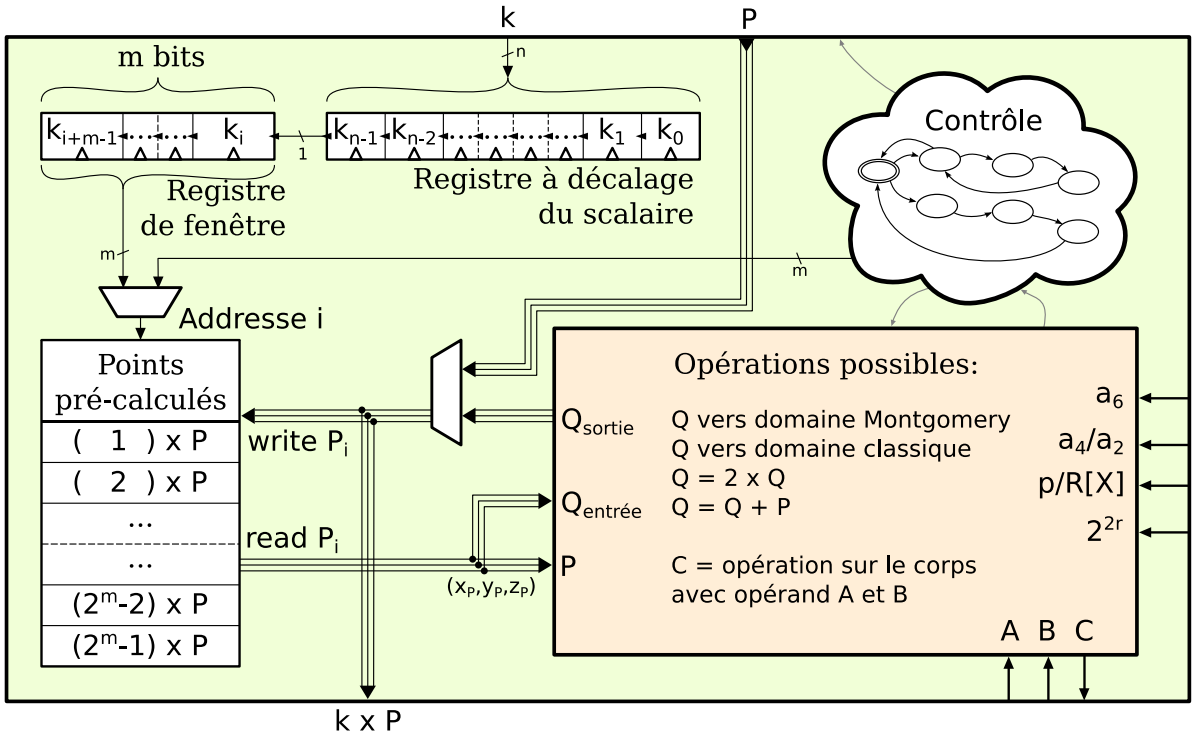


Figure II-7 Architecture de l'opérateur réalisant la multiplication scalaire $k \times P$

Il existe quelques limitations. La première apparaît si quelques bits de poids fort du scalaire sont nuls, les premières opérations de doublage étant alors impossibles à réaliser car elles doivent théoriquement avoir pour opérande le point à l'infini P_∞, mais ceci n'est pas supporté par les formules de doublage sur des courbes de Weierstrass. La seconde limitation survient lorsqu'une fenêtre contenant uniquement des bits nuls est rencontrée : l'addition ne peut pas être réalisée car l'addition par $0 \times P = P_\infty$ n'est pas supportée par la formule d'addition sur des courbes de Weierstrass. Pour éviter les fuites d'information via le temps de calcul et la puissance consommée, lorsqu'une de ces deux limitations est rencontrée, des opérations fictives sans conséquence sur le résultat final sont utilisées. Pour la première limitation, l'accumulateur est initialisé avec un point choisi au hasard dans la table pour pouvoir être doublé ou additionné à un point. Le véritable point initial écrasera l'accumulateur à la première fenêtre non vide rencontrée. Lorsqu'une fenêtre vide (contenant uniquement des bits nuls) est rencontrée, une addition est quand même réalisée : c'est une addition fictive entre le point accumulateur et un point choisi au hasard dans la table de points pré-calculés. Cette contre-mesure est une généralisation de l'algorithme « Doublages et Additions Systématiques » à la méthode de fenêtrage. Ces contre-mesures naïves peuvent être

activées/désactivées avant synthèse. La première limitation peut en réalité être évitée de deux manières : instaurer la contrainte d'utiliser des scalaires avec un bit de poids fort non nul ou ajouter l'ordre du point générateur de manière à obtenir un poids fort non-nul. La seconde limitation (fenêtres vides) peut être évitée en utilisant une formule d'addition complète comme celle que j'utilise pour les courbes quartiques de Jacobi. Il est alors possible d'additionner un point à P_∞. Malheureusement, l'utilisation d'un point de forme spéciale tel que le point à l'infini risque de générer un motif reconnaissable dans la trace de puissance consommée. Je propose au Chapitre IV une solution alternative à ce problème complexe qui est utilisable aussi bien pour les courbes quartiques de Jacobi que pour les courbes de Weierstrass.

4. Protocole et interfaces

L'implémentation du protocole de communication et de l'interface est moins critique car les secrets peuvent être cantonnés au bloc de niveau inférieur déjà décrit. Dans les chapitres suivants je m'intéresse plus particulièrement aux fuites d'informations par canaux auxiliaire des unités décrites dans les sections II-1, 2 et 3. Pour ces études, je me focalise sur la multiplication scalaire avec une interface UART permettant l'exécution de multiplications scalaires depuis un PC hôte. D'autres interfaces ont été développées mais ne seront pas utilisées dans cette thèse, notamment une interface AXI permettant le dialogue avec un processeur Microblaze [REFIVSW].

5. Résultats

La généricité de la description de mon crypto-processeur permet d'obtenir des configurations variées. Je m'attache dans cette section à comparer et évaluer ces différentes configurations. J'étudierai l'influence de la cible (FPGA ou ASIC), l'influence de la courbe elliptique supportée ou de l'algorithme de multiplication utilisé ainsi que le coût du support de plusieurs courbes. Chaque configuration est comparée à une référence qui est un circuit implémentant l'algorithme de multiplication scalaire « Doublages et Additions » sur la courbe du NIST `P-256` [FIPS00] sur une cible FPGA Kintex-7 du fabricant Xilinx : XC7K32T.

Pour estimer la consommation de ressources j'utilise les métriques de surface en mm^2 ou porte équivalente (« Gate equivalente », GE) pour une cible ASIC (« Application-Specific Integrated Circuit ») après l'étape de synthèse. Pour la même estimation sur des cibles FPGA j'utilise trois métriques : le nombre de bascules ou « Flip-Flops » (FFs), le nombre de « Look-

Up Tables » (LUTs) et le nombre de Slices. Les éléments spécifiques aux FPGA tels que les DSPs ne sont pas utilisés. Les FFs sont une métrique représentant la quantité d'éléments mémorisant dans le circuit, tandis que les LUTs représentent la quantité de logique combinatoire. Ces deux quantités sont relevées après l'étape de « mapping » dans le flot de conception. Cette étape correspond à la transposition de la description en sortie de synthèse logique en éléments disponibles dans la technologie cible. Un slice est un conteneur équipé de 4 LUT-6 (des tables de vérité à 6 entrées) et 8 FFs pour des FPGA de la famille 7 du constructeur Xilinx. Pour des contraintes de routage, il n'est pas possible d'utiliser tous les éléments d'un slice. Le nombre de slice réellement nécessaire n'est donc connu qu'après l'étape de « packing » qui dépend des étapes de placement des éléments et de routage des signaux. C'est cette troisième mesure que nous utiliserons pour représenter la surface car c'est une image de la surface occupée par notre circuit dans le FPGA. Cette mesure peut varier suivant la contrainte de vitesse d'horloge que l'on souhaite atteindre. Dans cette thèse, je tenterai toujours d'atteindre la vitesse maximale, cependant j'interdis à l'outil de conception d'utiliser les optimisations trop consommatrices de ressources telles que la duplication de registres. Les performances atteintes sont calculées à partir du nombre de cycles requis pour réaliser une multiplication scalaire (seulement dépendant de la description du circuit) et de la fréquence d'horloge maximale à laquelle il est possible de cadencer le circuit après l'étape de routage. Il est alors possible de calculer le temps minimal permettant de calculer $k \times P$. Toutes les implémentations sur cibles Kintex 7 ou Spartan 6 ont été obtenues avec la suite « ISE 14.6 » de Xilinx, tandis que celles sur Virtex 2 utilisent la suite « ISE 10.1 » et enfin les implémentations sur cibles Artix 7 sont générées avec la suite « Vivado 15.2 » . Ces choix ont été contraints par l'utilisation d'outils dans leurs versions supportant chacun des FPGAs.

5-A. Comparaison avec l'état de l'art

Avant d'étudier les différentes configurations de mes circuits, je propose ici de comparer le circuit correspondant à la classe de référence choisie avec les crypto-processeurs existants présentés dans le premier chapitre (voir section I-4 page 38). Mon circuit utilise l'algorithme « Doublages et Additions » sur la courbe du `P-256` et la cible de référence est le FPGA Xilinx XC7K32T. Les circuits appartenant tous à cette classe de circuits comparables (par rapport à la cible FPGA) sont représentés sur la Figure II-8. Seules les deux circuits présentés dans [Dond15] ont une protection contre les attaques par canaux auxiliaires.

Mon circuit représenté par un losange gris est plus lent que tous les circuits de référence choisis. Ceci s'explique par le fait que je me suis restreint à ne pas utiliser de DSP pour garder

une compatibilité avec d'autres technologies que les FPGAs. Tous les travaux existant sélectionnés visent une optimisation des performances alors que mon circuit a été développé avec pour objectif de mener les travaux sur l'analyse de la sécurité vis-à-vis des attaques par canaux auxiliaires. La plupart des circuits ont subi des optimisations qui ne peuvent être offertes que par quelques courbes spécifiques : utilisant un nombre premier de forme spécial [SaGü14; RoMu14; AlRa14; KoDe16] ou avec un cofacteur supérieur à 4 [SaGü14; KoDe16; Dond15]. Quelques crypto-processeurs peuvent supporter une courbe générique avec potentiellement un co-facteur < 4 et nombre premier aléatoire. Il s'agit du circuit décrit dans ce chapitre mais aussi des circuits présentés dans [MaLi13] et [BaMe14].

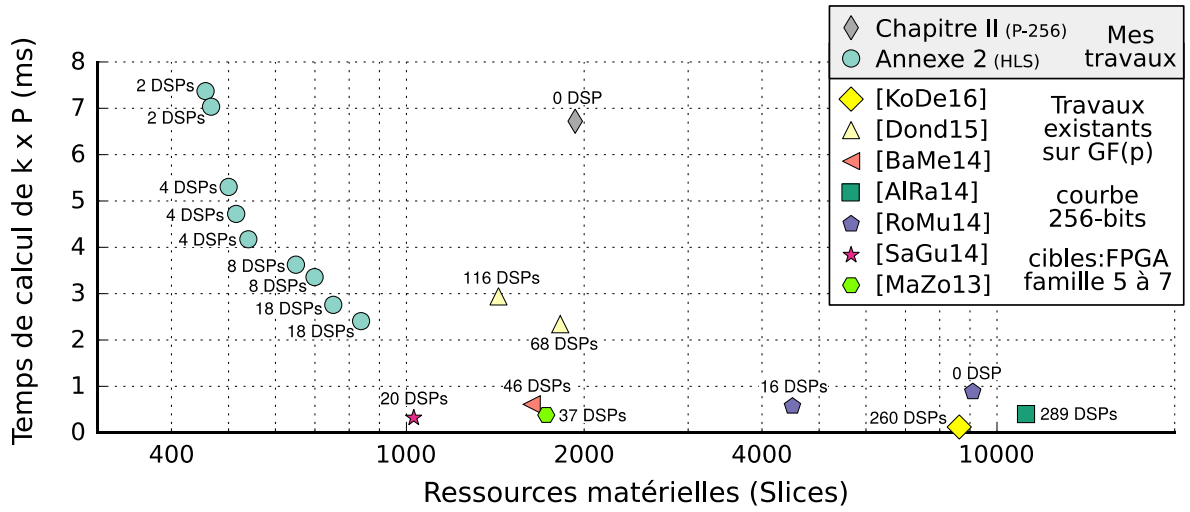


Figure II-8 Comparaison avec des crypto-processeurs existants

J'ai aussi développé des circuits spécifiquement pour cibles FPGAs (utilisant des DSPs) supportant des courbes génériques (cercles bleues sur la Figure II-8). Contrairement au travail présenté dans ce chapitre, ils utilisent un multiplieur de Montgomery par digits (voir Annexe 1) et non un multiplieur série-parallèle. Pour l'instant, ces circuits sont justes protégés contre les attaques par analyse du temps de calcul avec l'algorithme « Doublages et Additions Systématiques », ils sortent donc un peu du cadre de cette thèse consacrée aux attaques par canaux auxiliaires. La très faible consommation de ressources n'a pu être obtenue qu'avec une méthode de conception totalement différente de celle utilisée pour les circuits de ce chapitre. Il s'agit d'utiliser la synthèse de haut niveau pour permettre une conception rapide à travers un développement itératif incrémental. Plus de détails sur cette méthode de conception sont disponibles dans l'Annexe 3.

5-B. Impact de la courbe elliptique utilisée

Pour étudier l'impact de la courbe elliptique sur les caractéristiques de mon crypto-processeur, j'ai généré 12 circuits différents sur cible FPGA Kintex 7 implémentant chacun l'algorithme « Doublages et Additions » sur des courbes différentes. Tous ces crypto-processeurs ne supportent que la courbe elliptique pour laquelle ils ont été conçus. Les courbes sélectionnées sont 3 courbes de Weierstrass sur $GF(2^d)$ provenant des standards du consortium Certicom [Cert00] et du NIST [FIPS00], 7 courbes de Weierstrass sur GF(p) provenant des standards du consortium Certicom, du NIST et de l'ANSI [Amer98] et 2 courbes quartiques de Jacobi sur $GF(p)$ qui ont été générées par Marie-Angela Cornélie [Corn16] (voir l'Annexe 4). Certaines courbes apparaissent dans plusieurs standards sous des dénominations différentes. Je choisi en priorité la dénomination du standard FIPS186-2 utilisant une lettre pour préciser le corps et un nombre pour indiquer la taille de celui-ci (exemples : `B-571` ou `P-256`). Si la courbe n'est pas définie dans ce standard j'utilise la dénomination du standard SEC-2 composée du préfixe « `sect` » ou « `secp` » correspondant au corps suivi de la taille du corps et complété par un suffixe indiquant le mode de génération et un identifiant numérique (exemple la courbe `secp256k1` utilisée dans le protocole bitcoin [Naka08]). Seule la courbe prime239v1 est seulement présente dans le standard X9.62 de l'ANSI. Il faut noter que la seule courbe à être présente dans ces trois standard et dans le référentiel général de sécurité [ANSS14] est la courbe la plus utilisée : `P-256`. Pour plus d'informations sur l'équivalence des courbes elliptiques entre les standards, je renvoie le lecteur vers le tableau de l'annexe A du RFC4492 [MoBo13] décrivant l'usage des courbes elliptiques dans le protocole TLS.

La Figure II-9 montre le coût de ces 12 circuits en termes de LUT-6s, FFs et Slices. Même si deux circuits supportent des courbes différentes mais de même type et de même taille, par exemple `P-256` et `secp256k1`, le coût des circuits peut être différent. Dans l'exemple cité, le nombre premier du corps de `P-256` contient 127 bit nuls contre 6 pour `secp256k1`. Cette grande quantité de bits nuls a simplifié le routage du nombre premier vers l'additionneur. Il en résulte un nombre de Slice occupé beaucoup plus faible pour `P-256` (1931 contre 3150) alors que le nombre de FFs et de LUT-6 est sensiblement identique (seulement 24 LUTs supplémentaires alloués pour le routage). Le routage complexe qui a eu lieu pour `secp256k1` a convergé vers une solution où les slices sont remplis de manière beaucoup moins dense : 70% des couples de FFs sont accompagnés d'une LUT utilisée pour `secp256k1`, contre 50% pour `P-256`. La différence en quantité de Slices est illustrée par la Figure II-10. Même si je

n'ai pas implémenté la réduction optimisée associée au nombre premier de forme spéciale utilisé dans `P-256` (voir section I-3-A.i page 27), cette forme spéciale a tout de même bénéficié aux outils du flot de conception qui ont pu produire un circuit plus compact.

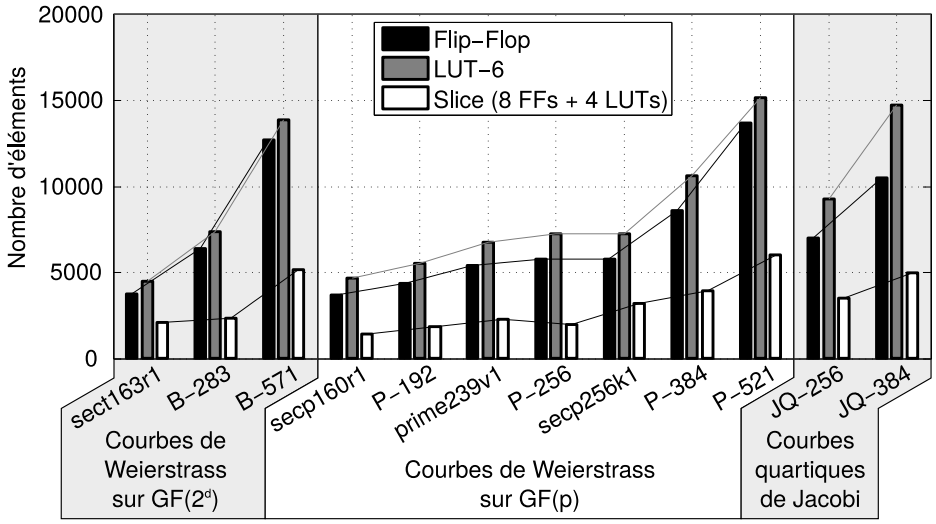


Figure II-9 Impact de la courbe, pour laquelle le crypto-processeur a été spécialisé, sur la consommation de ressources (Circuits sur cible FPGA Kintex 7 supportant une seule courbe)

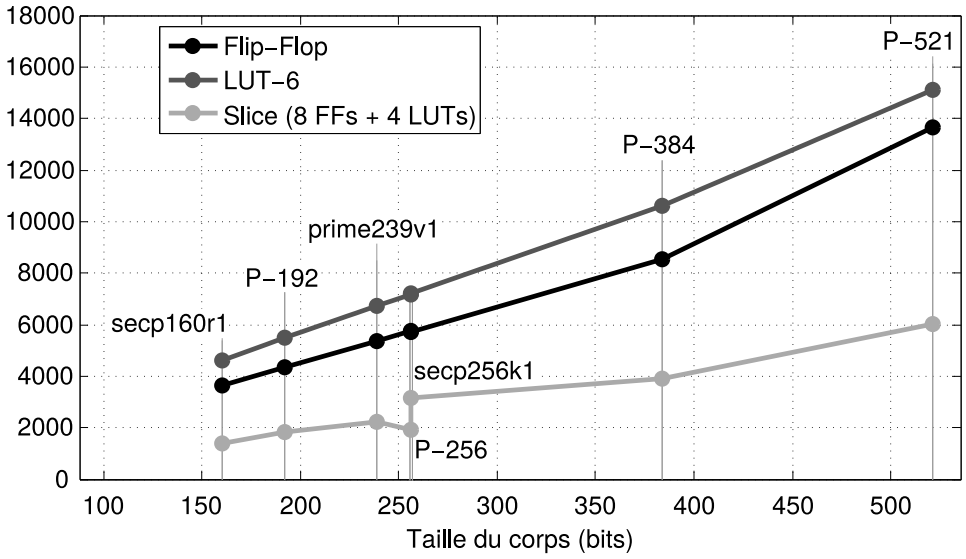


Figure II-10 Impact de la taille de p sur la surface occupée par un circuit supportant une seule courbe sur $GF(p)$ (Circuits sur cible FPGA Kintex 7)

Il est intéressant de comparer le coût d'un circuit supportant une courbe sur $GF(p)$ avec celui d'un circuit sur $GF(2^d)$. Comme `sect163r1` et `secp160r1` ont des tailles de paramètres proches, elles me serviront pour effectuer cette comparaison. Le polynôme irréductible de `sect163r1` à une taille plus grande de 2% par rapport au nombre premier de `secp160r1`. Bien que la taille du corps soit un peu plus grande pour `sect163r1`, le circuit supportant la courbe sur $GF(2^d)$ nécessite 250 LUT-6s de moins (5%) pour implémenter la logique. Nous verrons dans la partie 5-D que cela est dû à la loi d'addition qui est plus simple sur $GF(2^d)$.

L'implémentation d'un circuit basé sur une courbe quartique de Jacobi coûte plus cher que celle d'un circuit basé sur une courbe de Weierstrass sur $GF(p)$. À taille donnée (256 ou 384 bits), les courbes quartiques de Jacobi ont un surcoût de 21% en FFs, ce qui correspond à l'ajout de deux registres supplémentaires (surplus théorique de 25%). On retrouve environ le même surcoût en nombre de Slices ; par exemple l'implémentation sur `JQ-384` nécessite 25% de Slices supplémentaires par rapport à `P-384`.

La Figure II-11 montre que les circuits supportant une courbe sur $GF(2^d)$ sont beaucoup plus rapides que les autres. En effet, comme il n'y a pas de chaîne de retenue dans l'additionneur sur le corps, le chemin critique est beaucoup plus court et il est possible d'atteindre des fréquences de fonctionnement plus élevées (+72% pour `sect163r1` par rapport à `secp160r1` et +100% pour `B-571` par rapport à `P-521`). Les circuits implémentant une multiplication scalaire sur une courbe quartique de Jacobi doivent être cadencés à une fréquence légèrement plus faible que lorsqu'un nombre premier de forme spéciale est utilisé (-2,5% pour la courbe `JQ-256` par rapport à `P-256` et −5% pour `JQ-384` par rapport à `P-384`).

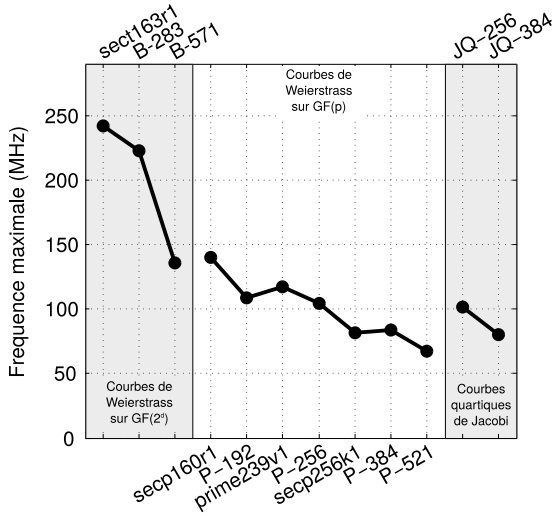


Figure II-11 Influence de la courbe elliptique sur la fréquence maximale atteignable (Circuits sur cible FPGA Kintex 7 supportant une seule courbe)

Du point de vue des performances, les courbes quartiques de Jacobi ont un avantage par rapport aux courbes de Weierstrass sur $GF(p)$: seules 11 multiplications sont nécessaires pour réaliser l'addition de points. Sur une courbe de Weierstrass en coordonnées Jacobiennes, avec les même contraintes (minimisation du nombre de registres et un seul opérateur sur $GF(p)$) mais seulement 3 coordonnées, il me faut 16 multiplications pour une addition de points et 10 pour un doublage de point (voir Figure II-4 et Figure II-5 page 60). Avec une multiplication scalaire par l'algorithme « Doublages et Additions », il y a toujours un doublage par bit du scalaire et en moyenne une addition tous les deux bits. Le coût par bit du scalaire sur une

courbe quartique de Jacobi est donc de 16,5 multiplications sur $GF(p)$ contre 18 pour une courbe de Weierstrass. La Figure II-12 montre combien de cycles d'horloge sont nécessaires pour calculer une multiplication scalaire sur chacun des circuits. On remarque que l'estimation faite en comptant les multiplications sur $GF(p)$ est toujours valable avec cette mesure plus précise. En effet, elle prend en compte aussi les additions et soustractions sur $GF(p)$. Une multiplication scalaire sur une courbe quartique de Jacobi nécessite moins de cycles que sur une courbe de Weierstrass sur $GF(p)$ (mais avec tout de même certaines variations dans l'architecture). Pour les courbes de Weierstrass aussi bien sur $GF(p)$ que sur $GF(2^d)$, la proportion de temps passé sur les additions de points est plus faible que sur les courbes quartiques de Jacobi puisqu'une formule spécifique est utilisée pour le doublage de point et qu'elle est plus rapide que la formule d'addition.

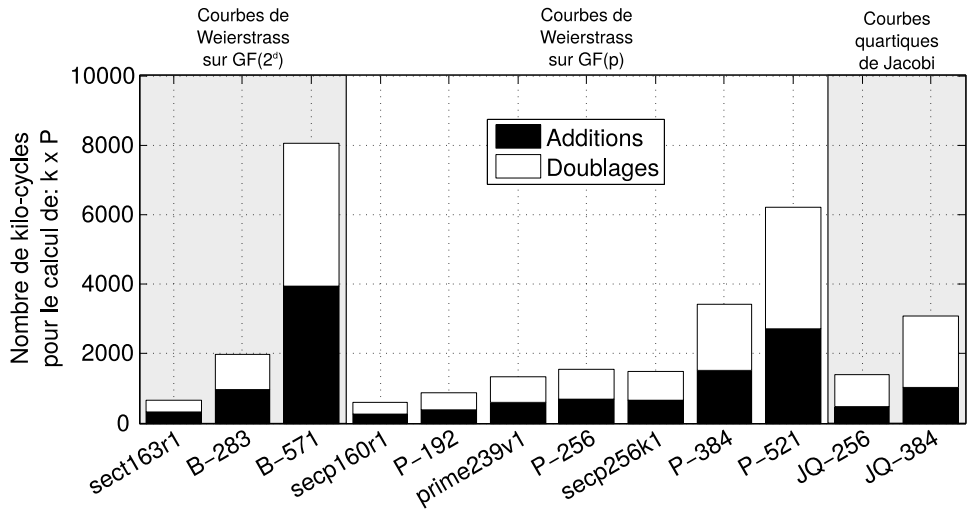


Figure II-12 Nombre de cycles requis par l'algorithme « Doublages et Additions » sur différentes courbes ellitpiques

Les circuits sur $GF(2^d)$ nécessitent un peu plus de cycles que ceux sur $GF(p)$ (un surcoût de 10% pour `sect163r1` par rapport à `secp160r1`) mais ceci est compensé par le gain sur la fréquence de fonctionnement. La Figure II-13 permet d'estimer les performances réelles de chaque circuit. Grâce à l'absence de chaîne de retenue, une multiplication scalaire sur $GF(2^d)$ (2,77 ms pour `sect163r1`) est plus rapide que sur $GF(p)$ (4,35 ms sur `secp160r1`). Il faut tout de même relativiser sur deux points : le premier est que l'existence d'algorithmes permettant de calculer le logarithme discret de manière plus efficace sur $GF(2^d)$ nous oblige à utiliser des tailles de clés plus importantes que sur $GF(p)$ pour atteindre le même niveau de sécurité ; le second point est que ce gain en performance n'existe pas ou peu si on effectue le calcul de la multiplication scalaire sur un processeur généraliste, en conséquence ce sont les courbes sur $GF(p)$ qui sont les plus utilisées et supporter uniquement les courbes sur $GF(2^d)$ peut poser des problèmes d'interopérabilité.

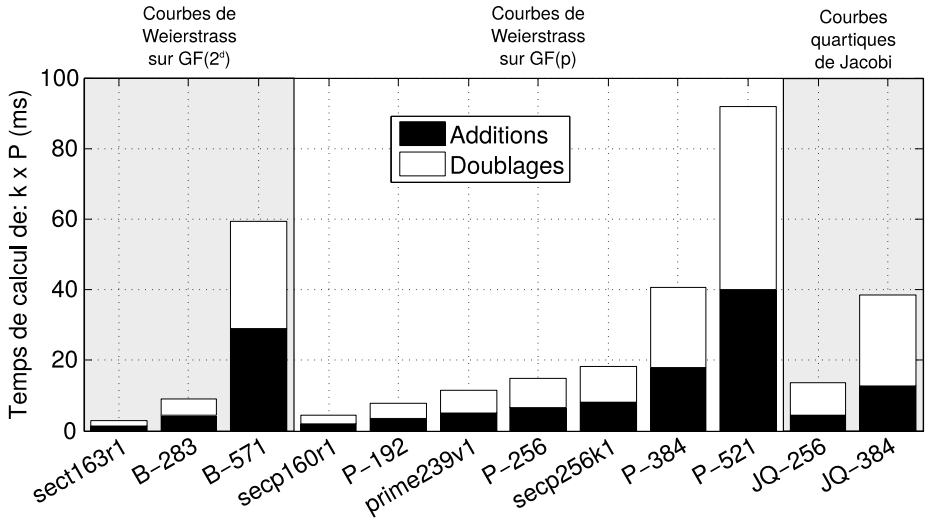


Figure II-13 Performance en temps de calcul de la multiplication scalaire pour différentes courbes elliptiques (Circuits sur cible FPGA Kintex 7 supportant une seule courbe)

La Figure II-13 et la Figure II-14Figure II-15 montrent, qu'à taille donnée, les circuits implémentant la multiplication scalaire sur une courbe quartique de Jacobi sont légèrement plus efficaces que les circuits utilisant une courbe de Weierstrass sur $GF(p)$. Le gain obtenu grâce à l'économie de cycles permet de contrebalancer la faible perte sur la vitesse d'horloge. La multiplication scalaire sur `JQ-256` est 8,5% plus rapide que sur `P-256` et 5,4% pour `JQ-384` par rapport à `P-384`. Je rappelle que ce gain n'est pas gratuit puisqu'il a un coût en surface d'environ 25%. Il est important de rappeler que la sécurité vis-à-vis des attaques par canaux auxiliaires est meilleure sur une courbe quartique de Jacobi grâce à l'utilisation d'une formule unifiée (voir section I-5-B page 41) ; je propose une étude détaillée de ce point dans le Chapitre III.

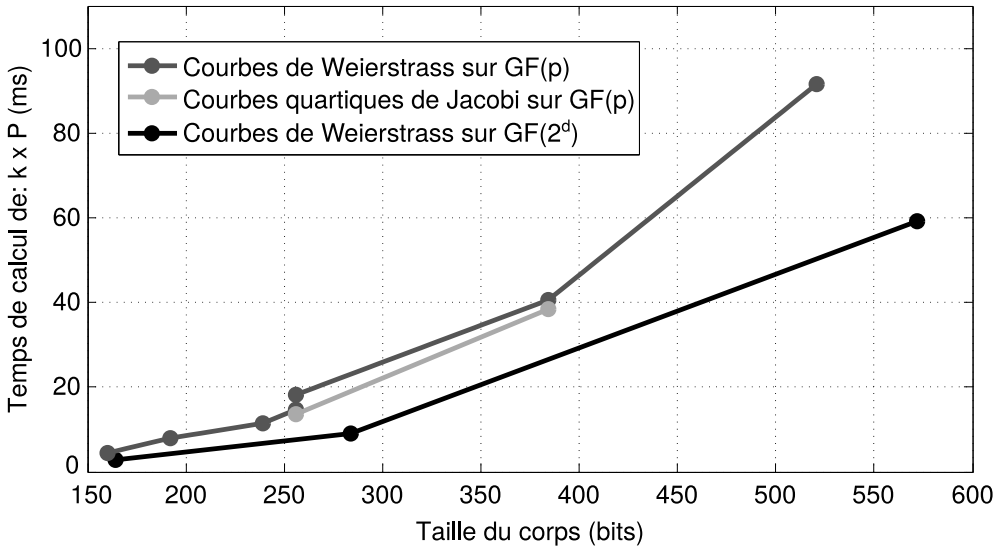


Figure II-14 Performance de la multiplication scalaire en fonction de la taille du corps pour différents types de courbes (Circuits sur cible FPGA Kintex 7 supportant une seule courbe)

5-C. Coût de différentes multiplications scalaires

La description du crypto-processeur permet de synthétiser des circuits implémentant différentes multiplications scalaires parcourant les bits de gauche à droite. La Figure II-15 représente le coût de circuits implémentant 3 types de multiplications scalaires : l'algorithme de référence « Doublages et Additions », l'algorithme « Doublages et Additions Systématiques » protégé contre les attaques par SPA (voir section I-5-B page 40) et l'algorithme avec méthode par fenêtrage « m-ary » (voir section I-3-C page 37). La différence entre le circuit implémentant l'algorithme « Doublages et Additions » et celui implémentant l'algorithme « Doublages et Additions Systématiques » se situe principalement dans la partie contrôle de l'unité de multiplication scalaire (voir Figure II-7 de la section II-3, page 64). Il n'y pas de surcoût en mémoire significatif mais l'utilisation des opérations fictives requière 8% de LUT-6 supplémentaires. Par contre, il y a un surcoût bien plus important en Slice (21%), qui s'explique par la différence d'optimum atteint par les outils du flot de conception qui a permis d'atteindre une fréquence de 107MHz contre 104MHz pour l'algorithme « Doublages et Additions ».

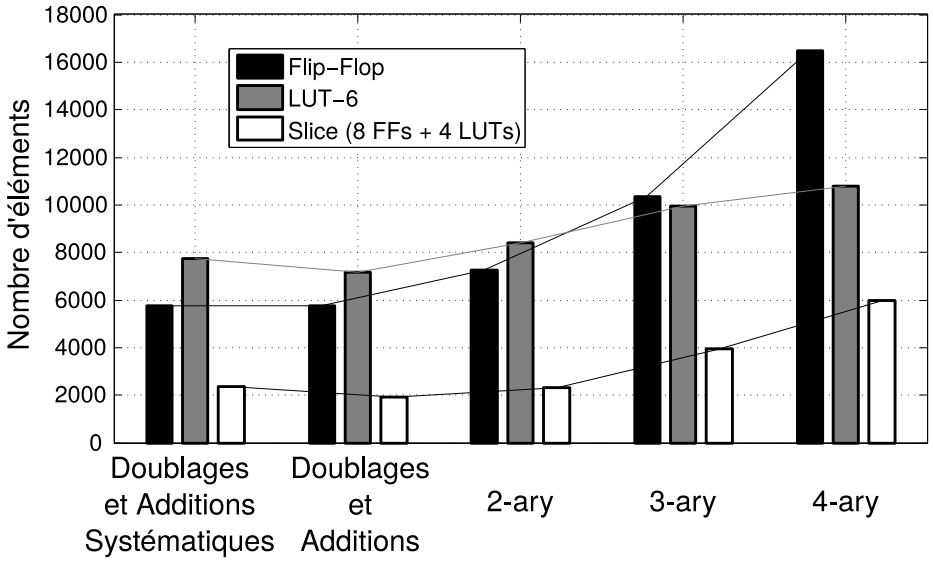


Figure II-15 Coût en surface des opérations fictives et des méthodes par fenêtrage (courbe P-256 sur cible FPGA Kintex 7)

La fréquence de fonctionnement est peu impactée par la méthode de fenêtrage. Une baisse significative n'apparaît que pour un fenêtrage de plus de 4 bits (exemple -7% pour un

fenêtrage de 6 bits, configuration non représentée dans la Figure II-15) mais je rappelle que des fenêtres d'une telle taille ne sont pas efficaces pour des clés de 256 bits (voir le Tableau I-1 page 37). L'utilisation de fenêtrage nécessite de stocker des points pré-calculés, il y a donc un surcoût naturel en terme de FFs. On observe que la logique (les LUT-6) est aussi impactée mais dans une moindre mesure par rapport à la mémoire. Un fenêtrage de 2 bits (algorithme « 2-ary ») nécessite 27% de FFs supplémentaires par rapport à l'algorithme « Doublages et Additions ». Mais le surcoût en nombre de Slices (20%) est plus réduit car un plus grand nombre de FFs a pu être intégré avec des LUT-6 utiles dans des Slices. L'augmentation de la quantité de mémoire a permis aux outils du flot de conception de réaliser un meilleur « packing » (arrangement) des LUT-6 et FFs dans les Slices. Ceci n'est plus vrai pour les fenêtres de tailles supérieures où le « packing » a été moins efficace mais pour permettre d'atteindre des fréquences d'horloge équivalentes à l'algorithme « Doublages et Additions Systématiques ». Les 36% de cycles économisés par un fenêtrage de 4 bits pour des clés de 256 bits à tout de même un coût important en Slices (200%). La mise en œuvre de telles méthodes sur FPGA serait bien plus efficace en utilisant des éléments spécifiques à cette cible telle que les BRAMs (Blocs spécifiques aux FPGA permettant d'instancier des mémoires) pour stocker les points. Cette optimisation n'a pas été réalisée pour garder une compatibilité avec d'autres cibles.

5-D. Influence du support de plusieurs courbes elliptiques

Jusqu'à maintenant, je me suis seulement intéressé à des circuits ne supportant qu'une seule courbe. En pratique cela peut convenir si l'on maîtrise parfaitement tout le matériel qui sera utilisé par l'application. Si le circuit doit interagir avec un écosystème non maîtrisé, il peut être intéressant de produire des circuits supportant plusieurs courbes elliptiques. La Figure II-16 représente trois nouveaux circuits supportant des courbes avec des clés de maximum 256 bits. Par rapport au circuit ne supportant que `P-256`, supporter toutes les courbes de Weierstrass sur 256 bits limite les optimisations que peut réaliser l'outil de synthèse : il y a donc une augmentation de la quantité de logique de 20% du nombre de LUT-6. Une partie de cette logique supplémentaire est aussi utilisée pour la configuration de la courbe utilisée. Le routage des signaux permettant de transmettre les paramètres de la courbe à la partie opérative a un coût non-négligeable puisque 79% de slices supplémentaires sont nécessaires. Ceci permet d'atteindre une fréquence de fonctionnement équivalente (environ 105 MHz) par rapport au circuit ne supportant que `P-256`.

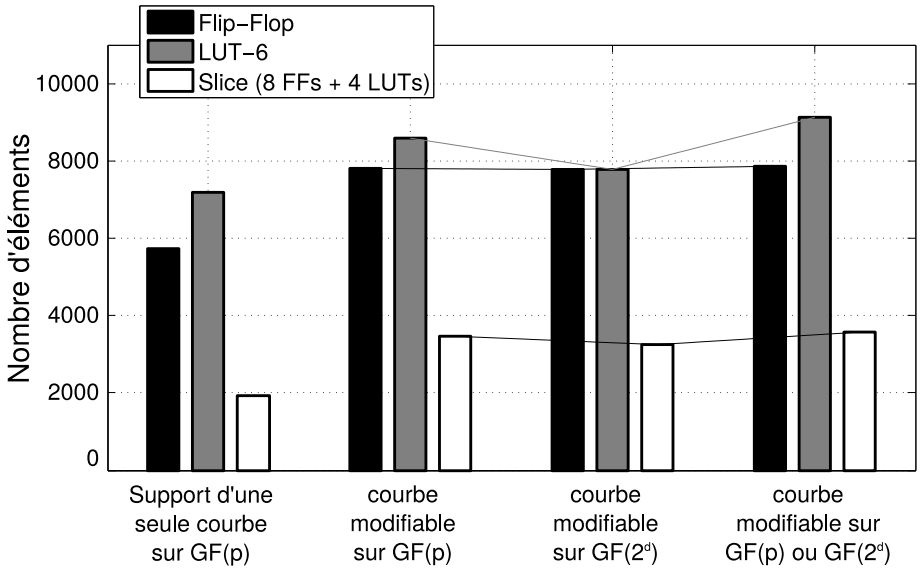


Figure II-16 Coût du support de plusieurs courbes de au plus 256 bits (cible FPGA Kintex 7)

Sur la Figure II-16 qui compare des circuits supportant des courbes de Weierstrass arbitraires (avec des tailles de clés plus petites que 256 bits) sur $GF(p)$ et $GF(2^d)$, l'économie qu'apporte le corps $GF(2^d)$ peut être observée indépendamment de la courbe utilisée. Cette économie se voit sur la logique avec l'utilisation de 10% de LUT-6 en moins par rapport à $GF(p)$. Comme il n'y a pas de chaîne de retenue dans l'additionneur sur $GF(2^d)$, il est possible d'atteindre une fréquence de 200 MHz. Le support de courbes sur les deux corps n'est pas optimal lorsqu'on utilise une courbe sur $GF(2^d)$ car l'horloge est bridée à 90MHz à cause de la chaîne de retenue de l'additionneur sur $GF(p)$. Il semble difficile d'utiliser des fréquences d'horloge différentes selon le corps sans abandonner le gain en surface apporté par la mutualisation des opérations sur les deux corps.

5-E. Influence de la cible

Les circuits obtenus ont des performances et une consommation de ressources matérielles qui dépendent de la cible. La Figure II-17 montre que la différence entre l'architecture des FPGA Virtex 2 et celle des FPGA récents (des LUT à 4 entrées au lieu de 6 et autant de LUT que de FFs) nous empêche de faire une comparaison pertinente. En se focalisant sur trois autres FPGAs qui utilisent des Slices avec architectures identiques (un Spartan 6, un Artix 7 et un Kintex 7) on observe que les circuits obtenus consomment une quantité de ressources similaire. La petite différence qui apparaît sur le nombre de LUT-6 utilisées pour la cible Artix 7 (+10% par rapport aux deux autres FPGA récents) est due à l'utilisation d'outils de conception différents (utilisation de la suite Vivado au lieu de la suite ISE).

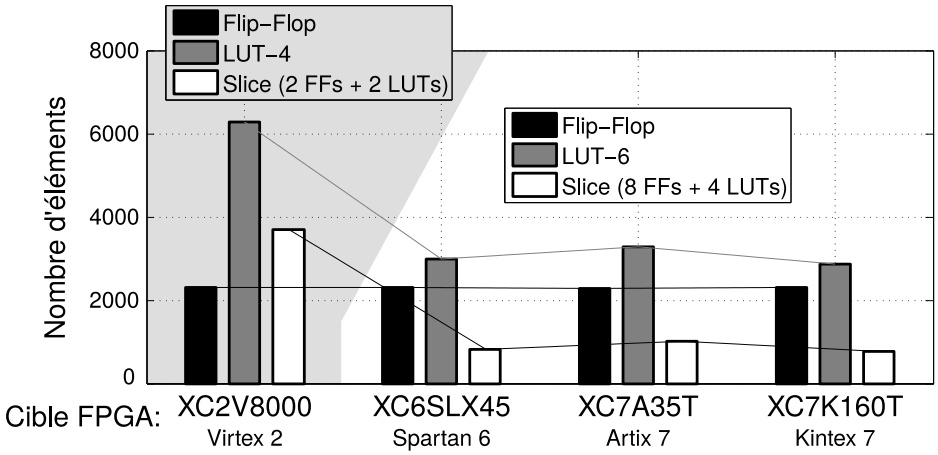


Figure II-17 Impact de la cible FPGA sur la consommation de ressources (pour la courbe P-256)

La différence la plus notable entre les cibles FPGA est la fréquence de fonctionnement maximale atteignable. Le circuit sur cible Virtex 2 utilisant une technologie de 150 nm doit utiliser une fréquence de fonctionnement inférieure à 36 MHz. Cette technologie de FPGA a été annoncée en 2001. La famille Spartan 6 annoncée en 2009 et utilisant une technologie de 45 nm permet d'atteindre une fréquence plus importante (53 MHz). Le passage à une technologie 28 nm permet d'atteindre des fréquences encore plus importantes jusqu'à 104 MHz pour la cible Kintex 7.

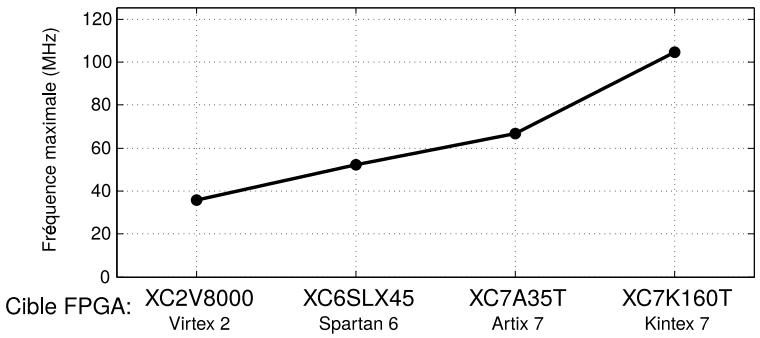


Figure II-18 Impact de la technologie sur la fréquence d'horloge maximale atteignable (différentes générations de FPGAs pour la courbe P-256)

Les technologies ASIC offrent plus de liberté aux outils du flot de conception puisque le « mapping » technologique est beaucoup moins contraint que sur cible FPGA. La description du circuit est optimisée pour cible FPGA. A partir de cette même description, j'utilise l'outil « Design Compiler » de Synopsys sur une ancienne technologie de 350 nm (AMS 0.35) pour produire différents circuits. La Figure II-19 montre qu'il est possible d'obtenir une grande gamme de circuits avec différentes fréquences de fonctionnement occupant plus ou moins de surface. Le circuit le plus rapide obtenu sur cette technologie peut fonctionner à 125 MHz en occupant 70 kGE (kilo-portes équivalentes). Il peut être cadencé à plus grande vitesse que le

circuit sur FPGA Kintex 7 même sur une technologie ASIC datée de 1995 (il faut tout de même noter que les résultats sur ASIC sont avant les étapes de placement et routage). Il est aussi possible de générer des circuits occupant 10% moins de surface en relaxant la contrainte sur la fréquence d'horloge. Le coût de fabrication de circuits ASIC ne peut être rentabilisé que par une production de masse. Dans la suite de cette thèse, nous nous concentrerons sur les cibles FPGA qui permettent un prototypage plus rapide et à faible coût.

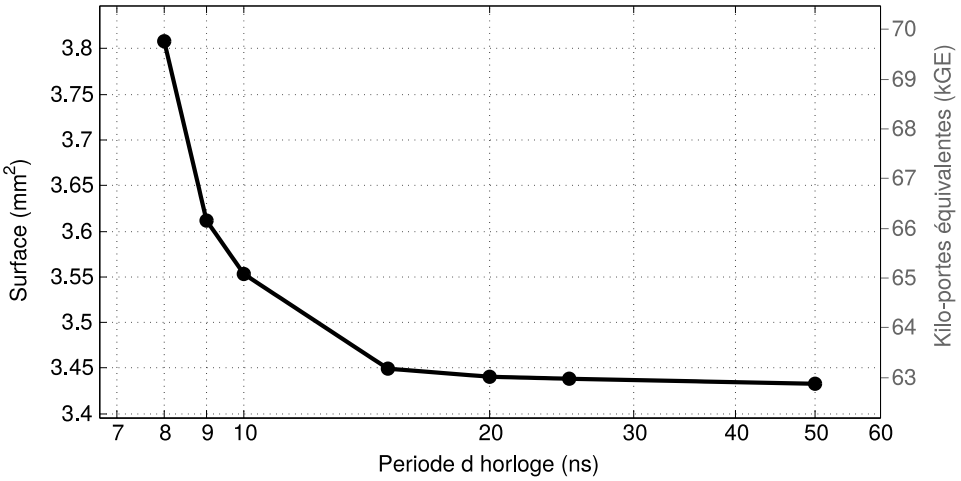


Figure II-19 Influence du cadencement maximal de l'horloge sur la surface occupée sur cible ASIC (technologie AMS 0.35 µm et courbe P-256)

6. Conclusion

Le crypto-processeur développé pour mes travaux de thèse et décrit dans ce chapitre est polyvalent. Il supporte les corps $GF(p)$ et $GF(2^d)$ mais peut également être spécialisé pour un seul de ces deux corps. Tous les nombres premiers sont supportés car aucune hypothèse n'a été faite sur la forme du nombre premier p. Bien qu'optimisée pour cible FPGA, la description du crypto-processeur se prête aussi bien à un flot ASIC que FPGA car aucun bloc spécifique à l'un ou l'autre n'a été utilisé. Il est capable de supporter toutes les courbes elliptiques de Weierstrass et les courbes quartiques de Jacobi. Il peut supporter plusieurs courbes ou alors ne supporter qu'une seule courbe après synthèse pour laisser plus de place aux optimisations réalisées par les outils du flot de conception. Il peut être configuré pour implémenter différents algorithmes de multiplication scalaire de gauche à droite, tels que les méthodes par fenêtrage ou l'algorithme « Doublages et Additions ». Il supporte les contre-mesures par opérations fictives. Le circuit reste sensible à certaines attaques, notamment des attaques par fautes. Une solution alliant optimisation des performances et réduction des fuites d'information apportant plus de sécurité vis-à-vis de ces différentes attaques est proposée dans le Chapitre IV.

Chapitre III Étude de la résistance des opérations unifiées vis-à-vis des attaques par analyse de la puissance consommée

Les circuits non-protégés implémentant des opérations cryptographiques basées sur les courbes elliptiques peuvent être la cible d'attaques à faible coût telles que l'analyse simple de puissance consommée (SPA). Depuis quelques années, plusieurs méthodes basées sur des formules unifiées ont été proposées pour permettre de réaliser les doublages et additions de points avec une seule formule et ainsi harmoniser leurs motifs de consommation. Dans ce chapitre, je propose une nouvelle attaque pour évaluer la sécurité des contre-mesures basées sur une formule unifiée. Je me focalise sur un circuit implémentant cette protection sur des courbes quartiques de Jacobi. L'attaque purement passive que je propose est non-supervisée et ne requière aucune connaissance de l'architecture ou de l'algorithme de la cible. L'idée de l'attaque est d'identifier parmi les opérations basiques sur les points celles qui sont des additions (avec deux points opérandes différents) de celles qui sont des doublages (avec deux points opérandes identiques). L'origine de la fuite d'information que je propose d'exploiter est la réutilisation d'un même point opérande dans plusieurs opérations basiques sur les points. Cette similitude, caractéristique des additions dans les multiplications scalaires de gauche à droite, se retrouve dans la trace de puissance consommée et est exploitée pour retrouver le scalaire utilisé. Je montre que sur des traces de consommation à faible bande passante, seulement une trentaine de traces suffisent à contrer l'efficacité de la formule unifiée. Ce petit nombre de traces n'a pu être atteint qu'après avoir identifié une métrique de comparaison efficace et un algorithme de classification adapté au problème. Une extension de cette attaque est l'identification du point pré-calculé utilisé dans une addition de points pour attaquer des multiplications scalaires par fenêtrage. Toutes les attaques ont été réalisées sur des traces réelles obtenues à partir d'un banc d'attaque par analyse de puissance consommée que j'ai développé pour l'occasion.

1. Cas d'étude

Dans cette partie je m'attache à présenter le circuit ciblé et le banc d'attaque utilisé. L'unification de l'addition et du doublage de points, dans le circuit qui sert de cible à la nouvelle attaque que je présente, est obtenue en utilisant des courbes quartiques de Jacobi sur lesquelles il existe une formule d'addition complète (voir section I-5-B page 43). Le cas d'étude se restreint à une seule méthode d'unification et une seule implémentation. Cependant, l'attaque présentée dans les sections suivantes ne nécessite aucune connaissance de l'architecture du crypto-processeur ou de la technique d'unification. Cela signifie que l'attaque pourrait être appliquée à d'autres unifications (ex : utilisation des courbes de Edwards tordues) et à d'autres architectures (ex : utilisation d'un multiplieur par digit) mais ceci n'a pas été validé expérimentalement.

1-A. Crypto-processeur basé sur une courbe quartique de Jacobi

L'architecture du crypto-processeur attaqué a été décrite dans le Chapitre II. Différentes configurations ont été attaquées : 4 configurations supportant chacune une courbe quartique de Jacobi différente sur 256 bits (c'est-à-dire 4 courbes) et 1 configuration supportant une courbe sur 384 bits. L'algorithme de multiplication scalaire principalement utilisé est « Doublages et Additions », quelques attaques contre un algorithme par fenêtrage « m-ary » ont aussi été montées. Ces multiplications scalaires sont sensibles aux attaques par analyse du temps de calcul (voir I-5-A page 40) mais ceci ne sera pas exploité, nous nous restreindrons à étudier si l'unification permet ou non de dissimuler de manière efficace le type des opérations sur les points (addition ou doublage) contre des analyses de puissance consommée. De même, le circuit n'est pas protégé contre les attaques par petits sous-groupes [AnBr03], la mise en œuvre d'une validation du point opérande n'influencerait pas l'étude que je mène ici. Toutes les traces de puissance consommée proviennent d'une cible FPGA Xilinx Virtex 2 que je cadence à 20 MHz.

La Figure III-1 montre un extrait de trace de puissance consommée par le dispositif cible durant une multiplication scalaire. Le motif de consommation de l'addition de points apparaît clairement et le découpage en sous-traces n'est pas une opération difficile. L'utilisation de la même formule pour les opérations basiques sur les points permet d'obtenir des sous-traces d'additions qui ressemblent fortement à des sous-traces de doublages. Contrairement à la l'extrait de trace de la Figure I-22 (multiplication scalaire sur une courbe de Weierstrass, page 41) il semble difficile de réaliser une attaque SPA. Le but de ce chapitre est d'explorer

une attaque horizontale plus évoluée que la classique SPA (comparant des sous-traces entre elles) pour identifier si les additions entre deux points opérandes identiques (doublages) sont distinguables des autres (additions).

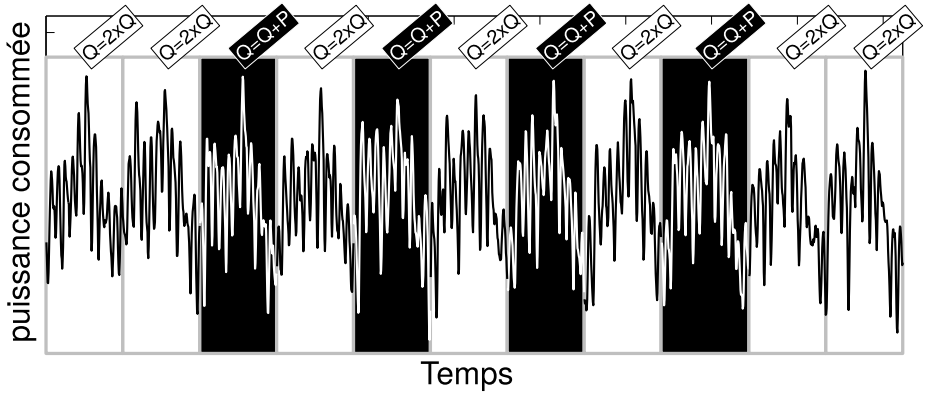


Figure III-1 Extrait d'une trace de puissance consommée d'une multiplication scalaire par l'algorithme « Doublages et Additions » sur une courbe quartique de Jacobi de 256 bits (cible FPGA Virtex 2).

Les 4 courbes quartiques de Jacobi sur 256 bits et la courbe sur 384 bits sont disponibles dans l'Annexe 4.

1-B. Banc d'attaque par analyse de puissance consommée

Pour réaliser les attaques par analyse de puissance, j'ai développé un banc constitué de plusieurs éléments. L'élément central est une carte munie d'un FPGA Xilinx Virtex 2 XC2V8000. J'utilise une sonde de courant (par effet Hall) avec une bande passante de 100MHz pour observer le courant consommé par le cœur du FPGA (Alimentation VCCINT). Il reste 4 capacités de 10 µF et 12 capacités de 100 nF entre le point de mesure et le FPGA. Cela signifie que la bande passante des traces est limitée. D'autres expérimentations ont été mises en œuvre contre une cible plus récente (Kintex 7, carte Sakura-X) mais la faible dynamique obtenue avec la sonde à effet Hall produit un bruit de quantification trop important pour les attaques. Sur cette cible récente, j'ai aussi obtenu des traces de puissance consommée à travers une sonde différentielle et une résistance « shunt » de 10 mOhm sur la ligne d'alimentation : bien que la qualité des traces obtenues soit meilleure qu'avec la sonde de courant, cela n'a pas suffi pour réaliser une attaque horizontale plus évoluée que la SPA classique. Comme l'objectif est de vérifier la sécurité apportée par l'utilisation d'une formule unifiée, je me place dans le meilleur cas pour développer mon attaque (Cible Virtex 2).

Un autre élément important du banc d'attaque est l'oscilloscope qui permet d'échantillonner la trace. Un des éléments pouvant être limitant pour la cryptographie asymétrique est la profondeur mémoire de celui-ci. L'oscilloscope Tektronix MDO4104B-6 que j'utilise permet de stocker 20M échantillons par trace. À titre d'exemple, l'attaque

développée dans ce chapitre utilise des traces de 20M échantillons sur 8 bits en mode haute-résolution (« HiRes ») pour la courbe de 384 bits et 10M échantillons pour les courbes de 256 bits, échantillonnés à 100MHz (il n'y pas de repliement de spectre car le signal mesuré est déjà filtré par les capacités). La profondeur mémoire utilisée est proche des capacités maximales de l'appareil tandis que le cadencement de l'échantillonnage est bien en deçà de la fréquence maximale (5GHz). Pour rester dans les conditions d'une attaque horizontale réelle, je n'utilise pas d'amélioration du rapport signal à bruit (SNR) des traces par calcul d'une trace moyenne (impossible en cas de contre-mesure par ajout de hasard).

Un ordinateur connecté au réseau local se charge de la communication avec le circuit via une interface UART (« Universal Asynchronous Receiver Transmitter »). C'est par ce biais que le point et le scalaire opérande sont envoyés au circuit. Pour chaque trace enregistrée, la conformité du point résultat avec les opérandes de la multiplication scalaire est toujours vérifiée, hors ligne et de manière logicielle, à l'aide de la bibliothèque `mphell` développée à l'Institut Fourier. Un serveur de calcul connecté au même réseau local coordonne les opérations. Il communique avec l'oscilloscope via une interface VXI-11 qui est une implémentation étendue du protocole GPIB par-dessus le modèle TCP/IP. Après avoir passé l'oscilloscope en mode « Single » (attente de l'évènement déclencheur : « Trigger »), il lance le calcul de la multiplication scalaire et télécharge l'oscillogramme représentant la trace de puissance consommée de cette opération. Ces traces peuvent être ensuite chargées dans une instance du logiciel Matlab de MathWorks pour mettre en œuvre des attaques par analyse de puissance consommée.

2. Attaque horizontale

Le schéma de l'attaque que je propose est décrit dans la Figure III-2. Comme je l'ai montré (voir I-5-D page 50), les attaques horizontales existantes qui nécessitent peu d'information sur le circuit attaqué (voir Tableau I-3) peuvent être divisées en trois étapes : un traitement initial sur la trace (étape 1), le calcul d'une mesure pour distinguer les opérations (étape 2) et une dernière étape de classification (étape 3). L'attaque que je présente ne déroge pas à cette règle : on retrouve ces trois étapes dans le schéma de mon attaque (voir Figure III-2). Je n'effectue que peu de prétraitements sur la trace de la multiplication scalaire : un découpage de la trace en sous-traces et une normalisation des sous-traces. Ce découpage est particulièrement facile à cause de la présence d'un seul motif. Les deux autres étapes qui différencient mes travaux des attaques existantes sont présentées dans la suite de cette partie.

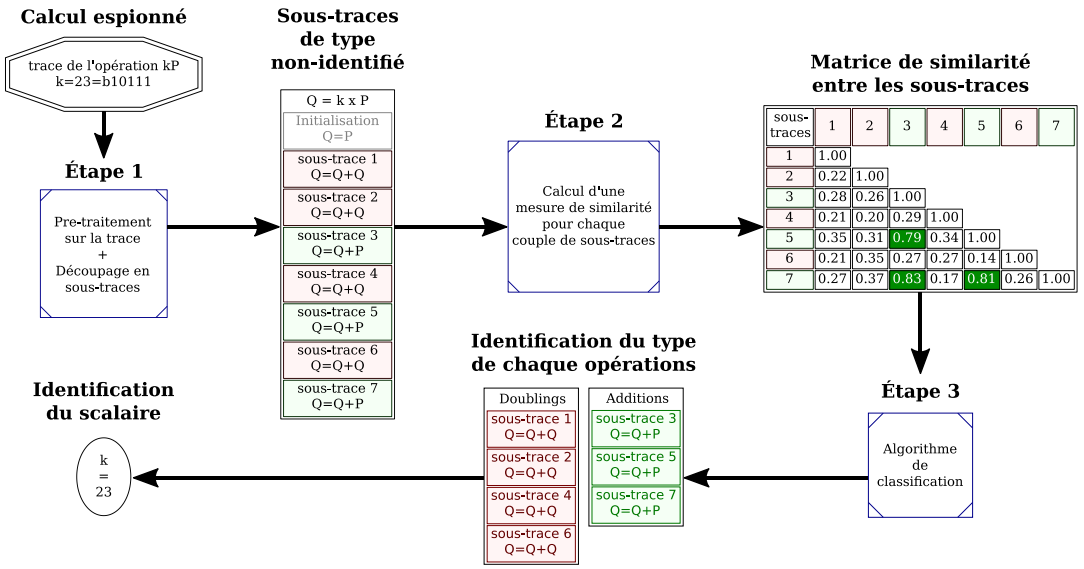


Figure III-2 Schéma de l'attaque horizontale contre la contre-mesure par formule unifiée

Dans cette section je présente une attaque purement horizontale sur une seule trace qui n'a pas pu être mise en place avec succès. Nous verrons par la suite comment modifier cette attaque pour exploiter plusieurs traces; les résultats expérimentaux d'attaques réelles ayant permis de retrouver le scalaire secret seront décrits dans la partie suivante.

2-A. Synchronisation et mesure de similarité par corrélation croisée

Le circuit attaqué utilise une multiplication scalaire de gauche à droite, plus précisément l'algorithme « Doublages et Additions ». Durant la multiplication scalaire, il y a donc deux types d'opérations basiques sur les points : $Q = Q + Q$ et $Q = Q + P$ (voir Figure I-18, page 36). Pour chacune de ces opérations, le point accumulateur Q est différent, mais le point P est constant et correspond à l'opérande de la multiplication scalaire $k \times P$. Comme toutes les additions ont cet opérande en commun, elles ont quelques opérations sur $GF(p)$ partageant une certaine ressemblance. Il devrait donc apparaitre une similarité plus élevée entre des sous-traces qui correspondent à des additions. Une majorité d'opérations sur $GF(p)$ dépend du second opérande (Q) qui est différent pour chaque addition de points. Cela va donc impacter négativement l'attaque, mais l'observation de toutes les mesures de similarité entre les opérations avant de prendre une décision d'identification du type de l'opération permet d'améliorer les résultats. C'est l'algorithme de classification qui sera décrit dans la partie suivante. Il est possible de sélectionner des points d'intérêt si l'ordonnancement de l'addition de points est connu, mais cela ne correspond pas au modèle d'attaquant que j'ai choisi : aucune connaissance de l'implémentation.

Comme je l'ai montré (voir section I-5-D page 48), il existe déjà plusieurs mesures de

ressemblance ou de distance entre les traces utilisées dans des attaques horizontales. Dans notre cas le découpage de la trace en sous-traces est grossier. Avant chaque mesure de similarité d'un couple d'opérations, je dois donc tout d'abord synchroniser les deux sous-traces. Cette synchronisation est faite en identifiant le déphasage qui permet de maximiser la corrélation croisée. Cette méthode a été choisie car elle peut être grandement accélérée en effectuant ce calcul dans le domaine spectral.

Pour simplifier la suite nous décrirons les calculs dans le domaine temporel, mais ils sont en réalité effectués dans le domaine discret (échantillonné). Avant la synchronisation des sous-traces, un prétraitement est utilisé pour les normaliser. Une sous-trace $X_i(t)$ est normalisée en $\widetilde{X_i}(t)$ en annulant sa valeur moyenne et en imposant une valeur efficace égale à 1 :

$$\widetilde{X_i}(t) = \frac{X_i(t) - \frac{1}{T}\int_0^T X_i(t)dt}{\sqrt{\frac{1}{T}\int_0^T \left(X_i(t) - \frac{1}{T}\int_0^T X_i(t)dt\right)^2 dt}}$$

On peut ainsi assurer que la sous-trace normalisée $\widetilde{X_i}(t)$ aura une autocorrélation de 1. Ce prétraitement n'est pas essentiel pour la synchronisation, mais permet de s'affranchir de perturbations basse fréquence dans le calcul de similarité.

La Figure III-3 décrit le calcul de la corrélation croisée $c(\tau)$ pour plusieurs déphasages τ entre deux sous-traces normalisées :

$$c_{ij}(\tau) = \frac{1}{T}\int_{-T}^{2T} \widetilde{X_i}(t-\tau) \times \widetilde{X_j}(t)$$

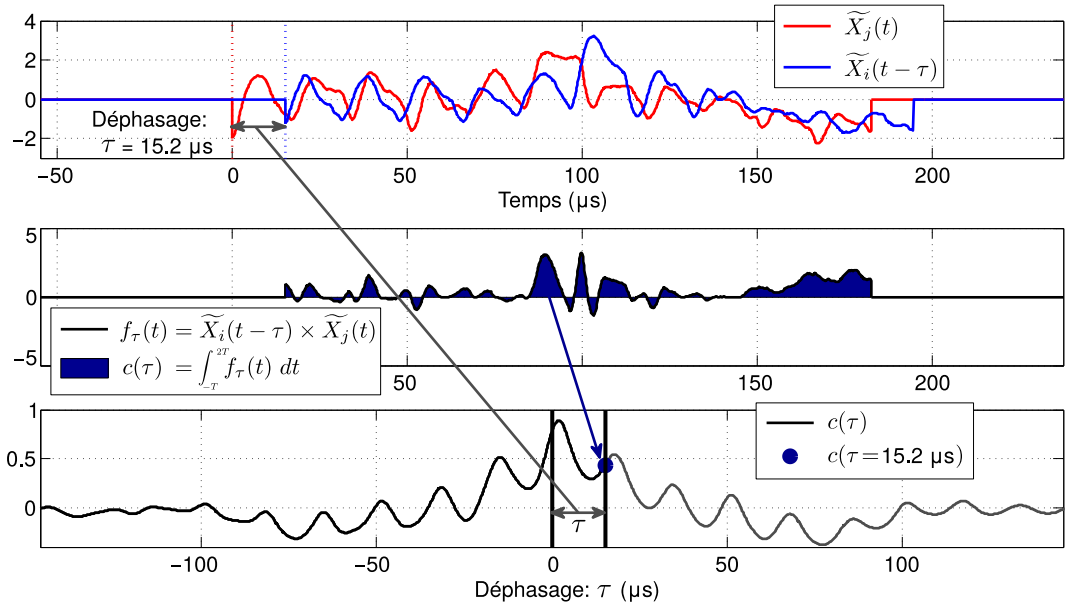


Figure III-3 Calcul de la corrélation croisée entre deux sous-traces : $\widetilde{X_i}(t)$ et $\widetilde{X_j}(t)$

Nous pouvons observer que l'enveloppe de $c(\tau)$ forme un triangle avec son sommet autour de l'abscisse $\tau \approx 0$. Ceci est dû à un biais de la corrélation croisée : pour des déphasages importants il y a moins d'instants t où $\widetilde{X}_\iota(t-\tau) \times \widetilde{X}_J(t)$ est non nul. Il est possible de corriger ce biais en remplaçant $1/T$ par $1/(T-|\tau|)$. Le maximum de $c(\tau)$ corrigé sera alors souvent identifié pour un très grand déphasage ne prenant en compte que très peu de points des sous-traces. Pour réaliser une synchronisation cohérente il devient alors important de définir un déphasage maximal dans la recherche du maximum de la corrélation croisée. Pour que l'attaque soit non-supervisée, j'ai choisi de ne pas corriger le biais de la corrélation car cela permet de ne pas définir ce déphasage maximal. En pratique, les sous-traces ne sont que faiblement désynchronisées, ce biais n'impact donc que très peu le processus de synchronisation. La corrélation croisée maximale identifiée dans la Figure III-3 est de 2,24 µs. Les sous-traces synchronisées sont représentées sur la Figure III-4.

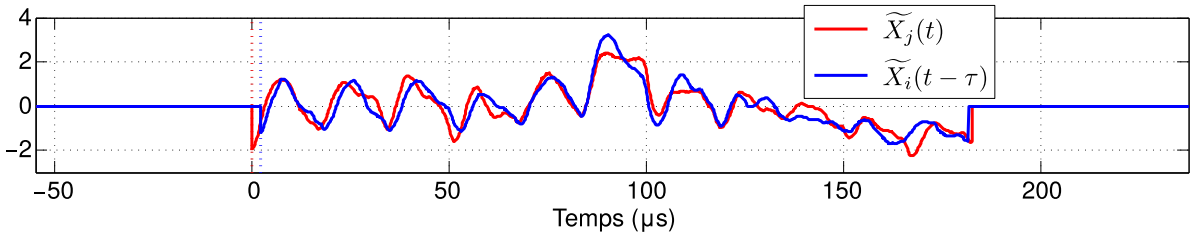


Figure III-4 Deux sous-traces synchronisées en identifiant le déphasage qui maximise la corrélation croisée

Pour éviter le calcul d'un grand nombre d'intégrales, la corrélation croisée est calculée dans le domaine spectral où ce produit de convolution devient un simple produit terme à terme. Pour réaliser cette optimisation, toutes les sous-traces sont transposées dans le domaine spectral avec une transformée de Fourier rapide. Une fois les corrélations croisées calculées dans ce domaine pour chaque couple de sous-traces, elles sont transposées dans le domaine temporel par transformée de Fourier rapide inverse. Il est alors possible d'identifier les maximums pour synchroniser les sous-traces. C'est cette dernière étape qui nécessite le plus de temps de calcul car il y a autant de transformées inverses à calculer que de couples de sous-traces.

Une fois les sous-traces synchronisées, il faut choisir une métrique pour définir un degré de similarité entre elles. J'ai essayé plusieurs métriques, mais je ne présenterai que les plus usuellement utilisées : la distance euclidienne et la corrélation. La métrique qui donne les meilleurs résultats est la corrélation : ceci sera étudié en détails dans la section III-3 (page 92). Je définis donc le degré de similarité entre deux sous-traces comme la valeur maximale de la

corrélation croisée : $\max_\tau(c(\tau))$ dans la Figure III-3. C'est aussi la corrélation pour un déphasage nul lorsque les sous-traces ont été synchronisées. Le calcul de la corrélation permet donc de réaliser deux opérations : la synchronisation des sous-traces et l'évaluation d'une métrique de similarité. Cette métrique est identique au « Pearson factor » : c'est un facteur calculable entre deux traces et qui réalise la normalisation décrite précédemment et le calcul de corrélation pour un déphasage nul. Cette métrique (sans synchronisation) est souvent utilisée dans des attaques théoriques car il existe un modèle de fuite où ce facteur est directement lié à la distance de Hamming entre les valeurs manipulées.

Cette métrique n'est pas une distance au sens mathématique (l'inégalité triangulaire n'est pas respectée) mais elle donne de meilleurs résultats que la distance euclidienne. Pour que la qualité de cette métrique de similarité soit cohérente, les sous-traces de chaque couple sont tout d'abord synchronisées entre elles : c'est une synchronisation locale à chaque couple et non globale au groupe des sous-traces. Le résultat de ces deux opérations est une matrice de similarités symétrique comme celle représentée dans la Figure III-2 (page 83).

2-B. Algorithme de classification : la méthode de Ward

La dernière étape avant l'identification du scalaire secret est la classification des sous-traces en deux groupes (celui des additions et celui des doublages) à partir de la matrice de similarité. Deux algorithmes différents ont été étudiés : l'algorithme itératif « k-mean » et la classification ascendante hiérarchique. Je ne présenterai ici que le second car c'est celui qui m'a permis d'obtenir les meilleurs résultats. Mes essais ont montré que l'algorithme « k-mean » est moins efficace pour identifier un groupe d'opérations similaires (les additions) parmi des opérations avec un degré moindre de ressemblance. La première attaque horizontale [Walt01] utilisait déjà un algorithme de classification hiérarchique (non formalisé dans l'article); il s'agit d'une version proche d'algorithmes existants mais modifiée en vue de respecter certaines contraintes comme le nombre de groupes. Ici je me base sur les travaux existant en classification et je montre qu'il est possible d'obtenir deux groupes sans pour autant modifier l'algorithme hiérarchique.

La Figure III-5 décrit la sortie d'un algorithme de classification hiérarchique ascendant à partir d'un extrait d'une matrice de similarité (seulement 8 opérations). L'opération 1 n'est pas proche de l'opération 6 alors que ce sont deux additions (voir Figure III-5). Dans notre exemple, trois opérations (1, 6 et 8) forment le groupe des additions que l'algorithme de classification a pour tâche d'identifier. Deux opérations (1 et 6) semblent différentes mais l'algorithme doit prendre en compte le fait qu'elles sont toutes les deux proches de l'opération

8 pour réussir l'attaque. Ceci montre qu'il ne suffit pas de regarder la similarité entre deux opérations pour décider si elles appartiennent ou non au même groupe (cluster). Il faut aussi prendre en compte le point commun qu'elles ont (être similaires à 8) et utiliser cette information durant la classification.

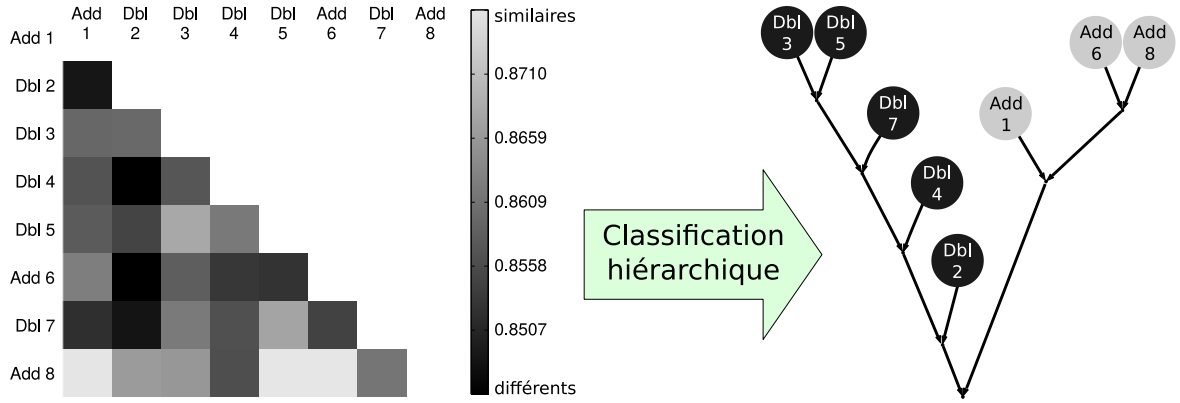


Figure III-5 Exemple de graphe des fusions, sorti d'un algorithme de classification ascendante hiérarchique

Les trois premières étapes de la Figure III-6 (page 89) décrivent une itération de l'algorithme. Il y a autant d'itérations qu'il y a d'éléments à classer (ici, des opérations). Initialement, chaque opération est dans un groupe différent. À chaque itération, les deux groupes les plus proches (étape 1 de la Figure III-6) sont fusionnés pour former un seul groupe contenant plusieurs opérations (étape 2). Après la dernière itération (étape n), il n'y a plus qu'un seul groupe contenant toutes les opérations. L'identification du groupe contenant les additions est alors effectuée en étudiant le graphe des fusions (aussi nommé dendrogramme). Celui correspondant à la Figure III-6 est présenté dans la Figure III-5. La première étape est simple à réaliser car il suffit d'identifier la plus grande similarité dans la matrice d'entrée de l'algorithme. Dans l'exemple, il s'agit des opérations 6 et 8. Pour toutes les autres itérations, il faut définir une manière d'évaluer la similarité entre deux groupes contenant plusieurs opérations. Si la mesure de cette distance entre groupes est bien choisie, la matrice de similarité peut être modifiée pour réduire la taille du problème en fusionnant deux lignes et deux colonnes (étape 3 de la Figure III-6). La classification pour notre attaque est une situation particulière : les additions se ressemblent entre elles tandis que les doublages ne se ressemblent pas entre eux. Nous cherchons donc à identifier un seul groupe cohérent. Les méthodes classiques de comparaison entre groupes comprenant plusieurs opérations sont la distance minimale, moyenne ou maximale de tous les couples avec une opération dans chacun

des groupes. Ces méthodes ne sont pas efficaces dans notre situation particulière car le groupe d'additions, même s'il se forme durant la classification, sera fusionné à une branche de doublages avant la dernière itération. L'identification de la branche des additions est alors complexe et de bons résultats n'ont pas pu être obtenus avec une identification non-supervisée. C'est pourquoi j'utilise la méthode de Ward [Ward63] pour comparer des groupes contenant plusieurs opérations. Cette méthode retarde la fusion des gros groupes. Il en résulte une fusion du groupe des additions à la dernière itération, comme cela est représenté sur le graphe de fusion la Figure III-5 et à l'étape n de la Figure III-6. L'identification du groupe d'additions est alors simple car il s'agit seulement de choisir le groupe contenant le moins d'opérations après l'avant-dernière fusion (il y a moins d'opérations d'addition que d'opérations de doublage). Cette mesure de distance entre deux groupes a été initialement proposée pour des matrices d'entrée exprimant la distance euclidienne de points à classer dans un espace à plusieurs dimensions. Pour ce cas spécifique, il est prouvé qu'à chaque itération la somme des inerties des groupes est toujours minimale [Ward63]. Cette méthode est aussi nommée méthode à variance minimale. La méthode de Ward a été généralisée pour des matrices d'entrée basées sur d'autres métriques que la distance euclidienne [Bata88], la minimisation systématique de l'inertie n'étant alors plus respectée. Cette méthode de comparaison de groupes (comme celles basées sur le minimum, la moyenne ou le maximum) peut être utilisé dans une classification hiérarchique ascendante optimisée. Il s'agit de mettre à jour la matrice de similarité après chaque fusion pour qu'une ligne et une colonne représentent le nouveau groupe. On peut alors supprimer les lignes et colonnes correspondant au groupe fusionné. Après chaque itération, la taille de la matrice diminue (étape 3 sur la Figure III-6) et il suffit alors d'identifier le nouveau maximum de similarité. Les différentes méthodes diffèrent dans la manière de calculer cette nouvelle ligne et colonne après la fusion. L'équation suivante décrit comment calculer la distance entre un groupe A et un groupe résultant de la fusion entre les groupes B et C :

$$dissimilarité(A, B \cup C) = \frac{Card(A \cup B)}{Card(A \cup B \cup C)} \times dissimilarité(A, B)^w + \frac{Card(A \cup C)}{Card(A \cup B \cup C)} \times dissimilarité(A, C)^w - \frac{Card(A)}{Card(A \cup B \cup C)} \times dissimilarité(B, C)^w$$

Il existe deux implémentations de la méthode de Ward, celle pour $w = 1$, et celle pour $w = 2$. Pour une distance euclidienne elles sont complètement équivalentes, mais le graphe des fusions peut être différent en fonction de ce paramètre lorsque l'on utilise la méthode de Ward sur une matrice qui n'a pas été construite avec une distance euclidienne [MuLe14] (c'est le cas pour l'attaque que je décris). Dans mes expérimentation, j'obtiens des résultats très proches avec les deux paramètres, mais pas rigoureusement identiques : bien que les arbres de fusions ne soient pas rigoureusement identique, la branche d'addition contient les même opérations.

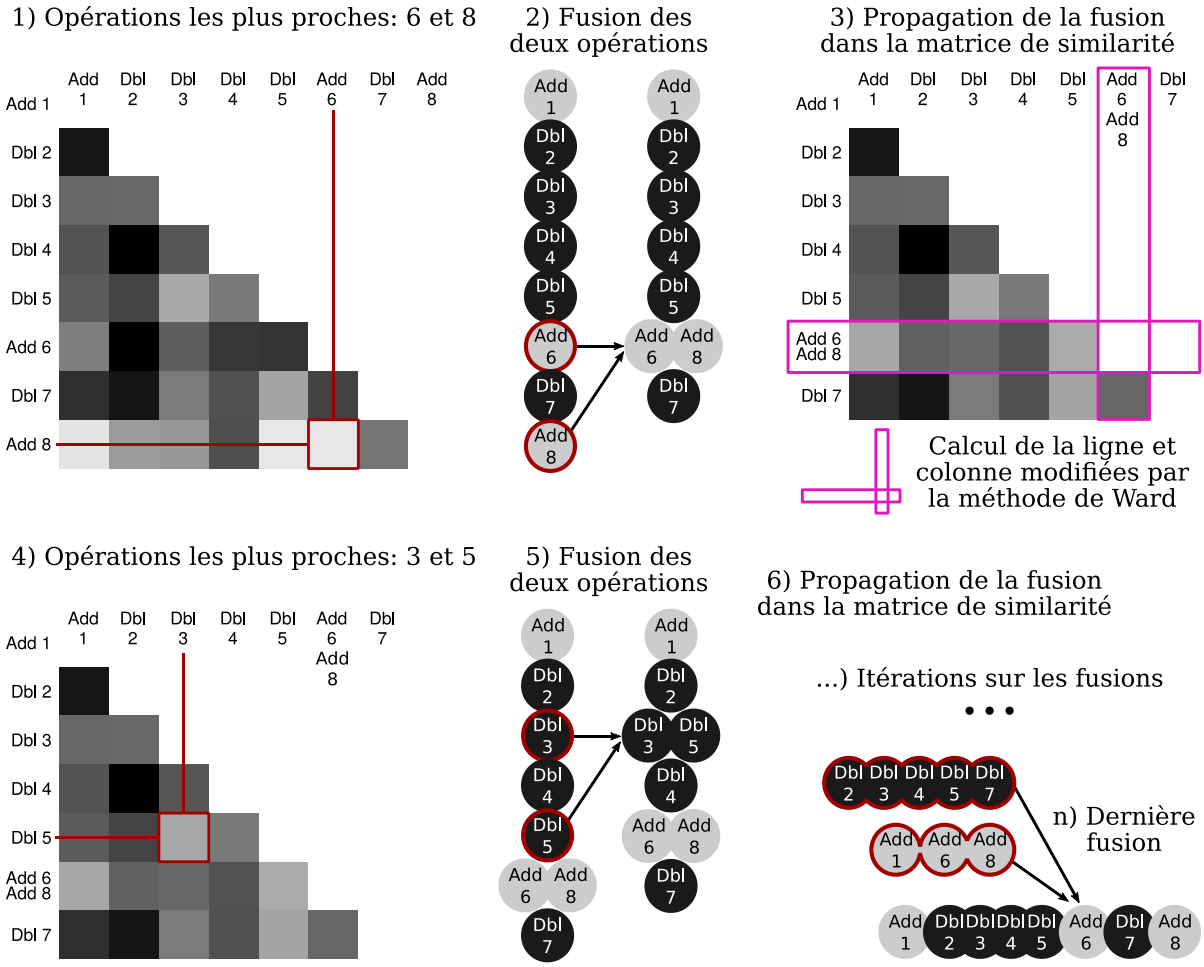


Figure III-6 Classification hiérarchique ascendante

Le résultat de la mesure de corrélation est un degré de similarité. Or les méthodes de classification sont plutôt basées sur des mesures de dissimilarité. Pour pouvoir réutiliser les méthodes existantes, je transforme la matrice de similarité en matrice de dissimilarité :

$$dissimilarité = 1 - similarité = 1 - corrélation\ croisée\ maximale$$

Après la première itération (fusion de 6 et 8), la matrice est réduite en une matrice 7×7 avec à la méthode de Ward. La distance entre le nouveau groupe $6 \cup 8$ et les autres opérations est disponible directement dans la nouvelle matrice. L'itération suivante est la fusion entre deux doublages (3 et 5) qui sont proches l'un de l'autre. Comme ils ne sont pas proches de beaucoup d'additions, leur fusion avec le groupe des additions est retardé jusqu'à la dernière itération. Le doublage 7 rejoint le groupe des doublages avant que l'addition 1 ne rejoigne son groupe. Comme décrit précédemment, le groupe des additions est identifié comme étant le plus petit des deux derniers groupes : la classification est finie. L'attaquant connaît l'ordre des opérations, il est donc très simple de reconstruire la chaîne temporelle des additions et doublages et retrouver le scalaire utilisé comme pour une attaque classique par SPA.

2-C. Exploitation de plusieurs traces

Comme nous allons le voir (section III-3), je n'ai pas réussi à mettre en œuvre avec succès une attaque purement horizontale (exploitant une seule trace). La qualité de la matrice de similarité basée sur une seule trace est trop faible pour effectuer une bonne classification des opérations. Plusieurs optimisations sont possibles pour améliorer la qualité de la matrice : augmenter la bande passante des mesures, choisir des points d'intérêt dans la trace ou utiliser plusieurs traces. J'ai choisi d'espionner plusieurs traces de multiplication scalaire utilisant le même scalaire. Cela restreint tout de même les possibilités de l'attaque. Comme le scalaire doit être utilisé plusieurs fois, l'attaque ne peut pas être menée contre l'algorithme ECDSA (voir Figure I-6, page 24) où un scalaire aléatoire est utilisé pour chaque signature. Si l'attaquant maîtrise le message qui est signé, il est possible d'attaquer l'algorithme EdDSA [BeDu12]. Dans cette variante de l'algorithme ECDSA initialement proposée pour des courbes d'Edwards (permettant aussi l'unification de la formule d'addition), le scalaire n'est plus choisi aléatoirement mais est dérivé à partir du message à signer par une fonction de hachage. En faisant signer le même message plusieurs fois, il est donc possible d'espionner plusieurs multiplications scalaires avec le même scalaire. Cette variante ouvre donc la porte à des attaques exploitant la verticalité. L'échange de secret Diffie-Hellman n'est pas forcément protégé contre les attaques horizontales car il existe des cas d'utilisation ou le scalaire utilisé est toujours la clé privée (voir section I-1 page 24). L'utilisation de plusieurs traces empêche l'attaque de pouvoir être utilisée sur un circuit où le scalaire est modifié aléatoirement avant chaque multiplication scalaire (en ajoutant à ce scalaire un petit nombre aléatoire, fois l'ordre du point générateur : il s'agit de la première contre-mesure proposée dans l'article [Coro99]).

La transformation de l'attaque purement horizontale (voir Figure III-2, page 83) en une

attaque exploitant plusieurs traces est simple. Le résultat de cette transformation est visible sur la Figure III-7. La nouvelle attaque résiste à plusieurs contre-mesures initialement proposées pour protéger les circuits d'attaque DPA grâce à une première phase purement horizontale. Ma proposition est de générer autant de matrices de similarité qu'il y a de traces à exploiter : pour chaque trace, les étapes 1 et 2 sont effectuées (phase horizontale). Suit ensuite la phase verticale où les matrices de similarité (autant que de traces exploitées) sont moyennées pour produire une nouvelle matrice de meilleure qualité. Le point opérande de la multiplication n'a pas à être connu par l'attaquant et les matrices moyennées entre elles peuvent provenir de multiplications scalaires avec des points opérandes différents grâce à la première phase horizontale. La moyennisation n'est pas faite directement sur les traces mesurées, mais sur les matrices de similarité après l'étape 2. Cela signifie que cette attaque peut aussi être mise en œuvre contre des circuits protégés par la contre-mesure DPA où la représentation du point opérande est modifiée aléatoirement en début de calcul (c'est la troisième contre-mesure proposée dans [Coro99]). Bien que j'utilise plusieurs traces, la phase horizontale de l'attaque permet de résister à certaines contre-mesures proposées contre les attaques par DPA. Comme nous le verrons dans la suite, la matrice moyennée a une meilleure qualité. C'est cette nouvelle matrice qui est utilisée en entrée de l'étape 3 (classification) avant de pouvoir identifier le scalaire utilisé.

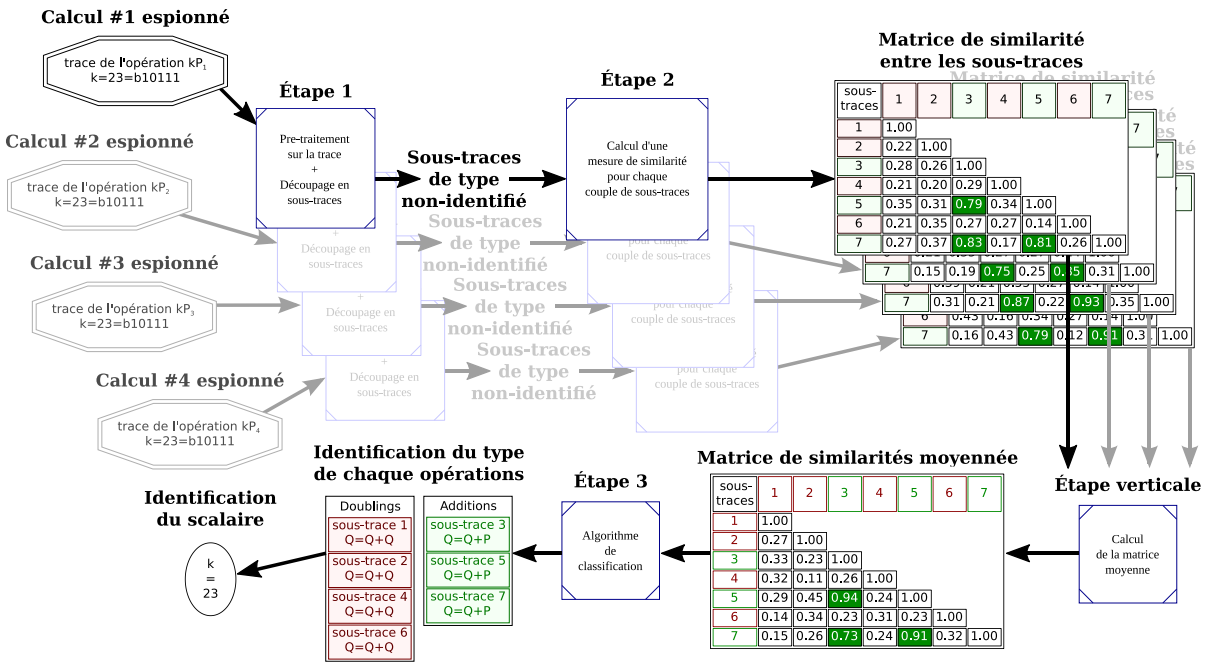


Figure III-7 Schéma de l'attaque exploitant plusieurs traces : une première phase horizontale suivie d'une phase verticale

3. Résultats expérimentaux et perspectives

Je propose tout d'abord d'estimer l'efficacité des étapes 1 et 2 de l'attaque. Pour cela, je définis une métrique permettant d'évaluer la qualité d'une matrice de similarité. Cette métrique n'est pas utilisable par un attaquant car elle requiert de connaître le type de chaque opération (addition et doublage) ; elle ne servira donc qu'à évaluer l'attaque et justifier mes choix. La Figure III-8 décrit la méthode que j'ai mise en œuvre pour calculer la qualité d'une matrice de similarité. La première étape consiste à isoler deux groupes de couples d'opérations : le groupe contenant uniquement des couples d'additions et le groupe contenant des couples addition-doublage. Les similarités de chacun de ces groupes sont triées dans un ordre différent. Une normalisation du nombre de couples dans chacun des groupes permet de faire une comparaison quantitative. Pour formaliser la construction présentée dans la Figure III-8 : la qualité est définie comme le pourcentage q maximal tel que q pourcent des similarités entre additions soient supérieurs à $100 - q$ pourcent des similarités entre additions et doublages. Cela signifie que plus les additions sont similaires entre elles ou plus elles sont différentes des doublages, plus cette qualité sera grande (proche de 100%). Un des avantages de cette méthode de mesure de qualité est qu'elle est indépendante de la similarité moyenne de la matrice. De plus, elle est rapide à calculer même sur des grandes matrices de similarité : il suffit d'effectuer deux tris.

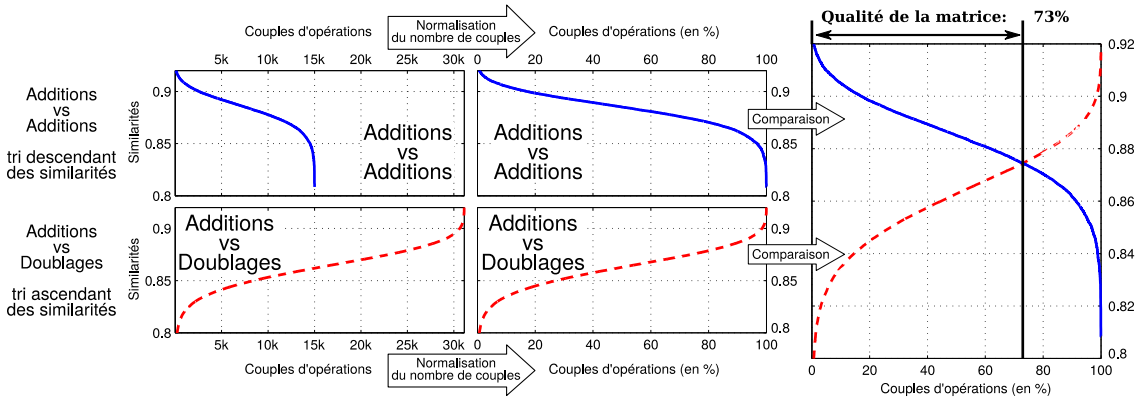


Figure III-8 Méthode de mesure de la qualité d'une matrice de similarité

La Figure III-9 montre que cette qualité atteint seulement 60% pour une attaque purement horizontale (attaque sur une seule trace) : ceci est insuffisant pour que l'algorithme de classification effectue une identification cohérente des opérations. Par contre on observe que la qualité de la matrice s'améliore si le nombre de traces augmente. Comme expliqué précédemment (voir section III-2-A page 85), la métrique basée sur la corrélation permet

d'obtenir des matrices de similarité de meilleure qualité. La Figure III-9 compare cette métrique à la distance euclidienne entre sous-traces après synchronisation. Cette comparaison est faite à partir de traces de consommation réelles (dans les conditions décrites dans la section III-1 page 80) d'un crypto-processeur effectuant plusieurs multiplications scalaires entre un scalaire fixé et des points différents (choisis au hasard) sur la courbe quartique de Jacobi de 256 bis #1 décrite dans l'Annexe 4.

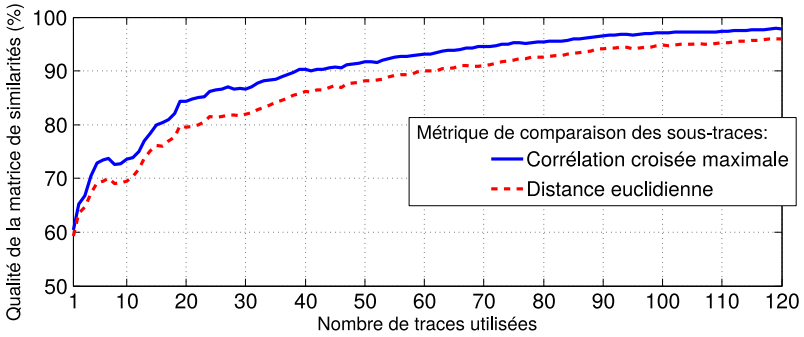


Figure III-9 Qualité de la matrice de similarité pour deux métriques de comparaison des sous-traces

L'attaque que je propose dans ce chapitre a été mise en œuvre dans différentes conditions pour évaluer son efficacité. Cinq courbes quartiques de Jacobi générées aléatoirement (voir l'Annexe 4) ont été attaquées. Pour chacune de ces courbes, 12 scalaires ont été générés aléatoirement. Pour chaque courbe et chaque scalaire, 80 traces de puissance consommée de 80 multiplications scalaires ont été enregistrées. Ceci représente environ 60 Go de traces compressées. Pour chacune des 4800 multiplications scalaires espionnées, le point opérande a été choisi au hasard sur la courbe. Contrairement à une attaque par DPA, ni les points opérandes ni leurs représentations ne sont connus de l'attaquant. Une trace est obtenue en 20 secondes (temps de téléchargement depuis l'oscilloscope). La trace est ensuite traitée pour en déduire la matrice de similarité associée ; cette opération est effectuée en une minute sur un processeur XEON (Intel) cadencé à 2 GHz. Le temps de calcul pour exécuter l'algorithme de classification est négligeable devant le temps de calcul des matrices de similarité. Cela signifie qu'il est peu couteux d'ajouter une trace à un attaque déjà exécutée si les matrices de similarité ont été sauvegardées.

La Figure III-10 montre l'efficacité de différentes attaques contre des courbes quartiques de Jacobi sur 256 bits en fonction du nombre de traces utilisées. Elle représente le nombre d'erreurs après l'étape de classification. En utilisant une seule trace, la classification effectuée contient en moyenne 110 opérations mal classifiées sur les 378, soit 35% d'opérations bien classifiées supplémentaires par rapport à une classification au hasard. Comme nous le verrons dans la suite, une seule trace ne suffit pas pour retrouver le scalaire. L'utilisation de plusieurs

traces permet d'améliorer la qualité de la matrice de similarité. Le nombre moyen d'opérations mal classifiées descend à 80 pour 4 traces, 40 pour 10 traces, 10 pour 20 traces et 2 pour 40 traces. Sur les 48 attaques contre des courbes de 256 bits, la plus efficace a permis une classification exacte en exploitant seulement 28 traces.

Comme le montre l'attaque sur la courbe quartique de Jacobi sur 256 bit #3 sur la Figure III-10, l'efficacité de l'attaque est variable. Par exemple une attaque contre cette courbe utilisant une seule trace a permis de classifier les opérations avec seulement 40 erreurs contre 110 en moyenne. Chaque trace représente la puissance consommée par une multiplication scalaire avec un point opérande différent. L'explication semble donc être que certains points permettent d'obtenir des traces plus exploitables. Les coordonnées (dans le domaine de Montgomery) des points permettant d'obtenir des matrices de similarité de bonne qualité ont été étudiées mais aucune particularité n'a pu être identifiée. Par exemple les poids de Hamming des coordonnées de ces points spécifiques ne sont pas différents de ceux des autres points.

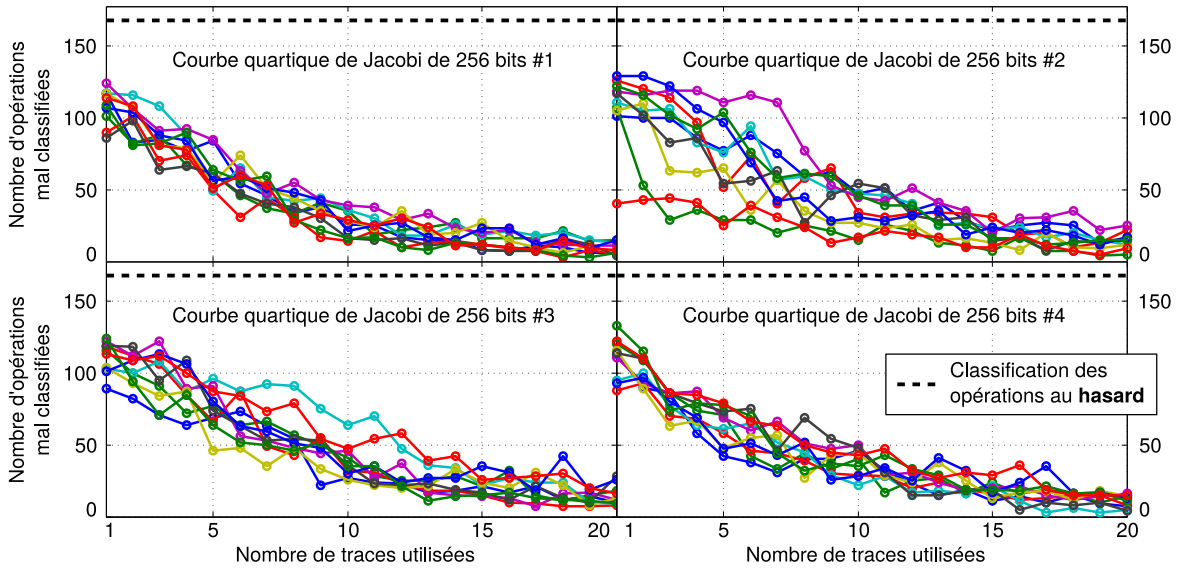


Figure III-10 Efficacité de 48 attaques exploitant quelques traces de multiplications scalaires

Il est imaginable que l'utilisation d'un niveau de sécurité plus important puisse faciliter les attaques par canaux auxiliaires [Walt04]. En effet, la taille de la matrice de similarité étant plus grande (car il y a plus d'opérations basiques sur les points), il y a plus d'information en entré de l'algorithme de classification. De plus le temps de calcul des opérations sur $GF(p)$ nécessitant plus de temps (car les nombres manipulés sont plus grands), il est imaginable que la qualité des similarités soit plus importante.

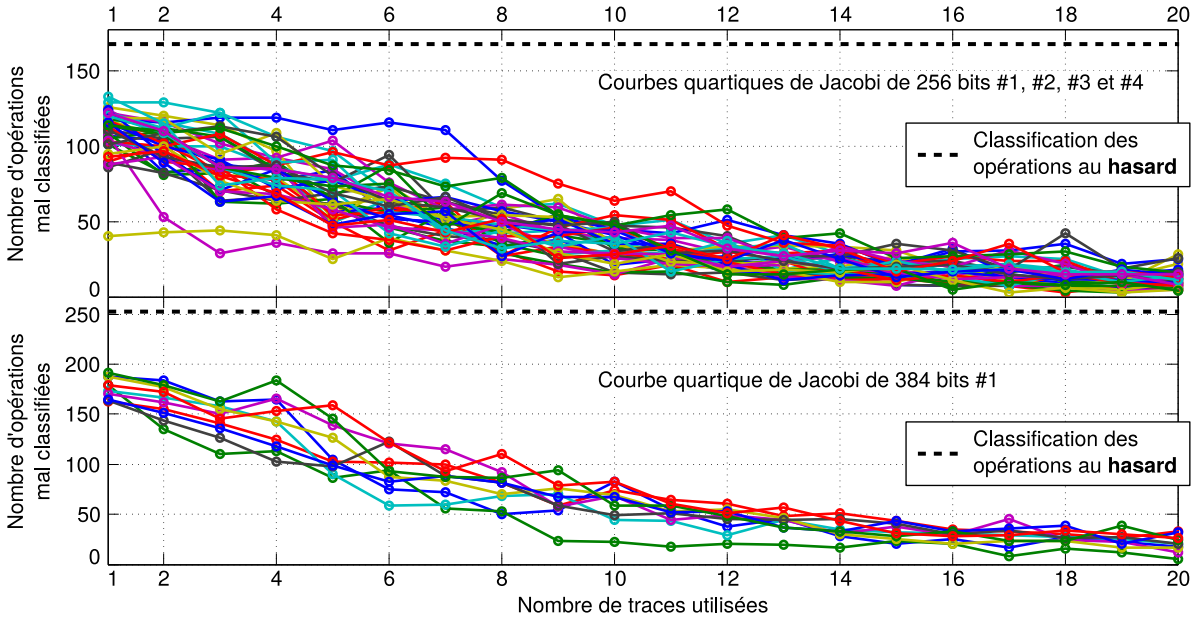


Figure III-11 Comparaison entre des attaques contre une courbe de 384 bits et des courbes de 256 bits

La Figure III-11 compare les résultats de la classification d'attaques contre une courbe quartique de Jacobi de 384 bits par rapport à ceux contre une courbe de 256 bits. En moyenne les résultats des classifications sont identiques pour les deux tailles de courbes par rapport au nombre d'opérations à classifier. L'hypothèse selon laquelle une attaque par canal auxiliaire sur une courbe de taille plus grande est plus efficace n'a pas pu être vérifiée.

Le nombre d'opérations mal classifiées permet d'observer l'évolution de la qualité des résultats de classification. Il est possible de définir une autre métrique. Même si la classification est possiblement erronée, l'attaquant peut essayer de retrouver le scalaire secret à partir de celle-ci. Je définis la seconde métrique comme étant la complexité d'une attaque par force brute orientée, exploitant la sortie de l'étape de classification (la chaîne d'additions et de doublages) pour retrouver le scalaire. Pour cette force brute orientée, il est possible de faire des hypothèses sur l'algorithme de multiplication scalaire utilisé. Je me restreindrai ici aux hypothèses déjà utilisées dans les premières étapes de l'attaque : la multiplication scalaire est de gauche à droite et il n'y a ni points pré-calculés ni opérations fictives. Cela signifie que la multiplication scalaire est uniquement constituée d'opérations sur un point accumulateur : les additions ($Q = Q + P$) et les doublages ($Q = Q + Q$). Je ne suppose pas que l'algorithme de multiplication scalaire est « Doublages et Additions ». Ceci permet d'attaquer des multiplications scalaires où plusieurs additions de points pourraient s'enchainer sans être séparées par un doublage de points. La première étape de l'attaque par force brute orientée est de vérifier si la chaîne d'additions et doublages sortant de l'algorithme de classification

correspond à un scalaire cohérent. Pour une attaque contre ECDSA, l'attaquant peut retrouver la clé secrète associé au scalaire à tester en utilisant les informations disponibles dans la signature avec seulement quelques opérations sur $GF(p)$ (voir section I-2-E page 25). Il peut alors valider si le scalaire est cohérent en vérifiant si la clé privée qu'il a déduite correspond à la clé publique du signataire. La même opération est possible contre ECDH, l'attaquant vérifie que la multiplication scalaire entre le point générateur et le scalaire à tester correspond bien à la clé publique. Pour ECDH avec des clés éphémères, l'attaquant réalise la même opération mais en utilisant un point transmis entre les deux interlocuteurs au lieu de la clé publique. Dans toutes ces situations, le test d'un scalaire coûte à l'attaquant une multiplication scalaire. Si le test échoue, cela signifie que la chaîne d'additions et doublages est erronée. L'attaquant effectue alors la même vérification à partir des 380 chaînes d'opérations basiques sur les points obtenues en inversant le type d'une seule opération. Le coût calculatoire de ces vérifications est de 380 multiplications scalaires, autant qu'il y a de chaînes à tester. Tant que l'attaquant n'a pas validé un scalaire potentiel, il continue cette attaque en inversant de plus en plus d'opérations. Si l'attaquant sait que l'algorithme de multiplication scalaire « Doublages et Additions » est utilisé, il peut éliminer toutes les chaînes contenant deux additions consécutives ; comme expliqué précédemment, je ne fais pas cette hypothèse. Pour chaque attaque (une des 48 attaques contre une courbe sur 256 bits exploitant un certain nombre de traces), il est possible de prévoir le nombre de multiplications scalaires moyen permettant d'identifier le scalaire secret avec l'attaque force brute orientée par la sortie de l'algorithme de classification. La complexité de cette dernière étape en fonction du nombre de traces est représentée pour les 48 attaques dans la Figure III-12. À partir de 13 traces, l'attaque par force brute orientée est moins complexe qu'une attaque mathématique (le logarithme discret avec l'algorithme « Rho de Pollard »). En faisant l'hypothèse que l'attaquant dispose d'une semaine et est capable de calculer une multiplication scalaire en 1 ms (puissance de calcul accessible à tout un chacun), il pourra tester 600 millions de scalaires, donc réussir une attaque par force brute d'une complexité de 29 bits. L'attaquant réussira en une semaine la moitié des 48 attaques avec seulement 32 traces, 90% des attaques avec 37 traces et la totalité des attaques avec 46 traces. Avec la même puissance de calcul, l'attaquant peut réussir une attaque par force brute orientée d'une complexité de 22 bits en 1 heure et 90% des 48 attaques sont réalisables en exploitant 50 traces dans ces conditions.

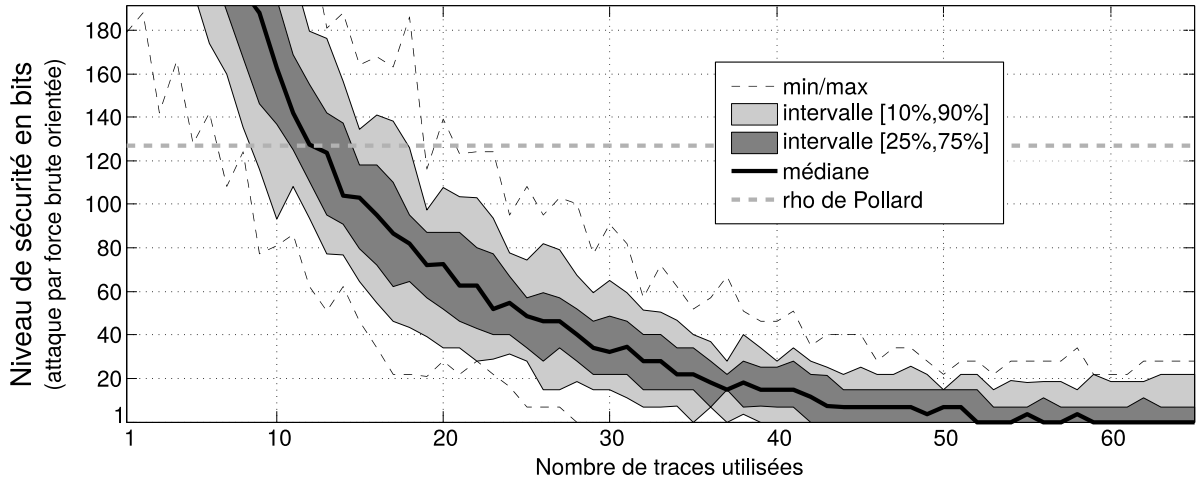


Figure III-12 Complexités d'attaques par force brute orientée en fonction du nombre de traces sur des courbes de 256 bits

Ces résultats montrent que l'attaque proposée est possible avec seulement quelques dizaines de multiplications scalaires espionnées. Il est donc imaginable qu'une attaque purement horizontale (utilisable contre une implémentation du protocole ECDSA) soit possible avec quelques améliorations. C'est pourquoi je propose ici quelques pistes pour améliorer l'attaque. La première des améliorations qui peut être mise en œuvre est une amélioration de la qualité des traces de puissance consommée ou l'utilisation de canaux auxiliaires alternatifs tels que le champ électromagnétique émis. La localité offerte par une mesure électromagnétique permettrait possiblement de différencier les additions des doublages en détectant si le point P est lu ou non. L'exploitation de seulement quelques points d'intérêt sur les traces n'a pas été étudiée car je suppose l'ordonnancement inconnu de l'attaquant. Cependant, les résultats montrent qu'en exploitant une seule trace, la prédiction du type des opérations n'est pas parfaite mais meilleure qu'un choix au hasard ; il est donc imaginable d'exploiter cette première classification infructueuse en vue de raffiner l'attaque en identifiant des points d'intérêt [PeIm14]. L'étape verticale de moyennisation des matrices de similarité fait disparaître de l'information ; par exemple il est impossible d'identifier sur la matrice moyennée que deux opérations sont estimées proches l'une de l'autre par toutes les traces sauf une seule qui les jugerait très différentes. Un algorithme de classification qui prendrait en entrée toutes les matrices de similarité (multi-variables) permettrait de s'affranchir de l'étape de moyennisation pour pouvoir, par exemple, être plus robuste en éliminant des informations qui semblent incohérentes. Il est aussi possible d'utiliser la sortie de l'algorithme de classification comme état initial d'une seconde classification itérative telle que l'algorithme « k-mean ». Cependant mes essais ont montré que cet algorithme converge

majoritairement vers la même classification en partant d'un état initial choisi au hasard ou imposé par l'algorithme de classification hiérarchique. Comme je l'ai montré précédemment, l'attaque par force brute orientée peut être améliorée en faisant des hypothèses sur l'algorithme de multiplication scalaire utilisé. Une autre piste d'amélioration est d'utiliser la matrice de similarité ou l'ordre des fusions dans le graphe (dendrogramme) pour inverser en priorité le type des opérations dont la classification semble incertaine. Il serait aussi intéressant d'étudier d'autres circuits pour valider l'indépendance de l'attaque vis-à-vis de l'architecture de la cible.

4. Extension de l'attaque aux multiplications scalaires par fenêtrage

J'ai montré que l'attaque proposée permet de distinguer les opérations d'addition de points des doublages, par espionnage de l'exécution de l'algorithme « Doublages et Additions ». Cette identification est faite en détectant la réutilisation d'un point opérande (P). Si la multiplication scalaire est effectuée de gauche à droite avec un fenêtrage (par exemple de 2 bits), alors quelques points sont préliminairement calculés (P, $2 \times P$ et $3 \times P$). La chaîne d'opérations basiques sur les points est alors constituée de 4 types d'opérations : le doublage de points ($Q = Q + Q$) et des trois types d'additions correspondant à trois second opérandes ($Q = Q + P$, $Q = Q + 2 \times P$ ou $Q = Q + 3 \times P$). Je vais montrer qu'une extension de l'attaque présentée précédemment permet de distinguer les doublages des additions mais aussi de classifier les additions en autant de groupes qu'il y a de points pré-calculés. La Figure III-13 montre un extrait du graphe de fusion en sortie de l'algorithme de classification d'une attaque dans les conditions suivantes : utilisation d'une formule unifiée sur une courbe quartique de Jacobi de 256 bits; multiplication scalaire avec l'algorithme « 3-ary » ; classification hiérarchique basée sur la moyenne des distances entre groupes au lieu de la méthode de Ward et exploitation de 70 traces de multiplications scalaires utilisant un scalaire secret identique. La première observation est qu'il apparaît trois grandes branches correspondant chacune à des additions par un des points pré-calculés. L'analyse du graphe des fusions (dendrogramme) peut donc permettre d'identifier que le circuit espionné utilise une multiplication scalaire par fenêtrage. Il est même possible d'identifier les doublages et de classer les additions en 3 groupes correspondant à 3 points, petits multiples de P. L'attaquant ne connaît pas a priori le digit associé à chacun des groupes d'additions. Le nombre de possibilités étant restreint, une attaque par force brute est tout de même possible. Une autre possibilité est d'effectuer une comparaison entre ces additions et celles de la phase de pré-

calcul [Walt01]. Une fois la chaîne d'opérations ou les quelques chaînes possibles identifiées, l'attaquant peut vérifier que le scalaire secret a été découvert.

L'attaque que je propose peut aussi être utilisée contre un algorithme de multiplication scalaire par fenêtrage de gauche à droite reposant sur des formules unifiées (sur des courbes elliptiques classiques). Une attaque préliminaire par SPA permet de sélectionner uniquement les sous-traces des additions. C'est sur ces sous-traces d'additions de points qu'il est alors possible de mettre en œuvre l'extension de mon attaque en vue de classer les additions en groupes associés chacun à un digit différent.

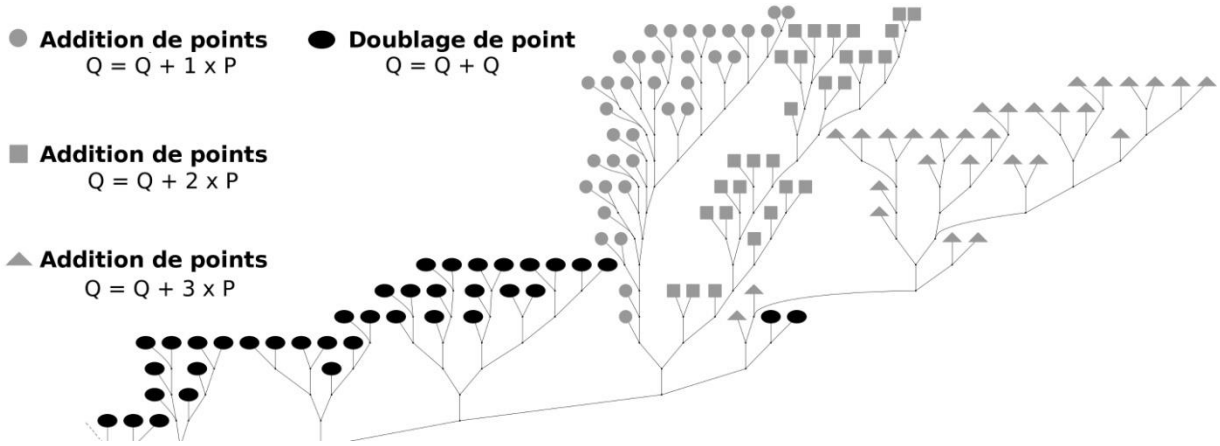


Figure III-13 Extension de l'attaque contre une multiplication scalaire par fenêtrage de 2 bits

5. Contre-mesures

Pour se protéger contre cette attaque, il faut que la réutilisation d'opérandes soit indépendante de la clé. Pour arriver à cette fin, je propose des contre-mesures permettant d'éliminer la réutilisation d'opérandes. Comme je l'ai montré, choisir une représentation de P au hasard avant de commencer le calcul de $k \times P$ ne protège pas d'attaques horizontales. Cette technique peut tout de même être adaptée pour éviter la réutilisation d'opérandes dans un algorithme de multiplication de gauche à droite. Pour cela, il faut changer la représentation de P après chaque addition. Pour éviter que la détection du changement de représentation permette à l'attaque de différentier les doublages et les additions, ce changement doit être effectué après chaque opération basique sur les points. Ce changement de représentation peut être effectué en tirant au hasard un élément de $GF(p)$ non nul et en multipliant chacune des 4 coordonnées (X, Y, Z, T) par ce nombre. Après chaque utilisation de la formule d'addition (11 multiplications sur $GF(p)$), il faut ajouter 4 multiplications. Le surcoût de cette contre-mesure est donc d'environ 36% sur le temps de calcul.

Il est aussi possible d'éviter la réutilisation d'opérande en effectuant la multiplication scalaire de droite à gauche. Dans cette situation, le point accumulateur (Q_1) qui contiendra le résultat final n'est jamais doublé, c'est un second accumulateur (Q_2) contenant initialement le point P qui est doublé (voir Figure I-19, page 36). Les deux opérations basiques sur les points sont :

$$Q_2 = Q_2 + Q_2 \qquad \text{et} \qquad Q_1 = Q_1 + Q_2$$

Dans cette situation, il n'y jamais réutilisation d'un point opérande. Par contre, l'optimisation des performances basée sur du fenêtrage n'est plus possible avec une multiplication scalaire de droite à gauche (voir section I-3-C page 38). Cette contre-mesure reste sensible à des attaques détectant un doublage de points en identifiant les élévations au carré sur $GF(p)$ [ClFe12]. De la même manière, l'échelle de Montgomery peut être utilisée pour protéger le circuit. La pertinence de l'unification de la formule d'addition de points peut être remise en question pour un algorithme régulier tel que celui-ci. Il en est de même pour l'algorithme « Doublages et Additions systématiques » sauf qu'il est peut être possible de détecter une addition fictive si elle est effectuée sur le point accumulateur et que celui-ci est réutilisé par le doublage qui suit comme cela a été réalisé sur une implantation du RSA [Wiva11]. Les contre-mesures permettant d'ajouter du hasard au niveau de la multiplication scalaire ou des opérations sur $GF(p)$ tout au long du calcul restent efficaces contre les attaques horizontales (par exemple, l'utilisation d'un domaine de Montgomery aléatoire [LeHs12]).

6. Conclusion

Pour conclure, l'attaque que je présente permet de détecter la réutilisation d'un point même quand une protection par formule unifiée est utilisée. Ceci permet de retrouver le scalaire utilisé par de nombreux algorithmes de multiplication scalaire. L'efficacité de l'attaque a été évaluée dans des conditions réelles sur cible FPGA ; il ne s'agit pas d'une évaluation sur des traces simulées ou sur un modèle de fuite. Aucune hypothèse sur le multiplieur sur $GF(p)$ utilisé n'a été faite contrairement à la majorité des attaques proposées et utilisables contre la formule unifiée [AmFe09; Walt04; StTh06; Walt01; ClFe12; BaJa15]. L'implantation du multiplieur n'a même pas à être connue de l'attaquant. Il n'est pas non plus nécessaire de connaître l'ordonnancement de la formule d'addition contrairement à toutes les attaques visant le multiplieur sur $GF(p)$. Contrairement à des attaques par collisions [YeKo05; HoMi08; DuPa16], il n'est pas nécessaire que l'attaquant choisisse le point opérande de la

multiplication scalaire. Contrairement à une attaque DPA [Coro99], le point opérande n'est jamais utilisé par l'attaquant et n'a même pas à être connu. Pour l'instant, l'attaque n'est pas purement horizontale mais le nombre de traces utilisées étant petit (quelques dizaines), il semble possible de réussir une attaque exploitant une seule trace en améliorant l'attaque et/ou la qualité des traces (bande passante et rapport signal à bruit). L'attaque ne prend aucun paramètre en entrée, elle est totalement non-supervisée : aucun seuil n'a à être défini. Une classification s'apparentant à de la classification hiérarchique modifiée a déjà été proposée [Walt01] et appliquée sur des traces de produits par digits simulés. Mon étude sur différents algorithmes de classification a permis d'identifier que la méthode de classification hiérarchique ascendante non modifiée qu'est la méthode de Ward est efficace dans un cas d'attaque réel par canaux auxiliaires, notamment pour résoudre le problème complexe d'identification d'un groupe cohérent parmi d'autres opérations qui ne se ressemblent pas. Ce problème est celui qu'il faut résoudre dans le cas d'une attaque contre une multiplication scalaire de gauche à droite protégée par l'utilisation d'une formule unifiée. Comme je l'ai montré, la représentation du graphe en sortie de la classification peut être utilisée par l'attaquant pour découvrir des informations sur le circuit, comme par exemple détecter l'utilisation d'un fenêtrage même si une formule unifiée est utilisée.

Chapitre IV Utilisation d'opérations fictives contre les attaques par analyse de la puissance consommée

Dans le Chapitre III, j'ai étudié une contre-mesure existante au niveau de l'addition de points. Dans ce Chapitre, je propose une contre-mesure au niveau supérieur en vue de protéger un circuit contre des attaques par canaux auxiliaires : il s'agit d'un nouvel algorithme de multiplication scalaire. L'étude a été appliquée à une cible matérielle mais ma contre-mesure est aussi compatible avec des implantations logicielles. Cette protection peut se combiner avec l'unification de la formule d'addition mais elle a été initialement conçue pour des courbes de Weierstrass (cas étudié) sensibles à une attaque par SPA. L'étude de la faisabilité et du coût de ma contre-mesure sera faite sur une version modifiée du crypto-processeur présenté dans le Chapitre II. Ce travail se différencie de l'existant par l'objectif visé : ne pas impacter les performances du crypto-processeur. L'utilisation d'une méthode de fenêtrage me permet d'économiser des opérations qui peuvent alors être dépensées dans le but de perturber l'attaquant en insérant, à des moments aléatoires, des opérations fictives. Pour perturber encore plus l'attaquant, la taille des fenêtres utilisées est définie aléatoirement. Je montrerai qu'une utilisation astucieuse d'opérations fictives présente de nombreux avantages. La contre-mesure proposée est résistante contre les attaques par SPA ou par « C-Safe » erreurs. La désynchronisation induite permet aussi d'augmenter la résistance du circuit contre les attaques par DPA. Ma contre-mesure est modulable car elle peut affecter plus ou moins le niveau de sécurité contre des attaques par canaux auxiliaires, les performances du circuit et le coût en surface. Cela signifie, par exemple, qu'un concepteur de circuit peut facilement mesurer le coût de la contre-mesure qui lui permettra de satisfaire des exigences fixées en termes de performance et de niveau de sécurité. Pour évaluer le niveau de sécurité de la contre-mesure, une attaque basée sur l'identification de motifs dans plusieurs traces de puissance consommée a été développée. Je montre que l'utilisation d'additions fictives n'est pas suffisante pour protéger un circuit contre des attaques exploitant plusieurs traces et qu'un bon niveau de sécurité ne peut être atteint qu'en insérant également quelques doublages fictifs. Ces travaux ont fait l'objet de publications dans des conférences [PoMa14a, PoMa14b, PoMa14c] et dans un journal [PoMa16].

Il existe déjà quelques contre-mesures permettant d'allier l'amélioration des performances et du niveau de sécurité vis-à-vis des canaux auxiliaires (voir section I-5-C page 44). La méthode O-WM [ItYa02] est la plus proche de celle que nous proposons. Je propose aussi de tirer bénéfice du fenêtrage mais contrairement à la méthode O-WM, j'utilise des opérations fictives. Je m'interdis de biaiser le choix aléatoire des tailles de fenêtres pour améliorer les performances (comme dans [AhHa03]). En effet, un attaquant pourrait optimiser son attaque sachant que la probabilité des fenêtres de petites tailles est plus faible.

Pour simplifier, dans tout ce Chapitre j'utiliserai le terme « aléatoire » ou « hasard » mais il s'agit en réalité de données provenant d'un générateur de nombres pseudo-aléatoires (« PRNG »). Le générateur que j'utilise est un bloc matériel implémentant le chiffrement par flux « Trivium » [DePr08]. Dans un crypto-processeur complet, un générateur de nombres véritablement aléatoires (« TRNG ») est disponible et utilisable en remplacement du « PRNG » que j'utilise. Si son débit n'est pas assez important pour la quantité d'aléas que consomme ma contre-mesure, il est possible d'utiliser un « PRNG » initialisé par le « TRNG ».

1. Fenêtrage aléatoire

Comme je l'ai montré précédemment, l'utilisation d'une méthode de fenêtrage parcourrant le scalaire par digits permet d'économiser des opérations d'addition et quelques opérations de doublage dans une multiplication scalaire de gauche à droite (voir section I-3-C : Figure I-20 et Tableau I-1 page 37). En vue de protéger un circuit contre un attaquant exploitant des canaux auxiliaires, je propose ici une méthode par fenêtrage aléatoire. Ceci permet de combiner amélioration des performances et protection de la multiplication scalaire. Je montre ici que cette protection n'est pas suffisante et je propose par la suite des améliorations.

J'ai déjà montré comment mettre en œuvre du fenêtrage à l'aide d'une table permettant de stocker des points pré-calculés (voir section II-3 page 37). Ceci a été appliqué à l'algorithme de multiplication scalaire « m-ary ». Je note la capacité de la table en nombre de points : $2^{W_{max}} - 1$ avec W_{max} qui correspond au m de « m-ary ». Cette table permet de traiter des digits du scalaire de W_{max} bits avec seulement une addition de points et W_{max} doublages de points. Mais elle permet aussi de traiter des digits de taille plus réduite : cette flexibilité n'est pas exploitée par l'algorithme « m-ary » car elle impacterait les performances . Il est possible de parcourir le scalaire par digits de taille aléatoire choisie entre W_{min} bits et W_{max} bits ($W_{max} \geq W_{min} \geq 1$) ; c'est la méthode par fenêtrage aléatoire. Ces seuils de taille de fenêtre

sont fixés pendant l'étape de synthèse logique. La Figure IV-1 montre un exemple de cet algorithme pour un scalaire de taille réduite (2995) avec des fenêtres aléatoires dont la taille varie entre 2 bits (W_{min}) et 4 bits (W_{max}). Il n'est pas nécessaire d'effectuer un recodage du scalaire, la taille de chaque fenêtre étant choisie aléatoirement avant d'être traitée. L'architecture présentée dans la Figure II-7 (page 64) s'adapte parfaitement à un fenêtrage de taille non fixe car il suffit de réinitialiser le registre de fenêtre et d'effectuer un nombre variable de décalages (à chaque doublage de points) du registre à décalage du scalaire. En notant q le nombre de bits du scalaire, cela correspond à q/W_{min} additions et $q - W_{min}$ doublages. Dans toutes les autres situations où moins d'opérations sont nécessaires, chaque opération économisée sera dépensée en une opération fictive insérée à un instant aléatoire. C'est ce pire cas qui fixera donc les performances du circuit : en choisissant un W_{min} de 1 bit on atteindra les performances de l'algorithme « Doublages et Additions Systématiques » ; en augmentant la taille des fenêtres, on améliore les performances du circuit en réduisant le nombre d'additions.

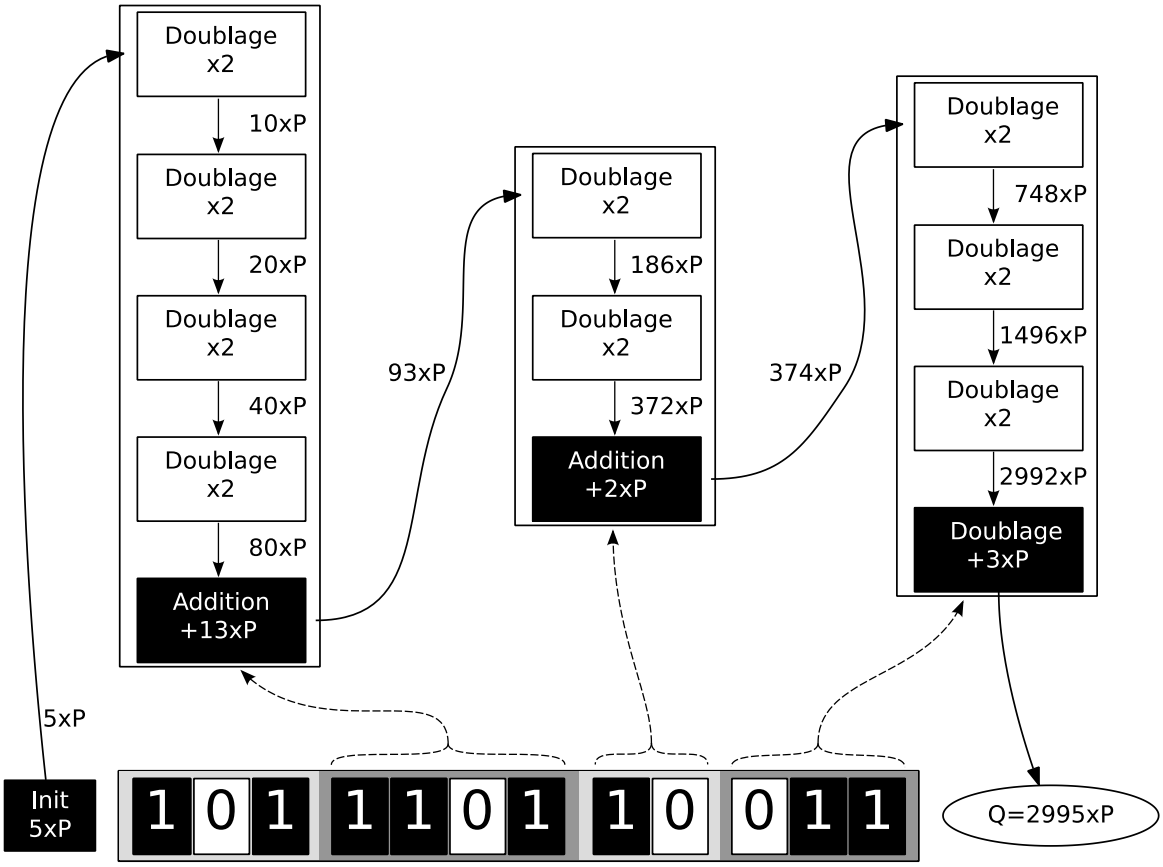


Figure IV-1 Exemple de fenêtrage aléatoire

Il reste tout de même impossible de traiter des digits nuls avec une addition de points car la formule d'addition de points sur une courbe de Weierstrass ne peut pas être utilisée avec pour opérande le point à l'infini P_∞. Quand cette situation se présente l'opération d'addition est tout simplement non effectuée et le traitement de la fenêtre suivante commence. Dans les cas où une addition complète est utilisée, comme c'est possible sur les courbes quartiques de Jacobi, on peut sommer le point à l'infini à un autre, mais les coordonnées nulles dans la représentation de ce point laissent à penser que son utilisation restera identifiable avec une attaque par canaux auxiliaires Je rappelle que ce chapitre se focalise sur l'algorithme de multiplication scalaire et que je ne fais pas d'hypothèses sur les niveaux inférieurs tels que l'addition de points ou le corps utilisé. Je me place donc dans le pire cas, c'est-à-dire que l'addition et le doublage ne sont pas unifiés. Mon modèle d'attaquant suppose qu'il a les capacités de réaliser une SPA pour retrouver la chaîne d'additions et de doublages de points et qu'il peut effectuer une attaque du type « C-Safe » erreur.

2. Utilisation astucieuse d'additions fictives

Même si cela semble difficilement exploitable, pour éviter toute fuite à travers le canal auxiliaire du temps de calcul, j'ai choisi de réaliser un algorithme de multiplication scalaire en temps constant. Pour obtenir cette propriété, il faut tout d'abord connaître le nombre maximal d'additions et de doublages qui pourrait être effectué. Dans notre situation, cela correspond au pire cas (du point de vue des performances) où toutes les fenêtres ont été tirées au hasard de taille minimale (W_{min}) et aucun digit nul n'a été rencontré.

2-A. Minimisation des fuites d'informations

Pour minimiser la lecture du secret qu'est le scalaire, j'ai choisi de réaliser le découpage aléatoire du scalaire tout au long du calcul. Ce n'est qu'à la fin du traitement d'une fenêtre qu'il est possible de connaitre la taille de la fenêtre suivante et si une addition vient d'être économisée à cause d'un digit nul ou non. Lorsqu'une fenêtre de taille plus grande que W_{min} est utilisée ou si un digit nul est rencontré, un compteur d'additions fictives disponibles est mis à jour. Le nombre d'additions fictives disponibles est donc incrémenté en cours de calcul en fonction du scalaire et de son découpage.

Le découpage du scalaire et la position des digits nuls sont des informations sensibles qui

peuvent être utilisées par un attaquant pour monter une attaque par canaux auxiliaire ou resynchroniser une attaque. Cela signifie que les additions économisées ne peuvent pas être dépensées instantanément en additions fictives. En effet l'attaquant étant capable d'identifier une opération fictive par multiplication scalaire avec une attaque par « C-Safe » erreur, si la position de cette opération est liée à une information sensible, il y a fuite d'information. C'est le cas pour l'algorithme « Doublages et Additions Systématiques » où la présence d'une addition fictive est directement liée à un bit nul dans le scalaire secret. L'utilisation astucieuse des additions fictives que je propose est la suivante : les additions économisées ne sont dépensées en additions fictives qu'après un délai aléatoire. Même si l'attaquant est capable de découvrir qu'une (et une seule) addition est fictive par multiplication scalaire, il ne peut pas identifier la position exacte dans le scalaire où elle a été économisée.

2-B. Implémentation

Les détails d'implémentation sont décrits dans l'Algorithme IV-1. Cet algorithme est implémenté dans une version modifiée de la machine de contrôle du bloc matériel que j'ai décrit dans la Figure II-7 (page 64). Il y a aussi quelques modifications dans la partie opérative, pour effectuer les calculs liés au nombre d'additions économisées ou à la distribution des nombres aléatoires. L'entrée de cet algorithme est le scalaire k et les points pré-calculés, petits multiples de P. La sortie est le résultat de la multiplication scalaire $Q = k \times P$. Les lignes utiles au calcul de Q sont : 1; 3-6; 10-12; 14; 16-18; 20-22. Au commencement, le point accumulateur est noté comme vide (ligne 1). Ensuite le scalaire est parcouru de gauche à droite (ligne 3). À la fin de chaque fenêtre, une nouvelle taille de fenêtre est sélectionnée aléatoirement (lignes 4-6). Pour chaque bit du scalaire, un doublage de l'accumulateur est effectué (ligne 14) et le bit correspondant rentre dans le registre à décalage de la fenêtre (lignes 10 et 11). À la fin du traitement d'une fenêtre, si le digit associé est non nul (ligne 17 ou 21), un point pré-calculé est ajouté au point accumulateur (ligne 24). Le premier digit subit un traitement différent puisqu'il n'est pas possible de manipuler le point à l'infini : l'addition n'est pas effectuée, l'accumulateur est seulement initialisé avec le point pré-calculé correspondant (ligne 18).

Algorithme IV-1 Fenêtrage aléatoire et insertion d'additions fictives à des positions aléatoires

Entrée: $k = (k_{n-1},...,k_0)_2$, 1P, 2P, 3P, ..., $(2^{Wmax}-1)P$

Sortie: Q = kP

```
 1:Q_vide=1; w=0; t=0; ta=0; s=nombre initial d'additions fictives disponibles
 2:Q = (choisi aléatoirement entre 1 et 2^Wmax-1) x P
 3:for i = n-1 .. 0
 4:   | if w==0
 5:   |    | w = choisi aléatoirement entre Wmin et Wmax
 6:   |    | Digit = 0
 7:   | if t==0
 8:   |    | t = Wmin
 9:   |    | ta = 0
10:   | Digit = Digit << 1 | k(n-1)
11:   | k = k << 1
12:   | w = w - 1
13:   | t = t - 1
14:   | Q = 2 x Q
15:   | essayer d'insérer une addition fictive; s-- if insérée
16:   | if Q_vide
17:   |    | if w==0 and Digit!=0
18:   |    |    | Q = Digit x P
19:   |    |    | Q_vide=0
20:   | else
21:   |    | if (w==0 or i==0) and Digit!=0
22:   |    |    |    | Q = Q + Digit x P
23:   |    |    |    | ta=1
24:   |    |    |    | essayer d'insérer une addition fictive; s-- if insérée
25:   | if t==0 and ta==0
26:   |    | s = s + 1
27:while not(s==0)
28:| insérer une addition fictive; s--
```

Comme expliqué précédemment, pour implémenter ma contre-mesure il faut compter le nombre d'additions économisées (noté `s` dans l'Algorithme IV-1). Cette valeur représente le nombre d'additions fictives disponibles à un instant donné. Pour calculer celui-ci, en parallèle de la détection de la fin de la fenêtre en cours, il est vérifié si le bit courant du scalaire correspond à la fin d'une fenêtre correspondant au découpage du pire cas (lignes 7 et 8). À la fin d'une fenêtre du pire cas (ligne 25), si aucune addition n'a été effectuée pendant celle-ci, une addition est comptabilisée comme économisée (lignes 9, 23 et 26). Après chaque opération légitime (lignes 15 et 24), un essai d'insertion d'addition fictive sera réalisé. Cette insertion sera effective ou non avec une probabilité dépendant du nombre d'additions fictives disponibles. Si beaucoup d'opérations fictives sont disponibles, alors la probabilité d'insertion est plus grande. Ceci permet de terminer le calcul de la multiplication scalaire avec un petit nombre d'additions fictives encore disponibles. Si aucune addition n'est disponible alors aucune opération fictive ne sera insérée car il n'est pas possible de prévoir si des opérations vont être économisées par la suite. Pour perturber l'attaquant même en début de calcul, le

nombre d'additions fictives disponibles n'est cependant pas initialement nul (ligne 1). J'ai choisi de commencer le calcul avec 2 opérations disponibles mais ceci pourrait être modifié. Il est possible qu'en fin de calcul, il reste des additions non dépensées. Les additions fictives restantes sont alors faites en fin de calcul (ligne 27 et 28). Toutes ces opérations permettent d'assurer qu'il y a toujours $2 + q/W_{min}$ additions durant la multiplication scalaire. Pour masquer la taille de la première fenêtre à un attaquant comptant les doublages, il y a toujours $q - 1$ doublages. Même si le point accumulateur n'a pas encore été réellement initialisé (ligne 18), des doublages inutiles sont réalisés. Pour que cela soit possible, le point accumulateur est initialisé en début de calcul par un point choisi au hasard dans la table de points pré-calculés. Dès que le premier digit non nul est rencontré (ligne 18), l'accumulateur est écrasé par un point réellement utile au calcul.

2-C. Loi de probabilité d'insertion d'addition fictive

L'insertion d'une opération fictive ne doit arriver que lorsqu'il y en a au moins une disponible. Pour éviter une longue séquence d'additions fictives en fin de calcul, il faudrait qu'il n'en reste que peu après la dernière opération légitime. Les additions fictives sont insérées à l'issu d'un test aléatoire après une addition ou un doublage légitime. Ce test dépend du nombre d'additions fictives disponibles. Pour que l'implémentation de ce test soit rapide et peu coûteuse, j'utilise une loi de probabilité avec une distribution linéaire. Il n'y a donc qu'un seul paramètre à cette loi : le nombre d'opérations disponibles correspondant à une probabilité d'insertion de 100%. Ce nombre correspond aussi au nombre maximal d'additions fictives disponible à chaque instant car s'il est atteint, une opération fictive sera dépensée instantanément. La Figure IV-2 décrit un exemple d'une telle loi de probabilité.

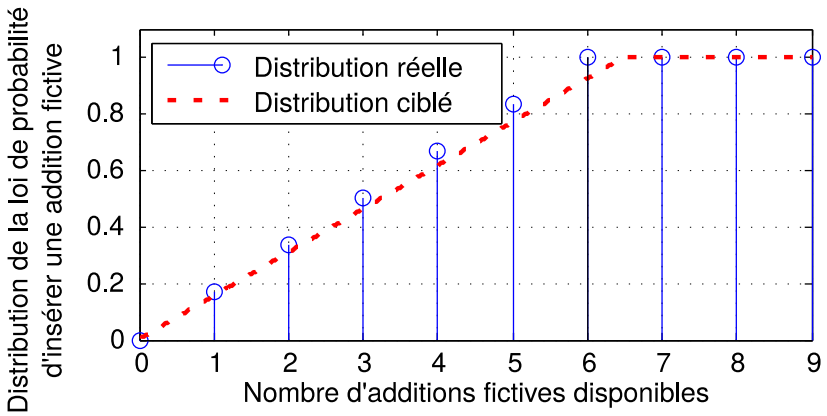


Figure IV-2 Probabilité d'insertion d'une addition fictive ($W_{min} = 2, W_{max} = 6$)

L'objectif est d'obtenir un compromis permettant d'avoir peu d'additions fictives en fin de calcul mais d'en avoir toujours un certain nombre disponibles durant le calcul. Je définis donc le nombre moyen d'additions fictives utilisables durant le calcul. J'ai choisi une valeur de 1,5 qui permet d'avoir en moyenne seulement une ou deux additions fictives en fin de calcul.

Il reste à identifier à partir de combien d'additions fictives disponibles l'insertion sera systématique pour atteindre ce nombre moyen d'opérations disponibles. Je vais expliquer ce calcul sur un exemple avec $W_{min} = 2$ et $W_{max} = 6$ sur la courbe `secp256k1`, où les scalaires sont codés sur 256 bits ($q = 256$). Nous montrerons ensuite que cette valeur reste valide même pour des courbes de taille plus grande. Il y a toujours $q - 1$ doublages de points (255) et $2 + n/Wmin\ (= C)$ additions de points. Le nombre moyen de fenêtres est D :

$$D = \frac{q}{(W_{min} + W_{max}) \div 2} = 64$$

Sachant que le nombre moyen de fenêtres contenant un digit nul est le suivant :

$$E = \frac{\sum_{i=W_{min}}^{W_{max}} \frac{1}{2^i}}{W_{max} - W_{min} + 1} = 9{,}69\%$$

On peut en conclure le nombre moyen d'additions économisées à cause de l'occurrence d'un digit nul :

$$\frac{2 \times q \times \sum_{i=W_{min}}^{W_{max}} \frac{1}{2^i}}{(W_{min} + W_{max}) \times (W_{max} - W_{min} + 1)} = 6{,}2$$

Les additions légitimes ne pouvant traiter que des digits non nuls, il y en aura en moyenne $q \times F$ avec :

$$F = \frac{D \times (1 - E)}{q} = \frac{2}{W_{min} + W_{max}} \times (1 - \frac{\sum_{i=W_{min}}^{W_{max}} \frac{1}{2^i}}{(W_{max} - W_{min} + 1)})$$

Dans notre exemple, il y a donc en moyenne 57,8 ($q \times F$) additions légitimes et 72,2 ($C - q \times F$) additions fictives par multiplication scalaire dont 2 sont des additions fictives initialement disponibles. En prenant en compte les doublages, il y a en moyenne 312,8 opérations légitimes. Pour éviter que deux additions fictives s'enchainent, j'ai décidé que le test d'insertion n'a lieu qu'après chacune de ces opérations légitimes. Donc le nombre moyen de tests effectués est de 312,8 soit $q \times (1 + F) - 1$. Sachant que 72,2 ($C - q \times F$) additions fictives seront insérées après 312,8 tests ($q \times (1 + F) - 1$), le taux de succès du test quand il

est à son niveau moyen (1,5) doit être de :

$$\text{Probabilité}(\text{op. disponibles} = 1{,}5) = \frac{2 + q \times (\frac{1}{Wmin} - F)}{q \times (1 + F) - 1} = \frac{72{,}2}{312{,}8}$$

Ayant choisi une distribution linéaire, il est simple de calculer le nombre d'opérations disponibles maximum :

$$\text{Probabilité}(\text{op. disponibles} = 1{,}5 \times \frac{q \times (1 + F) - 1}{2 + q \times (\frac{1}{Wmin} - F)}) = 100\%$$

La Figure IV-3 montre l'évolution de ce nombre maximum d'additions fictives disponibles en fonction de la taille en bit des scalaires manipulés. Ce nombre converge vers 6,7 opérations. La taille des courbes utilisées usuellement varie entre 160 bits et 521 bits. Pour toutes les tailles de scalaires usuelles le nombre maximal théorique d'additions fictives disponibles est toujours entre 6 et 7. Comme je souhaite que le test soit peu coûteux en matériel, je dois choisir un maximum qui est un nombre entier. En choisissant l'entier supérieur je risque d'avoir une grande quantité d'opérations fictives en fin de calcul car le test d'insertion ne serait que trop rarement réussi. C'est pourquoi j'ai choisi l'entier inférieur : 6. Le nombre moyen d'additions fictives disponibles sera donc légèrement inférieur à 1,5. La Figure IV-2 décrit la loi de probabilité qui a été utilisée. Si d'autres valeurs sont choisies pour les tailles minimale et maximale de fenêtre, alors la loi de probabilité est différente. Par exemple pour $W_{min} = 3$ et $W_{max} = 4$, le nombre maximal d'additions fictives disponibles est maintenant de 10 au lieu de 6. Le test d'insertion est plus difficile car il y aura moins d'additions économisées.

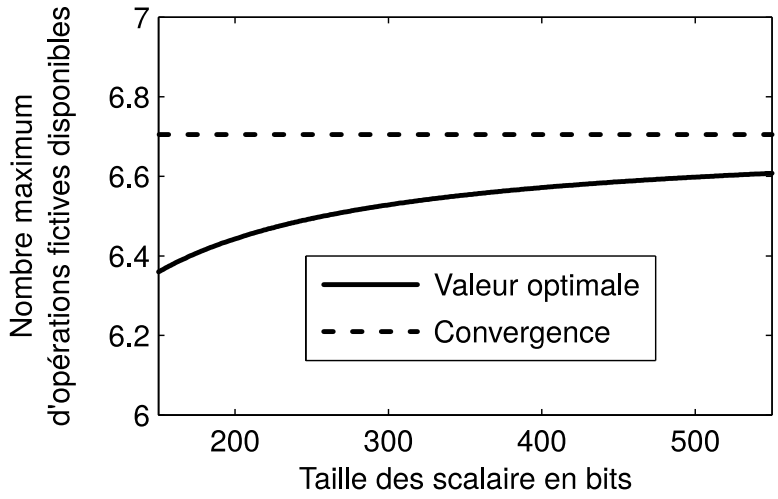


Figure IV-3 Nombre maximum d'opérations fictives disponibles en fonction de la taille du scalaire ($W_{min} = 2, W_{max} = 6$)

L'implémentation de ce test est réalisée en tirant un nombre entre 0 et 5 $(6-1)$. Si ce nombre est strictement inférieur au nombre d'additions fictives disponibles, une addition fictive est insérée et le compteur d'opérations disponibles est décrémenté. Dans le cas contraire, le calcul continue et le compteur n'est pas modifié.

La Figure IV-4 montre un exemple d'évolution du nombre d'additions fictives disponibles au cours d'une multiplication scalaire complète dans le cas décrit précédemment. Une augmentation de ce nombre représente une addition économisée par l'utilisation d'une fenêtre de grande taille ou le traitement d'un digit nul. Une diminution de ce nombre correspond à un test réussi qui a été suivi d'une addition fictive. On observe qu'ici le nombre maximum d'opérations n'a jamais été atteint. Le nombre moyen d'opérations fictives disponibles a été calculé sur 1000 multiplications scalaires. Ce nombre est de 1,22 opérations ce qui est, comme prévu, légèrement inférieur à 1,5. L'écart-type du nombre d'opérations fictives disponibles est 0,15.

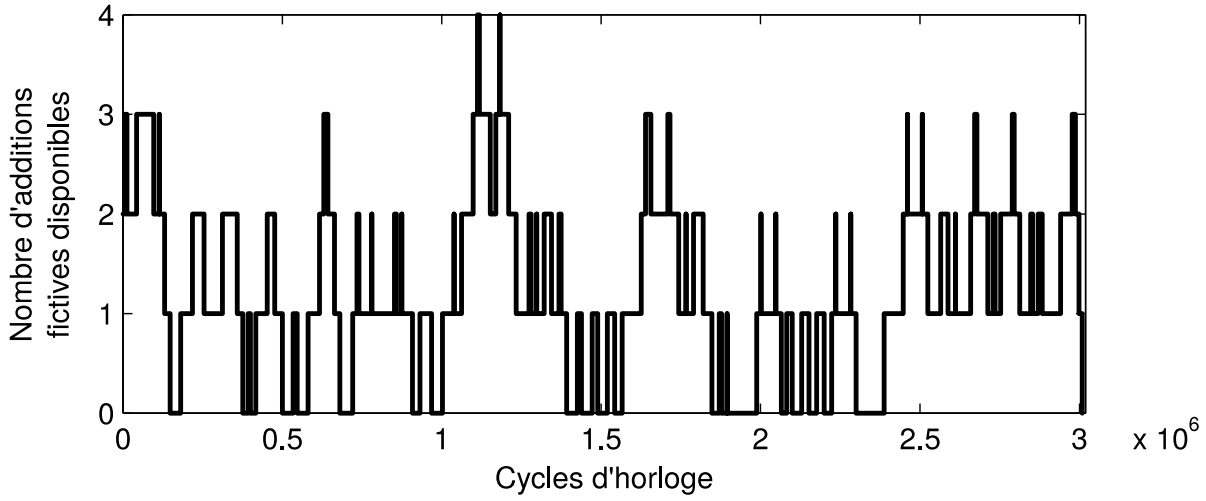


Figure IV-4 Évolution temporelle du nombre d'additions fictives disponibles pendant une multiplication scalaire ($W_{min} = 2, W_{max} = 6$)

Pour éviter qu'un attaquant puisse détecter qu'une opération basique sur les points est fictive, les coordonnées de l'accumulateur subissent toujours un mouvement rotatif (voir Figure II-3, page 59) que l'opération soit légitime ou fictive. Cela est expliqué en détail dans la section II-2-B.

2-D. Attaque par resynchronisation

Dans cette partie, je montre que la contre-mesure en l'état protège contre les attaques par canaux auxiliaires exploitant une seule trace, mais qu'en exploitant plusieurs traces il est possible de retrouver le scalaire. Dans la partie suivante, je proposerai une évolution de la contre-mesure afin d'améliorer la résistance aux attaques exploitant plusieurs traces.

Si un attaquant est capable d'obtenir plusieurs traces de multiplications scalaires utilisant le même scalaire secret, il est alors possible d'identifier des séquences de bits nuls dans le scalaire. Toutes les données expérimentales proviennent d'une implémentation sur une cible FPGA Xilinx Kintex 7 (XC7K160T) utilisant la courbe `secp256k1`. Sans injection de faute, il semble délicat de distinguer les opérations légitimes des opérations fictives. Par contre en réalisant des attaques par SPA, l'attaquant peut retrouver la chaîne d'additions et de doublages qui a été utilisée. Il peut alors identifier à quelles positions du scalaire chaque addition a eu lieu en comptant le nombre de doublages qui suivent ces additions. S'il connaît la taille de la table de points pré-calculés ou W_{max}, il peut aussi identifier les bits du scalaire étant potentiellement impactés par chaque addition. Si l'attaquant ne connaît pas cette valeur, il peut toujours essayer de mener l'attaque pour plusieurs valeurs de W_{max}.

La Figure IV-5 montre la position des additions de 1000 multiplications scalaires utilisant un scalaire fixé et différents points. Comme expliqué précédemment, l'attaquant peut construire une telle représentation en mettant en œuvre 1000 attaques par SPA. Chaque point noir représente une addition, sa ligne correspond à l'une des 1000 multiplications scalaires et sa colonne correspond à sa position dans le scalaire. Il apparaît clairement dans la Figure IV-5 (a) des zones du scalaire où il y a très peu d'additions de points (trainées blanches verticales). Dans la seconde configuration de la contre-mesure visible sur la Figure IV-5 (b), où W_{max} est plus important, cela semble plus difficile à observer car il y a plus d'additions fictives pour cacher ces zones. Ces zones avec de rares additions correspondent à des séquences de bits nuls dans le scalaire. En connaissant W_{max}, l'attaquant peut aller plus loin en affirmant que certaines zones n'ont pas pu être impactées par une addition et correspondent à des bits nuls dans le scalaire. Plus l'attaquant espionne de multiplications scalaires, plus il pourra identifier de bits nuls.

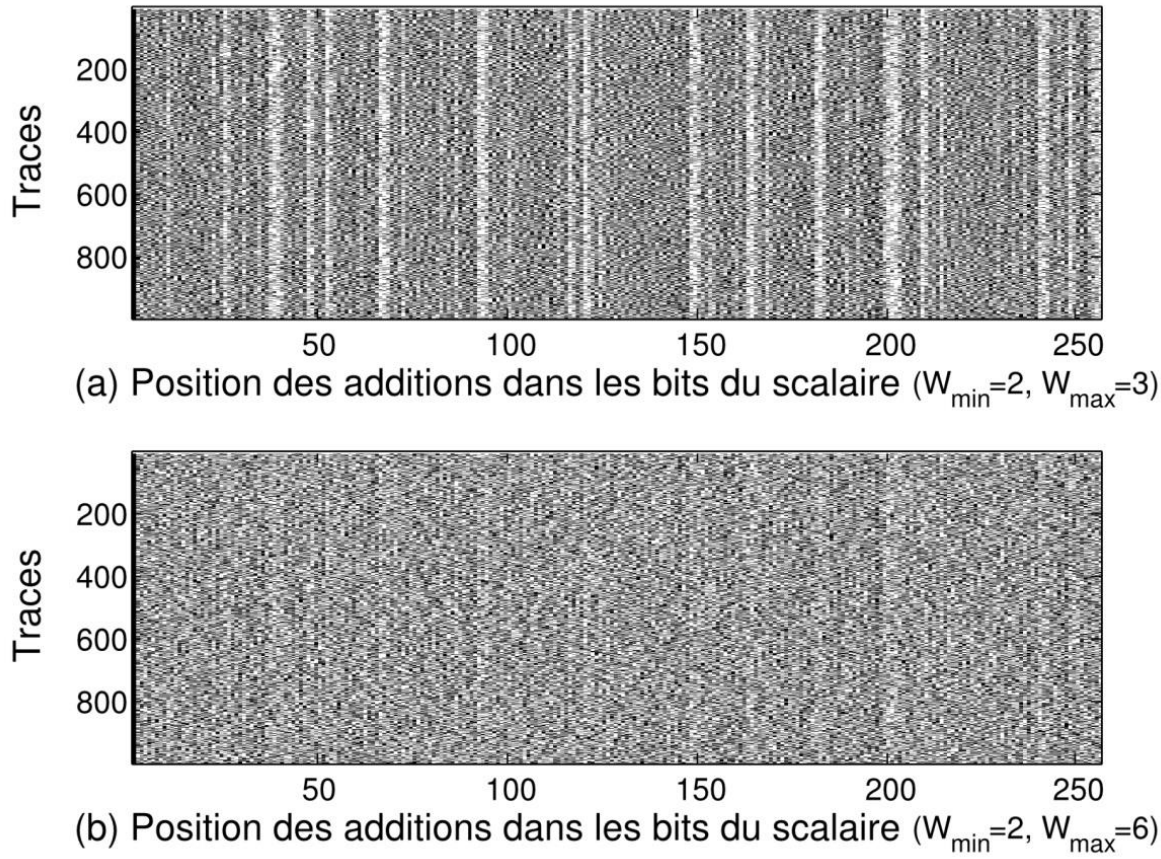


Figure IV-5 Position des additions dans un scalaire fixé pour 1000 multiplications scalaires sur une courbe de 256 bits (les points noirs représentent une addition, chaque ligne correspond à une multiplication scalaire différente et les colonnes correspondent à un bit du scalaire)

Cette attaque a été menée contre 5 configurations de la contre-mesure. Pour chacune d'elles, 12 scalaires secrets ont été attaqués en exploitant un nombre de traces variant entre 1 et 1000. La Figure IV-6 montre le nombre moyen de bits nuls (sur les 12 scalaires) identifié par rapport au nombre total de bits nuls des scalaires en exploitant un nombre variable de traces. Les barres d'erreur représentent l'écart-type par rapport aux 12 scalaires attaqués. Pour estimer le niveau de sécurité de la contre-mesure, je compare ces résultats à ceux d'une attaque par SPA contre les algorithmes de multiplication scalaire « Doublages et Additions », « Doublages et Additions Systématiques » et « m-ary ».

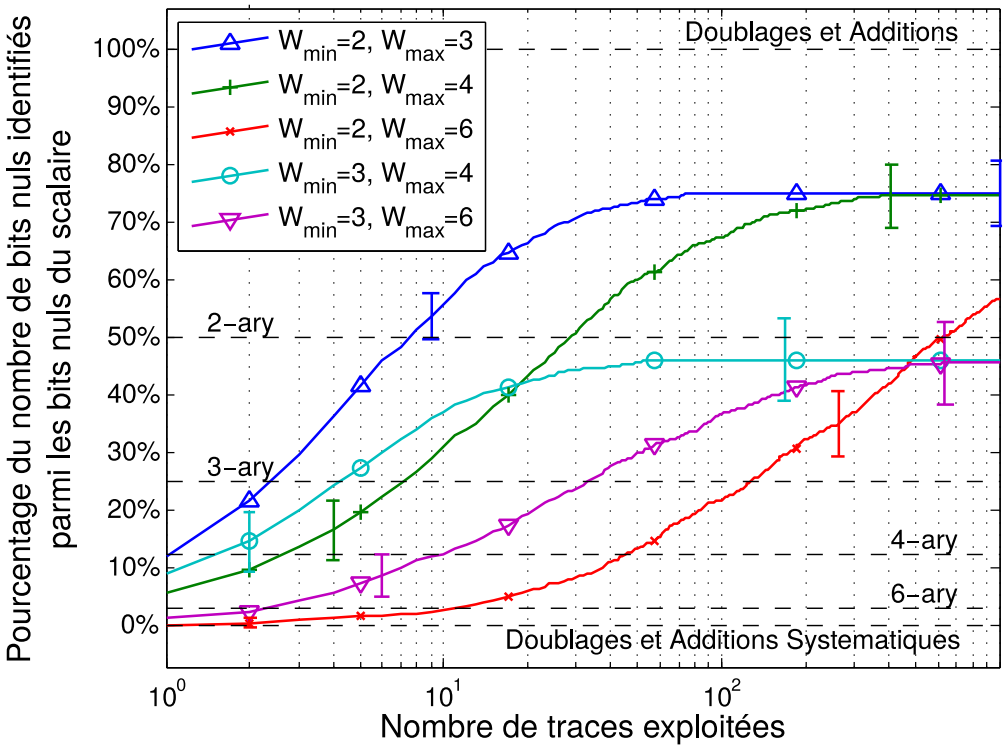


Figure IV-6 Pourcentage du nombre de bits nuls détectés dans le scalaire (moyenne sur 12 scalaires)

Le pourcentage de bits nuls découvert par l'attaquant converge quand le nombre de traces exploitées augmente. La valeur de convergence est définie par W_{min} tandis que la vitesse de convergence est définie par W_{max}. En effet, comme nous l'avons vu dans la Figure IV-5 (page 114), augmenter W_{max} ralentit l'attaque car cela augmente le nombre d'additions fictives qui perturbent l'attaquant. Une attaque par SPA contre l'algorithme « m-ary » permet d'identifier en moyenne $128 \times 2^{-(m-1)}$ bits nuls dans un scalaire de 256 bits, ou de façon plus générale $2^{-(m-1)}$ bits nuls par bit de scalaire. Il s'agit des digits nuls qui sont à des positions fixes dans le scalaire car le découpage du scalaire est lui-même fixé. Avec ma contre-mesure, l'attaquant peut identifier seulement les bits nuls qui forment une séquence d'au moins W_{min} bits. Pour être détectée, une zone de bits nuls doit former une fenêtre (au moins W_{min} bits pour qu'il n'y ait pas d'addition légitime) et être suivie d'assez de doublages pour que la prochaine addition n'impacte pas la zone. Si la fenêtre suivante fait W_{max} bits et qu'il n'y a pas d'addition fictive insérée, il y aura alors assez de doublages pour que l'attaquant détecte la zone. Bien que cela semble être une situation rare, en espionnant un grand nombre de multiplications scalaires utilisant le même scalaire l'attaquant peut trouver toutes les séquences de bits nuls détectables (plus de W_{min} bits nuls consécutifs). C'est la convergence que l'on observe sur la Figure IV-6.

Une attaque similaire peut être menée pour détecter quelques bits non nuls. Une addition fictive ne peut être insérée qu'après une opération légitime. De plus il n'est pas possible que deux additions légitimes soient consécutives. En conséquence, si trois additions consécutives sont détectées, alors la première et la dernière sont forcément fictives et l'addition centrale est légitime. Cette attaque a été menée 1000 fois en exploitant une seule trace contre 12 scalaires de 256 bits pour plusieurs configurations de ma contre-mesure. Le Tableau IV-1 résume les résultats de ces attaques. L'attaque est à la fois plus efficace pour des fenêtres de taille minimale réduite car il y a plus d'opérations légitimes et pour des fenêtres de taille maximale importante car il y a plus d'additions fictives. Pour une configuration avec un W_{min} de 2 bits, l'attaquant peut retrouver en moyenne un seul bit. Comme pour l'attaque précédente sur les bits nuls, si l'attaquant peut exploiter plusieurs traces d'utilisation du même scalaire alors il pourra détecter plus de bits. Il est possible de protéger la contre-mesure de cette attaque en empêchant cette situation d'arriver (interdire 3 additions consécutives). Mais je propose plutôt une amélioration de la contre-mesure qui permet de se protéger des deux attaques présentées.

Tableau IV-1 Moyenne et écart-type du nombre de bits non nuls identifiés dans un scalaire de 256 bits pour 5 configurations de la contre-mesure

Wmin	2	2	2	3	3
Wmax	3	4	6	4	6
Nombre moyen de bits identifiés	0,9	1,3	1,7	0,2	0,4
Écart-type	0,9	1,1	1,2	0,4	0,6

3. Désynchronisation par insertion de doublages fictifs

Comme nous l'avons vu, les deux attaques présentées contre la première version de la contre-mesure sont basées sur un comptage des doublages en vue de positionner les additions dans le scalaire. Je propose ici de désynchroniser ces attaques en insérant des doublages fictifs à des positions aléatoires.

Pour réaliser cette amélioration de la contre-mesure, la gestion de l'insertion de doublages fictifs a été ajoutée. Elle est analogue à la gestion des additions fictives, sauf que seulement quelques doublages peuvent être économisés en début de calcul. Il n'est pas possible de prévoir combien de doublages seront économisés car cela dépend du nombre de bits nuls dans les bits de poids fort du scalaire. Il faut noter que la majorité des doublages fictifs proviendront du stock initial de doublages économisés et donc disponibles.

La contre-mesure améliorée (par rapport à Algorithme IV-1) est décrite par l'Algorithme IV-2. Comme pour les additions, j'ai ajouté un compteur de doublages fictifs

disponibles (ligne 2) qui est initialisé à un nombre non nul. Les seuls doublages économisés le sont à la ligne 16 lorsque un bit du scalaire est traité et que l'accumulateur n'est pas encore initialisé légitimement. Après chaque opération légitime (doublage ligne 18 et addition ligne 27), un essai d'insertion d'une addition fictive est réalisé (lignes 19 et 29) suivi d'un essai d'insertion d'un doublage fictif (lignes 20 et 30). En fin de calcul s'il reste des opérations fictives à dépenser, elles seront dépensées en prenant garde à entremêler les doublages fictifs avec les additions fictives (lignes 33-35).

Algorithme IV-2 Fenêtrage aléatoire et insertion d'additions et de doublages fictifs à des positions aléatoires

Entrée: $k = (k_{n-1},...,k_0)_2$, 1P, 2P, 3P, ..., $(2^{Wmax}-1)P$

Sortie: $Q = kP$

```
 1:Q_vide=1; w=0; t=0; ta=0; s=nombre initial d'additions fictives disponibles
 2:                          d=nombre initial de doublages fictifs disponibles
 3:Q = (choisi aléatoirement entre 1 et 2^Wmax-1) x P
 4:for i = n-1 .. 0
 5:  | if w==0
 6 :  |   | w = choisi aléatoirement entre Wmin et Wmax
 7:  |   | Digit = 0
 8:  | if t==0
 9:  |   | t = Wmin
10:  |   | ta = 0
11:  | Digit = Digit << 1 | k(n-1)
12:  | k = k << 1
13:  | w = w - 1
14:  | t = t - 1
15:  | if Q_vide
16:  |   | d++
17:  | else
18:  |   | Q = 2 x Q
19:  |   | essayer d'insérer une addition fictive; s-- if insérée
20:  |   | essayer d'insérer un  doublage fictif ; d-- if inséré
21:  | if Q_vide
22:  |   | if w==0 and Digit!=0
23:  |   |   | Q = Digit x P
24:  |   |   | Q_vide=0
25:  | else
26:  |   | if (w==0 or i==0) and Digit!=0
27:  |   |   |   | Q = Q + Digit x P
28:  |   |   |   | ta=1
29:  |   |   |   | essayer d'insérer une addition fictive; s-- if insérée
30:  |   |   |   | essayer d'insérer un  doublage fictif ; d-- if insérée
31:  | if t==0 and ta==0
32:  |   | s = s + 1
33:while not(s==0) and not(d==0)
34:| insérer une addition fictive if not(s==0); s—-
35:| insérer un  doublage fictif  if not(d==0); d--
```

Je nomme G le nombre de doublages initialement disponibles et je considère que le premier digit rencontré sera non nul. Dans ces conditions, il y aura $G + w$ doublages fictifs pendant une multiplication scalaire avec w entre W_{min} et W_{max}. Cette valeur dépend de la taille de la première fenêtre mais varie très peu. Pour les cas moins probables et non traités où le premier digit du scalaire est nul, il y aura tout simplement plus de doublages économisés. Atteindre un nombre de doublages disponibles de $G + W_{max}$ semble plus probable. Je limite donc la valeur maximale du compteur de doublages à la puissance de 2 supérieure la plus proche de ce seuil. Dans les situations peu probables où cette limite est atteinte, un doublage fictif est réalisé instantanément.

Le calcul de la distribution de la loi de probabilité d'insertion des doublages est plus simple que pour les additions. En effet, comme tous les doublages économisés le sont en début de calcul et de manière quelque peu prévisible, il est possible d'utiliser un taux de succès constant indépendant de la quantité de doublages disponibles. Il y a autant d'essai d'insertion de doublages fictifs que d'additions fictives. Comme je l'ai montré précédemment (voir la section IV-2-C page 110), il y a $q \times (1 + F) - w$ opérations légitimes ; c'est aussi le nombre d'essais d'insertion effectués. Sur la totalité de ces tentatives d'insertions il faudra réussir en moyenne $G + w$ fois. Pour une configuration avec $W_{min} = 2$, $W_{max} = 3$ et un scalaire sur 256 bits, cela correspond à une probabilité entre 7,8% et 8% en fonction de la taille de la première fenêtre. J'ai choisi cette fois le taux de probabilité comme étant l' inverse d'une puissance de 2 entière inférieur le plus proche $(1/n^2)$. Il s'agit de $1/16 = 6{,}25\%$. Sélectionner un taux inférieur à ce qui est requis en moyenne permet d'augmenter la probabilité d'avoir des doublages fictifs répartis sur tout le calcul (jusqu'à la fin). Le choix d'un taux étant l'inverse d'une puissance de 2 a été fait pour simplifier l'implantation de celui-ci. Il suffit de tirer un nombre au hasard entre 0 et $2^4 - 1$: c'est simplement un vecteur de 4 bits aléatoire. Il n'y a pas besoin de filtrage pour assurer que ce nombre sera dans l'intervalle désiré. Si ce nombre est différent de zéro, alors le test échoue. Dans le cas contraire, un doublage fictif est inséré dans le calcul.

L'attaque présentée contre la première version de ma contre-mesure (voir section IV-2-D page 113) est impossible sur la version améliorée car les doublages fictifs perturbent l'étape de positionnement des additions dans le scalaire par comptage des doublages. J'ai tout de même mené l'attaque contre les zones de bits nuls sur la version améliorée et le résultat est illustré en Figure IV-7 (b). En comparant ce positionnement des additions avec les résultats obtenus contre la première version de la contre-mesure dans la même configuration ($W_{min} =$

2 et $W_{max} = 3$) sur la Figure IV-7 (a), on distingue nettement que les trainées blanches verticales ont quasiment disparu (surtout en début de calcul, à gauche de la figure).

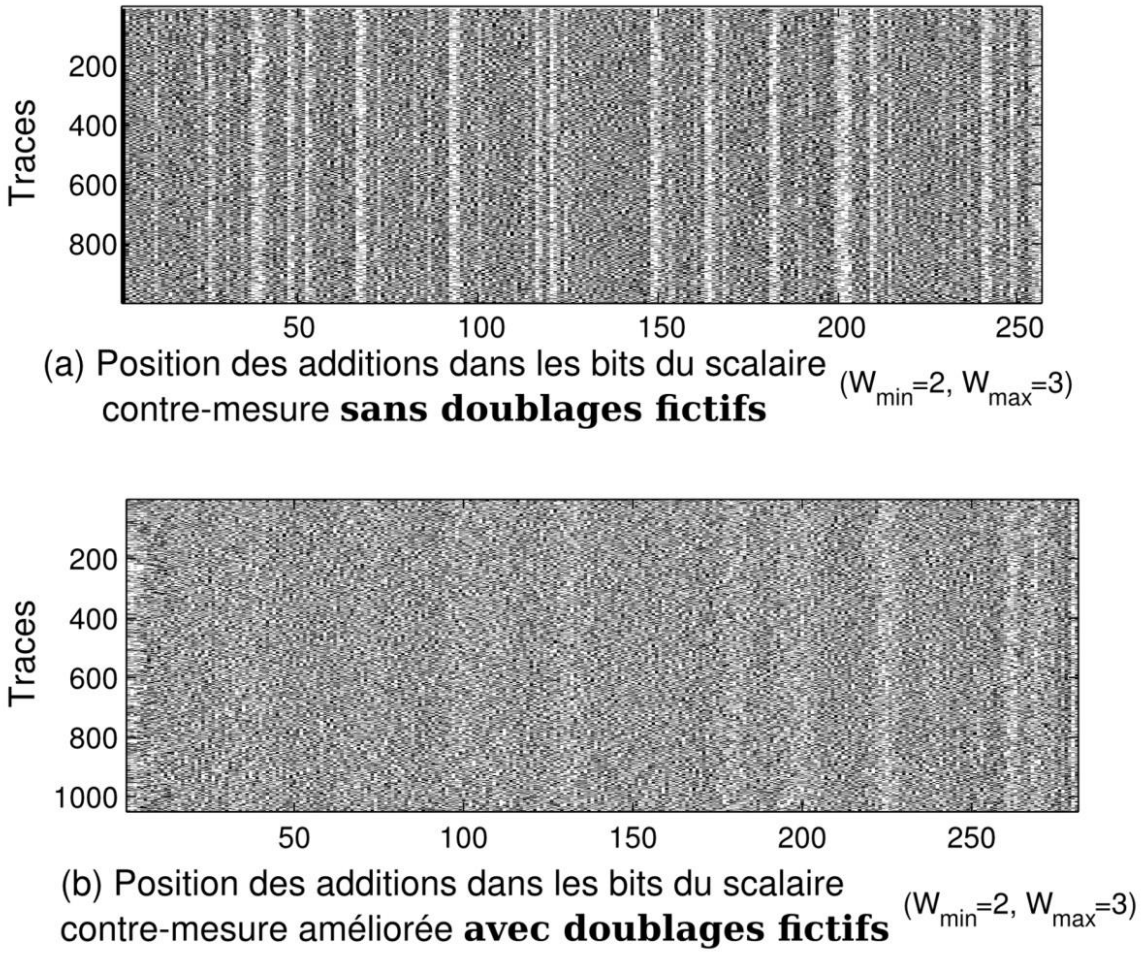


Figure IV-7 Position des additions dans un scalaire fixé pour 1000 multiplications scalaires avec la contre-mesure améliorée ($W_{min} = 2, W_{max} = 3$)

La Figure IV-8 représente un exemple de l'évolution du nombre d'additions et de doublages fictifs disponibles pendant une multiplication scalaire. Sur les graphes des doublages, la valeur initiale du compteur semble légèrement supérieure à 25 (le nombre de doublages initialement disponibles). En réalité, il a été initialisé à 25 mais le parcours des premiers bits du scalaire où quelques doublages sont économisés ne nécessite pas d'opération sur les points. Les incrémentations du compteur sont donc réalisées dans un délai de quelques cycles après le début du calcul et non visibles à l'échelle de la Figure IV-8. Chaque décrémentation du compteur correspond à l'insertion d'un doublage fictif ; on peut donc vérifier que des doublages ont été insérés tout au long du calcul.

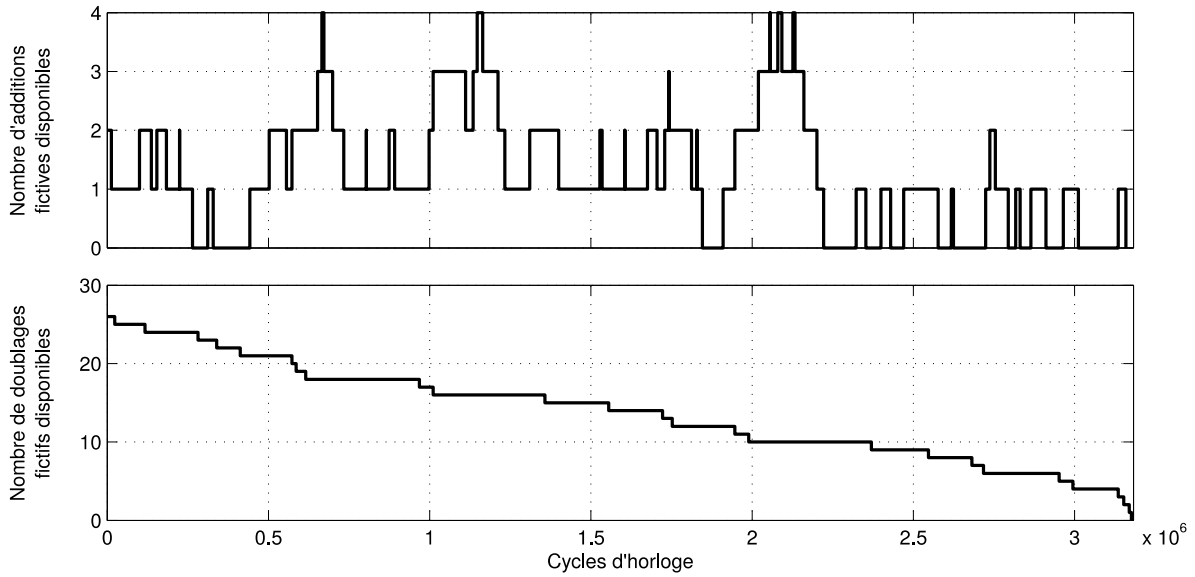


Figure IV-8 Évolution temporelle du nombre d'opérations fictives disponibles pendant une multiplication scalaire avec la contre-mesure améliorée ($W_{min} = 2, W_{max} = 6$)

4. Évaluation de la contre-mesure

Dans cette partie, je vais évaluer la contre-mesure que je propose à travers une analyse de son coût matériel, des performances qui peuvent être attendues et d'une estimation du niveau de sécurité à partir des attaques présentées précédemment. La contre-mesure a été implémentée sur le crypto-processeur décrit dans le Chapitre II. Les niveaux hiérarchiques inférieurs (à droite dans la Figure I-1, page 15) permettant de réaliser les opérations sur le corps des coordonnées et les additions et doublages de points n'ont pas été modifiés. Seul le niveau correspondant à la multiplication scalaire a subi des modifications pour pouvoir exécuter l'Algorithme IV-1 et l'Algorithme IV-2. Ce niveau correspond au bloc matériel décrit par la Figure II-7 (page 64). Le circuit protégé par ma contre-mesure hérite donc de certaines propriétés du crypto-processeur du Chapitre II : il est possible de générer un circuit généraliste supportant plusieurs courbes elliptiques sur les deux corps $GF(p)$ et $GF(2^d)$ ou de générer un circuit supportant un seul corps ou même une seule courbe. Il est aussi possible de générer un circuit supportant une courbe quartique de Jacobi en utilisant la configuration du bloc matériel d'additions et doublages de points (voir Figure II-2, page 58) qui utilise la formule d'addition unifiée (voir Chapitre III). Comme cette contre-mesure a été initialement développée pour protéger un circuit dont les additions et doublages sont distinguables dans la trace de puissance consommée, je vais me focaliser sur des courbes de Weierstrass de taille usuelle : 256 bits. Cette évaluation sera faite sur un circuit supportant une seule courbe de Weierstrass sur $GF(p)$: `secp256k1` (la courbe utilisée dans le protocole bitcoin).

4-A. Coût en surface

Je me limiterai ici à l'étude de différents circuits sur une seule cible : un FPGA Kintex 7 (XC7K160T) du fabricant Xilinx. L'évaluation du coût en surface sera faite sur la première version de la contre-mesure (sans doublages fictifs) car l'amélioration n'a qu'un coût modique et négligeable. Sans prendre en compte la table de points, et pour la configuration $W_{min} = 2$ et $W_{max} = 6$, le surcoût de la contre-mesure améliorée est de seulement 0,6% en LUTs et 0,43% en bascules (Flip-Flops). Le circuit protégé est comparé avec des circuits implémentant les algorithmes de multiplication scalaire « Doublages et Additions Systématiques » et « m-ary ». Le coût du générateur pseudo-aléatoire n'est pas pris en compte car il est toujours présent dans un circuit utilisant un crypto-processeur.

L'arithmétique sur le corps $GF(p)$ utilise 1975 LUTs et 529 Flip-Flops. Les deux registres sur 256 bits (accumulateur et registre à décalage, voir Figure II-1 page 56) correspondent à la majeure partie des Flips-Flops pour cette partie du circuit. L'unité chargée de l'addition et du doublage de points est composée d'un banc de 9 registres sur 256 bits. Ce sont ces points mémoires de la partie opérative qui occupent majoritairement les 2360 Flip-Flops du bloc réalisant les opérations basiques sur les points (voir Figure II-2 page 58). Cette unité consomme aussi 3200 LUTs. Le circuit étudié n'utilise pas d'éléments spécifiques au FPGA tels que les BRAMs ou les DSP. Grâce à cette restriction, il est possible de cibler d'autres technologies.

Le Tableau IV-2 résume les ressources occupées par la table de point pour différentes configurations. Pour l'algorithme « Doublages et Additions Systématiques » cette table ne sert qu'à stocker le point opérande P (en occupant 256 Flip-Flops par coordonnée). Dans cette configuration, un seul point est stocké ; il n'y donc aucune consommation de logique. Pour toutes les autres configurations, des LUTs sont utilisés pour mettre en place le mécanisme d'adressage de la table. Le nombre de Flip-Flops est quant à lui proportionnel au nombre de points stockés.

La seconde ligne du tableau permet de voir le coût de chaque algorithme de multiplication scalaire. Il s'agit du coût du bloc matériel lui-même sans prendre en compte ses sous-blocs tels que l'unité d'addition et de doublage de points. Ma contre-mesure n'occupe que légèrement plus d'espace que l'algorithme « m-ary » avec autant de points pré-calculés (quand $m = W_{\max}$). Dans le pire cas ($W_{\max} = 6$), la contre-mesure à un surcoût de 14% en Flip-Flops et 39% en LUTs comparativement à l'algorithme de multiplication scalaire sans fenêtrage « Doublages et Additions Systématiques ». Cette consommation supplémentaire de

ressources matérielles est due à l'implantation du test d'insertion d'additions fictives; du choix aléatoire d'un point de la table (opérande des additions fictives ou initialisation de l'accumulateur) et de la gestion du nombre d'opérations fictives disponibles. Ce surcoût peut sembler important mais le bloc de multiplication scalaire est un petit bloc compartivement aux blocs tels que l'unité d'opérations basiques sur les points ou les opérateurs sur le corps.

Tableau IV-2 Comparaison de plusieurs configurations de ma contre-mesure avec des algorithme multiplication scalaires classiques (ressources matérielles utilisées après l'étape de « mapping » sur un XC7K160T)

	Wmin = 2			**Wmin = 3**		**Doublages et Additions Systématiques**	**m-ary**	
	Wmax =3	**Wmax =4**	**Wmax =6**	**Wmax =4**	**Wmax =6**		**m=2**	**m=6**
Table de points (LUTs/Flip-Flops)	2350 5376	3333 11520	16767 48384	3236 11520	16384 48384	0 LUT 768 Flip-Flops	775 2304	16178 48384
Multiplication Scalaire (sans additions et doublages de points) (LUTs/Flip-Flops)	1224 313	1236 320	1641 327	1241 323	1640 329	1177 LUTs 287 Flip-Flops	1120 278	1523 287
Crypto-processeur (Slices)	3133	5322	13469	5270	13817	1981 Slices	2430	14292

Je propose maintenant d'observer le surcoût sur la multiplication scalaire relativement aux autres blocs. La Figure IV-9 et la dernière ligne du Tableau IV-2 décrivent les ressources occupées par différentes configurations de circuits complets incluant la multiplication scalaire, le stockage de points pré-calculés et l'unité de doublages et additions de points (comprenant l'arithmétique sur $GF(p)$). Le principal coût de ma contre-mesure est dans la table de points pré-calculés qu'elle nécessite. Il est mis en évidence sur la Figure IV-9 le surcoût de la multiplication scalaire (sans prendre en compte la table de points) entre la configuration la plus coûteuse de ma contre-mesure et l'algorithme « Doublages et Additions Systématiques ». Il s'élève à 1,3% en Flip-Flops et 7,2% en LUTs. Ceci confirme que la protection, que je propose, a son coût en ressource concentré dans la table de points pré-calculés. Ceci peut aussi s'observer Figure IV-9 en observant que ma contre-mesure pour un W_{max} donné coûte sensiblement autant que l'algorithme « m-ary » avec $m = W_{max}$.

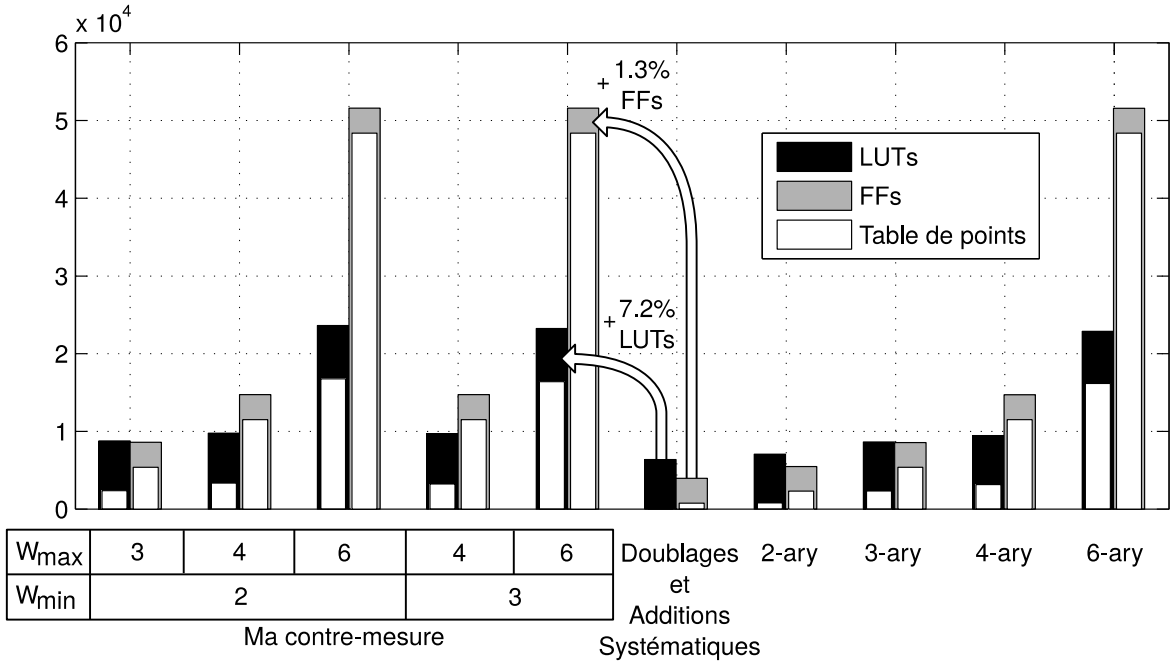


Figure IV-9 Ressources utilisées après l'étape de « mapping » sur un XC7K160T : nombre de LUTs et Flip-Flops occupées par 5 configurations de ma contre-mesure, comparées à 5 algorithmes classiques de multiplication

4-B. Performance

La première partie de cette discussion à propos des performances est consacrée à l'impact de ma contre-mesure sur la fréquence maximale de fonctionnement. Ici également, je compare ma contre-mesure à des circuits implantant les multiplications scalaires « Doublages et Additions Systématiques » et « m-ary ». Les rectangles noirs de la Figure IV-10 représentent la période d'horloge minimale pour 6 configurations après l'étape de « placement routage » sur un FPGA de la famille Kintex 7. Comme on peut l'observer ma contre-mesure a un impact négligeable sur le chemin critique qui passe de 9 ns à 9,25 ns soit une modification de fréquence maximale de 111MHz à 108MHz. Cette faible variation est due à des optima différents atteints par les algorithmes itératifs du flot de conception (placement et routage). Pour analyser de manière plus précise le chemin critique de chaque configuration, les 15 chemins les plus longs sont représentés pour chacune d'elles. En observant la décomposition des chemins critiques des trois circuits utilisant des multiplications scalaires classiques, on peut observer qu'ils passent tous à travers les unités d'opérations sur les points : le multiplieur et l'additionneur. C'est aussi le cas pour le circuit protégé par ma contre-mesure avec la configuration $W_{min} = 2$ et $W_{max} = 6$.

Pour les deux autres configurations de ma contre-mesure, le chemin critique peut se situer dans d'autres fonctions, sans pour autant réellement impacter le délai. Cela signifie que les

algorithmes d'optimisation des outils Xilinx ont identifié un optimum différent pendant le « placement routage ». Une de ces deux configurations a des chemins critiques passant par la logique de l'unité de multiplication scalaire avant d'atteindre la machine d'état de l'unité d'addition et de doublage de points. L'autre configuration est contrainte par un chemin entre la table de points et l'unité d'opérations sur les points. Cette dernière configuration correspond à la plus grande table de points ($W_{max} = 6$) qui commence donc à complexifier le routage. Ma contre-mesure affecte donc l'optimum identifié par les outils Xilinx mais n'affecte quasiment pas la fréquence de fonctionnement du circuit.

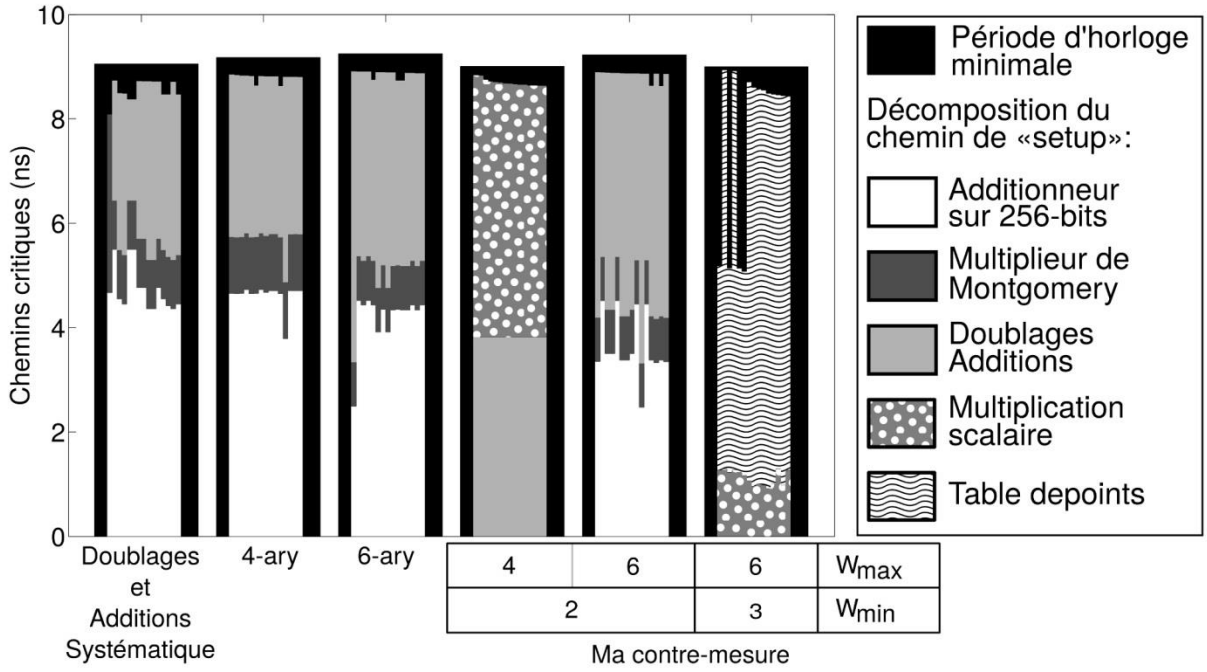


Figure IV-10 Chemin critique après l'étape de « placement routage » sur XC7K160T : période d'horloge minimale et décomposition des 15 chemins les plus critiques pour 6 configurations

Choisir une contre-mesure avec un W_{min} et W_{max} plus ou moins grand n'affecte la fréquence de fonctionnement que de manière insignifiante. Par contre cela a un impact réel sur le nombre d'additions et de doublages de points qui permettront de calculer une multiplication scalaire. Les tests permettant de contrôler les opérations fictives ainsi que le choix d'une taille de fenêtre aléatoire n'ont pas d'impact sur le nombre de cycles consommés car ils sont réalisés sur du matériel dédié et exécutés en parallèle des opérations sur les points.

La particularité de ma contre-mesure est la possibilité de réduire la latence de la multiplication scalaire en augmentant la taille de la fenêtre minimale (W_{min}). En effet, c'est ce paramètre qui fixe le nombre exact d'additions de points par multiplication scalaire. Le nombre de doublages n'est pas dépendant de la taille des fenêtres ; il y aura toujours $q - 1$

doublages dans la première version de ma contre-mesure. L'amélioration de la contre-mesure basée sur des doublages fictifs augmentera légèrement le nombre de doublages (le niveau de cette augmentation est paramétrable). Pour la première version de la contre-mesure, une configuration avec une taille de fenêtre minimale de 1 bit utilise autant d'opérations sur les points que l'algorithme « Doublages et Additions Systématiques ». En autorisant un surcoût en surface (dû à la table de points), ma contre-mesure apportera un meilleur niveau de sécurité vis-à-vis des canaux auxiliaires. Cette configuration peut-être choisie lorsqu'il n'y a pas de contrainte forte sur le niveau de performance à atteindre. Avec une taille de fenêtre minimale légèrement plus grande (2 bits), les performances atteintes seront celles de la multiplication scalaire « Doublages et Additions », ou légèrement dégradées dans le cas de la version améliorée. Sans toucher à la latence du calcul, la contre-mesure que je propose peut améliorer le niveau de sécurité en protégeant un circuit contre les attaques par SPA et par analyse du temps de calcul ainsi que ralentir une attaque par DPA. Pour les situations avec une contrainte très forte sur la latence de la multiplication scalaire, il est même possible d'améliorer les performances du circuit en utilisant une taille minimale de fenêtre de 3 bits, c'est-à-dire $q/3$ additions de points. C'est 11% de moins en moyenne que le nombre d'additions requis par l'algorithme « 2-ary » qui n'est pas protégé contre une SPA : $q \times (2^2 - 1)/(2 \times 2^2)$ additions. Il faut tout de même noter qu'atteindre un niveau de sécurité acceptable dans cette configuration requiert une grande taille de fenêtre maximale dont le coût en surface risque de ne pas être acceptable pour la plupart des applications.

4-C. Sécurité

La contre-mesure que je propose n'est pas sensible à l'attaque par timing classique car une multiplication scalaire sera toujours calculée avec le même nombre d'additions et de doublages de points. Des fuites de l'opérateur sur le corps des coordonnées restent potentiellement exploitables [Walt04] bien que la contre-mesure complexifie de telles attaques. Bien que je me sois focalisé sur une cible matérielle dans mon étude, les algorithmes de multiplication scalaire protégés que je propose peuvent aussi être implantés de manière logicielle sur un processeur généraliste. Dans cette situation, il faudra faire attention au comportement de la mémoire cache qui pourrait induire des variations de temps de calcul dues à une grande quantité de données manipulées (pour une grande taille maximale de fenêtre).

L'utilisation d'opérations fictives consistant à décorréler leurs positions dans le calcul de l'instant où elles ont été économisées permet de protéger le circuit contre le principal danger

qu'est l'attaque par « C-Safe » erreur. Des attaques par fautes plus complexes (par exemple, par injection de fautes à des instants multiples) pourraient permettre à un attaquant d'obtenir plus d'informations. De plus, le circuit est résistant contre les attaques par SPA sur une seule trace. Bien que l'attaquant puisse effectuer une détection de motifs en vue de reconstruire la chaîne d'additions et de doublages, cette information ne permet pas de retrouver le scalaire secret utilisé. L'attaque DPA est ralentie par deux propriétés de ma contre-mesure. La première est une désynchronisation ; l'attaquant peut difficilement identifier les opérations légitimes mais surtout de quels bits du scalaire elles dépendent lorsqu'il y a des doublages fictifs. S'il décide de traiter tous les cas possibles pour resynchroniser les traces, il ne pourra exploiter que quelques multiplications scalaires à cause de l'explosion du nombre de possibilités. C'est pour la même raison qu'un attaquant cherchant à identifier les opérations fictives à partir de fuite par un canal auxiliaire (par exemple, EM) devra réussir en exploitant une seule trace. Cela rend la détection d'opérations fictives délicates avec des moyens réduits. La deuxième propriété de ma contre-mesure est induite par le fenêtrage aléatoire. La chaîne d'additions et doublages légitime n'est pas constante pour un scalaire donné car le découpage de ce scalaire est aléatoire. La représentation du point accumulateur (sa coordonnée projective) est dépendante de toute la chaîne d'additions et doublages qui a permis son calcul. Même si deux découpages différents peuvent conduire à ce que l'accumulateur de chacune des deux multiplications scalaires passe par un même point de la courbe elliptique, la représentation de ce point sera certainement différente. Ces deux propriétés, la désynchronisation et la représentation projective, dépendant du découpage aléatoire du scalaire, augmenterons la complexité d'une attaque par DPA. Les techniques classiques de resynchronisation dans une attaque par canal auxiliaire permettent de s'affranchir de fronts d'horloge désynchronisés : par détection des cycles d'horloge [MoBe01], avec une analyse multi-échelles [ChPe05] ou avec une analyse de la forme des traces [RéCa08]. Mais elles ne sont pas adaptées pour attaquer une désynchronisation au niveau algorithme comme la contre-mesure que je propose.

La contre-mesure présentée utilise du fenêtrage dans une multiplication scalaire de gauche à droite. Une technique analogue existe pour un parcours du scalaire de droite à gauche sans points pré-calculés mais en ajoutant des opérateurs basiques sur les points (triplement ou quintuplement de point). Cette alternative nécessite un recodage du scalaire secret mais celui-ci peut être effectué en temps réel et du hasard peut être introduit dans cette opération pour lutter contre les attaques par canaux auxiliaires [ChTi13]. Combiner l'ajout d'opérateurs et une table de points pré-calculés est imaginable pour démultiplier le nombre de combinaisons

utilisables pour lutter contre les attaques par canaux auxiliaires mais cela risque de complexifier le recodage du scalaire.

L'extension de l'attaque contre les formules unifiées (voir section III-4 page 98) pour identifier la réutilisation de points pourrait sembler utilisable contre ma contre-mesure. Cette attaque nécessite plusieurs traces pour améliorer la matrice de similarité et effectuer une bonne classification. La désynchronisation induite par ma contre-mesure n'est pas compatible avec l'étape verticale de l'attaque (voir Figure III-7, page 91) donc la technique d'amélioration de la qualité de la matrice n'est pas utilisable. Comme je l'ai montré, des attaques spécifiquement développées contre ma contre-mesure (sur les zones de bits nuls ou non nuls) semblent plus efficaces que les attaques classiques mais l'utilisation de doublages fictifs permet de les ralentir.

Il est aussi possible de réaliser les additions et doublages avec la formule unifiée présentée dans le Chapitre III. Comme je l'ai montré, il semble difficile de différentier les additions des doublages en une trace (au moins dans les conditions avec lesquelles j'ai mené des attaques horizontales). Comme expliqué précédemment, l'attaque multi-traces du Chapitre III qui a permis d'obtenir une bonne identification des opérations sur les points n'est pas utilisable avec ma contre-mesure. Dans ces conditions, les attaques spécifiques (les plus efficaces) que j'ai présentées pour identifier les zones de bits nuls ou non nuls ne sont plus possibles car l'attaquant ne peut pas différentier les opérations de doublage des opérations d'addition de points. L'utilisation d'une formule unifiée permet donc de limiter les fenêtres à des tailles plus petites pour un même niveau de protection car les attaques utilisables sont moins efficaces. Cette économie sur la taille de fenêtre maximale permet de protéger un circuit avec un coût moins important en surface. Si le protocole utilisé assure qu'il n'est pas possible d'espionner plusieurs multiplications scalaires utilisant le même scalaire (exemple : ECDSA), alors ma contre-mesure peut être configurée de manière relâchée car les attaques possibles sont restreintes.

Si les performances du crypto processeur ne sont pas essentielles, la taille de fenêtre minimale doit être choisie petite pour maximiser le nombre d'additions fictives et augmenter la perturbation des attaques par canaux auxiliaires. Si les attaques exploitant plusieurs traces sont possibles, la taille de fenêtre maximale doit être choisie en fonction du nombre de traces que l'attaquant peut obtenir (voir Figure IV-6, page 115). Dans la contre-mesure améliorée (doublages fictifs) ou si une formule d'addition unifiée est utilisée, la taille de la fenêtre maximale peut être diminuée pour réduire le coût en surface.

5. Conclusion

Il existe déjà des contre-mesures basées sur du fenêtrage aléatoire [AhHa03; ItYa02]. J'ai identifié une fuite d'information qui peut potentiellement être exploitée contre ce type de contre-mesure. Je propose de réduire cette fuite avec des opérations fictives. Ma contre-mesure, combinant les techniques de fenêtrage aléatoire avec l'utilisation d'opérations fictives, est particulièrement adaptée à une situation où le concepteur doit protéger un circuit dans des conditions contraintes d'occupation en surface, de performance ou de sécurité vis-à-vis des attaques par canaux auxiliaires. Les choix du concepteur sont facilités car les deux paramètres que sont les tailles de fenêtres minimale et maximale sont liés de manière indépendante et prévisible à la surface occupée et aux performances du circuit. Deux paramètres supplémentaires permettent d'affiner le niveau de sécurité : les nombres initiaux de doublages et d'additions disponibles.

La protection opère au niveau de la multiplication scalaire, elle est donc compatible avec des contre-mesures opérant à d'autres niveaux hiérarchiques de la cryptographie sur les courbes elliptiques. Il est aussi possible d'augmenter encore plus la dispersion des chaînes d'additions et doublages utilisables pour une multiplication scalaire en s'autorisant à réaliser des soustractions entre deux points. La protection que je propose s'adapte à de nombreuses situations : si le concepteur a déjà protégé son circuit contre la détection des additions et doublages (par exemple avec une formule unifiée) ; si le protocole utilisé n'est pas compatible avec des attaques exploitant plusieurs traces ou au contraire si un attaquant peut espionner plusieurs multiplications scalaires. Ma contre-mesure est résistante aux attaques par analyse du temps de calcul ou par SPA. Les attaques par injection de fautes telles que celle basée sur les « C-Safe » erreurs ne permettent pas de retrouver des informations sur le scalaire utilisé. L'utilisation d'aléas à deux niveaux (découpage du scalaire et position des opérations fictives) devrait permettre de ralentir grandement une attaque par DPA. Dans les situations où l'attaquant peut obtenir plusieurs traces, ma contre-mesure peut être adaptée pour ralentir des attaques verticales. La contre-mesure a été validée sur une cible matérielle FPGA mais elle peut aussi s'appliquer à des implantations logicielles.

Conclusion et Perspectives

Le champ des possibles pour attaquer un système cryptographique est au moins aussi large que celui des protections qui peuvent être mises en place. Je me suis attaché à en explorer une partie pour renforcer la sécurité des implantations basées sur les courbes elliptiques vis-à-vis de certaines attaques par canaux auxiliaires (ou canaux cachés).

Mes contributions sont de plusieurs ordres. J'ai développé plusieurs crypto-processeurs matériels, chacun capable de réaliser des multiplications scalaires de plusieurs manières et sur différentes courbes. Ils supportent aussi bien des courbes classiques comme celles de Weierstrass que des courbes permettant l'usage d'une formule d'addition unifiée telles que les courbes quartiques de Jacobi. Les multiplications scalaires supportées vont des algorithmes classiques parcourant le scalaire de gauche à droite à l'utilisation de méthodes de fenêtrage aléatoire pour protéger un secret. Ces circuits étant tous des variations d'un même composant, ils me permettent d'isoler facilement une modification pour évaluer son impact intrinsèque sur le coût, les performances ou la sécurité. Je dispose ainsi de nombreux circuits implantant les techniques usuelles, que je peux comparer à une nouvelle méthode dans des conditions stables : même technologie, même architecture pour l'arithmétique de base.

Protéger un circuit en utilisant une formule unifiée est possible depuis longtemps et de multiples manières. J'ai montré que la solution basée sur les courbes quartique de Jacobi est intéressante car elle a un impact favorable sur les performances pour un surcoût en ressources acceptable (25%). Il existe différentes attaques contre cette unification mais la plupart nécessitent de connaître les détails architecturaux ou l'ordonnancement interne du circuit attaqué. J'ai proposé une nouvelle attaque, avec une mise en œuvre réelle. Cette attaque exploite quelques dizaines de traces de puissance consommée mesurées à faible bande passante lors de multiplications scalaires de gauche à droite. J'ai montré qu'il est possible dans ces conditions de monter une attaque sans connaître de détail sur l'implantation du circuit ciblé et sans hypothèse sur l'architecture du multiplieur de grands entiers. La première phase de mon attaque est purement horizontale et permet de s'affranchir de la connaissance du point opérande ou résultant (contrairement à une DPA). Elle permet aussi de s'affranchir de quelques contre-mesures telles que la représentation aléatoire du point opérande avant de passer à la seconde phase, verticale, exploitant plusieurs traces. J'ai identifié une méthode de classification, la méthode de Ward, qui permet de distinguer les additions des doublages. J'ai

aussi montré qu'une telle attaque est utilisable pour détecter la réutilisation d'opérande dans une multiplication scalaire par fenêtrage. J'ai finalement identifié les bonnes pratiques à mettre en œuvre pour utiliser une formule unifiée de manière plus sûre.

L'utilisation de méthodes de fenêtrages aléatoires dans la multiplication scalaire a déjà été proposée pour implanter des contre-mesures aux attaques par canaux auxiliaires. J'ai proposé une nouvelle contre-mesure de ce type mais en la combinant avec des opérations fictives permettant de désynchroniser les attaques exploitant plusieurs traces et d'obtenir une multiplication scalaire en temps constant résistant à l'attaque par SPA. L'utilisation d'opérations fictives peut ouvrir la porte à de nouvelles attaques, notamment par injection de fautes. J'ai montré que ma contre-mesure reste cependant utilisable de manière sûre en décorrélant par un délai aléatoire l'instant où l'opération fictive devient disponible du moment où elle est exécutée. Ce délai aléatoire permet de limiter le nombre d'opérations fictives détectables mais aussi de ralentir les attaques exploitant plusieurs traces de multiplications scalaires. Cette technique, couplée à un découpage du scalaire en fenêtres de tailles aléatoires, permet d'obtenir une contre-mesure hautement paramétrable. En modifiant les bornes des tailles de fenêtres utilisées, un concepteur de circuit peut maîtriser indépendamment le coût en surface de ma contre-mesure et les performances de la multiplication scalaire. Il est même possible d'utiliser la protection que je propose pour réduire la latence de la multiplication scalaire bien que cela ait un coût non négligeable en surface si l'on souhaite atteindre un niveau de sécurité satisfaisant. De plus, le coût de ma contre-mesure peut être réduit en la combinant avec l'utilisation d'une formule unifiée.

Mes travaux ouvrent de nombreuses **perspectives**. Le fenêtrage aléatoire permet d'utiliser une chaîne d'additions et de doublages légitimes aléatoire que je dissimule parmi des opérations fictives. Il existe plusieurs solutions qui pourraient permettre de relaxer encore plus la chaîne légitime. La première pourrait exploiter la soustraction entre deux points qui peut être réalisée sans coût matériel. Par contre, comme le montre l'algorithme wNAF, identifier un recodage du scalaire efficace en temps réel pendant la multiplication scalaire semble être délicat. Il est aussi possible d'exploiter plus d'opérateurs basiques tels que le triplement ou quintuplement d'un point pour relaxer la chaîne des opérations basiques. Ceci est facilement exploitable pour un algorithme de multiplication scalaire de droite à gauche. Il serait intéressant d'étudier la faisabilité et le coût d'une multiplication scalaire exploitant à la fois des points pré-calculés et des opérateurs supplémentaires. Toutes ces relaxations de la chaîne d'opération basique sur les points peuvent être étudiées pour remplacer ou venir en complément du fenêtrage aléatoire simple que je propose. Si le recodage du scalaire et le coût

matériel d'opérateurs supplémentaires ne sont pas trop grands, la sécurité apportée par ma contre-mesure vis-à-vis des attaques par canaux auxiliaires pourrait encore être améliorée.

Les conclusions de mon attaque sur les courbes quartiques de Jacobi semblent extrapolables aux autres méthodes d'unification des opérations sur les points. Il me semble indispensable de valider cette hypothèse sur des courbes tordues de Edwards car elles permettent d'utiliser une formule unifiée et commencent à être intégrées dans certains protocoles. L'architecture ciblée pour valider mon attaque n'utilise pas de DSP. Il serait intéressant d'étudier l'influence de l'utilisation de tels composants matériels sur le niveau de sécurité. Ceci pourrait être réalisé en attaquant le circuit présenté dans l'Annexe 3 qui exploite un multiplieur par digits au lieu d'un multiplieur série-parallèle. C'est ce bloc qui semble être à la source de la fuite d'information. Il serait donc aussi intéressant d'étudier des multiplieurs classiques n'exploitant pas le domaine de Montgomery.

Le nombre de traces qu'il est nécessaire d'espionner pour mon attaque contre la formule unifiée peut sûrement être réduit. Comme il est déjà petit, il sera peut-être possible de réaliser une attaque purement horizontale, n'exploitant qu'une seule trace, avec quelques améliorations. Réussir une attaque en exploitant une seule trace permet d'attaquer des cibles même si le scalaire secret est utilisé une seule fois (ex : ECDSA) ou si sa représentation a été modifiée (par exemple en lui ajoutant un certain nombre de fois l'ordre du point générateur). Plusieurs pistes me semblent exploitables pour essayer d'atteindre l'objectif d'une attaque purement horizontale. La première piste est d'améliorer le rapport signal à bruit et la bande passante de la mesure de consommation existante. La seconde opportunité serait d'exploiter la localité que peut apporter une attaque par mesure du champ électromagnétique émis. La troisième piste serait d'explorer plus de techniques alternatives à la méthode de Ward permettant elles aussi d'identifier de manière non-supervisée un groupe cohérent par classification. La validation de l'attaque sur un modèle plus récent de FPGA ou sur une cible ASIC est aussi un axe de développement qui me paraît important.

Les deux contre-mesures que j'ai étudiées sont compatibles car elles interviennent à des niveaux hiérarchiques différents : multiplication scalaire et opération basique sur les points. Je pense qu'une telle combinaison est intéressante car l'étude que j'ai menée laisse à penser que cette contre-mesure combinée n'aurait que peu ou pas d'impact sur les performances du système. La formule unifiée s'ajouterait aux doublages fictifs pour limiter les possibilités d'une attaque par comptage des doublages. L'amélioration de l'attaque horizontale contre la formule unifiée en exploitant plusieurs traces semblerait compromise par le fenêtrage aléatoire et la désynchronisation due aux opérations fictives.

Bibliographie

Bibliographie externe

[AhHa03] MahnKi Ahn, JaeCheol Ha, HoonJae Lee, et SangJae Moon.
«A random M-ary method based countermeasure against side channel attacks.» dans *International Conference on Computational Science and Its Applications*, p. 338-347. Springer. **2003**.

[AlRa14] Hamad Alrimeih et Daler Rakhmatov.
«Fast and flexible hardware support for ECC over multiple standard prime fields.» dans *VLSI Systems, IEEE Transactions on*, vol. 22, p. 2661-2674. IEEE. **2014**.

[Amer98] X9-Financial Services American National Standards Institute.
«ANSI X9.62, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).» . **1998**.

[AmFe09] Frederic Amiel, Benoit Feix, Michael Tunstall, Claire Whelan, et William P Marnane.
«Distinguishing multiplications from squaring operations.» dans *Selected Areas in Cryptography*, p. 346-360. Springer. **2009**.

[ANSS14] ANSSI.
«Mécanismes cryptographiques: Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques.» <*http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf*>. **2014**.

[AnBr03] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, et Scott Vanstone.
«Validation of elliptic curve public keys.» dans *International Workshop on Public Key Cryptography*, p. 211-223. Springer. **2003**.

[BaDe16] Thomas Baignères, Cécile Delerablée, Matthieu Finiasz, Louis Goubin, Tancrède Lepoint, et Matthieu Rivain.
«Million Dollar Curve.» <*http://cryptoexperts.github.io/million-dollar-curve/*>. **2016**.

[BaMe14] Jean-Claude Bajard et Nabil Merkiche.
«Double level Montgomery Cox-Rower architecture, new bounds.» dans *Smart Card Research and Advanced Applications*, p. 139-153. Springer. **2014**.

[Bata88] Vladimir Batagelj.
«Generalized Ward and related clustering problems.» dans *Classification and related methods of data analysis*, p. 67-74. North-Holland Amsterdam. **1988**.

[BaJa15] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, Jean-René Reinhard, et Justine Wild.
«Horizontal collision correlation attack on elliptic curves.» dans *Cryptography and Communications*, vol. 7, p. 91-119. Springer. **2015**.

[Bern06] Daniel J Bernstein.
«Curve25519: new Diffie-Hellman speed records.» dans *Public Key Cryptography-PKC 2006*, p. 207-228. Springer. **2006**.

[BeBi08] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, et Christiane Peters.
«Twisted edwards curves.» dans *International Conference on Cryptology in Africa*, p. 389-405. Springer. **2008**.

[BeDu12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, et Bo-Yin Yang.
«High-speed high-security signatures.» dans *Journal of Cryptographic Engineering*, vol. 2, p. 77-89. Springer. **2012**.

[BiMe00] Ingrid Biehl, Bernd Meyer, et Volker Müller.
«Differential fault attacks on elliptic curve cryptosystems.» dans *Annual International Cryptology Conference*, p. 131-146. Springer. **2000**.

[BiTi15] Karim Bigou et Arnaud Tisserand.
«Single base modular multiplication for efficient hardware RNS implementations of ECC.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 123-140. Springer. **2015**.

[Brai05] E C Brainpool.
«ECC Brainpool standard curves and curve generation.» <*http://www.ecc-brainpool.org/download/Domain-parameters.pdf*>. **2005**.

[BrJo02] Eric Brier et Marc Joye.
«Weierstraß elliptic curves and side-channel attacks.» dans *Public Key Cryptography*, p. 335-345. Springer. **2002**.

[Cert00] S E Certicom.
«SEC 2: Recommended elliptic curve domain parameters.» dans *Proceeding of Standards for Efficient Cryptography, Version*, vol. 1,. **2000**.

[ChTi13] Thomas Chabrier et Arnaud Tisserand.
«On-the-fly multi-base recoding for ECC scalar multiplication without pre-computations.» dans *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, p. 219-228. IEEE. **2013**.

[ChPe05] Xavier Charvet et Herve Pelletier.
«Improving the DPA attack using Wavelet transform.» dans *NIST Physical Security Testing Workshop*, vol. 46,. **2005**.

[ChCi04] Benoît Chevallier-Mames, Mathieu Ciet, et Marc Joye.
«Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity.» dans *Computers, IEEE Transactions on*, vol. 53, p. 760-768. IEEE. **2004**.

[ClFe12] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylene Roussellet, et Vincent Verneuil.
«ROSETTA for single trace analysis.» dans *Progress in Cryptology-INDOCRYPT 2012*, p. 140-155. Springer. **2012**.

[CoFr05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, et Frederik Vercauteren.
«Handbook of elliptic and hyperelliptic curve cryptography.» CRC press. **2005**.

[Corn16] Marie-Angela Cornelie.
«Efficient Implementation of a Generic Coprocessor for Elliptic Curve Cryptography on Reconfigurable Hardware.» . thèse de doctorat, Université Grenble-Alpes. **2016**.

[Coro99] Jean-Sébastien Coron.
«Resistance against differential power analysis for elliptic curve cryptosystems.» dans *Cryptographic Hardware and Embedded Systems*, p. 292-302. Springer. **1999**.

[DePr08] Christophe De Canniere et Bart Preneel.
«Trivium.» dans *New stream cipher designs*, p. 244-266. Springer. **2008**.

[DiGe05] Mario Di Raimondo, Rosario Gennaro, et Hugo Krawczyk.
«Secure off-the-record messaging.» dans *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, p. 81-89. ACM. **2005**.

[DiHe76] Whitfield Diffie et Martin Hellman.
«New directions in cryptography.» dans *IEEE transactions on Information Theory*, vol. 22, p. 644-654. IEEE. **1976**.

[DoIm06] Christophe Doche et Laurent Imbert.
«Extended double-base number system with applications to elliptic curve cryptography.» dans *International Conference on Cryptology in India*, p. 335-348. Springer. **2006**.

[Dond15] Ariano-Tim Donda.
«Efficient Implementation of a Generic Coprocessor for Elliptic Curve Cryptography on Reconfigurable Hardware.» <*http://www.torsten-schuetze.de/reports/MasterThesis_Donda.pdf* >. **2015**.

[DuPa16] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, et Sylvain Guilley.
«Dismantling Real-World ECC with Horizontal and Vertical Template Attacks.» dans *Constructive Side-Channel Analysis and Secure Design*, p. 88-108. Springer. **2016**.

[FIPS00] PUB FIPS.
«186-2. digital signature standard (DSS).» dans *National Institute of Standards and Technology (NIST)*. **2000**.

[FoMu04] Pierre-Alain Fouque, Frédéric Muller, Guillaume Poupard, et Frédéric Valette.
«Defeating countermeasures based on randomized BSD representations.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 312-327. Springer. **2004**.

[GaGa16] Steven D Galbraith et Pierrick Gaudry.
«Recent progress on the elliptic curve discrete logarithm problem.» dans *Designs, Codes and Cryptography*, vol. 78, p. 51-72. Springer. **2016**.

[GaGe14] Steven D Galbraith et Shishay W Gebregiyorgis.
«Summation polynomial algorithms for elliptic curves in characteristic two.» dans *International Conference in Cryptology in India*, p. 409-427. Springer. **2014**.

[Guer02] Shay Gueron.
«Enhanced montgomery multiplication.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 46-56. Springer. **2002**.

[GüPa08] Tim Güneysu et Christof Paar.
«Ultra high performance ECC over NIST primes on commercial FPGAs.» dans *Cryptographic Hardware and Embedded Systems–CHES 2008*, p. 62-78. Springer. **2008**.

[GuPa04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, et Sheueling Chang Shantz.
«Comparing elliptic curve cryptography and RSA on 8-bit CPUs.» dans *Cryptographic hardware and embedded systems-CHES 2004*, p. 119-132. Springer. **2004**.

[Hamb15] Mike Hamburg.
«Decaf: Eliminating cofactors through point compression.» dans *Annual Cryptology Conference*, p. 705-723. Springer. **2015**.

[HaMo02] Jae Cheol Ha et Sang Jae Moon.
«Randomized signed-scalar multiplication of ECC to resist power attacks.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 551-563. Springer. **2002**.

[HeIb14] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, et Georg Sigl.
«Clustering algorithms for non-profiled single-execution attacks on exponentiations.» dans *Smart Card Research and Advanced Applications*, p. 79-93. Springer. **2014**.

[HiWo08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, et Ed Dawson.
«Twisted Edwards curves revisited.» dans *Advances in Cryptology-ASIACRYPT 2008*, p. 326-343. Springer. **2008**.

[HiWo09] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, et Ed Dawson.
«Jacobi quartic curves revisited.» dans *Information Security and Privacy*, p. 452-468. Springer. **2009**.

[HoMi08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, et Adi Shamir.
«Collision-based power analysis of modular exponentiation using chosen-message pairs.» dans *Cryptographic Hardware and Embedded Systems–CHES 2008*, p. 15-29. Springer. **2008**.

[ItYa02] Kouichi Itoh, Jun Yajima, Masahiko Takenaka, et Naoya Torii.
«DPA countermeasures by improving the window method.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 303-317. Springer. **2002**.

[Jivs12] A Jivsov.
«Elliptic Curve Cryptography (ECC) in OpenPGP.» <*http://www.rfc-editor.org/rfc/rfc6637.txt*>. RFC Editor. **2012**.

[John67] Stephen C Johnson.
«Hierarchical clustering schemes.» dans *Psychometrika*, vol. 32, p. 241-254. Springer. **1967**.

[Joye07] Marc Joye.
«Highly regular right-to-left algorithms for scalar multiplication.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 135-147. Springer. **2007**.

[JoQu01] Marc Joye et Jean-Jacques Quisquater.
«Hessian elliptic curves and side-channel attacks.» dans *Cryptographic Hardware and Embedded Systems–CHES 2001*, p. 402-410. Springer. **2001**.

[JoYe02] Marc Joye et Sung-Ming Yen.
«The Montgomery powering ladder.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 291-302. Springer. **2002**.

[KoAc96] C Kaya Koc, Tolga Acar, et Burton S Kaliski.
«Analyzing and comparing Montgomery multiplication algorithms.» dans *IEEE micro*, vol. 16, p. 26-33. IEEE. **1996**.

[Koch96] Paul C Kocher.
«Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems.» dans *Advances in Cryptology–CRYPTO 96*, p. 104-113. Springer. **1996**.

[KoJa99] Paul Kocher, Joshua Jaffe, et Benjamin Jun.
«Differential power analysis.» dans *Annual International Cryptology Conference*, p. 388-397. Springer. **1999**.

[KoDe16] Philipp Koppermann, Fabrizio De Santis, Johann Heyszl, et Georg Sigl.
«X25519 Hardware Implementation for Low-Latency Applications.» <*Article sous presse*> dans *19th Euromicro Conference on Digital System Design (DSD)*. IEEE. **2016**.

[LeHs12] Jen-Wei Lee, Ju-Hung Hsiao, Hsie-Chia Chang, et Chen-Yi Lee.
«An efficient DPA countermeasure with randomized montgomery operations for DF-ECC processor.» dans *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, p. 287-291. IEEE. **2012**.

[LiSm01] P.-Y. Liardet et Nigel P Smart.
«Preventing SPA/DPA in ECC systems using the Jacobi form.» dans *Cryptographic Hardware and Embedded Systems–CHES 2001*, p. 391-401. Springer. **2001**.

[LoMi08] Patrick Longa et Ali Miri.
«New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (extended version).» dans *IACR Cryptology ePrint Archive*, vol. 2008, p. 52. **2008**.

[LóDa99] Julio López et Ricardo Dahab.
«Fast multiplication on elliptic curves over GF (2m) without precomputation.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 316-327. Springer. **1999**.

[Maot67] James MacQueen et others.
«Some methods for classification and analysis of multivariate observations.» dans *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, p. 281-297. Oakland, CA, USA. **1967**.

[MaLi13] Yuan Ma, Zongbin Liu, Wuqiong Pan, et Jiwu Jing.
«A High-Speed Elliptic Curve Cryptographic Processor for Generic Curves over GF(p).» dans *Selected Areas in Cryptography–SAC 2013*, p. 421-437. Springer. **2013**.

[Mill86] Victor Miller.
«Use of elliptic curves in cryptography.» dans *Advances in Cryptology–CRYPTO 85 Proceedings*, p. 417-426. Springer. **1986**.

[MoBo13] Bodo Moeller, Nelson Bolyard, Vipul Gupta, Simon Blake-Wilson, et Chris Hawk.
«Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).» <*http://www.rfc-editor.org/rfc/rfc4492.txt*>. RFC Editor. **2013**.

[Möll02] Bodo Möller.
«Parallelizable elliptic curve point multiplication method with resistance against side-channel attacks.» dans *International Conference on Information Security*, p. 402-413. Springer. **2002**.

[Mont85] Peter L Montgomery.
«Modular multiplication without trial division.» dans *Mathematics of computation*, vol. 44, p. 519-521. **1985**.

[MoOl90] François Morain et Jorge Olivos.
«Speeding up the computations on an elliptic curve using addition-subtraction chains.» dans *Informatique théorique et Applications*, vol. 24, p. 531-543. **1990**.

[MoBe01] D Moyart et R Bevan.
«A method for resynchronizing a random clock on smart cards.» dans *Smart Card Security Conference*. **2001**.

[MuLe14] Fionn Murtagh et Pierre Legendre.
«Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion?» dans *Journal of Classification*, vol. 31, p. 274-295. Springer. **2014**.

[Naka08] Satoshi Nakamoto.
«Bitcoin: A peer-to-peer electronic cash system.» . **2008**.

[OkSc04] Katsuyuki Okeya, Katja Schmidt-Samoa, Christian Spahn, et Tsuyoshi Takagi.
«Signed binary representations revisited.» dans *Annual International Cryptology Conference*, p. 123-139. Springer. **2004**.

[OsAi01] Elisabeth Oswald et Manfred Aigner.
«Randomized addition-subtraction chains as a countermeasure against power attacks.» dans *International Workshop on Cryptographic Hardware and Embedded Systems*, p. 39-50. Springer. **2001**.

[PeIm14] Guilherme Perin, Laurent Imbert, Lionel Torres, et Philippe Maurine.
«Attacking Randomized Exponentiations Using Unsupervised Learning.» dans *Constructive Side-Channel Analysis and Secure Design*, p. 144-160. Springer. **2014**.

[Plût12] Jérôme Plût.
«On various families of twisted jacobi quartics.» dans *Selected Areas in Cryptography*, p. 373-383. Springer. **2012**.

[Poll78] John M Pollard.
«Monte Carlo methods for index computation mod p.» dans *Mathematics of computation*, vol. 32, p. 918-924. **1978**.

[PrMu14] Adrien Prost-Boucle, Olivier Muller, et Frédéric Rousseau.
«Fast and standalone Design Space Exploration for High-Level Synthesis under resource constraints.» dans *Journal of Systems Architecture*, vol. 60, p. 79-93. Elsevier. **2014**.

[RéCa08] Denis Réal, Cécile Canovas, Jessy Clédière, Mhamed Drissi, et Frédéric Valette.
«Defeating classical hardware countermeasures: a new processing for side channel analysis.» dans *Proceedings of the conference on Design, automation and test in Europe*, p. 1274-1279. ACM. **2008**.

[RoMu14] Debapriya Basu Roy, Debdeep Mukhopadhyay, Masami Izumi, et Junko Takahashi.
«Tile before multiplication: An efficient strategy to optimize DSP multiplier for accelerating prime field ECC for NIST curves.» dans *Proceedings of the 51st Annual Design Automation Conference*, p. 1-6. ACM. **2014**.

[SaGü14] Pascal Sasdrich et Tim Güneysu.
«Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices.» dans *Reconfigurable Computing: Architectures, Tools, and Applications*, p. 25-36. Springer. **2014**.

[Sema04] Igor Semaev.
«Summation polynomials and the discrete logarithm problem on elliptic curves.» dans *IACR Cryptology ePrint Archive*, vol. 2004, p. 31. **2004**.

[Shan71] Daniel Shanks.
«Class number a theory of factorization and genera.» dans *Proceedings of Symposia in Pure Mathematics*, vol. 20, p. 415-440. **1971**.

[StTh06] Douglas Stebila et Nicolas Thériault.
«Unified Point Addition Formulae and Side-Channel Attacks.» dans *Cryptographic Hardware and Embedded Systems–CHES 2006*, vol. 4249, p. 354-368. Springer Berlin Heidelberg. **2006**.

[SuKi01] Yen Sung-Ming, Seungjoo Kim, Seongan Lim, et SangJae Moon.
«A countermeasure against one physical cryptanalysis may benefit another attack.» dans *International Conference on Information Security and Cryptology*, p. 414-427. Springer. **2001**.

[Vans92] Scott Vanstone.
«Responses to NISTs Proposal.» dans *Communications of the ACM*, vol. 35, p. 50-52. ACM. **1992**.

[VaDr15] Michal Varchola, Milos Drutarovsky, Marek Repka, et Pavol Zajac.
«Side channel attack on multiprecision multiplier used in protected ECDSA implementation.» dans *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, p. 1-6. IEEE. **2015**.

[Walt01] Colin D Walter.
«Sliding windows succumbs to Big Mac attack.» dans *Cryptographic Hardware and Embedded Systems–CHES 2001*, p. 286-299. Springer. **2001**.

[Walt04] Colin D Walter.
«Longer keys may facilitate side channel attacks.» dans *Selected Areas in Cryptography*, p. 42-57. Springer. **2004**.

[Walt04] Colin D Walter.
«Simple power analysis of unified code for ECC double and add.» dans *Cryptographic Hardware and Embedded Systems–CHES 2004*, p. 191-204. Springer. **2004**.

[WaGu05] Arvinderpas S Wander, Nils Gura, Hans Eberle, Vipul Gupta, et Sheueling Chang Shantz.
«Energy analysis of public-key cryptography for wireless sensor networks.» dans *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, p. 324-328. IEEE. **2005**.

[Ward63] Joe H Ward Jr.
«Hierarchical grouping to optimize an objective function.» dans *Journal of the American statistical association*, vol. 58, p. 236-244. Taylor & Francis. **1963**.

[What16] Inc WhatsApp.
«WhatsApp Encryption Overview.»
<*https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf*>. **2016**.

[Wiva11] Marc F Witteman, Jasper GJ van Woudenberg, et Federico Menarini.
«Defeating RSA multiply-always and message blinding countermeasures.» dans *Topics in Cryptology–CT-RSA 2011*, p. 77-88. Springer. **2011**.

[YeKo05] Sung-Ming Yen, Lee-Chun Ko, SangJae Moon, et JaeCheol Ha.
«Relative doubling attack against montgomery ladder.» dans *Information Security and Cryptology-ICISC 2005*, p. 117-128. Springer. **2005**.

Bibliographie personnelle

Journal

[PoMa16] Pontie, Simon, Paolo Maistri et Régis Leveugle.
«Dummy Operations in Scalar Multiplication over Elliptic Curves: a Tradeoff between Security and Performance» dans Microprocessors and Microsystems. Article sous presse. Elsevier. 2016.

Actes de Conférences à Comité de Lecture

[PoBo16] Pontie, Simon; Alban Bourge, Adrien Prost-Boucle, Paolo Maistri, Olivier Muller, Régis Leveugle et Frédéric Rousseau.
«HLS-Based Methodology for Fast Iterative Development Applied to Elliptic Curve Arithmetic» dans *19th Euromicro Conference on Digital System Design (DSD)*, p. 511-518. IEEE. **2016**.

[BaBl16] Backenstrass, Thibaud, Mathieu Blot, Simon Pontie et Régis Leveugle.
«Protection of ECC Computations against Side-Channel Attacks for Lightweight Implementations» dans *1st International Verification and Security Workshop (IVSW)*, p. 1-6. IEEE. **2016**.

[PoMa14c] Pontie, Simon, Paolo Maistri et Régis Leveugle.
«An Elliptic Curve Crypto-Processor Secured by Randomized Windows» dans *17th Euromicro Conference on Digital System Design (DSD)*, p. 535-542. IEEE. **2014**.

[PoMa14b] Pontie, Simon, et Paolo Maistri.
«Randomized windows for secure scalar multiplication on elliptic curves» dans *25th International Conference on Application-specific Systems, Architectures, and Processors (ASAP)*, p. 78-79. IEEE. **2014**.

[PoMa14a] Pontie, Simon et Paolo Maistri.
«Design of a secure architecture for scalar multiplication on elliptic curves» dans *10th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, p. 1-4. IEEE. **2014**.

Autres interventions

Simon Pontié.
«Étude de la sécurité des courbes quartiques de Jacobi vis à vis des attaques par analyse de puissance consommée.» *Séminaire à l'École des Mines de Saint-Etienne*, Gardanne. **2016**.

Simon Pontié.
«Prise en compte des fuites d'informations par canaux auxiliaires dans une implémentation ECC.» *Séminaire sécurité des systèmes électroniques embarqués*, Rennes. **2016**.

Simon Pontié.
«Attaque par analyse de la puissance consommée contre un crypto-processeur basé sur les courbes Jacobi quartiques.» *Journées Codage et Cryptographie*, Toulon. **2015**.

Simon Pontié, Paolo Maïstri et Régis Leveugle.
«Tuning of randomized windows against simple power analysis for scalar multiplication on elliptic curves.» *TRUDEVICE 2015: Workshop on Trustworthy Manufacturing; Utilization of Secure Devices*, Grenoble. **2015**.

Simon Pontié et Marie-Angela Cornelie.
«Fast and secure crypto-processor based on Elliptic Curve Cryptography.» *2eme Journée SCCyPhy: Security; Cryptology for CyberPhysical systems*, Grenoble. **2015**.

[PoMa14d] Simon Pontié.
«Architecture d'un crypto processeur ECC sécurisé contre les attaques physiques.» *Journées Nationales du Réseau Doctoral en Micro-nanoélectronique*, Lille, p. 1-4. **2014**.

Simon Pontié.
«Multiplication scalaire avec fenêtrage aléatoire pour la protection d'un coprocesseur de chiffrement basé sur les courbes elliptiques.» 1er Journée SCCyPhy: Security; Cryptology for CyberPhysical systems, Grenoble. **2014**.

Annexe 1 Multiplieur de Montgomery par digits

J'ai présenté le fonctionnement d'un multiplieur série-parallèle dans la section I-3-A.iii (page 29). Il est aussi possible de manipuler des nombres découpés en digits plus larges que 1 bit [KoAc96]. Ceci a un intérêt dans les situations où la cible matérielle peut utiliser un multiplieur câblé efficace : c'est le cas de la plupart des cibles FPGA qui contiennent des DSPs. La Figure 1 présente une seconde version du multiplieur de Montgomery. Contrairement à la première version qui ne requière qu'un additionneur sur $n + 2$ bits, cette version utilise un multiplieur d'entiers élémentaire manipulant des opérandes de r bits et produisant un résultat sur $2r$ bits. Les opérandes et le nombre premier p sont alors représentés sous forme de plusieurs digits de r bits. L'opérateur noté $\lceil . \rceil$ correspond un arrondi à l'entier supérieur. Le nombre premier p étant sur n bits, il faudra $s = \lceil n \div r \rceil$ digits pour représenter les nombres : la Figure 1 illustre le cas où 3 digits sont utilisés. On a donc toujours $p < 2^{r.s}$ puisque $2^n < 2^{r.}$ et $p < 2^n$.

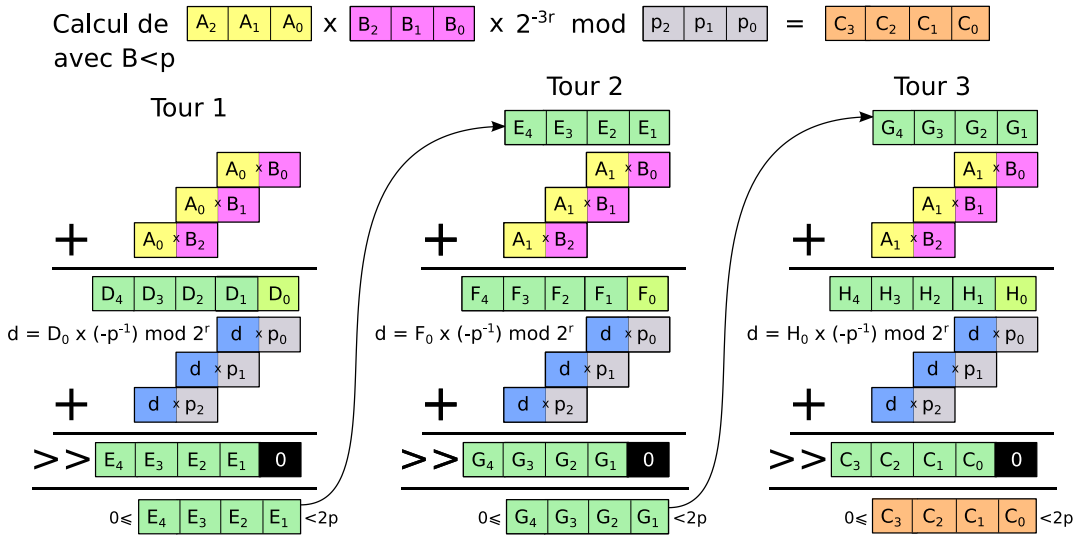


Figure 1 Multiplieur de Montgomery par digits: exemple avec p sur 3 digits sans réduction finale

On retrouve des boucles proches de la version série-parallèle mais ici l'accumulateur est divisé par 2^r au lieu de 2. L'élément ajouté à l'accumulateur en début d'itération n'est plus seulement B ou 0 comme dans la première version, ici c'est $A_i \times B$ qui est accumulé avec A_i. Comme on le voit sur la Figure 1, l'accumulation du produit $A_i \times B$ peut aussi être réalisée en

accumulant $A_i \times B_j$ pour les j de 0 à $\lceil n \div r \rceil - 1$. Ajouter p ou 0 n'est plus suffisant car il faut annuler tous les bits du digit de poids faible de l'accumulateur. On va donc ajouter $d \times p$. La valeur du digit d est calculée de manière à annuler le digit de poids faible de l'accumulateur. Par exemple dans le deuxième tour de boucle, on choisira $d = F_0 \times (-p^{-1}) \ mod \ 2^r$. Le calcul efficace de d consiste à récupérer le poids faible du résultat du produit des digits F_0 et p_c. La valeur du digit p_c est calculée hors-ligne et vaut $-p^{-1} \ mod \ 2^r$. Pour de nombreuses courbes elliptiques et des $r < 32$ bits, p_c peut valoir 1. Cette seconde version du multiplieur de Montgomery respecte aussi $C < 2p$ si on a $B < p$ et $p < 2^{r\lceil n \div r \rceil}$ (que l'on a assuré par construction) [Guer02]. Cela signifie que si le bit de poids fort du digit de poids fort de p (p_2 dans l'exemple) vaut 1, alors le digit C_3 (comme $K_5 \ et \ G_5$) ne peut valoir que 0 ou 1. Dans le cas contraire : $p_2 < 2^{r-1}$, C_3 sera toujours nul. Il faut quand même réaliser une réduction finale pour assurer $C < p$.

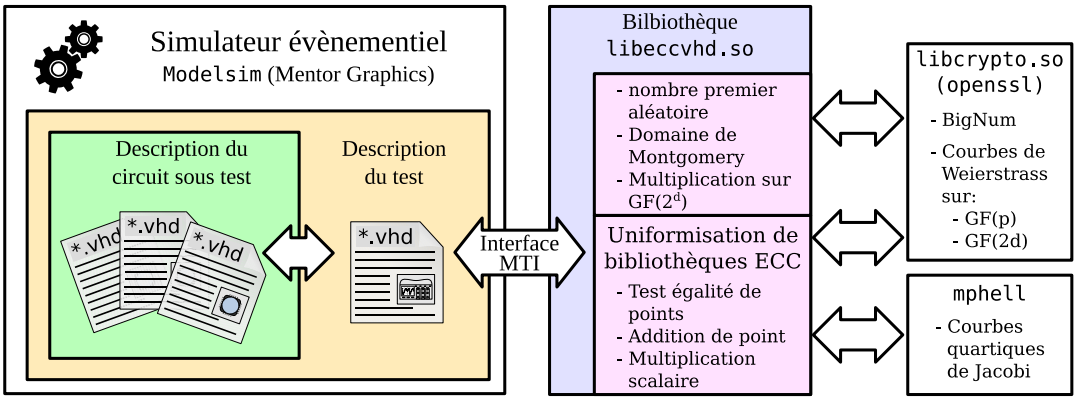
Il existe une alternative intéressante à la réduction finale. Si l'on choisit de coder les nombres sur m digits tels que $4p < 2^{r \times m}$ alors on effectuera m tours de boucle et on assurera l'invariance de la précondition. C'est-à-dire que $C < 2p$ sera respecté si $B < 2p$ [Guer02] : la précondition a été relaxée. Il devient inutile de réaliser une réduction finale car la sortie du multplieur de Montgomery peut être réutilisée directement en opérande d'entrée. Il est possible que $m > \lceil n \div r \rceil$, c'est-à-dire qu'un tour de boucle a été ajouté. Cela dépend de r et de p, si lorsque l'on code p en digits de r bits, le poid fort de p a ses deux bits de poids fort nuls alors $m = \lceil n \div r \rceil$ et il n'y aura pas de tour de boucle supplementaire dans le multiplieur de Montgomery tout en assurant l'invariance de la précondition. Sinon le coût sera simplement l'ajout d'un digit pour coder les nombres et donc un tour de boucle suplémentaire. Pour le cas particulier de $r = 1$ (qui correspond au multiplieur série-parallele, voir Figure I-13 page 31) cette astuce est plus coûteuse car il faudra ajouter deux tours de boucle.

Le multiplieur de Montgomery par digit est particulièrement adapté aux cibles FPGA car elles sont munies de blocs DSP qui permettent une implémentation efficace des produits partiels. Les dépendances de données dans cette version du multiplieur sont compatibles avec des techniques de pipeline. Il est possible de réduire encore plus les dépendances de données en retardant les réductions (ajout de $d \times p$ et division/décalage) de un ou plusieurs tours de boucle : c'est le multiplieur de Montgomery à quotient pipeliné [MaLi13]. L'accumulateur doit pouvoir stocker autant de digits supplémentaires que de retard à ajouter. D'autres optimisations nécessitant plus de mémoire sont possibles [KoAc96].

Annexe 2 Validation d'un crypto-processeur par simulation numérique

Tous les circuits présentés dans cette thèse ont subi une vérification poussée. L'automatisation des tests est essentiels lorsque qu'il y a de nombreux circuits et pour permettre au concepteur d'effectuer des tests de non-régressions. Pour les travaux présentés dans cette thèse, j'ai automatisé le test fonctionnel de chaque élément du crypto-processeurs. Ces tests sont réalisés par le simulateur évènementiel `Modelsim` de Mentor Graphics qui simule l'élément sous test à partir de sa description. Il simule aussi le testeur qui fournit les vecteurs d'entrées au circuit sous test. C'est aussi son rôle de vérifier à l'aide d'assertions la validité des vecteurs de sorties.

Le jeu de fonctions à la disposition d'un testeur décrit en « `vhdl` » est limité et ne permet pas d'effectuer une multiplication sur $GF(2^d)$ ou manipuler des points sur une courbe elliptique. Le test de nombreux éléments d'un crypto-processeur doit donc être écrit d'une manière statique. Il en résulte des tests peu flexibles et un nombre restreint de vecteurs. Pour remédier à ce problème j'ai développé un jeu de fonctions « `vhdl` » permettant au testeur d'effectuer des opérations cryptographiques classiques pour générer des vecteurs d'entrées aléatoires et vérifier les vecteurs de sorties associés. Ceci a été réalisé sous la forme d'une bibliothèque dynamique (`libeccvhd.so`) a l'intermédiaire entre `Modelsim` et des bibliothèques cryptographiques. Je peux ainsi comparer les sorties de mes circuits avec ceux d'une implémentation de référence, dans mon cas une bibliothèque logicielle cryptographique largement auditée et validée : `openssl`.



Fifure 1 Bibliothèques cryptographiques et simulation de circuits numériques

Les capacités offertes au testeur couvre l'arithmétique sur $GF(p)$ et $GF(2^d)$ et la manipulation de points sur une courbe elliptique. La vérification fonctionnelle devient dès lors plus robuste : par exemple, il est possible de vérifier l'implémentation de l'arithmétique sur $GF(p)$ ou $GF(2^d)$ pour de nombreux nombres premiers ou polynômes irréductibles choisis au hasard. L'implémentation logicielle de référence utilisée pour l'arithmétique sur ces deux corps est celle d'`openssl`.

Ma bibliothèque offre aussi la possibilité de décrire des tests faisant appel à des fonctions tels que : générer un point aléatoire sur une courbe donnée (en coordonnées affines, ou avec une représentation projective aléatoire), vérifier que deux points sont équivalents même s'ils ont des représentations différentes et bien sûr, effectuer les opérations sur les points (addition, doublage, multiplication scalaire). La référence des opérations sur les courbes de Weierstrass sur $GF(p)$ ou $GF(2^d)$ est fournie par `openssl` avec la possibilité d'utiliser une courbe déjà intégrée dans cette bibliothèque ou une courbe arbitraire. La référence des opérations sur les courbes quartiques est fournie par la bibliothèque logicielle `mphell` développée à l'institut Fourier (Grenoble).

Cette bibliothèque est réutilisée pour les tests et les attaques sur carte. Elle permet de générer les points choisis au hasard sur une courbe qui sont ensuite envoyés au circuit à travers une interface UART. Que ce soit pour le test sur carte ou les attaques, le point résultat est toujours reçu et vérifié via une bibliothèque cryptographique de référence.

Annexe 3 Synthèse de haut niveau appliquée à la cryptographie sur les courbes elliptiques

Suite à mes travaux présentés dans le Chapitre II basés sur une méthode classique de conception (de bas en haut), j'ai exploré les possibilités que peut offrir la conception de haut niveau (HLS) appliqué à la cryptographie basée sur les courbes elliptiques. Ce travail a été mené en collaboration avec l'équipe SLS du laboratoire TIMA, plus particulièrement avec Alban Bourge pour la partie applicative, flot de conception et Adrien Prost-Boucle pour le support sur l'outil de synthèse de haut niveau AUGH [PrMu14] dont il est le développeur principal. Je me contente, ici, de fournir un aperçu de ce travail et de ces conclusions. Pour plus de détails, je renvois le lecteur vers l'article [PoBo16].

La méthode de conception est constituée de plusieurs boucles décrites par la Figure 1. Durant la première phase de développement, de premières itérations sur la description haut niveau permettent d'arriver à une description fonctionnelle. S'en suit alors des itérations pour optimiser les performances estimées par l'outil de HLS. Le développement se termine par une optimisation des performances en sortie du flot complet.

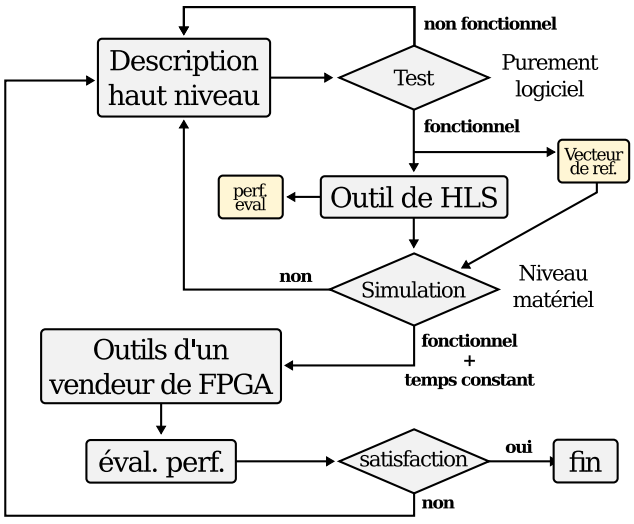


Figure 1 Flot et méthode de conception

La première ligne de la Figure 2 décrit les outils que nous avons utilisés pour la première boucle. Le langage de haut niveau que nous utilisons est le langage C et la vérification

fonctionnelle est faite par comparaison avec `openssl`. Le logiciel AUGH [PrMu14] effectue une synthèse de haut niveau pour générer des estimateurs et une description matérielle (en langage `vhdl`) à partir de la description haut niveau. C'est un logiciel libre supportant un grand nombre de FPGAs (Xilinx, Altera, Lattice). Dans nos expérimentations, nous avons optimisé la description de haut niveau pour des FPGA Xilinx.

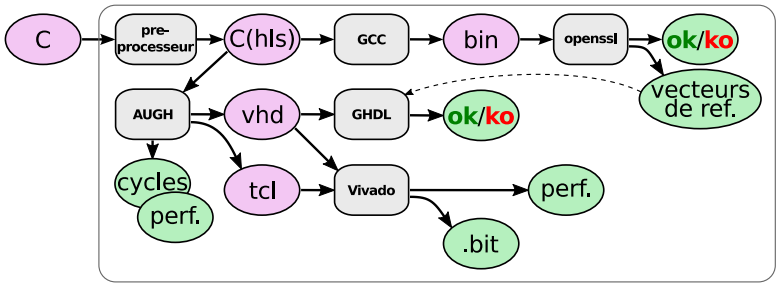


Figure 2 Outils sélectionnés pour en mettre en œuvre le flot de conception

Cette méthode de conception est rapide car les itérations en vue d'obtenir une description fonctionnelle sont purement logicielles et l'on dispose en quelques secondes d'estimateur sur le circuit matérielle en amont des outils du vendeur de FPGA. Par exemple, un multiplieur de Montgomery par digit a été développé en 14 jours d'optimisations après de nombreuses modifications architecturales et algorithmiques (voir Figure 3).

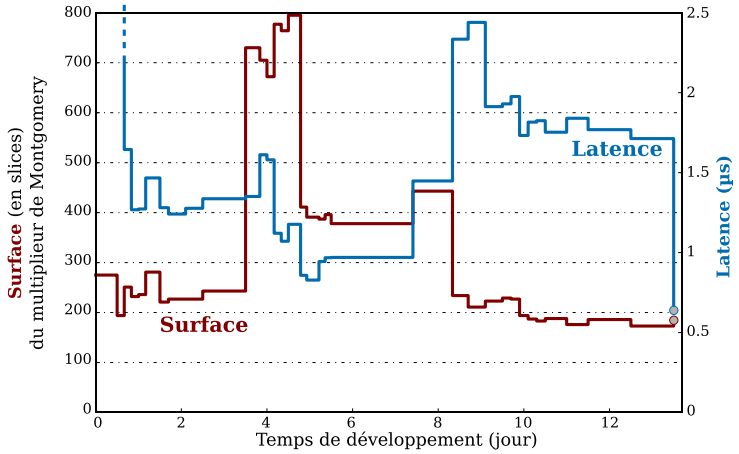


Figure 3 Développement d'un Multiplieur de Montgomery en quelques jours

Nous avons montré que la synthèse de haut niveau appliqué à la cryptographie basée sur les courbes elliptiques permet d'obtenir rapidement des circuits compétitifs (voir Figure II-8, page 67). Elle peut s'adapter à des contraintes spécifiques tels que la manipulation de grand entiers et le calcul en temps constant. Par contre certaines étapes de la méthode proposée nécessitent une intervention humaine. Le degré de parallélisme atteignable a quelques limites avec l'outil de HLS, mais des développements en ce sens sont en cours.

Annexe 4 Courbes quartiques de Jacobi

Les courbes quartiques de Jacobi utilisé dans ma thèse ont été générées par Marie-Angela Cornelie. La méthode de génération est une génération itérative de courbe de Weierstrass aléatoire jusqu'à obtenir une courbe qui satisfasse des critères de sécurité. Cette courbe elliptique sûre vis-à-vis d'attaques mathématiques est ensuite transposée en une courbe quartique de Jacobi. Plus d'informations sur la génération des courbes à partir de la bibliothèque `mphell` développée à l'institut Fourier (Grenoble) sont disponibles dans la thèse de Marie-Angela Cornelie [Corn16].

1. Courbe de 256 bits #1

$p =$ `0xd7c794d676fb1dba22ebf0fbd62bf53e9997b994059fdcf28d6d6e4f7c324327`

$a =$ `0x3d3be73995339a2b1ad3967e03ed4caf85f8ddfa57770c445fde382f10c0a9a0`

$G_x =$ `0x6cbe5e1919960c93e35155d1333c5586bb3b407bf455d8184a5a495b908074ac`

$G_y =$ `0xcd1f7be594cbc6c40bc40d2896a9e1cc80fc033276a5fd57b6e22070c69bde15`

$G_z =$ `0x871bbb4658f18e91a1cf02f0b0e153eacb342329a5d58f877cf10ad2a51e3458`

$G_t =$ `0xc436472c7e22157d6af7a8db595ae957d0aef74d97937bd132201151bf04bc85`

$Order =$ `0x35f1e5359dbec76e88bafc3ef58afd4f5ea21a6b25716bf3f6f99c73bf314f5b`

$Co - factor =$ `0x4`

2. Courbe de 256 bits #2

$p =$ `0xd996abb73c9e96f2bfc5092b673c62617183b49a8ab625c88cb91e82a3386f5f`

$a =$ `0x55a3b8e806e20a8baddf27f9471ab978a78d2552b55d2a870f7efa1628d498f2`

$G_x =$ `0x19424f36c251009898f7671093429439cd19a33506191d1bec74aa0cd4ca8c5a`

$G_y =$ `0x822e63dcd8c5636a50028d49f02475d9286e4de5f7d3c9246e50e7c168e8c870`

$G_z =$ `0x743e8547e38d4d8cef9aa58afe9855c7a1c9ec91aba5df68184464c6cfb3a68f`

$G_t =$ `0xd699a93ab583c63951a723d062723f5c0a3e62d97113e45e6bf0af912a68b2e2`

$Order =$ `0x6ccb55db9e4f4b795fe28495b39e3130cc32f64bc147e7d71749028cf49c2ece`

$Co - factor =$ `0x4`

3. Courbe de 256 bits #3

$$p = \texttt{0xe13c785ed201e065f98fcfa6f6f40def4f92b9ec7893ec28fcd412b1f1b32d33}$$
$$a = \texttt{0x5989099806b987f64e89f5945d6ac68d76fb285628312a21d2b357f5782c9ce7}$$
$$G_x = \texttt{0xbfdf67eeb6f3ac87cb60bd13e6c2da68062d0670721cffc0265e28c856091570}$$
$$G_y = \texttt{0x3697fdb8b8047998abc6aaa12b50026305745b357c0d154675ecf60b661f13f4}$$
$$G_z = \texttt{0xd3513cd930ceabdc9ac77f7253a1f5c1dc448e3debc7f1e6ebe48bf03b79c125}$$
$$G_t = \texttt{0x3b92c29d28cce0ff4af013176477d278a904bd96c1237490bacfad34de95e7b8}$$
$$Order = \texttt{0x384f1e17b48078197e63f3e9bdbd037c2628ec2f2fd86f34571fa58e3b20d7a7}$$
$$Co - factor = \texttt{0x4}$$

4. Courbe de 256 bits #4

$$p = \texttt{0xb09f075797da89f57ec8c0265b014c5ba37bcb84208a4c4d83199a31d17cb1af}$$
$$a = \texttt{0x1468727718e87fae1d94287a36205fa0e3b70bbaad31a097c1ffa5a5db263683}$$
$$G_x = \texttt{0x7e99fa90a889802939e8704d3cc8abc5c0c879540dea85075eabfb6c73d515c5}$$
$$G_y = \texttt{0x5d5db0901ca66a952304e220fcf6581c55fb7fb7986522096911c09d1e76de9b}$$
$$G_z = \texttt{0x86eccefd9aa9174b1e61505ba174452525f7fd2989304574b298978e79690c20}$$
$$G_t = \texttt{0x77d5d37af2ee21e362a68ec5e277b40538434a020c6d1a3c885b6eb8fec53dd3}$$
$$Order = \texttt{0x2c27c1d5e5f6a27d5fb2300996c0531746f66dec9a2e0bd13eabeafa863611f7}$$
$$Co - factor = \texttt{0x4}$$

5. Courbe de 384 bits #1

$$p = \texttt{0xacd49a36ff1e46676e14a637afeb43d10f3984babf68bb26}$$
$$\texttt{db6d6266b0068743d816766326410428fa661d6bd4943d3f}$$
$$a = \texttt{0x3b4c751352a31b859999a6e561976c0a402a0b21ad95a255}$$
$$\texttt{b7bcfd92fd300738c26f86093b3533f993f0a122ff51be44}$$
$$G_x = \texttt{0x3183facd2992bbf1d81d87927178b6132f2cb48812381c4}$$
$$\texttt{e7283ceb15b42aec1fa6b24f14db1d927159d8296e883c1e7}$$
$$G_y = \texttt{0x58faf1120020cbb54ad6a6b25793d88ec4b80821e643579f}$$
$$\texttt{365f3f360c18c58c78e1faf010c19d5a491ea8a279208b1a}$$
$$G_z = \texttt{0x1136fa4a2ebb9799bc9fcaf79e5a8d88e09e45cc466310fe}$$
$$\texttt{0fa4a442f3b7d83ed09c86f986e8805d1c50162859698262}$$
$$G_t = \texttt{0x1fb38ea18a3332af67788940b650418acfecb8102ff1b1d8}$$

$$877a6b81a52bfc3bffd1ae75368bbff4c5c065c483f7d4d6$$

$$Order = 0x2b35268dbfc79199db85298debfad0f443ce612eafda2e$$

$$c9f933eed9c48d1e7f55bf1068ed54c925a3422176ae0d2cdb$$

$$Co - factor = 0x4$$

TITRE : Sécurisation matérielle pour la cryptographie à base de courbes elliptiques

RÉSUMÉ De nombreuses applications imposent des contraintes de sécurité élevées (notamment au sens confidentialité et intégrité des informations manipulées). Cette thèse porte sur l'accélération matérielle du système de cryptographie asymétrique basé sur les courbes elliptiques (ECC). L'environnement des systèmes visés étant rarement maîtrisé, je prends en compte l'existence potentielle d'attaquants avec un accès physique au circuit.

C'est dans ce contexte qu'un crypto-processeur très flexible, compatible aussi bien avec des cibles ASIC que FPGA, a été développé. Dans le but de choisir des protections contre les attaques dites matérielles (analyse de consommation, génération de fautes, etc.), j'évalue la sécurité vis-à-vis des attaques par canaux auxiliaires (ou canaux cachés) et le coût de la contre-mesure basée sur l'unification des opérations élémentaires sur des courbes elliptiques. En montant une nouvelle attaque contre un circuit mettant en œuvre des courbes quartiques de Jacobi, je montre qu'il est possible de détecter la réutilisation d'opérandes. Des expérimentations réelles m'ont permis de retrouver le secret en exploitant seulement quelques traces de puissance consommée. Je présente également une nouvelle protection permettant d'établir un compromis entre le niveau de sécurité, les performances et le coût. Elle est basée sur une accélération par fenêtrage aléatoire et l'utilisation optimisée d'opérations fictives.

Mots clés : *Sécurité matérielle · Courbes elliptiques · Canaux auxiliaires · Analyse de Puissance consommée*

TITLE : Hardware security for cryptography based on elliptic curves

ABSTRACT Many applications require achieving high security level (as confidentiality or integrity). This thesis deals with hardware acceleration of asymmetric cryptography based on elliptic curves (ECC). These systems are rarely in a controlled environment. With this in mind, I consider potential attackers with physical access to the cryptographic device.

In this context, I developed a very flexible crypto-processor that can be implemented as an ASIC or on FPGAs. In order to evaluate protections against physical attacks (power consumption analysis, fault injection, etc), I evaluate the security against side-channel attacks and the cost of the counter-measure based on operation unification. I present a new attack against a chip using Jacobi quartic curves and show that the reuse of operands is detectable. By exploiting only a few power consumption traces, I am able to recover the secret. I present also a new counter-measure allowing a configurable tradeoff between security level, performances, and overheads. It uses random windows to accelerate computation, mixed to an optimized usage of dummy operations.

Keywords: *Hardware security · Elliptic curves · Side channels · Power Consumption Analysis*

INTITULÉ ET ADRESSE DU LABORATOIRE
Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France

ISBN 978-2-11-129218-5