Leak study of cryptosystem implementations in randomized RNS arithmetic

Jérôme Courtois

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

LIP6

THÈSE

pour obtenir le titre de

Docteur de Sorbonne Université

Spécialité : Informatique

Jérôme COURTOIS

Étude des fuites d'implémentations de cryptosystème en arithmétique RNS randomisée

---

Directeur de thèse: Jean-Claude BAJARD

Co-encadrant de thèse: Lokmane ABBAS-TURKI

Soutenue le 10 septembre 2020 devant le **Jury composé de :**

Rapporteurs :

Pierre-Alain Fouque	-	Université de Rennes I, IRISA.
Florence Merlevéde	-	Université Paris-Est-Marne-La-Vallée, LAMA.
Stephane Vialle	-	CentraleSupelec, LRI, Université de Paris-Saclay.

Examinateurs :

Caroline Fontaine	-	CNRS/Université de Paris-Saclay, LSV.
Karine Heydemann	-	Sorbonne Université, Paris, LIP6.
Emmannuel Prouff	-	ANSSI, Paris.

Directeur :

Jean-Claude Bajard	-	Sorbonne Université, IMJ-PRG.

Co-Encadrant :

Lokmane Abbas-Turki	-	Sorbonne Université, LPSM.

A ma mère († 2008), docteur en médecine,
A mon père, docteur en sciences économiques.

Remerciements

Et voici la boucle est bouclée...Il faut maintenant que je remercie tout ce monde. Parce qu'il y en a des gens pleins d'intérêts, d'étonnements et de soutiens qui sont derrière cette thèse qui m'a rempli de doute du début à fin. Le problème de faire une thèse à l'âge de 52 ans, c'est qu'on est toujours entouré de gens que l'on considère plus brillants que soi. J'ai découvert le syndrome de l'imposteur et il a fallu que je me batte contre ce personnage maladroit qui habitait en moi.

Il faut tout d'abord que je remercie Jean-Claude et Lokmane qui ont accepté que je fasse cette thèse. Ils m'ont offert cette opportunité avec une générosité qui m'a profondément touché. Ils m'ont fait découvrir le monde de la recherche, sa beauté, sa dureté et ses errements. Tous les deux m'ont encouragé à publier un article dans une revue de bonne qualité. Son acceptation a été moteur pour aller au bout de cette thèse. Je remercie Jean-Claude pour sa patience et ses encouragements. Je n'imagine pas les inquiétudes qu'il a dû avoir quand il a fallu gérer mes angoisses et mes doutes qui revenaient avec une périodicité métronomique. Je suis étonné qu'il ait pu faire certaines taches que j'aurais dû faire. Il a été aux petits soins pour moi. Peut être que cette thèse a pu être menée au bout grâce à nos ressemblances. Je remercie Lokmane qui a dû souffrir encore plus de mes manques et de mes peurs. Il a su être m'accompagner avec bienveillance. J'aurais aimé parfois être plus à la hauteur de ses attentes. Il a souvent mieux vendu mon travail quand moi je le dévalorisais.

Je remercie Pierre-Alain Fouque, Stépnane Vialle et Florence Merlevéde qui ont bien voulu être rapporteur de mon manuscrit. Et je les en remercie d'autant plus que ce travail transversal abordait parfois des domaines qui n'étaient pas forcément proches de leurs sujets de prédilection. Je tiens également à remercier Caroline Fontaine, Karine Heydemann et Emmannuel Prouff qui ont accepté de faire partie de ce jury de thèse.

Ma première rencontre avec le monde de la recherche fut le bureau 336 avec Charles et Guillaume qui m'ont énormément aidé pour comprendre le langage C. Et ils furent des partenaires essentiels pour installer, sur une machine réticente, les outils nécessaires à ma recherche. Je remercie aussi les occupants des bureaux adjacents comme Anastasia et Yvan qui permirent que l'ambiance du bureau fut aussi souvent au beau fixe. Et il y en a d'autres qui participèrent à l'ambiance et que je me dois de citer : Romain, Daisuke, Sénan, Ramon, Thibault... J'ai peur d'en oublier. Comme j'étais à mi-temps et que j'avais une fille à aller chercher à l'école, j'ai raté trop de rendez-vous au bar ou au restaurant.

Merci à Damien, chef d'équipe d'Almasty, dont le bureau était toujours ouvert pour prendre un café et m'écouter, et qui a su avec Charles et Jean-Claude, créer une atmosphère détendue mais propice à un travail efficace. Merci à Charles pour son humour, humour très contextualisé mais toujours appréciable pour relativiser notre monde. Merci

aussi, Antoine, pour quelques conseils précieux. Je ne peux surtout pas ne pas penser à tous ceux de l'équipe qui m'ont accompagné parfois longtemps, parfois pour de courte période : Florette, Thomas, Anand, Alexandre, Lucas, Julia, Shi, Natacha, Noura....et surtout Vincent. Je n'oublierais pas les grandes discussions avec Florette et Thomas sur le Japon, le japonais, les Japonais et les animes. Je garde un souvenir ému d'Alexandre qui n'arrêtait pas de m'appeler "le vieux", irritant mais toujours sympathique surtout pendant les parties de fléchettes. Et puis quand Lucas passait de temps en temps, je me sentais bien entre camarade de promo. Anand, thank you for your smile, your incredible stories of your travels and very thank you for having corrected my horrible English. Et puis il y a Vincent, mon compagnon de route du début qui m'a accompagné pendant 3 ans. Quand il est parti, je lui en ai presque voulu. Le reste du temps, son cynisme et sa présence me manqueront de façon récurrente.

Heureusement qu'il y avait Babacar avec qui j'ai travaillé dans la bonne humeur quoique parfois nos cerveaux saturaient.

A cause, disons grâce à cette thèse je suis parti seul à l'étranger pour la première fois depuis l'age de mes 22 ans. C'était pour moi un saut dans l'inconnu de partir aux antipodes sans être accompagné, tout seul. Lors de ce voyage en Australie, à l'université de Wollongong, j'ai été accueilli par Thomas qui a été aux petits oignons. Et ce fut vraiment agréable d'être entouré d'Andréa et Arnaud qui ont travaillé avec moi comme si j'avais toujours été là.

Il faut que je remercie mes collègues de mathématiques du Lycée Duruy : Damien,Yves, Nicolas, Jean-Claude, Philippe, Laïla, Marie-Dominique, Anne-Valérie, Audrey, Irène, Pierre et Guillaume. Parce qu'il ne faut pas oublier que j'enseignais à mi-temps au Lycée Victor Duruy pendant ce long cheminement doctoral. Ils m'ont encouragé concrètement en faisant certains travaux à ma place. Ils ne m'ont jamais reproché mes absences à certaines réunions et ils ont souvent devancé mes manquements. J'ai une pensée pour Fayçal qui m'a sorti d'un blocage mathématique dans les derniers jours de rédaction. Et puis il faut que je remercie particulièrement Franck d'avoir travaillé comme un fou pour notre arrivée dans la nouvelle spécialité Informatique (NSI) de première. L'administration de lycée n'est d'ailleurs pas en reste. Mr Tournier et Mr Duma-Delage furent bienveillants sur quelques-unes de mes absences à certains conseils de classe et ils n'ont jamais refusé leur soutien pour les problèmes administratifs.

Quel enrichissement et que de rencontres pendant ces années. Mais ils ne faut pas que j'oublie mes amis qui m'ont encouragé à continuer. Tous par leurs petits mots, remarques, interrogations, petites admirations ou rappels à l'ordre, ils ont participé à la continuation de ce travail. Il faudra maintenant que je fasse plein d'invitations dans tous les sens pour rattraper mes absences et mes manquements. Il faut que je réapprenne à utiliser le téléphone. Une petite note pour Florence et Bruno qui m'ont fait répété la soutenance

et dont l'enthousiasme était communicatif.

Dans la famille il y a Garance et Clément qui furent là pour me communiquer leur énergie. Et puis il y a mon frère Jocelyn, sa compagne Véronique, mon neveu Jim et ma nièce Adèle. Ils ont toujours été présents dans mes pensées. Merci à Dominique et mon père ! Ils ont gardé ma fille Blanche chaque fois que je leur demandais pour me permettre de travailler au maximum. Je m'en veux encore d'avoir raté la cérémonie de retraite de mon père alors que j'étais tellement absorbé par un code réticent. Que je puisse oublier ce moment d'égarement! Encore désolé Papa.

Et puis j'étais entouré de l'amour de Sabine, Violaine et Blanche. Ce qui n'est pas le moindre avantage quand on veut écrire une chose aussi compliquée qu'une thèse. Tout cela n'aurait sûrement pas eu lieu sans Sabine qui, elle même encadrant des doctorants, savait la difficulté de rester concentré sur le long terme. Elle a relu mon manuscrit jusqu'à vouloir comprendre les moindres recoins de ce domaine qui lui était totalement étranger. Elle n'a pas seulement accepté que je me plonge dans la recherche, elle l'a intégré dans notre vie de famille comme un moment de folie nécessaire. A chaque moment de doute, elle a su trouver les mots pour me replonger dans l'aventure. Je voudrais prolonger cette dédicace rien que pour elle.

Sommaire

Chapter 1

Introduction

Sommaire

La cryptologie, la science du secret, est la science qui étudie le chiffrement suivant deux aspects: la cryptographie et la cryptanalyse. La cryptographie s'intéresse à la protection des données tandis que la cryptanalyse a pour objectif de corrompre les propriétés apportées par la cryptographie. On parle couramment d'attaque sur un cryptosystème. Les objectifs de la cryptographie sont de quatre ordres :

- La confidentialité garantit que seuls les participants à un échange d'information soient détenteurs de celle-ci.
- L'authenticité assure l'identité des participants lors d'un échange d'information.
- L'intégrité assure que l'information n'a pas été modifiée lors d'un échange.

- La non-répudiation empêche les participants de nier avoir participés à l'échange.

Les nouveautés en matière de cryptologie sont apparues pendant les différents conflits émaillés tout au long de l'histoire. Les cryptosystèmes les plus connus sont le chiffrement de César qui fut utilisé pendant la guerre des Gaules et la machine allemande Enigma utilisée pendant la seconde guerre mondiale et décryptée par l'équipe de Alan Turing. Dans les deux cas, le chiffrement, c'est-à-dire le brouillage du message échangé, est assuré par une même clef secrète possédée par tous les participants à l'échange d'information. On parle ici de chiffrement symétrique. Chaque participant à l'échange du secret possède la même clef de chiffrement. Le défaut de cette méthode nécessite la communication préalable de la clef de chiffrement. Comment communiquer cette clef de manière sécurisée ? Il faut attendre les travaux de W. Diffie et M. Hellman [38], et de R. Merkle [74] dans les années 70 sur le chiffrement asymétrique, aussi appelé chiffrement à clef publique pour avoir une réponse satisfaisante.

Le chiffrement asymétrique repose sur le principe qu'une personne génère deux clefs qui sont liées entre elles. Cette relation doit être difficile à retrouver du point de vue calculatoire. Une des clefs est secrète, c'est-à-dire connue de la seule personne qui veut déchiffrer le message. L'autre clef est publique et peut être utilisée par n'importe qui. L'analogie classique pour se représenter le chiffrement asymétrique est le principe de la boîte aux lettres. La clef publique est une boîte aux lettres et n'importe qui peut y déposer un message. Cette boîte ne peut être ouverte que par la clef "secrète" détenue uniquement par le possesseur de la boîte aux lettres.

Pour garantir la sécurité d'un tel système, il faut que la clef secrète soit difficile à retrouver à partir de la clef publique. On parle ici de difficultés calculatoires. D'un autre côté, les calculs permettant de créer le couple de clefs publique/secrète doivent pouvoir être effectués dans des délais raisonnables. En 1976, W. Diffie et M. Hellman n'avaient pas donné d'exemple utilisable en informatique de système asymétrique. Il a fallu attendre 1978 et les travaux de R. Rivest, A. Shamir et L. Adleman [86] qui ont donné leur initiale pour nommer la méthode RSA. Ils ont donné une construction concrète d'un système asymétrique. RSA repose sur la multiplication et la factorisation des entiers. La multiplication est une opération facile, en revanche la factorisation est difficile. Par exemple, si on se donne deux nombres $a = 71$ et $b = 37$ il est rapide de faire le produit $a \times b = 2627$. En revanche, si on a un nombre comme 5141, il sera difficile de trouver les deux nombres 53 et 97 dont le produit donne 5141. RSA est le système le plus utilisé. Dans ce mémoire, on parlera d'un autre système asymétrique : les courbes elliptiques pour la cryptographie (ECC : Elliptic Curve for Cryptographie) proposées dans les années 80 par N. Koblitz [58] et V.S. Miller [76].

Parallèlement à la cryptographie, les méthodes de cryptanalyse se développent. Souvenez-vous de la course à la Reine Rouge dans "Alice à travers le miroir" de Lewis Caroll [25].

Le décor se déplace et Alice et la Reine Rouge font une course endiablée pour y rester. Certains principes de la théorie de l'évolution [93] utilisent cette image pour expliquer l'évolution d'espèces concurrentes dans leur environnement. Ici l'environnement, c'est la cryptologie. La proie est la cryptographie et le prédateur la cryptanalyse. On constate qu'il y a une course à la Reine Rouge entre la cryptographie et la cryptanalyse. Jusqu'en 1996, les méthodes de cryptanalyse se basent sur les propriétés mathématiques et algorithmiques des protocoles cryptographiques. Mais en 1996, P.C. Kocher [59] surprend la communauté scientifique en présentant ses travaux sur les attaques qui utilisent des mesures des temps de calculs (Timing Attack). Il exploite une caractéristique physique du déroulement d'un algorithme sur une implémentation matérielle pour retrouver une clef de chiffrement. Comme cette propriété ne peut pas être déduite formellement, elle est appelée fuite par canal auxiliaire.

1.1 Les fuites en cryptographie

Les attaques par canaux auxiliaires ou les attaques par fautes deviennent maintenant des problèmes récurrents qu'il ne faut pas négliger lors de la conception des implémentations cryptographiques. Les fuites générées par les calculs cryptographiques génèrent des informations cruciales qui pourraient permettre de retrouver les secrets que l'on veut dissimuler. Les fuites par canaux auxiliaires sont le résultat d'un calcul qui induit une réponse physique sur le matériel. Elles peuvent être de différents types : le temps de calcul, la consommation électrique, les radiations électromagnétiques, la chaleur, les vibrations... Suite à ses travaux sur la Timing Attack, P.C. Kocher montre avec J. Jaffe et B. Jun [60], avec la Differential Power Analysis (DPA) que les fuites de consommation de courant peuvent être aussi un moyen pour remonter jusqu'au secret. Ces fuites sont essentiellement dues aux variations d'état au niveau de la mémoire quand un bit passe de 0 à 1 ou de 1 à 0. Un attaquant, qui a accès au système physique, enregistre des traces de mesure d'un type de fuite, comme la consommation de courant, pendant les calculs successifs où l'on utilise le même secret. On fait une hypothèse sur le secret, on choisit un ensemble de messages puis on les classe en deux groupes : ceux qui produisent un bas niveau de fuites et ceux qui produisent un haut niveau de fuites. La différence des moyennes crée alors un pic si l'hypothèse sur le secret est correct. Une autre méthode statistique, pour exploiter les fuites, consiste à déterminer le secret en faisant des corrélations avec des modèles de fuites comme avec la Correlation Power Analysis (CPA) [24]. Dans notre analyse, on s'intéressera tout particulièrement à la méthode de Template Attack [18, 39, 72]. Pour effectuer une Template Attack, l'attaquant doit avoir accès à une autre copie de l'appareil protégé qu'il peut contrôler entièrement. Ensuite, il doit créer un "dictionnaire" qui représente le modèle du système. Il peut être nécessaire d'avoir

des dizaines de milliers de traces pour obtenir un modèle exploitable. Cependant, un très petit nombre de traces issues du système attaqué peut suffire pour réussir l'attaque. Dans certains cas, une seule trace suffit pour retrouver la clef.

Avec l'évolution des technologies et des connaissances, d'autres approches apparaissent, en particulier les méthodes utilisant le deep learning [83, 68]. Cependant les distances de Hamming restent l'élément de base pour créer un modèle de fuite.

Ainsi, cette étude se focalise sur le poids de Hamming, (c'est-à-dire la somme des bits d'une représentation numérique) et la distance de Hamming (égale aux nombres de bit qui changent d'état). On récolte des séquences de distances de Hamming entre les états de transitions successifs et on mesure les effets de la randomisation des représentations arithmétiques.

L'objectif serait de donner des paramètres pour qu'il soit très difficile d'établir une corrélation entre les fuites et le secret.

Les états de transitions dépendent clairement de la représentation des données. Quand un attaquant fait des hypothèses sur l'état du système, il doit tenir compte de la représentation des données. Si la représentation des données est perturbée, l'attaquant aura plus de difficulté à faire des hypothèses sur l'état du système. On veut donc éviter de reproduire l'exécution d'un calcul avec les mêmes représentations.

Par exemple: Si on représente les nombres $A = 37_{10} = 0010\ 0101_2$ et $B = 83_{10} = 0101\ 0011_2$ en base 2 sur un registre de 8 bits, la distance de Hamming entre A et B est de 5 car 5 bits ont changé d'état. Mais si on prend un codage BCD (Binary Coded Decimal) qui concatène la représentation des chiffres en base 10, $A = \overbrace{0011}^{3}\ \overbrace{0111}^{7}$ et $B = \overbrace{1000}^{8}\ \overbrace{0011}^{3}$, la distance est 4 car 4 bits ont changé d'état. Donc la distance de Hamming change quand on change de représentation. On remarque aussi que le poids de Hamming change. Comme les modèles de fuites utilisent les distances ou les poids de Hamming, si on change la représentation, un attaquant aura des difficultés à deviner le secret à partir des fuites. Il faut donc utiliser une arithmétique qui puisse randomiser les représentations entre deux exécutions.

Pour cette raison, on choisit un type de représentation qui permet une randomisation assez simple : le Residue Number System (RNS). Cette utilisation du RNS associé à une randomisation offre une contre-mesure qui permet d'être indépendante du choix de protection utilisée, que ce soit pour RSA, ECC, Pairing, réseau euclidien ou autres. Il existe d'autres systèmes de représentation qui peuvent être randomisés comme par exemple le Polynomial Modular Number System (PMNS) [37] ou encore le Modular Arithmetic Adapted Base (MAAB) proposé en 2004 par Thomas Plantard [15] .

On présente le RNS dans la Section 1.2, la méthode de randomisation dans la Section 1.3 et les types de séquences utilisés dans notre étude dans la Section 1.4. La Section 1.5

est une bref description d'une ECC.

1.2 Residue Number System (RNS)

Le Residue Number System est basé sur le Théorème Chinois des Restes (CRT). Chaque valeur est codée à partir de ses résidus sur un ensemble de moduli tous premiers entre eux. Par exemple si on considère une base $\mathcal{B}_5 = \{3, 7, 13, 19, 29\}$ et une base $\widetilde{\mathcal{B}_5} = \{5, 11, 17, 23, 31\}$,

le nombre 35 sera représenté par

$\{35[3], 35[7], 35[13], 35[19], 35[29]\} = \{2, 0, 9, 16, 6\}$ dans la base $\mathcal{B}_5$

et par

$\{35[5], 35[11], 35[17], 35[23], 35[31]\} = \{0, 2, 1, 12, 4\}$ dans la base $\widetilde{\mathcal{B}_5}$.

En introduisant la base RNS, $\mathcal{B}_n = \{m_1, ..., m_n\}$ de moduli tous premiers entre eux, le théorème des restes chinois (CRT) garantit l'existence d'un isomorphisme d'anneaux entre $\mathbb{Z}_M$ et $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$ avec $M = \prod_{i=1}^{n} m_i$. Ainsi, pour tout entier positif X strictement inférieur à M,

$$X = \left(\sum_{i=1}^{n} x_i \left| M_i \right|_{m_i}^{-1} M_i \right) \bmod M \tag{1.1}$$

avec $x_i = |X|_{m_i} = X \bmod m_i$ et $M_i = M/m_i$.

Cet ensemble de moduli est la base du système [91]. Le RNS est devenu une méthode classique de représentation pour la cryptographie sur des implémentations de type GPU ou FPGA [48, 8].

Dans [14], les auteurs ont montré que l'on peut tirer au hasard une base RNS à partir d'un ensemble de moduli, pour randomiser une exécution avec un faible coût. Depuis leur publication, ces travaux ont été utilisés et cités dans différents articles [48, 83, 63, 43, 95]. À notre connaissance, personne n'a mené d'étude complète sur le comportement aléatoire d'une telle approche et sur le type de protection qu'elle peut apporter. En particulier, on ne sait pas quel est le lien entre la randomisation des moduli et l'aléa des séquences de distances de Hamming obtenues à l'issue d'une exécution. Quelques attaques ont été publiées [42, 90, 84, 83] mais on voit qu'elles utilisent certaines bases particulières et on lit dans [13] que le type de base pourrait avoir une influence sur la randomisation.

1.3 La méthode de randomisation

Dans l'article "Leak resistant arithmetic" [14], les auteurs font un tirage de n moduli parmi $2n$ nombres entiers positifs tous premiers entre eux. Ce tirage aléatoire est basé sur un tirage standard sans remise. Le nombre de configurations possibles est $\binom{2n}{n} = \frac{(2n)!}{n!^2}$

si on ne tient pas compte de l'ordre d'enregistrement des moduli. Dans notre l'étude, on a fait le choix de prendre des moduli de 32 bits.

1.4 Construction des séquences

Pour obtenir des suites d'exécutions, on doit implémenter des algorithmes classiques de cryptographie comme RSA, ECC, Pairing ou réseau euclidien. Dans cette étude on a fait le choix de se concentrer sur ECC car il est très utilisé dans de nombreux protocoles et il a l'avantage d'être complet en terme d'opérations arithmétiques. Le travail se traite autour de trois types d'implémentation:

- Les implémentations non protégées,
- Les implémentations protégées (Avec des contre-mesures comme Montgomery Ladder [56]),
- Les implémentations dédiées à notre analyse.

Pour mesurer l'impact de la randomisation sur l'arithmétique, on se sert des résultats d'une implémentation non protégée. Nous avons besoin à ce niveau d'obtenir une série aléatoire le plus proche de la perfection pour valider notre approche et vérifier l'impact réel de ce choix. La bibliothèque doit être indépendante de l'arithmétique, seuls les opérateurs de base dépendront du choix de l'arithmétique et elle a été développée en C.

1.5 Un modèle cryptographique: les courbes elliptiques

Notre analyse se fonde sur la multiplication scalaire d'un point d'une courbe elliptique. On définit une ECC $E(\mathbb{F}_N)$ par

- N un nombre premier;
- Deux éléments a et $b \in \mathbb{F}_N$ tels que $4a^3 + 27b^2 \not\equiv 0 \bmod N$;
- Une équation $E : y^2 \equiv x^3 + ax + b \bmod N$;
- $G(x_G, y_G)$ un point de base de $E(\mathbb{F}_N)$ et n_G l'ordre de G sur $E(\mathbb{F}_N)$.

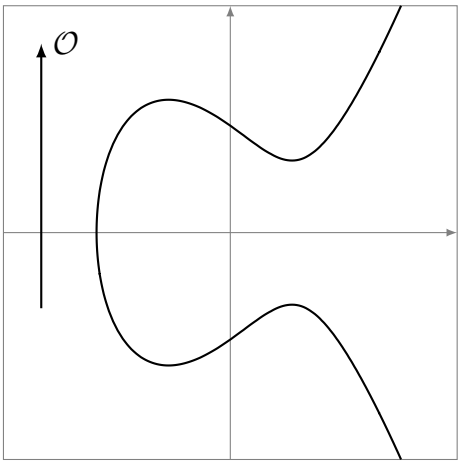
Une courbe elliptique sur $E(\mathbb{F}_N)$ est un ensemble de solutions $P(x, y)$ de E auquel on ajoute un point $\mathcal{O}$. $E(\mathbb{F}_N)$ est un groupe abélien additif avec les propriétés suivantes :

- $\mathcal{O}$ est l'élément neutre;
- $\mathcal{O} + \mathcal{O} = \mathcal{O}$;

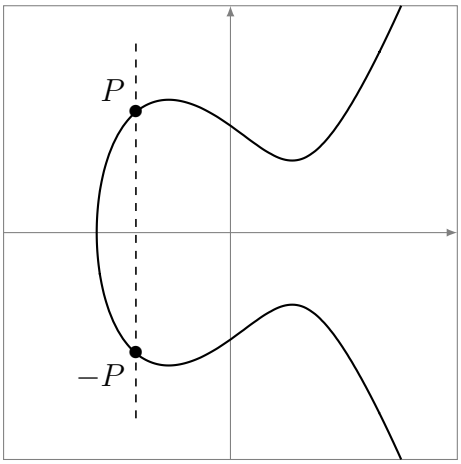
- $\mathcal{O} + P = P$;
- L'opposé : $-(x, y) = (x, -y)$;
- $P - P = \mathcal{O}$;
- L'addition de deux points $P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3)$ dans le cas où $x_2 \neq x_1$ est donnée par

  $x_3 \equiv \lambda^2 - x_1 - x_2 \bmod N$, $y_3 \equiv \lambda(x_1 - x_3) - y_1 \bmod N$ et $\lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} \bmod N$;
- Le doublement $P(x_1, y_1) + P(x_1, y_1) = R(x_3, y_3)$ dans le cas où $y_1 \neq 0$ par

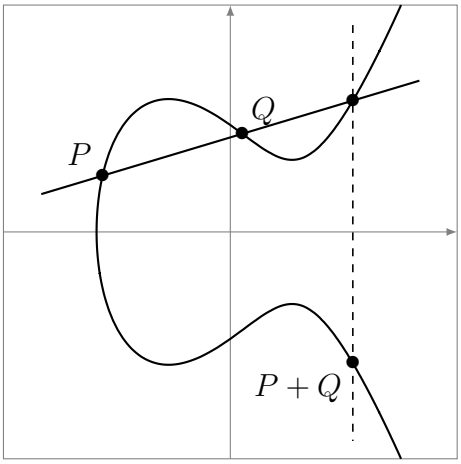
  $x_3 \equiv \lambda^2 - 2x_1 \bmod N$, $y_3 \equiv \lambda(x_1 - x_3) - y_1 \bmod N$ et $\lambda \equiv \frac{3x_1^2 + a}{2y_1} \bmod N$.



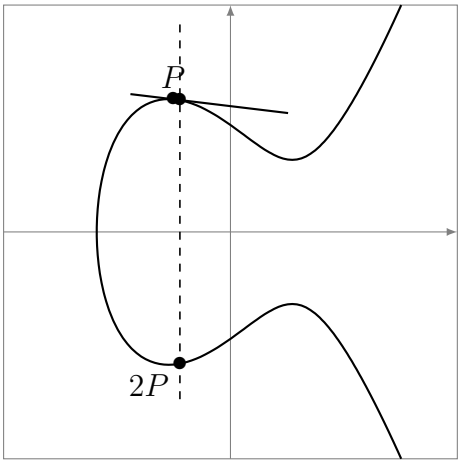
Element Neutre $\mathcal{O}$



Inverse $-P$



Addition $P + Q$



Doublement $P + P$

Pour faire un chiffrement, on se donne un point G de $E(\mathbb{F}_N)$ et une clef K un entier et on fait la multiplication scalaire $[K]G = \underbrace{G + G + \cdots + G}_{K \quad fois}$. Cette opération se retrouve dans plusieurs protocoles cryptographiques à courbes elliptiques : chiffrement, signature, etc... [71, 6]. Cela offre l'avantage d'être assez complet en termes d'opérations

arithmétiques. Plusieurs scénarios sont possibles : le secret est utilisé plusieurs fois pour déchiffrer comme dans PSEC-KEM [79], or une seule fois comme dans ECDSA [6]. Différentes attaques sont alors disponibles dans le premier cas via la CPA ou la DPA, dans le second cas via l'apprentissage. Quoi qu'il en soit, le secret est sous la forme d'un scalaire K appliqué à un point G d'une courbe elliptique E. On utilise dans toute l'étude des courbes de Montgomery recommandées par [26] et les courbes d'Edwards de 255 bits [22] pour confirmer certains résultats.

On utilise des coordonnées jacobiennes avec l'Algorithme 1 (Section 2.3.1) de multiplication modulaire qui évitent les divisions sur des grands nombres et réduisent les calculs [73, 29, 50]. Chaque point est défini par trois coordonnées jacobiennes $(X; Y; Z)$ dont la représentation affine est $(X/Z^2; Y/Z^3)$.

L'équation E devient $Y^2 = X^3 + aXZ^4 + bZ^6$, $(X; -Y; Z)$ est l'inverse de $(X; Y; Z)$ et le point infini est choisi égal à $(1; 1; 0)$. Pour calculer $[K]G$, on utilise des opérations d'addition et de doublement dans un algorithme d'exponentiation. Les opérations d'addition et de doublement se trouvent dans [73].

Pour calculer $[K]G$, on utilise, par exemple, l'algorithme d'exponentiation rapide ou l'Echelle de Montgomery (MPL) [78] que l'on utilise dans cette étude pour sa résistance à l'attaque Simple Power Analysis (SPA) [52]. L'algorithme produit une suite de $(H_{0,K}, ..., H_{i,K})$ de distance de Hamming. On utilise pour toute la suite X_K^i pour désigner le vecteur colonne de $i + 1$ variables aléatoires obtenue lors d'une réalisation de clef K i.e. $X_K^i = (X_{0,K}, ..., X_{i,K})^T$.

1.6 Quelques considérations au sujet des tests du NIST (National Institute of Standarts and Technology)

Dans cette Section, on rappelle brièvement en quoi consiste les tests du NIST puis on discute de leur pertinence dans le cadre de notre étude.

Le NIST [7] fournit des tests pour tester les générateurs de nombres aléatoires (RNG) et les générateurs de nombres pseudo-aléatoires (PRNG). Un type de RNG ou PRNG fournit une séquence de bits aléatoires qui pourrait être interprétée comme le résultat des lancés d'une pièce non biaisée avec un côté étiquetés "0" ou "1". Chaque lancé ayant une probabilité d'exactement 1/2 produisant un "0" ou "1". Chaque lancé doit être indépendant les uns des autres. Ainsi, la répétition des lancés produit un flux binaire de valeurs "0" et "1" qui sera distribué de manière aléatoire et uniforme dans $\{0; 1\}$. Tout élément d'une suite est généré indépendamment les uns des autres. On ne peut pas prédire la valeur de l'élément suivant quel que soit le résultat des éléments précédents.

Pour résumer, une suite de bits est construite par des générateurs de nombres aléatoires (RNG) à partir d'une source non déterministe comme par exemple une quantité

physique. Un générateur de nombres pseudo-aléatoires (PRNG) génère des valeurs à partir de plusieurs entrées appelées graines. Les graines sont généralement produites par des générateurs de nombres aléatoires (RNG).

En fait, il n'est pas possible de produire une véritable séquence de "0" et "1" aléatoires, c'est pourquoi on parle de générateur pseudo-aléatoire dans la plupart des situations. Les sorties d'un tel générateur pseudo-aléatoire sont utilisées dans de nombreuses applications cryptographiques. En particulier, ces sorties doivent être imprévisibles. On a donc besoin d'outils pour certifier un générateur pseudo-aléatoire. Le NIST fournit 15 tests statistiques pour tester l'uniformité et l'indépendance d'une séquence de bits de "0" et "1".

L'idée initiale de notre étude était de considérer que le système des moduli aléatoires appliqué à une multiplication modulaire pourrait être considéré comme un générateur de nombres pseudo-aléatoires. On génère des traces de distance de Hamming et on utilise les tests du NIST pour rendre compte du caractère suffisamment aléatoire de la méthode.

Cependant les distances de Hamming n'ont clairement pas une distribution uniforme. On suppose que les distances de Hamming ont une distribution binomiale que l'on approxime en une distribution normale continue.

- Description de la transformation :
  - Une variable aléatoire X binomiale discrète est traitée comme une variable gaussienne continue.
  - La variable X est transformée en variable U uniforme continue sur $[0, 1]$.
  - U est multipliée par un coefficient γ pour obtenir une variable U' uniforme et continue sur $[0, \gamma]$.
  - On introduit alors $\underline{U} = \lfloor U' \rfloor$
  - Les valeurs de $\underline{U}$ sont écrites sous la forme d'une séquence de "0" et "1" dans un fichier pour être traité par les tests du NIST.
- Le coefficient γ est l'inverse de l'écart minimum entre les réalisations de U obtenues. Cela réduit l'occurrence de la même valeur prise par $\underline{U}$. Malheureusement, il reste des trous dans cette distribution. Cela peut être une des raisons pour lesquelles les tests échoueront pour des grands nombres d'échantillons.
- On a essayé plusieurs méthodes d'enregistrement :
  1. Écriture de toutes les valeurs en base-2 (Sans fixer le nombre de bits) mais il y a trop de "1" par rapport aux "0". Par exemple la suite 1 2 3 4 5 6 7 s'écrit 1 10 11 100 101 110 111. Il y a 5 "0" et 12 "1". Pour la suite 1 2 3 4 5 6 7...$2^k - 1$, il y a $2^k - 1$ "1" de plus que de "0".

2. Ecriture sous la forme registre (En définissant un nombre fixe k de bits), en rajoutant des "0" si necessaire. Par exemple la suite 1 2 3 4 5 6 7 s'écrit 001 010 011 100 101 110 111. Cependant, il y a plus de "0" que de "1" parce que la valeur maximum est rarement du type $2^k - 1$.

3. Pour être sûr, il vaut mieux ne pas écrire toutes les valeurs. Si k est la taille maximum en bit, on écrit toutes les valeurs inférieures de taille inférieure à $k-1$ bits en rajoutant des "0" si nécessaire. Car si X est une variable uniforme sur $[0; m]$ alors pour $d \in [0; m[$, la restriction de $X_{[0;d]}$ de X est uniforme sur $[0; d]$.

   Par exemple, si la valeur maximum est 2035 de taille 11 bits. On écrit alors que les valeurs inférieures à 2^{10} et on complète avec des "0" si nécessaire pour obtenir toujours 10 bits.

On utilise les outils suivants décrits dans les Sections 1.6.1 et 1.6.2.

1.6.1 Transformation en distribution uniforme continue

Soit G une variable aléatoire avec une distribution normale centrée réduite et

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-u^2/2} du \text{ avec } x \in \mathbb{R}.$$

F est strictement croissante et continue donc inversible.

$$P(F(G) \leq t) = P(G \leq F^{-1}(t)) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{F^{-1}(t)} e^{-u^2/2} du = F(F^{-1}(t)) = t \qquad (1.2)$$

donc $F(G)$ a une distribution uniforme sur $[0; 1]$.
Si $X \sim \mathcal{N}(\mu, \sigma)$ alors $G = \dfrac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1)$.
A partir de la variable aléatoire G, on définit la variable aléatoire U uniforme sur $[0; 1]$ en utilisant la fonction

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt, \quad x \in \mathbb{R},$$

$$\begin{aligned} U = F(G) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{G} e^{-u^2/2} du = 0.5 + \frac{1}{\sqrt{2\pi}} \int_{0}^{G} e^{-u^2/2} du = 0,5 + \frac{1}{\sqrt{2\pi}} \int_{0}^{\frac{G}{\sqrt{2}}} e^{-t^2} \sqrt{2} dt \\ &= 0,5 + \frac{1}{\sqrt{\pi}} \int_{0}^{\frac{G}{\sqrt{2}}} e^{-t^2} dt = 0,5 \left(1 + \frac{2}{\sqrt{\pi}} \int_{0}^{\frac{G}{\sqrt{2}}} e^{-t^2} dt \right) = 0,5 \left(1 + erf\left(\frac{G}{\sqrt{2}} \right) \right) \end{aligned}$$

donc

$$U = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{G} e^{-u^2/2} du = 0,5 \left(1 + erf\left(\frac{G}{\sqrt{2}}\right)\right). \tag{1.3}$$

1.6.2 Transformation en distribution uniforme discrète

Pour utiliser les tests du NIST, on a besoin que la distribution soit uniforme discrète. Soit $X_1, ..., X_n$ une suite de réalisations ordonnées de X, une statistique d'ordre [33] pour une variable aléatoire normale, i.e. $X_1 < ... < X_n$.
On cherche la distance minimum entre les réalisations ordonnées de $U = F(\frac{X-\mu}{\sigma})$, i.e. $U_1 < ... < U_n$.

$$\Delta^* = \min_{i \in \{1,..,n-1\}} \Delta U_i = \min_{i \in \{1,..,n-1\}} (U_{i+1} - U_i) \tag{1.4}$$

La statistique d'ordre est strictement croissante car la loi est gaussienne. Le plus petit incrément numérique de deux réalisations successives de distance de Hamming est égale à un donc

$$\begin{aligned} U_{i+1} - U_i = F\left(\frac{X_{i+1}-\mu}{\sigma}\right) - F\left(\frac{X_i-\mu}{\sigma}\right) &= \frac{1}{\sqrt{2\pi}} \int_{\frac{X_i-\mu}{\sigma}}^{\frac{X_{i+1}-\mu}{\sigma}} e^{-u^2/2} du \\ \geq \frac{1}{\sqrt{2\pi}} \int_{\frac{X_i-\mu}{\sigma}}^{\frac{X_i+1-\mu}{\sigma}} e^{-u^2/2} du &= \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{\sigma}} e^{-(u+\frac{X_i-\mu}{\sigma})^2/2} du \end{aligned}$$

$$\min_{i \in \{1,..,n-1\}} (U_{i+1} - U_i) \geq \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{\sigma}} \min_{i \in \{1,..,n-1\}} e^{-(u+\frac{X_i-\mu}{\sigma})^2/2} du \tag{1.5}$$

$$\geq \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{\sigma}} e^{-(u+\frac{X_n-\mu}{\sigma})^2/2} du \tag{1.6}$$

Note : Le calcul expérimental de la valeur exacte du minimum a montré que cette estimation de l'écart minimum est suffisamment bonne.

On va donc multiplier les réalisations de U par

$$\gamma = \frac{1}{\Delta^*} = \frac{1}{\frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{\sigma}} e^{-(u+\frac{X_n-\mu}{\sigma})^2/2} du} \tag{1.7}$$

De manière pratique, on utilise la fonction erf, en faisant le calcul suivant:

Soit $G_{max} = \dfrac{X_n - \mu}{\sigma}$,

$$\Delta^* = \frac{1}{\sqrt{2\pi}} \int_0^{\frac{1}{\sigma}} e^{-(u+\frac{X_n-\mu}{\sigma})^2/2} du = \frac{1}{\sqrt{2\pi}} \int_{\frac{G_{max}}{\sqrt{2}}}^{\frac{\frac{1}{\sigma}+G_{max}}{\sqrt{2}}} e^{-t^2}\sqrt{2} dt = \frac{1}{\sqrt{\pi}} \int_{\frac{G_{max}}{\sqrt{2}}}^{\frac{\frac{1}{\sigma}+G_{max}}{\sqrt{2}}} e^{-t^2} dt$$

$$= 0.5 \left(\frac{2}{\sqrt{\pi}} \int_0^{\frac{\frac{1}{\sigma}+G_{max}}{\sqrt{2}}} e^{-t^2} dt - \frac{2}{\sqrt{\pi}} \int_0^{\frac{G_{max}}{\sqrt{2}}} e^{-t^2} dt \right)$$

$$\Delta^* = 0.5 \left(erf\left(\frac{\frac{1}{\sigma} + G_{max}}{\sqrt{2}} \right) - erf\left(\frac{G_{max}}{\sqrt{2}} \right) \right) \tag{1.8}$$

1.6.3 Résultats et conclusion

On a testé les traces de distances de Hamming générées par une ECC de 112 bits (sexp112r1 [26]) avec l'Algorithme 2 du Chapitre 2 en prenant un exposant égal à l'ordre du point de base recommandé. Les modulis ont été randomisés. La transformation donne des résultats corrects pour 100000 calculs. La majorité des tests réussissent. Pour 1000000 de calculs, la majorité des tests échoue. Ces résultats sont reportés dans l'Appendice A. On a essayé pour des tailles de calculs plus grandes avec des résultats similaires. Le fait peut être justifié par la transformation discrète utilisant 16 à 20 bits permettant une valeur maximale de 65535 à 1048576. De plus il y a des trous dans les suites de valeurs obtenues.

Le problème réside surtout dans la difficulté d'interpréter les résultats. Les résultats restent trop généraux et ne rendent pas compte du niveau de sensibilité de la randomisation vis à vis des attaques par canaux auxiliaires. Par conséquent, on a abandonné l'idée d'utiliser ces tests et nous avons décidé d'utiliser principalement des méthodes d'attaques par canaux auxiliaires comme outils d'analyse pour évaluer la randomisation.

1.7 Calculs GPU (Graphics Processing Unit)

Suite à une approche d'analyse avec les outils d'attaque par canaux auxiliaires, il est apparu nécessaire de bien comprendre le comportement des distances de Hamming avec l'utilisation de la méthode des moduli aléatoires. La méthode la plus efficace pour comprendre ce comportement semble être la Template Attack et l'utilisation de l'estimateur du maximum de vraisemblance (MLE : Maximum Likelihood Estimator). L'idéal est d'avoir un calcul complètement exhaustif en connaissant la taille d'une clef et le nombre de configurations de moduli possibles. Il faut calculer une matrice de covariance et un

vecteur moyen pour i distances de Hamming et $2n$ moduli, ce qui fait

- 2^{i-1} suites de bits possibles.
- $\binom{2n}{n}$ configurations de moduli possibles,
- $\frac{i(i+1)}{2}$ coefficients pour chaque matrice de covariance, correspondant à une suite de bits (la matrice de covariance étant symétrique),
- i coefficients pour le vecteur moyen correspondant à une suite de bits.

Cela fait donc

$$\left(\frac{i(i+1)}{2} + i\right) \times \binom{2n}{n} \times 2^{i-1} \text{ calculs.}$$

De plus, la création du dictionnaire nécessite, pour chaque suite de bits, l'inversion de la matrice de covariance en faisant une décomposition LDLt et le calcul du déterminant (Cf. Section 2.5, 3.2.2 et 3.4.3). Vu la quantité de calcul, il est nécessaire de faire un calcul en parallèle pour obtenir des résultats rapidement en ayant le plus de paramétrages possibles sur le nombre de moduli et le nombre de traces d'observations pour avoir des statistiques fiables. Un travail conséquent de programmation C/CUDA [80] est à l'origine de l'automatisation de l'analyse grâce à une implémentation sur une carte Nvidia Tesla K40C. On peut ainsi faire le calcul des vecteurs moyens et des matrices de covariances pour de nombreuses clefs. Pour le calcul d'inversion de matrice et de factorisation LDLt, on utilise une méthode développée par [3] pour un ensemble de matrices de petite taille (Batch Computing). Seulement cette méthode très performante pour des matrices bien conditionnées peut s'avérer insuffisante pour des matrices qui sont proches de la singularité. Or on voudrait pouvoir utiliser ce type de matrice dans le cas de l'utilisation de l'estimateur du maximum de vraisemblance conditionnel de la Section 3.4 du Chapitre 3. On essaye donc d'améliorer le code de [3] utilisant la méthode Cuppen Divide & Conquer [31] développée pour le batch computing pour avoir une meilleure précision sur des matrices proches de la singularité.

1.8 Organisation et contribution de la thèse

On parlera d'analyse forte pour une analyse qui permet de retrouver la clef d'un système cryptographique et d'une analyse faible dans le cas où on élimine des clefs candidates. Le but de cette thèse est essentiellement la compréhension du comportement de l'aléa des distances de Hamming produites par un système cryptographique de type ECC quand on utilise une représentation RNS avec la méthode des moduli aléatoires. On s'intéresse aux attaques de type verticales par analyse de trace, les autres, horizontales ou par faute

entre dans un autre cadre. On peut se se référer aux articles [20, 28, 85, 10] pour les attaques horizontales. On se réfère à [17, 42] pour les attaques par fautes.

Le Chapitre 2 introduit les différentes notions nécessaires à la compréhension de ce document. Il introduit brièvement l'algorithme de multiplication modulaire (Algorithme de Montgomery pour RNS) qui a inspiré la méthode des moduli aléatoires. Puis il décrit l'algorithme qui génère les séquences de distances de Hamming nécessaires à notre analyse. Ensuite il montre quel niveau de résistance apporte la méthode des moduli aléatoires contre différentes attaques classiques comme DPA, CPA, DPA du second ordre [75] et MIA (Mutual Information Analysis)[19]. On apporte une compréhension de la distribution des distances de Hamming considérées comme des variables aléatoires. Suite à cela, on ajoute l'hypothèse gaussienne sur les distances de Hamming. On utilise alors le MLE et une analyse forte comme pour faire des Template Attacks pour avoir une compréhension fine du niveau d'aléa apporté par la méthode des moduli aléatoires.

Le Chapitre 3 est une suite naturelle des conclusions apportées par le Chapitre 2 sur l'hypothèse gaussienne. Si des attaques sont possibles avec le MLE, c'est qu'il existe peut-être des relations fortes entre les distances de Hamming considérées comme des variables aléatoires. La Section 3.2 cherche à quantifier ce niveau de dépendance des distances de Hamming. Ensuite en restant dans l'hypothèse gaussienne, on remarquera qu'il est possible de construire un type de DPA qu'on a appelé DPA square reposant sur la covariance au lieu de la moyenne comme dans la DPA classique. Mais cela reste très gourmand en traces d'observation d'autant que dans de nombreux protocoles utilisant une ECC, on utilise une clef qu'une seule fois. La Section 3.4 s'efforce de montrer qu'il existe de l'information sur peu de traces de distances de Hamming malgré la randomisation des moduli. Pour cela, on fait un MLE par un conditionnement sur l'une des distances de Hamming en utilisant une analyse faible.

Le dernier Chapitre 4 commence par introduire brièvement les choix algorithmiques qui ont été faits pour résoudre les problèmes d'inversion de matrices de covariance (symétriques définies positives) de la Section 2.5 et l'analyse des relations fortes entre les Hamming dans la Section 3.2. On utilise ici des outils de Graphic Processing Unit (GPU) sur un très grand nombre de matrices de petite taille. On parlera de Batch Computing. La méthode LDLt présentée au début de ce chapitre s'est avérée insuffisante pour résoudre complètement le problème du MLE conditionné présenté dans la Section 3.4. On présente le travail sur l'amélioration d'un code de diagonalisation de matrice tridiagonale utilisant le principe de Diviser pour Régner (Divide & Conquer) développé par Lokmane Abbas-Turki et Stéphane Graillat dans [3]. On présente une généralisation de ce code, des optimisations en temps de calcul et une amélioration de la robustesse des calculs en simple précision pour des matrices de taille inférieure à 32.

Chapter 2

Resilience of randomized RNS arithmetic with respect to side-channel leaks of cryptographic computation

Sommaire

2.1 Introduction (français)

Ce chapitre est le résultat d'une publication dans la revue IEEE Transactions on Computers en 2019 sous le titre "Resilience of randomized RNS arithmetic with respect to side-channel leaks of cryptographic computation" [30].

En cryptographie asymétrique, quel que soit le niveau de sophistication des mathématiques validant la robustesse du modèle, le passage à l'implémentation reste un point très délicat. Une mise en œuvre naïve peut sérieusement compromettre la sécurité d'un système.

L'arithmétique randomisée est une solution possible pour améliorer la sécurité. Mais jusqu'à présent, aucune étude ne précisait la qualité de l'aléa généré par l'arithmétique randomisée. On veut donc combler ce manque par le biais de cette étude, en fournissant des éléments permettant de mesurer cette qualité.

Les fuites sont principalement dues aux variations des poids de Hamming [70], en d'autres termes, à la distance de Hamming des états successifs. On suppose le pire des scénarios : une personne mal intentionnée aurait accès aux mesures de distance de Hamming sans aucun bruit matériel supplémentaire. On se place clairement du côté de la protection en créant ce bruit, intrinsèquement lié à l'arithmétique, indépendamment des spécifications du support hébergeant la mise en œuvre.

La seule faiblesse de cette approche est le tirage au sort de la base qui, si elle est corrompue, compromet la sécurité. Par exemple, s'il existe une graine avec un défaut d'entropie pour un générateur supposé de bonne qualité [51], ou inversement, un générateur faible avec une graine de bonne qualité [21]. Dans cette étude, on ne considère que les attaques par observation. Par conséquent, les attaques physiques sur le générateur aléatoire ne sont pas traitées.

On trouve dans la littérature différentes manières d'utiliser les fuites observées: généralement DPA ou CPA [24, 61], ou d'autres méthodes plus récentes telles que DPA de second ordre [75], template attacks [27] et Mutual Information Analysis (MIA) [19]. Pour un aperçu sur différentes attaques et contre-mesures, on renvoie le lecteur à [41].

Les transitions d'état dépendent clairement de la représentation des données qui peuvent être conjecturées par une personne malveillante. La randomisation dans un système de représentation arithmétique garantit que les calculs utilisent différentes représentations d'une exécution à l'autre. Cela réduit considérablement les prévisions sur les transitions d'état. Dans cette thèse, on se concentre sur la représentation RNS basée sur le théorème chinois des restes. Chaque valeur est connue par ses résidus sur un ensemble de nombres premiers entre eux qui représente la base RNS.

Les auteurs de [18, 54] ont suggéré une randomisation utilisant des isomorphismes de courbe comme contre-mesure contre les templates attacks sur ECDSA. Ce travail a été prolongé en 2016 [39] sur les attaques visant le doublement des points sur les courbes elliptiques dans «mbed TLS». Par conséquent, une arithmétique RNS randomisée offre une bonne opportunité de randomiser indépendamment de l'algorithme cryptographique utilisé. Ainsi, les méthodes présentées dans ce chapitre conviennent à tout système cryptographique tel que RSA, ECC, Euclidean Lattice ou autres. Le RNS est évolutif et peut être adapté aux tailles de clef qui pourraient augmenter en fonction de la progression de la cryptanalyse. De plus, les calculs RNS peuvent être efficacement parallélisés [9]. Pour plus d'informations sur les avantages du RNS, on se réfère à [48].

On présente ici une étude établissant le lien entre le nombre d'éléments n de la base participant au tirage aléatoire et la taille S de l'échantillon nécessaire pour avoir une information exploitable. On utilise une courbe ECC de 112 bits ECCsecp112r1 [26], essentiellement pour illustrer les résultats et conjecturer que $S = O((2n)!/(n!)^2)$. Les résultats sont similaires avec les courbes d'Edwards de 255 bits [22] ou ECCsecp256r1 [26]. Dans toute l'étude, on utilise des moduli de 32 bits sachant que l'on a, pour l'instant, pas constater de différence du comportement aléatoire sur des moduli de 8, 16 ou 32 bits.

La présentation de ce chapitre est la suivante : dans la Section 2.3, nous introduisons brièvement la randomisation des moduli dans la représentation RNS, l'algorithme de Montgomery appliqué à ECC et l'objectif de la randomisation. La Section 2.4 explique les principales raisons pour lesquelles la résilience d'un système devrait plutôt se concentrer sur environ 10 distances de Hamming successives. La Section 2.5 rappelle les notions sur l'estimateur de vraisemblance maximale (MLE) et étudie la taille S des observations nécessaires à la réalisation d'une analyse.

2.2 Introduction

In asymmetric cryptography, whatever the level of sophistication of the mathematics validating the robustness of the model, the transfer to implementation remains a very delicate point. A naive implementation can seriously compromise the security of a system.

Randomized arithmetic is a possible solution to improve the security. But until now, we had no precise studies on the quality of the randomness generated by randomized arithmetic. We therefore want to fill this gap through this study, by giving elements to measure this quality.

Leaks are mainly due to the variations of the Hamming weights of successive execution states [70], in other words, to the Hamming distance of successive states. We assume the worst case scenario: a malicious person would have access to Hamming distances measurements without any added material noise. We place ourselves clearly on the side of protection by creating this noise intrinsically linked to arithmetic, independently of the specifications of the support hosting the implementation.

The only weakness of this approach is the random choice of the base which, if it is corrupted, makes randomization obsolete. For example, if there is a seed with an entropy defect for a supposedly good quality generator [51], or vice versa a weak generator with a good quality seed [21]. In our study, we consider only attacks by observation, thus physical attacks on the random generator are not tackled.

We find in literature different ways of using the observed leakages: usually DPA or CPA [24, 61] or other more recent methods like second-order DPA [75], template attacks [27] and Mutual Information Analysis (MIA) [19]. For a survey on different attacks and countermeasures, we refer the reader to [41].

The state transitions depend clearly on the data representation which can be an assumption made by a malicious person. The randomization with respect to an arithmetic system ensures that the computations use different representations from one execution to another. This reduces significantly predictions on the state transitions. We focus on RNS representation based on the Chinese Remainder Theorem. Each value is known by its residues over a set of co-prime numbers which represent the RNS base. The authors of [18, 54] suggested randomization using curve isomorphisms as counter measure against template attacks on ECDSA. This work was extended in 2016 [39] on attacks on doubling point operation on elliptic curves in "mbed TLS". Therefore, a randomized RNS arithmetic offers a good opportunity to randomize independently from the cryptographic algorithm used. Thus, the methods presented are suitable for any cryptosystem such as RSA, ECC, Euclidean Lattice or others. RNS is scalable, and can be adapted to key sizes which could increase in function of the cryptanalysis progress. Furthermore, RNS computations can be efficiently parallelized [9]. For further benefits of RNS, we refer to [48].

In [14], the authors showed that we can draw randomly a RNS base from a set of

moduli, to randomize an execution with a small cost. Since their publication, this work was used and cited in different papers [48, 83, 63, 43, 95]. To our knowledge, no one established a statistical study of the randomness behavior of such approach, and what kind of protection it can get.

We wish to fill this gap. We present here a study establishing the link between the number of elements n of the base participating in the draw and the size S of the sample necessary for exploitable information. We use 112 bits ECC curve essentially to illustrate the results and conjecture that $S = O\left(\frac{(2n)!}{(n!)^2}\right)$. The results are quite similar when dealing with Edwards curves of 255 bits [22] or ECCsecp256r1 [26]. Throughout the study, 32-bit moduli are used knowing that, for the moment, we have not noticed any difference in the random behavior with 8, 16 or 32-bit moduli.

The layout of this chapter is as follows: in Section 2.3, we briefly introduce the moduli randomization of the RNS representation in the Montgomery algorithm applied for ECC and the goal of randomization. Section 2.4 explains the main reasons why the resilience of a system should rather focus on about 10 successive Hamming distances. Section 2.5 recalls the Maximum Likelihood Estimator (MLE) and studies the size S of observations needed to perform the analysis.

2.3 Key elements of the study : Montgomery Power Ladder (MPL) using RNS representation applied to ECC and randomization.

In Section 2.3.1, we briefly explain the randomization technique based on RNS representation for Montgomery multiplication. Section 2.3.2 clarifies the way the Hamming distances are computed through the successive steps of MPL. Because ECC uses the main arithmetic operations as additions and multiplications, this makes the randomization efficient. We give in Section 2.3.3 some elements about our evaluation of the randomness.

2.3.1 Montgomery for RNS modular multiplication

In [77], P. Montgomery introduced an algorithm of modular multiplication to avoid trial division by large numbers. The RNS version of this algorithm is the starting point of the randomization used in [5]. We summarize this method with a presentation that is quite similar to the one in [12].

We denote $|a|_m = a \ mod \ m$ and $[\![1, n]\!] = \{1, ...n\}$. When a and m are coprime, we set $|a|_m^{-1} = a^{-1} \ mod \ m$ to be the inverse of a modulo m. Introducing the RNS base $\mathcal{B}_n = \{m_1, ..., m_n\}$ of pairwise coprime moduli, the Chinese Remainder Theorem ensures

the existence of a ring isomorphism between $\mathbb{Z}_M$ and $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$ with $M = \prod_{i=1}^{n} m_i$. Thus, for any positive integer X strictly smaller than M

$$X = \left(\sum_{i=1}^{n} x_i \left| M_i \right|_{m_i}^{-1} M_i \right) \bmod M \tag{2.1}$$

with $x_i = |X|_{m_i} = X \bmod m_i$ and $M_i = M/m_i$.

Let $\widetilde{\mathcal{B}_n} = \{\widetilde{m}_1, ..., \widetilde{m}_n\}$ be another RNS base of pairwise coprime moduli that are also coprime with $\mathcal{B}_n$, i.e. m_i and $\widetilde{m_j}$ are coprime for each $i \in [\![1, n]\!]$ and $j \in [\![1, n]\!]$. For a number X that is strictly smaller than $\widetilde{M} = \prod_{i=1}^{n} \widetilde{m}_i$, we use the notation $\{\widetilde{x}_1, ..., \widetilde{x}_n\}$ for the decomposition of X on $\widetilde{\mathcal{B}_n}$. Using these notations as well as the standard definition of the usual RNS operations, Algorithm 1 presents the modular multiplication. Addition $+_{RNS}$, multiplication $\times_{RNS}$ and opposite $(-X)_{RNS}$ are explained in [12].

Algorithm 1 RNSn modular multiplication

Require:

A residue base $\mathcal{B}_n = \{m_1, ..., m_n\}$ where $M = \prod_{i=1}^{n} m_i$

A residue base $\widetilde{\mathcal{B}_n} = \{\widetilde{m}_1, ..., \widetilde{m}_n\}$ where $\widetilde{M} = \prod_{i=1}^{n} \widetilde{m}_i$ with $gcd(M, \widetilde{M}) = 1$

A modulus N expressed in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}_n}$ with $gcd(N, M) = 1$ and $gcd(N, \widetilde{M}) = 1$, $0 < (n+2)^2 N < M$ and $0 < (n+2)^2 N < \widetilde{M}$

An Integer A expressed in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}_n}$

An Integer B expressed in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}_n}$ with $AB < NM$

Ensure: An integer R expressed in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}_n}$ such that $R \bmod N = ABM^{-1} \bmod N$

function

$Q \leftarrow ((-(A \times_{RNS} B))_{RNS}) \times_{RNS} N^{-1}$ in base $\mathcal{B}_n$

Extension of Q from $\mathcal{B}_n$ to $\widetilde{\mathcal{B}_n}$ (Cf. Appendix 2.A)

$R \leftarrow (A \times_{RNS} B +_{RNS} Q \times_{RNS} N) \times_{RNS} M^{-1}$ in base $\widetilde{\mathcal{B}_n}$

Extension of R from $\widetilde{\mathcal{B}_n}$ to $\mathcal{B}_n$ (Cf. Appendix 2.A)

end function

Before each modular exponentiation, we perform a random selection of n moduli $\{m_1, ..., m_n\}$ among $\{\mu_1, .., \mu_{2n}\}$ for base $\mathcal{B}_n$. The remaining moduli forms the base $\widetilde{\mathcal{B}_n}$. This random choice is based on a standard drawing without replacement.

Since many modular multiplications are needed in ECC or RSA, one should consider the Montgomery form of A and B as inputs to Algorithm 1. This trick allows to circumvent dealing with $ABM^{-1} \bmod N$ as an output. We recall that the Montgomery form of A is given by $AM \bmod N$. As proposed in [14], once $M\widetilde{M} \bmod N = \prod_{i=1}^{2n} \mu_i \bmod N$ is known, the Montgomery form can be obtained with Algorithm 1. It is applied to A and

$M\widetilde{M}$ mod N provided that we exchange $\mathcal{B}_n$ and $\widetilde{\mathcal{B}}_n$, since

$$A \times |M\widetilde{M}|_N \times \widetilde{M}^{-1} = AM \bmod N.$$

To recover the appropriate expression, we need to perform a final pass in Algorithm 1 to multiply 1 and $(AM)(BM)M^{-1}$ mod N, yielding

$$|(AM)(BM)M^{-1}|_N \times |1|_N \times M^{-1} = AB \bmod N.$$

We point out that pre-computing $|M\widetilde{M}|_N$, proposed in [14] instead of $|M^2|_N$, is justified by the randomization procedure. We refer to Appendix 2.A for extensions used in the RNS modular multiplication.

Remark : RNS has become a standard for randomization, especially since there is a great diversity of moduli. Moreover, with Montgomery multiplication algorithm [77], the Montgomery factor strengthens the random behaviour of Hamming distances.

2.3.2 Example on scalar multiplication on elliptic curve

We target the scalar product of a point of an elliptic curve. This operation is found in several Elliptic Curves Cryptographic protocols: encryption, signature, etc... [71, 6]. It offers the benefit of being quite complete in terms of arithmetic operations. Several scenarii are possible: the secret key is used several times as for decryption in PSEC-KEM [79], or only once as the random secret in ECDSA [6, 10]. Different attacks are then available in the first case via CPA or DPA, in the second case via learning.

Anyway, the secret is in the form of a scalar K applied to a point G of an elliptic curve E. Thus the computation of $[K]G$ must remain secret by being leak resistant.

To compute $[K]G$ on an elliptic curve and protect against Simple Power Analysis (SPA)[52], we use the binary version of MPL detailed in Algorithm 2. First, we compute both the Montgomery Form A_0 of G and A_1 the double of A_0. Then, if the bit value b_i of K is one, A_0 is added to A_1 memorized in A_0 and A_1 is doubled. Otherwise, A_1 is added to A_0 memorized in A_1 and A_0 is doubled.

The elliptic curve domain of $E(\mathbb{F}_N)$ is defined by a finite field $\mathbb{F}_N$ with N a prime number, two elements a and $b \in \mathbb{F}_N$, an equation $E : y^2 \equiv x^3 + ax + b \bmod N$, $G(x_G, y_G)$ a base point of $E(\mathbb{F}_N)$ and n_G is a prime number that is the order of G on $E(\mathbb{F}_N)$.

In our implementation, we use the elliptic curves recommended by Certicom [26] employing Jacobian coordinates to avoid the division and reduce computations [73, 29, 50]. Each point is defined by three Jacobian coordinates $(X; Y; Z)$ with the affine representation $(X/Z^2; Y/Z^3)$.

Associated to the equation E is $Y^2 = X^3 + aXZ^4 + bZ^6$, $(X; -Y; Z)$ is the inverse

of $(X;Y;Z)$ and the infinite point is chosen to be equal to $(1;1;0)$. The addition and doubling operations can be found in [73].

Algorithm 2 shows exactly at which step of MPL we choose to compute the Hamming distances for ECC in RNS. We remind that M is the product of the moduli of base $\mathcal{B}_n$.

Algorithm 2 Montgomery Powering Ladder for ECC in RNSn

Require:
A point $G = (X;Y;1)$ in RNS representation
A key $K = 2^{d-1}b_0 + 2^{d-2}b_1 + ... + 2b_{d-2} + b_{d-1}$
Ensure:
$A_0 = [K]G$
$(H_i)_{i\in\{0,..,d-1\}}$, the Hamming distances
function
Choose a random base permutation
$A_0 = (|XM|_N, |YM|_N, |M|_N)$, *Montgomery form of* G
$A_1 = [2]A_0$
H_0=**Hamming weight of** (A_0, A_1)
for $i = 1$ to $d-1$ **do**
$A'_0 = A_0$ and $A'_1 = A_1$
$A_{\overline{b_i}} = A_{\overline{b_i}} + A_{b_i}$
$A_{b_i} = [2]A_{b_i}$
$H_i =$ **Hamming distance between** (A_0, A_1) **and** (A'_0, A'_1)
end for
Result $A_0 = (|X'M|_N, |Y'M|_N, |Z'M|_N)$ *in Montgomery form*
Return to the Non-Montgomery form [16]
$A_0 = (|X'|_N, |Y'|_N, |Z'|_N)$
end function

2.3.3 Goal of randomization

The goal of randomization according to the involved moduli is to make as unpredictable as possible the secret from the Hamming distance (number of different bits) between two consecutive states. As sketched on Figure 2.1, we distinguish two randomness sources given by both the configuration of moduli C and the key K. K is also considered as a random variable in our analysis.

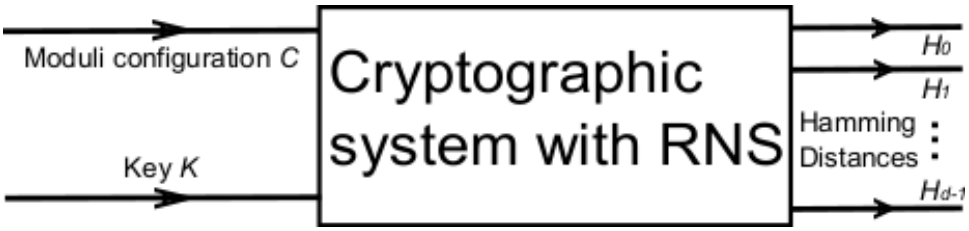


Figure 2.1 – Hamming distances with respect to randomness sources

$H = (H_0, ..., H_{d-1})$ are the Hamming distances observed through the execution of a cryptographic algorithm. For example, we study here MPL algorithm associated to ECC. The random vector $H = (H_0, ..., H_{d-1})$ can be considered as a deterministic function of the couple of random variables (K, C). Consequently, the link between K and H is difficult to establish when the noise generated by C is significant. The perfect noise would be the one that mimics an independence between K and H. Without loss of generality, let us denote informally:

- $L(H, K)$ the joint distribution of (H, K),
- $L(H|K)$ the conditional distribution of H given K,
- $L(H)$ and $L(K)$ the marginal distributions of H and K.

The perfect noise must fulfill

$$L(H, K) = L(H|K)L(K) = L(H)L(K) \tag{2.2}$$

or equivalently $L(H|K) = L(H)$, meaning that K must not provide any information on H. Although (2.2) is impossible to obtain because each H_i as a function of K and C, will always depend on K, it tells us that the independence of the coordinates of $H = (H_0, ..., H_{d-1})$ is not necessary to have a perfect noise.

We study the distinguishability between $L(H|K)$ and $L(H|K')$ $(K \neq K')$ with respect to n, the number of moduli in the RNS representation denoted by RNSn. Nevertheless, because studying $L(H|K)$ for the whole vector H is computationally barely possible, we develop a strategy based on few Hamming distances that provide the most valuable information on the key K. Unlike DPA and CPA that use only the marginal information associated to each step, our conditional strategy combined with MLE uses a cross-information based on about 10 Hamming distances. Therefore, our main contribution can be summarized by :

1. We show that a cryptographic system has to be resilient with respect to the cross-information.
2. With randomization CPA [24] fails and we explain why MIA [19] is inefficient when applied to randomization.
3. The DPA results are inconsistent with the level of randomization and we show that second-order DPA does not overcome this inconsistency.
4. We used the Maximum Likelihood Estimator (MLE) to measure the level of information leakage. We show that the information leakage, is monotonous, unlike the DPA.

5. We propose a number of randomized moduli to protect ECC from a template attack.

In real implementation, physical hardware noises will be added to the noise generated by the random selection of RNS bases. This makes harder the detection of the Hamming distances. In our study, as we want to measure the impact of the randomization of the RNS bases, we consider only the ideal case when the Hamming distances are known. We focus on ECC as it is well suited for the evaluation of moduli randomization countermeasure because its arithmetic is dominant and performing DPA can be efficient. Indeed, all the probability tools like: total variation, covariance matrix, asymptotic error of Monte Carlo (cf. Appendix 2.B) can be reused in the same fashion for other systems. We point out also that our work is different from [83] where the authors study protection against memory addressing attack. Indeed, our study focuses on the resilience of randomization when the system is supposed to be protected against memory addressing attacks. We use 112-bit ECC curve essentially to illustrate the results and conjecture that $S = O((2n)!/(n!)^2)$. The results are quite similar when dealing with Edwards curves of 255 bits [22] or ECCsecp256r1 [26].

2.4 A conditional strategy and limitations of CPA, DPA, second-order DPA and MIA

With randomized RNS, we verify that Hamming distance has a Gaussian distribution (see Figure 2.4). Unfortunately, most of the statistic tests, like NIST's ones [7], evaluate uniform distribution. Therefore, we mainly use methods from side-channel attacks as tools for assessing randomization.

The randomization of moduli effectively generates noisy data. Consequently, we should target the most sensitive values that provide exploitable information on the secret key K. When the latter fact is studied in Section 2.4.1, Section 2.4.2 shows that CPA is impossible to use and the size S of observations to achieve a DPA is not monotonous with respect to the number of moduli. Section 2.4.3 discusses the adaptation of other methods to RNS randomization.

2.4.1 Sufficient information and conditional strategy

From now on, we denote S the sample size of simulations. The following three properties of Hamming distances are presented:

α) For fixed choice of moduli and the secret is a random variable, the first Hamming distances are the one that provide the strongest information (dependence) on the secret K.

β) For fixed choice of moduli and the secret is a random variable, the correlation between Hamming distances decreases significantly with respect to the gap l that separates H_i and $H_{i\pm l}$.

γ) Under randomization of moduli and a fixed secret, each Hamming distance H_i has a normal distribution. This property shall not make absurd the assumption that the whole vector H is Gaussian.

Consequently, the first bits of a secret key K can be deduced from the information extracted from the first Hamming distances. Once these few first bits are known, the following ones can be found step by step, until we recover the whole key. It is a Divide and Conquer method. We make an hypothesis on a a part of the key then we reject or accept it with a statistic hypothesis test.

At each step of Algorithm 2 (MPL), we evaluate the dependency of the random variables K and H_i. The values of K and H_i are integers belonging to the intervals $I = [0, 2^p[$ $(p \leq d)$ and $\mathcal{H}^i = [min(H_i), max(H_i)]$. In order to reduce the complexity of computations and increase the Monte Carlo accuracy (cf. Appendix 2.B), we use appropriate subdivisions of those intervals I and $\mathcal{H}^i$ respectively into $2^{p'}$ and q sub-intervals. The details of the subdivisions are given in Appendix 2.C.

Monte Carlo is used for the estimation of probability terms involved in the Total Variation to Independence (TVI) [64] expression (2.3) given below. The Law of Large Numbers ensures the convergence and the Central Limit Theorem provides its rate. Consequently, we quantify the accuracy thanks to the 95% confidence interval, with 95% chance of having at most a 10% relative error. The relative error is defined as the width of the confidence interval normalized by the estimated value.

As introduced informally above, the perfect noise must fulfill $L(H, K) = L(H)L(K)$ (2.2) which yields

$$P\left(H_i \in \mathcal{H}_j^i, K \in I_k\right) = P\left(K \in I_k\right) \times P\left(H_i \in \mathcal{H}_j^i\right).$$

Thus the dependence is quantified through the distance between the probability of the product $P\left(H_i \in \mathcal{H}_j^i, K \in I_k\right) = P\left(K \in I_k\right) \times P\left(H_i \in \mathcal{H}_j^i | K \in I_k\right)$ and the product of probabilities $P\left(K \in I_k\right) \times P\left(H_i \in \mathcal{H}_j^i\right)$. We compute this distance using TVI [64] given by

$$\mathrm{TVI}_i = \frac{1}{2} \sum_{k=0}^{2^{p'}-1} \sum_{j=0}^{q-1} P\left(K \in I_k\right) \left| P\left(H_i \in \mathcal{H}_j^i\right) - P\left(H_i \in \mathcal{H}_j^i | K \in I_k\right) \right|. \tag{2.3}$$

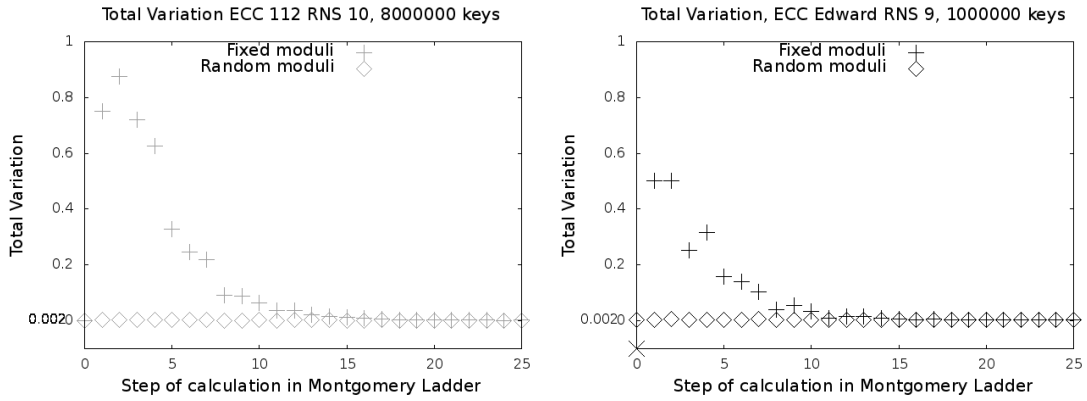


Figure 2.2 – Total variation as a function of the calculation step.

The value of $P\left(K \in I_k\right)$ is known since we draw uniformly an integer value on $[0, 2^p[$. However, the value $P\left(H_i \in \mathcal{H}_j^i | K \in I_k\right)$, and subsequently $P\left(H_i \in \mathcal{H}_j^i\right)$, is approximated using Monte Carlo simulation. For more mathematical details on Monte Carlo, we refer the reader to [53].

In Figure 2.2, we calculate TVI_i for each step in MPL either for a fixed choice of moduli or when they are randomized. When the moduli configuration is fixed we draw only independent keys $\{K^l\}_{1 \leq l \leq S}$. When the moduli are randomized we draw independent couples $\{(K^l, C^l)\}_{1 \leq l \leq S}$ of keys and moduli configurations. The Monte Carlo approximation is then given either by

$$P\left(H_i \in \mathcal{H}_j^i, K \in I_k\right) \approx \frac{1}{S} \sum_{l=1}^{S} 1_{\left\{H_i(K^l) \in \mathcal{H}_j^i \bigcap K^l \in I_k\right\}}$$

or by

$$P\left(H_i \in \mathcal{H}_j^i, K \in I_k\right) \approx \frac{1}{S} \sum_{l=1}^{S} 1_{\left\{H_i(K^l, C^l) \in \mathcal{H}_j^i \bigcap K^l \in I_k\right\}}.$$

We simulate with $S = 8 \times 10^6$ or $S = 10^6$ in order to have a sufficiently accurate results to compute TVI. We use the random number generator proposed in [62]. This choice is appropriate computationally and statistically for Monte Carlo simulation.

According to Figure 2.2, randomizing moduli reduces effectively TVI for all Hamming distances. Also, according to Figure 2.2, we see clearly that TVI almost vanishes for Hamming distances of a rank bigger than 10 which confirms property α). This observation can be explained by the fact that the first ~ 10 Hamming distances depend strongly on the first ~ 10 bits of the key. Because a large choice of combinations of bits can produce the same value on each $\{H_i\}_{i>10}$, the dependence between $\{H_i\}_{i>10}$ and the key is reduced significantly.

Regarding property β), we approximate the covariance of each couple of Hamming distances with a Monte Carlo simulation on keys for a fixed choice of moduli:

$$\mathrm{Cov}(H_i, H_j) \approx \frac{1}{S}\sum_{l=1}^{S} H_i(K^l)H_j(K^l) - \left[\frac{1}{S}\sum_{l=1}^{S} H_i(K^l)\right]^2.$$

We get the results presented in Figure 2.3 for the covariance H_1, H_4, H_8 and H_{10} with the other Hamming distances. In Figure 2.3, the fact that $|\mathrm{Cov}(H_i, H_{i\pm l})|_{l\geq 0}$ decreases with respect to the gap l is due to the difference in term of bits that separates H_i and $H_{i\pm l}$.

The importance of the covariance comes from the property γ. Indeed, in a multivariate Gaussian vector, each two coordinates are independent if and only if their covariance is equal to zero. A Chi Square test does not disapprove the Gaussian distribution of each Hamming distance H_i when moduli are randomized. We also present in Figure 2.4 an histogram associated to H_{10} that shows a bell-shaped distribution. This property does not contradict the assumption that the whole vector H is Gaussian. For more mathematical details on multivariate Normal distribution, we refer the reader to [53].

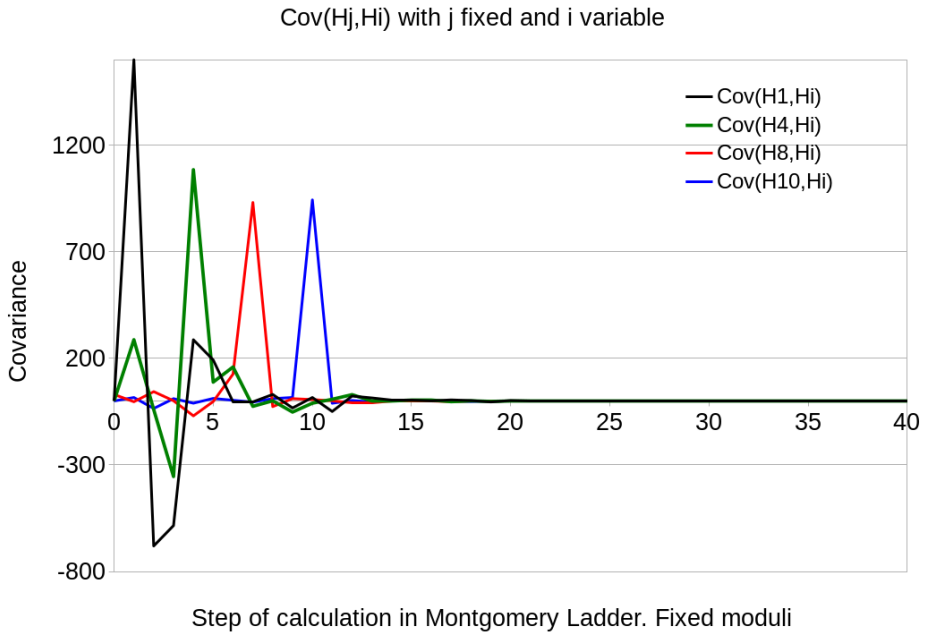


Figure 2.3 – $Cov(H_j, H_i)_{j=1,4,8,10}$.Fixed moduli, ECC112 RNS10.

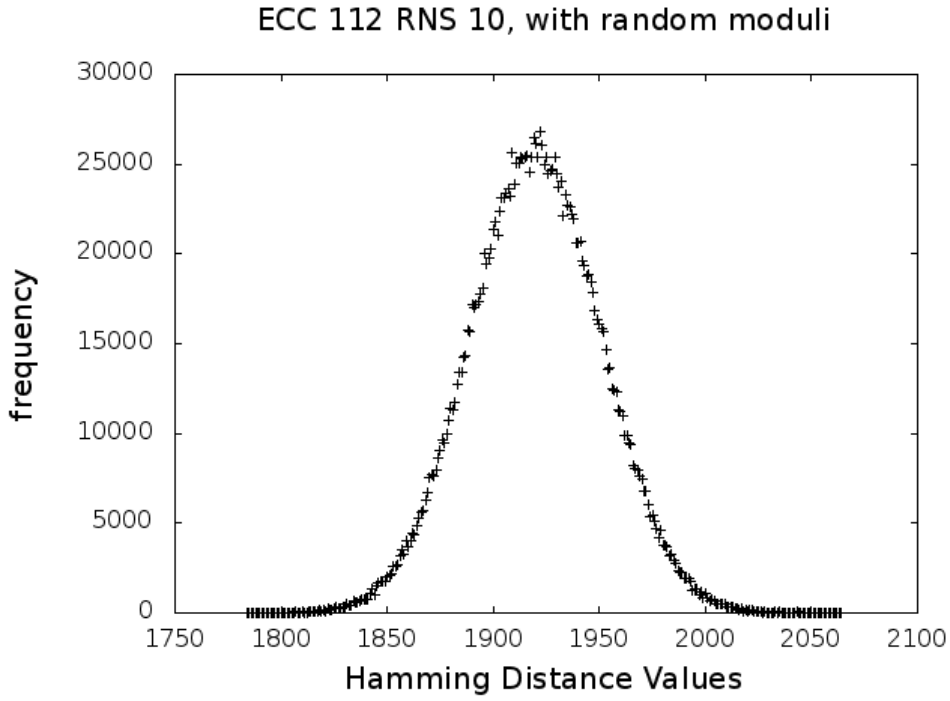


Figure 2.4 – Frequency of H_{10}, 2×10^6 computations. Random moduli and K random

2.4.2 Unreliable CPA and inconsistent DPA

The essential result of Section 2.4.1 is that exploitable information leakage should be based on the first ~ 10 computation steps. Once the bits associated to some of these steps are known, one should fix them to continue with the following ~ 10 computation steps and so on. This conditional strategy (conditioning on the bits found) not only uses the marginal information of each step but must use the cross-information of the ~ 10 successive steps. Indeed, according to Figure 2.2, the first ~ 5 Hamming distances have almost the same strength of dependence on the secret key K and we waste information if we use only the marginal distributions. Nevertheless, CPA and DPA are not conceived to take advantage of this cross-information.

We study the effect of the randomized moduli on Hamming distances. Our approach can be justified by the leakage models introduced in [70] between the power consumption and the Hamming distances. In this study, even with fixed K, the randomization of moduli generates different representations which induces different Hamming distances. We explore the link between the randomization of representation and Hamming distances, independently of the implementation support.

2.4.2.1 Methodology

In the following, we denote C^l the lth realization of moduli configuration. We denote $K = \sum_{l=0}^{d-1} b_l 2^{d-1-l}$ the key of d bits to guess. Assuming that we know the j first bits, we denote $K'_j = \sum_{l=0}^{j-1} b_l 2^{d-1-l} + \sum_{l=j}^{d-1} 2^{d-1-l}$ a key of d bits which begins like K and have only ones after. At each step i, $H_i(K, C^l)$ is an observation associated to the real key K and $H_i(K'_j, C^{l+S})$ is the simulation associated to the guessed one K'_j. The term $+S$ in C^{l+S} expresses the independence between the moduli configurations $\{C^l\}_{1\leq l\leq S}$ and $\{C^{l+S}\}_{1\leq l\leq S}$. This results from the independence of the whole sequence $C = \{C^1, C^2, \ldots, C^S, C^{S+1}, \ldots, C^{2S}\}$.

Using the discrepancy induced by $\left\{H(K, C^l)\right\}_{l=1}^{S}$ and by $\left\{H(K'_0, C^{l+S})\right\}_{l=1}^{S}$, we determine the position j_1 of the first zero in the binary expansion of K. Then, we do the same for K and K'_{j_1} which provides the position j_2 of the second zero and so on till we find K. For example, when $K = 11101101110_2$:

- We get $j_1 = 3$ from $K = 111\,0\,1101110_2$ and $K'_0 = 11111111111_2$.
- We get $j_2 = 6$ from $K = 111011\,0\,1110_2$ and $K'_3 = 11101111111_2$.
- We get $j_3 = 10$ from $K = 1110110111\,0\,_2$ to $K'_6 = 11101101111_2$.

Formerly speaking, denoting $H_i^l(K) = H_i\left(K, C^l\right)$, $H_i^{l+S}\left(K'_{j_{p-1}}\right) = H_i\left(K'_{j_{p-1}}, C^{l+S}\right)$ and setting $\min(\emptyset) = d$ then

$$\begin{aligned} j_0 &= 0 \\ j_p &= \min\left\{i > j_{p-1}, \left| f_i\left(\left\{H_i^l(K), H_i^{l+S}\left(K'_{j_{p-1}}\right)\right\}_{l=1}^{S}\right)\right| > \mathcal{T}\right\} \end{aligned} \tag{2.4}$$

and the key is recovered when $j_p = d$ which means that $K = K'_{j_{p-1}}$. $\mathcal{T} > 0$ is the distinguishing threshold and f_i is a distinguisher defined in Section 2.4.2.2 for CPA and DPA. In the worst case ($K = 2^{d-1}$), this method requires to check $d-1$ hypothesis instead of checking 2^{d-1} hypothesis.

With this approach, we search the number of moduli needed to make the key K indistinguishable from K'_j for a DPA or CPA. In classic DPA [61, 70], for a key hypothesis and a set of messages, messages are classified in two sets: those that give a high Hamming weight and those that give low Hamming weight. Thus, the leaks are classified and the difference of their average produces a peak if the hypothesis is correct. The randomization of moduli does not allow to classify Hamming distances with respect to messages. Consequently, the average on messages has been replaced by the average on

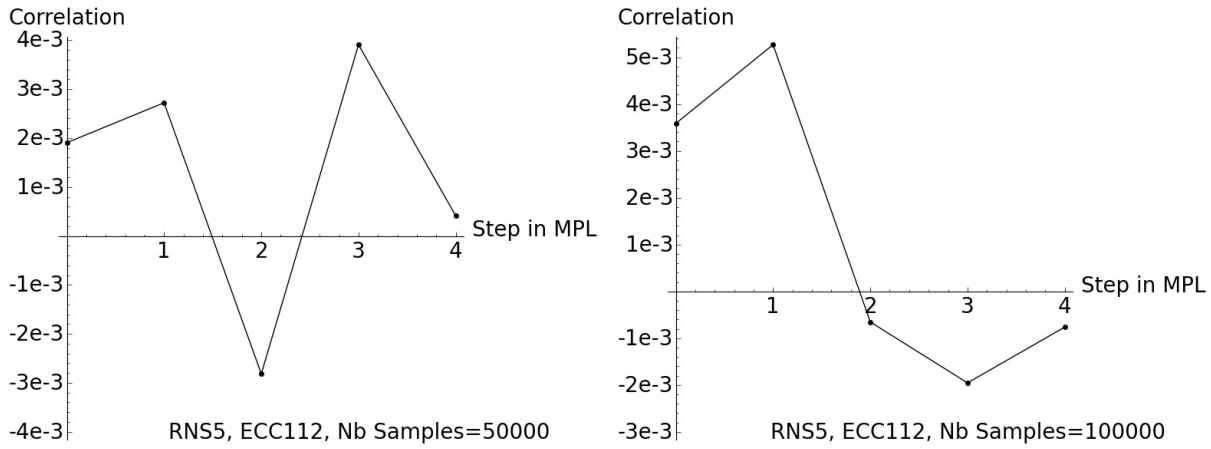


Figure 2.5 – Correlation between `0xdeeefbf7` and `0xffffffff`, 50000 and 100000 traces. Nothing appears at bit 2 and the correlations are too small. RNS5

configurations of moduli:

$$\overline{H}_i(K, C) = \frac{1}{S}\sum_{l=1}^{S} H_i(K, C^l) \quad \text{and} \quad \overline{H}_i(K'_j, C) = \frac{1}{S}\sum_{l=1}^{S} H_i(K'_j, C^{l+S}).$$

We point out that we are not allowed to classify Hamming distances with respect to moduli configurations. Indeed, an attacker does not control the moduli configuration of the system.

2.4.2.2 Results

A CPA on Hamming distances is based on the correlation ξ_i (2.5) that exists at step i between observations $H_i(K, C^l)$ on the real key K and simulations $H_i(K'_j, C^{l+S})$ on the guessed one K'_j which yields

$$\xi_i = f_i\left(\left\{H_i^l(K), H_i^{l+S}\left(K'_{j_{p-1}}\right)\right\}_{l=1}^{S}\right) \tag{2.5}$$

$$= \frac{\sum_{l=1}^{S}\left[H_i(K, C^l) - \overline{H}_i(K, C)\right]\left[H_i(K'_j, C^{l+S}) - \overline{H}_i(K'_j, C)\right]}{\sqrt{\sum_{l_1=1}^{S}\left[H_i(K, C^{l_1}) - \overline{H}_i(K, C)\right]^2 \sum_{l_2=1}^{S}\left[H_i(K'_j, C^{l_2+S}) - \overline{H}_i(K'_j, C)\right]^2}}. \tag{2.6}$$

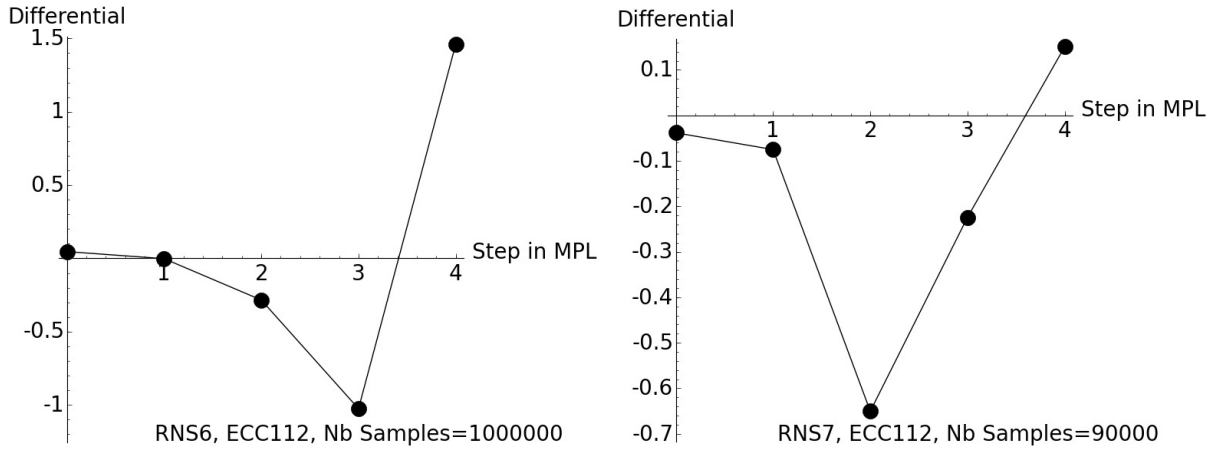


Figure 2.6 – Differencial between `0xffffffff` and `0xdeeefbf7` with respectively 1000000 and 90000 traces. A jump appears for the bit 2 for RNS7 as we expected but the jump is not obvious for RNS6 and we needed more traces to have this little jump. RNS6 and RNS7

where f_i is a sample correlation estimator :

$$f_i\left(X,Y,S\right)=\frac{\sum_{l=1}^{S}\left[X^l-\overline{X}\right]\left[Y^l-\overline{Y}\right]}{\sqrt{\sum_{l_1=1}^{S}\left[X^{l_1}-\overline{X}\right]\sum_{l_2=1}^{S}\left[Y^{l_2}-\overline{Y}\right]}} \tag{2.7}$$

with random variable X and Y and their realization X^t and Y^t.

The independence of the sequence $\{C^l\}_{1\leq l\leq 2S}$ makes the use of CPA completely irrelevant as shown in Figure 2.5. We use $K=$ `0xdeeefbf7` as a model key and $K'_0=$ `0xffffffff` is used as a distinguisher.

Regarding the DPA based on Hamming distances, the differential value to evaluate the distinction is given at each step i of the MPL by

$$\mathrm{DIFF}_i=f_i\left(\left\{H_i^l\left(K\right),H_i^{l+S}\left(K'_{j_{p-1}}\right)\right\}_{l=1}^{S}\right)=\overline{H}_i(K,C)-\overline{H}_i(K'_j,C).$$

Unlike for CPA, the lag $+S$ involved in the expression of DIFF_i is less disturbing because, by the law of large numbers, $\frac{1}{S}\sum_{k=1}^{S}H_i(K'_j,C^{k+S})$ and $\frac{1}{S}\sum_{k=1}^{S}H_i(K'_j,C^{k})$ converge to the same value as $S\to\infty$. Consequently, DPA can be used to exploit information leaks when S is big enough. The fact that we do not know how big S must be (except doing very coarse domination) makes DPA difficult to use. Indeed, as shown in Figure 2.6, sometimes we even need a bigger S for an RNS with less randomized moduli!

2.4.3 Further analysis : Second order DPA and MIA

Like in Section 2.4.2, we focus only on Hamming distances. Generally, DPA and MIA are used when the information leakage is observed with a hardware noise. In our study, the only noise is due to the RNS randomization that reduces the dependence between the secret K and Hamming distances.

2.4.3.1 Second order DPA

We use a simulation of second order DPA (2ODPA) as follows:

$$2ODPA_0 = DIFF_0$$
$$2ODPA_i = DIFF_{i+1} - DIFF_i \text{ if } i > 0$$

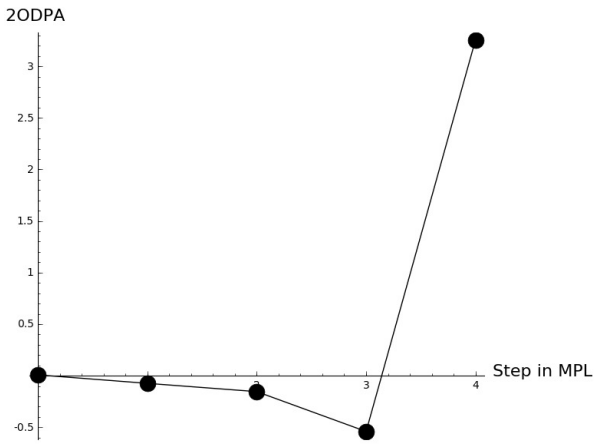


Figure 2.7 – Second order DPA between `0xffffffff` and `0xdeeefbf7` with 1000000 traces. RNS6 with Montgomery Power Ladder (MPL).

According to Figure 2.7, we see that the second order DPA on Hamming distances does not improve the results of DPA presented in Figure 2.6 (RNS6). This can be explained by the absence of a heterogeneity in the code between two steps of computations and thus between two successive Hamming distances. Moreover, the second order DPA defined in [75] (Proposition 2) involves marginal information since it averages on the realizations of one random variable defined as the difference between the power consumptions of two successive steps.

2.4.3.2 Mutual Information Analysis

Introduced in Section 2.4.1 and Appendice 2.C , the TVI computation involved the estimation of $P_1 = P\left(H_i \in \mathcal{H}_j^i\right)$ and of $P_2 = P\left(H_i \in \mathcal{H}_j^i | K \in I_k\right)$. These quantities have unbiased estimators with a Mean Square Error (MSE) (cf. [34]) equal to $\sigma^2\left(1_{H_i \in \mathcal{H}_j^i}\right)/S$

and to $\sigma^2\left(1_{H_i\in\mathcal{H}_j^i}|K\in I_k\right)/S$ respectively, where $\sigma^2(\cdot)$ is the variance and $\sigma^2(\cdot|K\in I_k)$ is the conditional variance. The approximations of P_1 and P_2 require many traces to reach an error smaller than 10%; more than 200000 traces in RNS10 for the first 10 Hamming distances, and more than 1000000 for 112 Hamming distances. Despite these relatively heavy computations when we randomized moduli, TVI was not able to measure the dependence structure. Randomized moduli TVI does not distinguish between the distribution of H_i and of H_i conditionally on $K\in I_k$, i.e between P_1 and P_2.

The MIA distinguisher (cf. [19]) is based on the mutual information of two random variables $(X,Y)\in\mathcal{X}\times\mathcal{Y}$:

$$I(X,Y)=\sum_{x\in\mathcal{X}}P(X=x)\sum_{y\in\mathcal{Y}}P(Y=y|X=x)\log\left(\frac{P(Y=y|X=x)}{P(Y=y)}\right).$$

In our case, using the subdivision described in Appendix 2.C, one has to approximate:

$$\sum_{k=0}^{2^{p'}-1}P(K\in I_k)\sum_{j=0}^{q-1}P(H_i\in\mathcal{H}_j^i|K\in I_k)\log\left(\frac{P(H_i\in\mathcal{H}_j^i|K\in I_k)}{P(H_i\in\mathcal{H}_j^i)}\right).$$

Therefore to apply MIA, we have to compute

$$\log\left(P\left(H_i\in\mathcal{H}_j^i\right)\right)\text{ and }\log\left(P\left(H_i\in\mathcal{H}_j^i|K\in I_k\right)\right)$$

that have biased Monte Carlo estimators as pointed out in Appendix 2.B. The quality of these biased estimators can be measured with MSE. For example, according to equality (5) in Appendix 2.D, the MSE of $\log\left(P\left(H_i\in\mathcal{H}_j^i\right)\right)$ is approximately $\dfrac{\sigma^2\left(\mathbf{1}_{\{H_i\in\mathcal{H}_j^i\}}\right)}{SP^2(H_i\in\mathcal{H}_j^i)}$. We notice that it is divided by the number $P^2\left(H_i\in\mathcal{H}_j^i\right)<<1$. The logarithm increases the distances but amplifies significantly the variance; it becomes difficult to estimate. This makes the use of MIA even harder than TVI to distinguish the dependence on the key K. Thanks to the Gaussian assumption on the distribution of Hamming distances, MLE distinguisher requires less computations.

2.5 Evaluation with Maximum Likelihood Estimator

2.5.1 Gaussian model and MLE to evaluate RNS randomization

Applying the Maximum Likelihood Estimator (MLE) [87], used for the Template attack [27], on Hamming distances provides better results than DPA. MLE requires a learning phase that precomputes the mean vector and the covariance matrix $(m_{k,i},\Gamma_{k,i})$ of $H^i=$

$(H_0, ..., H_i)^T$. Given the value of the secret $K = k$, for each $i \in \{0, ..., d-1\}$, we suppose $H^i = (H_0, ..., H_i)^T$ has a multivariate normal distribution with density

$$p_{k,i}(\mathbf{x}^i) = \frac{\exp\left(-\frac{(\mathbf{x}^i - m_{k,i})^T \Gamma_{k,i}^{-1} (\mathbf{x}^i - m_{k,i})}{2}\right)}{\left(\sqrt{2\pi}\right)^{i+1} \sqrt{\det(\Gamma_{k,i})}} \tag{2.8}$$

where $\mathbf{x}^i = (x_0, ..., x_i)^T \in \mathbb{R}^{i+1}$.

The MLE analysis that we performed can be summarized in the following steps: for $i = 0, 1, ..., d-1$,

- Either we compute the exact value of $(m_{k,i}, \Gamma_{k,i})$ with all $\frac{(2n)!}{(n!)^2}$ combinations in RNSn [1], or we estimate $(m_{k,i}, \Gamma_{k,i})$ using fewer combinations simulated with a Monte Carlo method. This is the learning phase.
- We observe S realizations $\mathbf{x}^i_{1 \le j \le S}$ of random vector $H^i = (H_0, ..., H_i)^T$.
- We select the secret $K = k$ that maximizes $\prod_{j=1}^{S} p_{k,i}(\mathbf{x}^i_j)$. Due to the floating-point precision, it is better to select the value that maximizes $\sum_{j=1}^{S} \log(p_{k,i}(\mathbf{x}^i_j))$. This is the estimation phase.
- We quantify the value of S that makes the information leakage exploitable.

This gives an MLE analysis algorithm :

Algorithm 3 MLE analysis

Require: Random Variable H^i_k for all k.
Ensure: Secret K and the value of S that makes the information leakage exploitable.
  Calculate or approximate $m_{k,i}$ and $\Gamma_{k,i}$ for all k.
  while S does not make the information leakage exploitable **do**
    We observe S réalizations $\left(x^i_j\right)_{1 \le j \le S}$ of $H^i_K = (H_{0,K}, ..., H_{i,K})$.
    We select $K = \arg\max_k \left\{ \prod_{j=1}^{S} p_{k,i}(x^i_j) \right\}$.
    increment S.
  end while

Indeed we have to perform statistical analysis for all k of $i+1$ bits, different values of S and RNS. We had to calculate $\Gamma_{k,i}^{-1}$ for all k. And we compute the exact value of $(m_{k,i}, \Gamma_{k,i})$

[1] This is our default choice in all the following figures and tables.

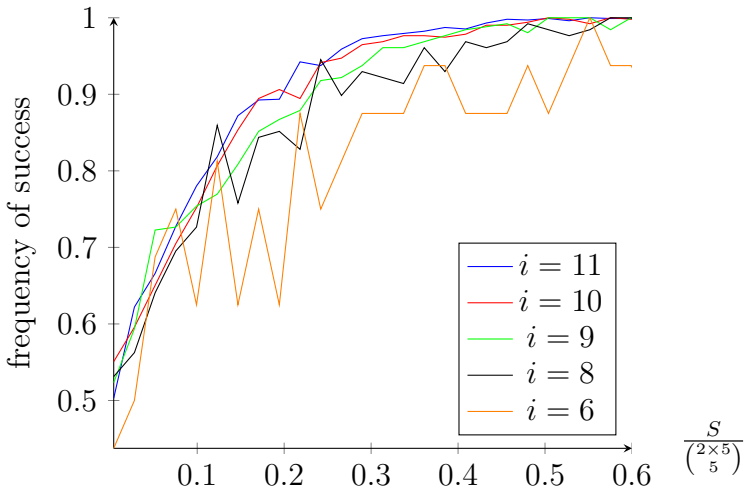


Figure 2.8 – Frequency of success to find the first bit of the key with MLE on ECC 112 in RNS5 ($n = 5$) for different size $(m_{k,i}, \Gamma_{k,i})$ of the template. $\binom{10}{5} \times 2^i$ samples to evaluate frequency

with all $\frac{(2n)!}{(n!)^2}$ combinations in RNSn. We use 2^i templates to estimate the success. We point out that this MLE analysis was possible largely to our GPU (Graphics Processing Unit) implementation that provides sufficient throughput to perform this study, especially for Figure 2.9.

Figure 2.8 shows that there is a remarkable information leak in the first ~ 10 successive Hamming distances. In particular, we see that there is not substantial amelioration between $i = 9$ and $i = 11$. This confirms what was already explained with TVI in Section 2.4.1.

According to Figure 2.9, with $S = \frac{(2n)!}{n!^2}$ we get almost 100% of success for ten bits. It is quite remarkable to see that S should be of the order of $(2n)!/(n!)^2$ that is the number of combinations in a RNSn. We point out that we obtain similar results with other curves including: Edwards 25519 [22] and ECCsecp256r1 [26].

Denoting $H_K^i = (H_{0,K}, ..., H_{i,K})^T$, a sequence of Hamming distances given by a key K, MLE shows that distribution of each H_K^i is completely distinguishable with its couple of mean and covariance matrix. Moreover, our study shows that the distribution differences reduce with an increase of the number of moduli till reaching the order of $\binom{2n}{n}$.

Let us denote f the frequency of success. Figure 2.9 shows that f depends on $\frac{S}{\binom{2n}{n}}$ and is almost linear with respect to $x = \frac{S}{\binom{2n}{n}}$ for $0 < x < 0.3$. Moreover, as we see in Table 2.1, the value of the slope barely changes for $n \in \{5, 6, 7, 8, 9, 10, 11\}$. Consequently, one can set $f \approx \beta \frac{S}{\binom{2n}{n}}$ with $2.2 < \beta < 2.4$ and $0 \leq \frac{S}{\binom{2n}{n}} < 0.3$.

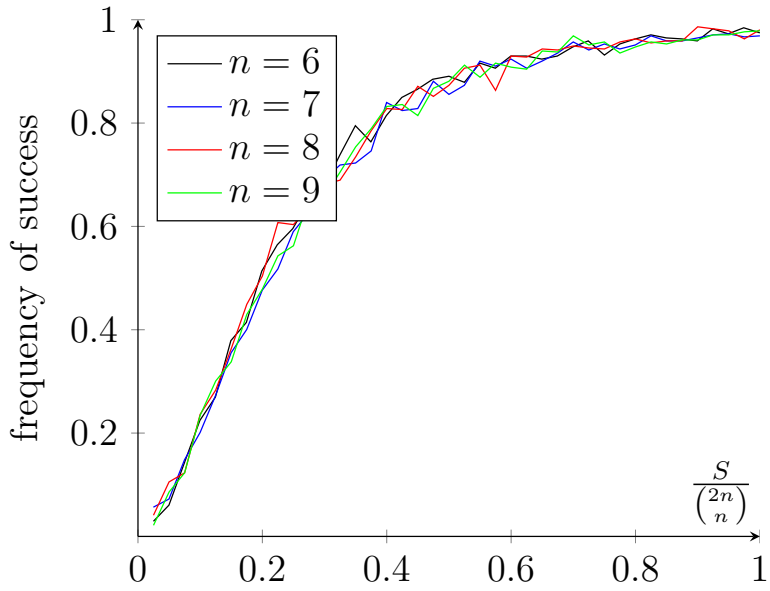


Figure 2.9 – Frequency of success to find a 10-bit key with MLE for different RNSn on ECC 112 Montgomery in Jacobian coordinates.

n	5	6	7	8	9	10	11
β	2.37	2.38	2.30	2.40	2.31	2.24	2.33

Table 2.1 – Slope of the linear regression

	p_t					
S	0.1	0.2	0.3	0.4	0.5	0.6
2^{10}	9	8	8	8	8	8
2^{15}	12	11	11	11	10	10
2^{20}	14	14	13	13	13	13
2^{25}	17	16	16	16	16	15
2^{30}	19	19	18	18	18	18
2^{35}	22	21	21	21	21	20
2^{40}	24	24	24	23	23	23
2^{45}	27	26	26	26	26	26
2^{50}	29	29	29	28	28	28

Table 2.2 – Minimum n to protect 10 first bits of the key till S traces with a success frequency smaller than p_t.

To protect the system, the frequency of success f should not be greater than a frequency threshold p_t:

$$\beta \frac{S}{\binom{2n}{n}} < p_t. \tag{2.9}$$

If we extrapolate the result for $n > 10$, we obtain, in Table 2.2, the minimum number of 32 bits moduli to fulfill the previous condition for the first 10 bits of the key. A good protection of significant bits could be important. Knowing some of the most significant bits of the key K for different base point, allows to mount attacks against the ECDSA signature scheme by means of an attack using LLL or the Bleichenbacher attack on DSA [10]. If we want to protect the whole key from a conditional strategy (cf. Section 2.4.1), we must consider another condition. In an ECC of size $\#ECC$, there are $\#ECC - 1$ unknown bits. We try to know each block of 10 bits knowing that we know the first bit of each block. We therefore analyze $\frac{\#ECC-1}{9}$ blocks. If we have a probability f to find each block of 10 bits. If we want to have a probability lower than p_t of finding the whole key, we have

$$f^{\frac{\#ECC-1}{9}} < p_t \quad \text{therefore} \quad f < p_t^{\frac{9}{\#ECC-1}} \quad \text{thus}$$

$$\beta \frac{S}{\binom{2n}{n}} < p_t^{\frac{9}{\#ECC-1}}, \tag{2.10}$$

knowing the value of the first bit and $\#ECC$ is the bit length of the cardinality of the ECC. Thus, Table 2.3 gives the minimum n to protect the whole key. This minimum is necessarily bigger than the smallest n required to implement an ECC of $\#ECC$ bits based on 32-bit moduli.

2.5.2 Additional considerations to choose the number of moduli

2.5.2.1 From which level we loose the random behaviour?

Let us denote the null hypothesis,
Hyp$_0$: "We obtain 10 bits of the key with a probability equal to 2^{-9}".
We calculate the 95% prediction interval with $p = 2^{-9}$:

$$\mathcal{I}_p = \left[p - 1.96\sqrt{\frac{p(1-p)}{SE}}; p + 1.96\sqrt{\frac{p(1-p)}{SE}} \right].$$

SE is a sample size. If $f \in \mathcal{I}_p$, we do not reject **Hyp$_0$** otherwise we reject **Hyp$_0$** .

$S \times \frac{\#ECC-1}{9}$	#ECC 112	256	384	521
2^{10}	6	9	13	18
2^{15}	8	9	13	18
2^{20}	11	10	13	18
2^{25}	13	13	13	18
2^{30}	16	15	15	18
2^{35}	19	18	18	18
2^{40}	21	20	20	20
2^{45}	24	23	23	22
2^{50}	26	26	25	25

Table 2.3 – Minimum n to protect the whole key till $S \times \frac{\#ECC-1}{9}$ traces from the target key: $p_t = 0.1$.

We can notice in Table 2.4 that we have to use $n > 7$ to avoid an attack with a single trace. This confirms the suggestion of [95].

n	5	6	7	8	9	10	11
S	1	1	1	5	7	16	130

Table 2.4 – Minimum size to reject $\mathbf{Hyp_0}$ with a sample size $SE = 32256$ ($error < 0.1\%$ for a 95% prediction interval)

2.5.2.2 The learning phase costs more than the estimation phase even with Monte Carlo

We used an exact value of $(m_{k,10}, \Gamma_{k,10})$ to set the template, so we needed $2^9 \times \binom{2n}{n}$ traces. Comparing this value to Table 2.3, we notice that the learning phase costs more than the estimation phase. Thus, the former determines the number of moduli needed for protection. Even with Monte Carlo, the success decreases by half when we use 80% of $2^9 \times \binom{2n}{n}$ traces to set the template (Fig. 2.10). If the attacker chooses to reduce computations via Monte Carlo, he decreases significantly his chance to find the secret.

2.6 Conclusion and future work

In this work, we used MLE that takes advantage of the cross-information in the Hamming distances. This provides an efficient evaluation of the information leakage even for cryptographic systems protected by RNS randomization. We showed however that this efficiency decreases as the number of needed observations is of the order of $\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$.

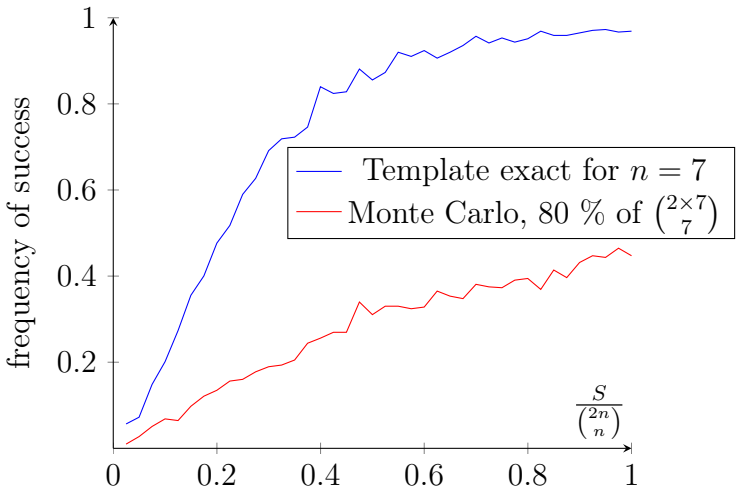


Figure 2.10 – Frequency of success to find a 10-bit key with MLE on ECC 112. Comparison between Template exact and Monte Carlo.

With this evaluation, we gave an estimation of the number of moduli to protect efficiently a system against a template attack. For example, since RNS12 requires $2^{30.36} = 2^9 \times \binom{24}{12}$ computations to construct a good template, it provides a good protection for the actual standard of ECC like Edwards curve 25519 [22] and ECCsecp256r1 [26]. A RNS15 protects also against the estimation phase at the same level and in any case, it is better to have $n > 7$ to have a correct random behaviour against a single trace attack.

As a future work, we would like to explore if there exists a method to decrease the number of moduli for a given level of randomness in order to reduce calculations. Furthermore, it would be good to prove theoretically the behaviour obtained in Figure 2.9 or at least establish an upper bound curve. It could be also interesting to do this same study when the template phase is evaluated with a Monte Carlo method and/or investigate variance reduction techniques.

Annexes

2.A Extensions in RNS Montgomery Algorithm

The first extension in Algorithm 1 is a raw method to prevent heavy computations due to mixed radix systems [16]. Thus, to extend in base $\widetilde{\mathcal{B}}_n = \{\widetilde{m}_1, ..., \widetilde{m}_n\}$, we calculate

$$\widetilde{q}_j = \left| \sum_{i=1}^{n} \left| q_i \left| M_i^{-1} \right|_{m_i} \right|_{m_i} M_i \right|_{\widetilde{m}_j} \quad \text{for each } j \in [\![1, n]\!]. \tag{2.11}$$

We obtain

$$\widetilde{Q} = \sum_{i=1}^{n} \left| q_i \left| M_i^{-1} \right|_{m_i} \right|_{m_i} M_i = Q + \alpha_Q \times M \text{ with } \alpha \in [\![1, n]\!]. \tag{2.12}$$

For the extension of R from $\widetilde{\mathcal{B}}_n$ to $\mathcal{B}_n$, we used a similar method, obtaining for R an expression like (4) with $\alpha_R \times \widetilde{M}$. Here, as R is not computed modulo $\widetilde{M}$, an extra modulo namely $m_x \geq n$ is used. Thus the first extension of Q is from $\mathcal{B}_n$ to $\tilde{\mathcal{B}}_n \cup \{m_x\}$, and the second extension from $\tilde{\mathcal{B}}_n$ to $\mathcal{B}_n$, using m_x for computing α_R. In contrast to Kawamura Extension [57], this Shenoy-Kumaresan method [89, 12] allows a larger choice of moduli. We obtained all the results with pseudo-Mersenne 32-bit moduli.

$$\alpha_R = \left| \left| \widetilde{M}^{-1} \right|_{m_x} \left(\sum_{j=1}^{n} \left| \widetilde{r}_j \left| \widetilde{M}_j^{-1} \right|_{\widetilde{m}_j} \widetilde{M}_j \right|_{m_x} - |R|_{m_x} \right) \right|_{m_x}. \tag{2.13}$$

Subsequently, $r_i = |R|_{m_i}$ can be computed using

$$r_i = \left| \sum_{j=1}^{n} \left| \widetilde{r}_j \left| \widetilde{M}_j^{-1} \right|_{\widetilde{m}_j} \widetilde{M}_j \right|_{\widetilde{m}_j} - |\alpha_R M|_{\widetilde{m}_j} \right|_{m_i}. \tag{2.14}$$

The Shenoy-Kumaresan extension requires that N has to fulfill $(n+2)^2 N < M$. The latter inequality with $AB < NM$ makes $R < (n+2)N$. As we obtain $R < 2N$ at the very end of an exponentiation, we can use mixed radix representation [16] to obtain the exact value.

2.B Quick review of Monte Carlo method

A detailed analysis of the effects of randomization is conducted in this chapter. The implemented probability tools use either brute-force that involves the whole set of scenarios or Monte Carlo simulation that samples it. The latter method is based on two theorems that are generally considered as the two pillars of the probability theory. The first one is the Strong Law of Large Numbers (SLLN) that announces the convergence of a certain series $\widetilde{\mathbb{E}}_S(W_l) := \sum_{l=1}^{S} \frac{W_l}{S}$ to $\mathbb{E}(W_l)$ as $S \to \infty$. The independent random variables $(W_l)_{l \geq 1}$ have the same distribution with $\mathbb{E}(|W_l|) < \infty$. The second one is the Central Limit Theorem (CLT) which determines the speed of the convergence when we assume $\mathbb{E}(|W_l|^2) < \infty$. These two classic theorems can be found, for instance, in [53]. In our study of randomization, $\mathbb{E}(|W_l|^2) < \infty$ is true since the studied random variables are bounded.

As $\mathbb{E}\left(\widetilde{\mathbb{E}}_S(W_l) - \mathbb{E}(W_l)\right) = 0$, the Monte Carlo estimator $\widetilde{\mathbb{E}}_S(W_l)$ is unbiased and its

efficiency is translated through its variance which asymptotically vanishes. Indeed, CLT allows even to have a confidence interval. For instance

$$P\left(\mathbb{E}(W_l) \in \left[\widetilde{\mathbb{E}}_S(W_l) - \frac{1.96\sigma(W_l)}{\sqrt{S}}, \widetilde{\mathbb{E}}_S(W_l) + \frac{1.96\sigma(W_l)}{\sqrt{S}}\right]\right) \approx 95\%$$

with relative error $= \dfrac{2 \times 1.96\sigma(W_l)}{\widetilde{\mathbb{E}}_S(W_l)\sqrt{S}}$. Thus, the complexity/accuracy of Monte Carlo is governed by the variance $\sigma^2(W_l) = \mathbb{E}\left[(W_l - \mathbb{E}(W_l))^2\right]$ which can be decreased using variance reduction techniques. There is not general procedure for variance reduction but various standard methods like conditioning, importance sampling, stratification and control variates.

Presented in Appendix 2.C and used in Section 3.1, the width of the sets $I_k \times \mathcal{H}_j^i$ was tuned to reduce variance without significantly compromising the sensitivity of the TVI estimator as a measure of dependence. We did not investigate other variance reduction methods in this study but we project to do it in future works.

In case of biased estimators as $\log(\widetilde{\mathbb{E}}_S(W_l))$, used in MIA, it is not sufficient to study only the variance. Indeed, applying Jensen's inequality with logarithm one proves that $\mathbb{E}\left(\log(\widetilde{\mathbb{E}}_S(W_l)) - \log(\mathbb{E}(W_l))\right) \neq 0$. Then, one should rather compute $MSE =$ variance $+$ bias^2 to study the efficiency of the estimator.

2.C Subdivision for evaluation of TVI

In order to reduce the complexity of computations and increase the Monte Carlo accuracy Appendix 2.B, we use appropriate subdivisions of I and $\mathcal{H}^i$ respectively into $2^{p'}$ and q sub-intervals.

$$\text{a) } I = [0, 2^p) = \bigcup_{k=0}^{2^{p'}-1}, \; I_k = \bigcup_{k=0}^{2^{p'}-1} [k2^{p-p'}, (k+1)2^{p-p'})$$

and

$$\text{b) } \mathcal{H}^i = [min(H_i), max(H_i)] = \bigcup_{j=0}^{q-1} \mathcal{H}_j^i$$

with

$$\begin{aligned}
&\mathcal{H}_0^i = [min(H_i), min(H_i) + \lambda), \\
&\mathcal{H}_{q-1}^i = [max(H_i) - \lambda, max(H_i)], \\
&\mathcal{H}_j^i = [min(H_i) + \lambda + j\epsilon, min(H_i) + \lambda + (j+1)\epsilon)
\end{aligned}$$

for

$$j = 1, \dots q-2 \text{ where } \epsilon = \frac{max(H_i) - min(H_i) - 2\lambda}{q-2}.$$

The choice of λ is closely linked to the mean and the standard deviation of H_i Referring to Appendix 2.B, we use $W_l = \mathbf{1}_{\{H_i^l \in \mathcal{H}_j^i\}}$. Defining $\mathcal{P} = P\left(\mathbf{1}_{\{H_i^l \in \mathcal{H}_j^i\}}\right)$, we get $\sigma^2\left(\mathbf{1}_{\{H_i^l \in \mathcal{H}_j^i\}}\right) = \mathcal{P} - \mathcal{P}^2 = \mathcal{P}(1-\mathcal{P})$.

The relative error is $= \dfrac{2 \times 1.96\sqrt{\mathcal{P}(1-\mathcal{P})}}{\mathcal{P}\sqrt{S}} = \dfrac{2 \times 1.96}{\sqrt{S}}\sqrt{\dfrac{(1-\mathcal{P})}{\mathcal{P}}}$. We remark that the relative error decreases when $\mathcal{P}$ increases and $\mathcal{P}$ is controlled by the size of $\mathcal{H}_j^i$.

2.D Mean Square Error (MSE) of $\log\left(P\left(V \in A\right)\right)$

Introducing the Bernoulli random variable $W = \mathbf{1}_{\{V \in A\}}$ with $\mathbb{E}(W) = P(V \in A)$, an estimation of $\log(P(V \in A))$ could be obtained by $\log\left(\widetilde{\mathbb{E}}_S(W)\right)$ with:

$$\widetilde{\mathbb{E}}_S(W) = \frac{1}{S}\sum_{j=1}^{S} \mathbf{1}_{\{V^j \in A\}}$$

where $\{\mathbf{1}_{\{V^1 \in A\}}, \dots, \mathbf{1}_{\{V^S \in A\}}\}$ are independent realizations that have the same distribution as W. Then, we compute

$$\begin{aligned}
MSE_{\log} &= \mathbb{E}\left(\left(\log(\mathbb{E}(W)) - \log\left(\widetilde{\mathbb{E}}_S(W)\right)\right)^2\right) \\
&= \mathbb{E}\left(\log^2\left(\frac{\widetilde{\mathbb{E}}_S(W)}{\mathbb{E}(W)}\right)\right) \\
&= \mathbb{E}\left(\log^2\left(1 - \frac{\mathbb{E}(W) - \widetilde{\mathbb{E}}_S(W)}{\mathbb{E}(W)}\right)\right) \\
&\approx \mathbb{E}\left(\left(\frac{\mathbb{E}(W) - \widetilde{\mathbb{E}}_S(W)}{\mathbb{E}(W)}\right)^2\right), \text{ since } \log(1-x) = -x + o(x).
\end{aligned}$$

Applying CLT:

$$\lim_{S \to \infty} \frac{\sqrt{S}}{\sigma(W)}\left(\mathbb{E}(W) - \widetilde{\mathbb{E}}_S(W)\right) \rightsquigarrow \mathcal{N}(0;1),$$

thus

$$MSE_{log} \approx \left(\frac{\sigma(W)}{\sqrt{S}\mathbb{E}(W)}\right)^2 = \frac{\sigma^2(W)}{S\mathbb{E}^2(W)} = \frac{\sigma^2\left(\mathbf{1}_{\{V\in A\}}\right)}{SP^2(V\in A)}. \tag{2.15}$$

Chapter 3

Dépendance Forte et Faible entre secret et distances de Hamming sous l'hypothèse gaussienne

Sommaire

Notation: d est la taille de la clef en binaire. Pour $1 \leq i \leq d-1$, la taille des vecteurs analysés est $i+1$, ici $+1$ car le début d'une clef est toujours 1 en binaire.

3.1 Introduction

Dans la suite, on parlera

- d'analyse forte pour une analyse qui permet de retrouver la clef.
- d'une analyse faible dans le cas où on élimine des clefs candidates.

On utilise X_K^i pour désigner le vecteur colonne de $i+1$ variables aléatoires obtenu lors d'une réalisation de clef K, i.e. $X_K^i = (X_{0,K}, ..., X_{i,K})^T$ où $X_{t,K}$ correspond à la variable aléatoire à la t-ième étape dans un calcul ECC. On considère toutes les clefs de $i+1$ bits commençant par 1. On note ainsi $(X_K^i, K = 2^i, ..., 2^{i+1}-1) := \left({X_{2^i}^i}^T | {X_{2^i+1}^i}^T | \dots | {X_{2^{i+1}-1}^i}^T\right)^T$, le vecteur colonne qui est la concaténation de toutes les coordonnées de X_K^i, pour $K \in \{2^i, ..., 2^{i+1}-1\}$.

On rappelle qu'une variable aléatoire multivariée $X^t = (X_0, ..., X_t)^T$ a une distribution gaussienne si sa fonction de densité est :

$$p_t(\mathbf{x}^t) = \frac{1}{\left(\sqrt{2\pi}\right)^{t+1}\sqrt{\det(\Gamma_t)}} \exp\left(-\frac{(\mathbf{x}^t - m_t)^T \Gamma_t^{-1} (\mathbf{x}^t - m_t)}{2}\right) \tag{3.1}$$

où $\mathbf{x}^t = (x_0, ..., x_t)^T \in \mathbb{R}^{t+1}$. $m_t = (\mathbb{E}(X_0), ..., \mathbb{E}(X_t))^T$ est le vecteur moyen de X^t. $\Gamma_t = \mathbb{E}\left[(X^t - m_t)(X^t - m_t)^T\right]$ est la matrice de covariance de X^t.

Dans le Chapitre 2, on a montré comment construire un vecteur de variables aléatoires $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ où $H_{t,K}$, avec $0 \leq t \leq i$, qui sont les distances de Hamming

générées par l'Algorithme 2 de la Section 2.3.2. On a montré dans la Section 2.5 que faire l'hypothèse gaussienne permettait de retrouver une partie de la clef cible en ayant suffisamment de traces. Cependant, dans le cas d'un système cryptographique asymétrique, il est courant de n'utiliser la clef qu'une seule fois. L'objectif de ce chapitre est d'élargir le cadre de l'hypothèse gaussienne pour trouver de l'information dans une seule trace. On considère donc que le vecteur des distances de Hamming est une variable aléatoire et on utilise l'hypothèse gaussienne dans trois types de configurations.

- Dans la Section 3.2, on considère que le vecteur $(H_K^i, K = 2^i, ..., 2^{i+1} - 1) := \left({H_{2^i}^i}^T | {H_{2^i+1}^i}^T | \dots | {H_{2^{i+1}-1}^i}^T\right)^T$ est gaussien. L'objectif est de chercher une combinaison linéaire sur les vecteurs de Hamming. En effet, il existe bien une relation entre les clefs K, les configurations de moduli et le vecteur de Hamming, mais elle est difficile à établir en pratique sans hypothèse supplémentaire. En utilisant l'hypothèse gaussienne pour $(H_K^i, K = 2^i, ..., 2^{i+1} - 1)$, nous testons la possibilité de trouver une relation linéaire entre les distances de Hamming générées par différentes clefs et différentes configurations de moduli. Par construction, on parlera de dépendance forte. Si elle existe cela entraînerait une faiblesse notable de la contre-mesure des moduli aléatoires, faiblesse qui pourrait être exploitable par un attaquant malveillant. Mais on verra que cela n'est pas le cas même si on utilise peu de moduli pour la représentation RNS.

- Dans la Section 3.3, on considère que chaque vecteur aléatoire H_K^i est un vecteur gaussien, i.e. pour chaque $K = k$, $H_k^i = (H_{0,k}, ..., H_{i,k})^T$ a une distribution normale multivariée. On parlera de dépendance faible lorsque les paramètres de la loi des variables aléatoires dépendent du secret. Faible car les vecteurs aléatoires $H_{2^i}^i, ..., H_{2^{i+1}-1}^i$ ne sont pas nécessairement dans le même espace de probabilité. Sachant que l'on a fait l'hypothèse gaussienne qui détermine le vecteur de Hamming par une moyenne et une matrice de covariance, on étend la notion de DPA dans la Section 2.4.2, qui ne considère que le vecteur moyen, à une DPA square qui utilise la matrice de covariance.

- Dans la Section 3.4, l'hypothèse gaussienne est faite sur le vecteur aléatoire H_K^i conditionnée par un événement A (par exemple "$H_{1,K} = 567$"). L'hypothèse gaussienne permet de déterminer complètement la variable (vecteur) aléatoire H_K^i en pré-calculant le vecteur moyen et la matrice de covariance, c'est-à-dire le couple $(m_{K,i}, \Gamma_{K,i})$. On parlera de dépendance faible avec un conditionnement. Cependant une analyse forte de la dépendance faible avec conditionnement n'est pas suffisante pour faire apparaître une fragilité d'une attaque sur une seule trace. On relaxe le problème avec une analyse faible. On élimine uniquement une partie des clefs candidates au lieu de garder la bonne. Il en résulte une stratégie d'attaque utilisant

l'hypothèse gaussienne comme dans la Section 2.5.1 avec l'estimateur du maximum de vraisemblance (MLE : Maximum Likelyhood Estimator). Cette stratégie permet d'être moins gourmande en traces du point de vue de l'analyse du secret.

3.2 Analyse de la Dépendance forte et ses limites

L'étude présentée dans cette section a pour but de démontrer la difficulté d'élaborer une stratégie d'attaque fondée sur un problème inverse. Dans la Section 3.2.1, nous présentons le modèle choisi pour simplifier cette étude. La Section 3.2.2 détaille les résultats obtenus et conforte la complexité à établir une dépendance forte entre distances de Hamming et clef. La problématique initiale : pour une configuration de moduli et un choix de clef, peut-on déduire une fonction analytique qui lie les Hammings de sortie avec ce choix de configuration et de clef ?

$$K = f(Hamming, \text{configuration de moduli}).$$

Les distances de Hamming observées sont des fonctions déterministes de la randomisation des moduli et de la clef choisie. Retrouver des approximations analytiques de ces fonctions pourrait rendre efficace une attaque fondée sur l'inversion de ces approximations. De telles attaques permettraient de retrouver la clef et la combinaison de moduli à partir d'une réalisation de vecteur de Hamming.

La complexité du système cryptographique liée à l'utilisation des courbes elliptiques, la randomisation RNS, la représentation en coordonnées Jacobiennes et le facteur de Montgomery rendent très difficile l'approximation analytique du système décrit en Figure 3.1.

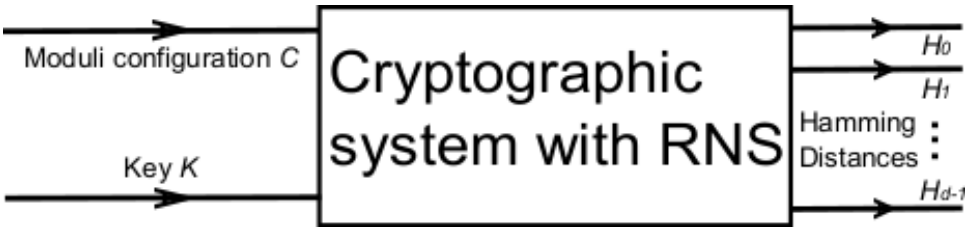


Figure 3.1 – Distances de Hamming en fonction d'une entrée aléatoire

Cette fonction existe puisque qu'elle est déterministe. On peut par exemple créer un tableau associant les $\binom{2n}{n}$ configurations de moduli et le vecteur de Hamming pour chaque clef. On obtient pour des clefs de taille d en RNSn, un tableau de taille $2^{d-1} \times \binom{2n}{n} \times d$ et chercher une fonction analytique qui va faire une approximation de ce résultat. Cette méthode s'assimile à la force brute et par construction, il est très difficile

d'établir une telle fonction analytique. Le fait de rajouter l'hypothèse gaussienne sur le vecteur $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ pourrait permettre de contourner la difficulté.

A titre d'exemple, réduisons le problème à la comparaison entre deux clefs K et K'. La première réflexion naïve serait de trouver une relation f entre les vecteurs de Hamming $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ et $H_{K'}^i = (H_{0,K'}, ..., H_{i,K'})^T$ telle que:

$$H_K^i = f\left(H_{K'}^i\right).$$

Prenons par exemple pour $K = 127 = 1111111_2$ et $K' = 112 = 1110000_2$, on remarque alors que les 3 premiers bits sont égaux donc $(H_{0,127}, H_{1,127}, H_{2,127}) = (H_{0,112}, H_{1,112}, H_{2,112})$ mais on ne pourra rien dire pour les distances de Hamming suivantes.

Plus généralement, pour certaines clefs et pour les $b < i$ premiers bits, on a $(H_{0,K}, ..., H_{b,K})^T = (H_{0,K'}, ..., H_{b,K'})^T$, mais on ne peut rien dire pour les bits qui suivent si on ne fait aucune hypothèse.

Dans le cadre de notre étude, on cherche à savoir s'il y a une faiblesse dans la contre-mesure des moduli aléatoires. Il serait donc intéressant de savoir si les vecteurs aléatoires représentés par les distances de Hamming ont une intersection significative des tribus engendrées. Si l'intersection des tribus engendrées de tous les vecteurs de Hamming est suffisamment grande, on pourrait trouver une relation forte entre ceux ci et mettre en avant une faiblesse du système. On montre dans l'annexe 3.A un exemple d'intersection pour un cas discret simple ou [53]

Si on modélise les distances de Hammming par la fonction suivante :

$$H_K^i : (\Omega, \mathcal{F}, P) \to \left(E_K^i, \mathcal{E}_K^i\right) \tag{3.2}$$

- Ω est l'espace d'état des configurations de moduli et des clefs possibles. Dans notre exemple de la Section 2.3, avant chaque exponentiation modulaire, on fait un tirage aléatoire de n moduli $\{m_1, ..., m_n\}$ parmi $\{\mu_1, .., \mu_{2n}\}$ pour une base $\mathcal{B}_n$. Le reste des moduli forme la base $\widetilde{\mathcal{B}}_n$. C'est un tirage aléatoire standard sans remise. Ω est donc l'ensemble des combinaisons de n parmi $2n$ moduli auquel on ajoute l'ensemble des clefs possibles.
- $\mathcal{F}$ la tribu associée à Ω.
- E_K^i est l'espace d'état des valeurs prises par H_K^i.
- P est une mesure de probabilité.
- $\mathcal{E}_K^i$ est la tribu associée à E_K^i.

Avec l'approximation par l'hypothèse gaussienne $E_K^{i+1} = \mathbb{R}^{i+1}$ et $\mathcal{E}_K^{i+1} = \mathcal{B}(\mathbb{R}^{i+1})$ qui est la tribu borélienne de $\mathbb{R}^{i+1}$.

3.2.1 Choix du modèle de la dépendance forte

On rappelle la notation $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ utilisée pour une suite de distances de Hamming obtenue à partir d'une clef K (cf. Section 2.5.1). Si on trouve une relation $F : (\mathbb{R}^{i+1})^{2^i} \to \mathbb{R}$ entre les H_K^i telle que :

$$F(H_{2^i}^i, H_{2^i+1}^i, \ldots, H_{2^{i+1}-1}^i) = 0 \tag{3.3}$$

on considère alors que l'on a une relation forte qui peut être utilisée pour attaquer le secret. On parle de dépendance forte car on cherche à établir une relation entre variables aléatoires. Par exemple, si $F(H_{2^i}^i, H_{2^i+1}^i) = (H_{2^i}^i)_t - (H_{2^i+1}^i)_{t'}$, où $(H_{2^i}^i)_t$ et $(H_{2^i+1}^i)_{t'}$ sont respectivement, des composantes du vecteur $H_{2^i}^i$ et $H_{2^i+1}^i$. Si on connait la distance de Hamming associée à $K = 2^i$, on connait alors celle associée à $K = 2^i + 1$

Une démarche constructive d'une relation vérifiant (3.3) est difficile à automatiser sans réduire l'espace de recherche. On va donc faire des hypothèses et des transformations pour simplifier le problème. D'abord on fait l'hypothèse que le vecteur $(H_K^i, K = 2^i, ..., 2^{i+1}-1) := \left({H_{2^i}^i}^T | {H_{2^i+1}^i}^T | \ldots | {H_{2^{i+1}-1}^i}^T\right)^T$ est gaussien, ce qui restreint la recherche de F sur l'ensemble des formes linéaires. Ensuite, on décorrèle les coordonnées de chaque composante H_K^i du vecteur $(H_K^i, K = 2^i, ..., 2^{i+1}-1)$ pour obtenir G_K^i composé de gaussiennes centrées constituant une base orthonormale pour les réalisations de H_K^i. Cela nous sera utile pour faciliter notre recherche des relations fortes (cf. figure 3.2). Puis on construit $\widetilde{\mathcal{G}}_i = (G_K^i, K = 2^i, ..., 2^{i+1} - 1)$. Finalement, pour chercher les relations, $\mathcal{G}_i$ résulte de la concaténation $\widetilde{\mathcal{G}}_i = (G_K^i, K = 2^i, ..., 2^{i+1} - 1)$ après suppression des redondances structurelles.

On a vu dans la Section 2.5.1 que chaque H_K^i peut être modélisé comme un vecteur aléatoire gaussien complètement déterminé par le couple moyenne et matrice de covariance $(m_{K,i}, \Gamma_{K,i})$. Comme dans une Analyse en Composante Principale, on cherche des directions privilégiées qui permettraient d'établir des relations entre l'ensemble ou une partie des composantes de $(H_K^i, K = 2^i, ..., 2^{i+1} - 1)$.

La matrice de covariance $\Gamma_{K,i}$ est une matrice symétrique définie positive. Il existe donc une matrice triangulaire inférieure notée $\sqrt{\Gamma_{K,i}}$, telle que $\sqrt{\Gamma_{K,i}}\sqrt{\Gamma_{K,i}}^T = \Gamma_{K,i}$. Comme $\Gamma_{K,i}$ est définie positive, $\sqrt{\Gamma_{K,i}}$ est inversible. Si on suppose que H_K^i est un vecteur gaussien, on peut écrire d'après [53, 67] :

$$H_K^i = m_{K,i} - \sqrt{\Gamma_{K,i}} G_K^i \tag{3.4}$$

où G_K^i est une gaussienne multivariée centrée et de covariance la matrice identité. On

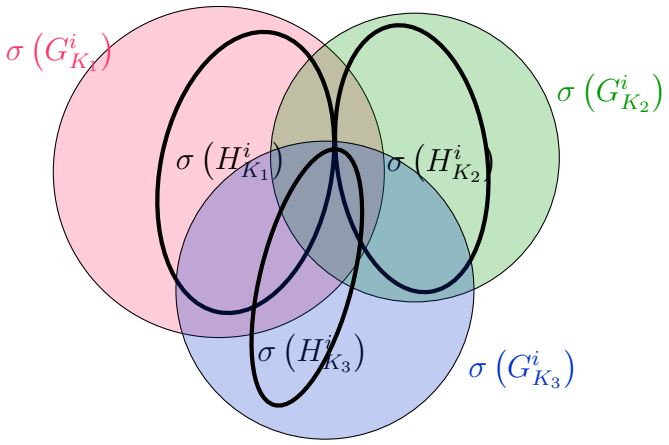


Figure 3.2 – Tribus engendrées par les Hammings H^i_K et par les Hammings décorrélés G^i_K.

peut donc décorréler les composantes de H^i_K. On en déduit que :

$$G^i_K = \left(H^i_K - m_{K,i}\right)\sqrt{\Gamma_{K,i}}^{-1}. \tag{3.5}$$

Décomposer les réalisations de H^i_K sur la base G^i_K permet de passer d'une distribution avec des directions privilégiées pour H^i_K à une distribution homogène pour G^i_K. On facilite ainsi la recherche d'intersection entre les tribus engendrées, comme représentée sur la Figure 3.2. Les directions privilégiées sont illustrées par des ellipses. Les cercles représentent l'homogénéisation.

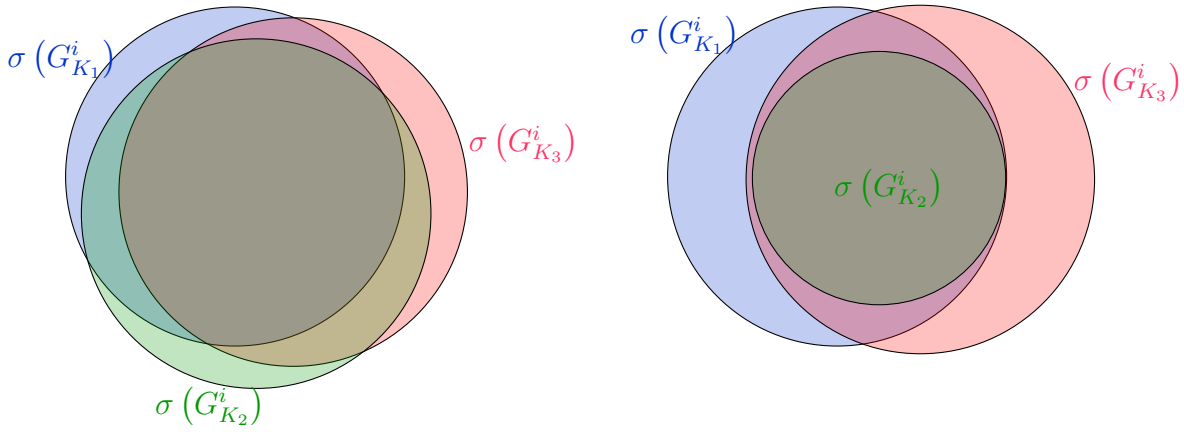
Si l'intersection des tribus engendrées était significative, comme représenté sur la Figure 3.3a, cela pourrait être une source majeure de fragilité du système. Ce dernier cas serait idéal pour un attaquant malveillant. Cela signifierait par exemple que l'on aurait deux vecteurs aléatoires $H^i_{K_1}$ et $H^i_{K_2}$ associées aux clefs K_1 et K_2 qui seraient corrélés par le même vecteur aléatoire centré réduit G^i.

$$H^i_{K_1} = m_{K_1,i} - \sqrt{\Gamma_{K_1,i}}G^i \text{ et } H^i_{K_2} = m_{K_2,i} - \sqrt{\Gamma_{K_2,i}}G^i.$$

Dans ce cas, avec S réalisations $\mathbf{x}^i_{1\leq j\leq S}$ données par la clef cible, on estime

$$\left(\mathbf{x}^i_j - m_{K_1,i}\right)\sqrt{\Gamma_{K_1,i}}^{-1} \text{ et } \left(\mathbf{x}^i_j - m_{K_2,i}\right)\sqrt{\Gamma_{K_2,i}}^{-1}. \tag{3.6}$$

et on élimine le cas qui ne suit pas une loi normale centrée réduite.



(a) Cas où les Hammings sont presque dans un même espace de probabilité.

(b) Cas où un Hamming dépend de deux autres.

Figure 3.3 – Tribus engendrées par les Hammings décorrélés G^i_K.

Une situation moins dramatique mais tout de même dangereuse serait le cas où les Hammings d'une clef seraient la combinaison linéaire des Hammings d'autres clefs, comme on pourrait le représenter sur la Figure 3.3b.

Après avoir fait la décorrélation, on construit le vecteur qui concatène toutes les distances de Hamming pour toutes les clefs de taille $i+1$ (Pour rappel commençant par 1). Il y a 2^i clefs de $i+1$ bits allant de 2^i à $2^{i+1}-1$. On construit donc $\widetilde{\mathcal{G}}_i$ tel que $\widetilde{\mathcal{G}}_i = (G^i_K, K = 2^i, ..., 2^{i+1}-1)$ qui est un vecteur colonne de taille $2^i \times (i+1)$.

On peut encore simplifier le calcul en enlevant les redondances. Certaines valeurs de clefs commencent par les mêmes valeurs de bits. Autrement dit, elles ont le même préfixe. Il existe donc des redondances dans le vecteur $\widetilde{\mathcal{G}}_i$. Pour une clef K et tout t tel que $t < i$, on enlève toutes les valeurs de $G_{t,K'}$ telles que $K' \neq K$ a le même préfixe de taille $t+1$ que K. Par exemple pour $i = 2$, en écrivant la valeur des clefs en binaire :

$$
\begin{aligned}
G^{2T}_4 &= (g_{0,100}, g_{1,100}, g_{2,100})\,, \\
G^{2T}_5 &= (g_{0,101}, g_{1,101}, g_{2,101})\,, \\
G^{2T}_6 &= (g_{0,110}, g_{1,110,1}, g_{2,110})\,, \\
G^{2T}_7 &= (g_{0,111}, g_{1,111}, g_{2,111})
\end{aligned}
$$

donc $\widetilde{\mathcal{G}}^T_2 = (g_{0,100}, g_{1,100}, g_{2,100}, g_{0,101}, g_{1,101}, g_{2,101}, g_{0,110}, g_{1,110}, g_{2,110}, g_{0,111}, g_{1,111}, g_{2,111})\,.$

On remarque qu'il y a des redondances dans les composantes de $\widetilde{\mathcal{G}}_i$. Par exemple pour

$i = 2$:

$g_{0,100} = g_{0,101} = g_{0,110} = g_{0,111}$ *car le premier bit de chaque clef (4, ,5, 6 et 7) est un* 1.
$g_{1,100} = g_{1,101}$ *car les deux premiers bits de chaque clef (4 et 5) sont* 10.
$g_{1,110} = g_{1,111}$ *car les deux premiers bits de chaque clef (6 et 7) sont* 11.

Donc si on ne veut pas écrire ces redondances, le vecteur $\widetilde{\mathcal{G}}_2$ devient :

$$\mathcal{G}_2^T = (g_{0,100}, g_{1,100}, g_{2,100}, g_{2,101}, g_{1,110}, , g_{2,111})$$

Si on fait cela pour le vecteur $\widetilde{\mathcal{G}}_i$ quand les premiers bits de chaque clef sont les mêmes, on obtient un vecteur $\mathcal{G}_i$ de taille $\omega_i = 2^{i+1} - 1$ sans redondances.

Preuve : Le problème revient à compter le nombre de mots qui contiennent entre 1 et $i+1$ caractères 0 ou 1 sachant que tous les mots commencent par 1. Il est clair qu'il y a 2^{t-1} mots de taille t avec $1 \leq t \leq i+1$. Il y a donc $1 + 2 + 2^2 + \cdots + 2^i = 2^{i+1} - 1$ mots.

Une construction formelle de $\mathcal{G}_i$: En posant $\left(X_0, X_1, X_2, \ldots, X_{(i+1)2^i-1}\right)$ une suite de variables aléatoires, on définit la fonction

$$V_i : \left(X_0, X_1, X_2, \ldots, X_{(i+1)2^i-1}\right) \to \left(X_{\ell_0}, X_{\ell_1}, \ldots, X_{\ell_{2^{i+1}-2}}\right)$$

où ℓ_t avec $0 \leq t < 2^{i+1} - 1$ est une suite telle que :

$\ell_0 = 0$
$\ell_t = min\left\{j \in [\![1; (i+1)2^i - 1]\!] \setminus \{\ell_0, \ell_1, \ldots, \ell_{t-1}\} \,\middle|\, X_j \notin \left\{X_{\ell_0}, X_{\ell_1}, \ldots, X_{\ell_{t-1}}\right\}\right\}$ pour $t > 0$.

On a alors

$$\mathcal{G}_i^T = V_i\left(\widetilde{\mathcal{G}}_i^T\right). \tag{3.7}$$

Le passage de l'équation (3.8) à l'équation (3.13) permet de schématiser la succession d'hypothèses et d'opérations permettant de simplifier la recherche de la relation forte F. En notant $\mathscr{E}(A)$ l'ensemble des fonctions $F : A \to \mathbb{R}$ et $\mathscr{L}(A)$ l'ensemble des formes linéaires $F : A \to \mathbb{R}$, on réduit l'espace de recherche de F comme suit :

$$F \in \mathscr{E}\left(\left(\mathbb{R}^{i+1}\right)^{2^i}\right) \text{ telle que } F(H^i_{2^i}, H^i_{2^i+1}, \ldots, H^i_{2^{i+1}-1}) = 0. \tag{3.8}$$

$$\Big\Downarrow \text{Avec l'hypothèse gaussienne}$$

$$F \in \mathscr{L}\left(\left(\mathbb{R}^{i+1}\right)^{2^i}\right) \text{ telle que } F(H^i_{2^i}, H^i_{2^i+1}, \ldots, H^i_{2^{i+1}-1}) = 0. \tag{3.9}$$

$$\Big\Downarrow \text{Avec la décorrélation}$$

$$F \in \mathscr{L}\left(\left(\mathbb{R}^{i+1}\right)^{2^i}\right) \text{ telle que } F(G^i_{2^i}, G^i_{2^i+1}, \ldots, G^i_{2^{i+1}-1}) = 0 \tag{3.10}$$

où G^i_K est défini par (3.5) .

Ce qui revient à écrire

$$F \in \mathscr{L}\left(\left(\mathbb{R}^{i+1}\right)^{2^i}\right) \text{telle que } F(\widetilde{\mathcal{G}}_i) = 0. \tag{3.11}$$

$$\Big\Downarrow \text{Sans les redondances}$$

$$F \in \mathscr{L}\left(\mathbb{R}^{2^{i+1}-1}\right) \text{ telle que } F\left(V_i\left(\widetilde{\mathcal{G}}_i\right)\right) = 0. \tag{3.12}$$

Ce qui revient à écrire

$$F \in \mathscr{L}\left(\mathbb{R}^{2^{i+1}-1}\right) \text{ telle que } F(\mathcal{G}_i) = 0. \tag{3.13}$$

où $\mathcal{G}_i$, défini par l'équation (3.7), est une partie du vecteur colonne $(G^i_K, K = 2^i, ..., 2^{i+1} - 1)$ auquel on a enlevé les redondances.

Notons $\mathcal{G}_i^T = \left(\mathcal{G}_{i,0}, \mathcal{G}_{i,1}, \ldots, \mathcal{G}_{i,2^{i+1}-2}\right)$, les composantes du vecteur $\mathcal{G}_i$. Ce sont les distances de Hamming décorrélées sans redondance. On peut résumer la séparabilité des tribus engendrées par ces composantes, en caractérisant trois cas :

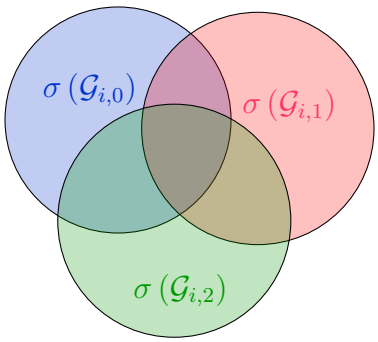
(a) Le cas défavorable à la protection et exploitable.

(b) Le cas défavorable à la protection et inexploitable.

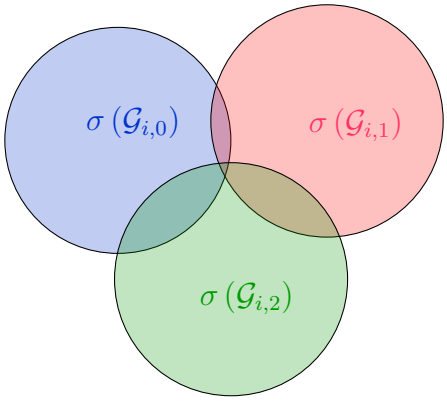
(c) Le cas favorable à la protection.

que l'on schématise, en ne représentant que 3 composantes pour plus de clarté, avec la Figure 3.4 :

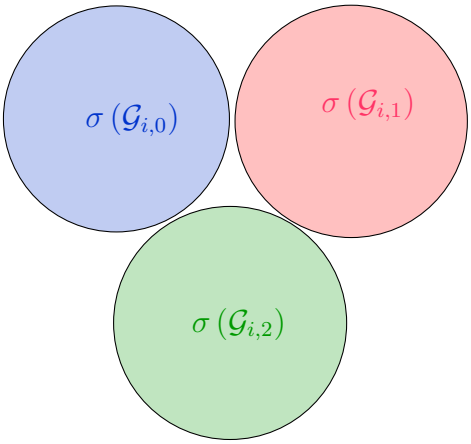
A partir d'ici, on utilise le vecteur $\mathcal{G}_i$ sans les redondances. La matrice de covariance de $\mathcal{G}_i$ est $\Psi_i = \mathbb{E}(\mathcal{G}_i\mathcal{G}_i^T)$ de dimension $(2^{i+1} - 1) \times (2^{i+1} - 1)$. Elle est symétrique positive donc $\Psi_i = Q_i D_i Q_i^T$ où Q_i est orthogonale et D_i est diagonale. Les valeurs propres positives de Ψ_i sont $\lambda_0, \lambda_1, \ldots, \lambda_{2^{i+1}-2}$ ordonnées dans le sens décroissant.



(a) Le cas défavorable à la protection et exploitable. Il y a une intersection importante.



(b) Le cas défavorable à la protection et inexploitable. Il y a une petite intersection.



(c) Le cas favorable à la protection. Il n'y a pas d'intersection.

Figure 3.4 – Evolution de l'intersection des Hammings décorrélés

3.2.2 Résultats et interprétation

Les séquences de Hamming proviennent de l'Algorithme 2 décrit dans la Section 2.3.2 programmé en C. Le calcul exhaustif de la moyenne et de la matrice de covariance pour chaque clef de $i+1$ bits a nécessité un calcul parallèle sur GPU. Un travail conséquent de programmation C/CUDA [80] est à l'origine de l'automatisation de l'analyse grâce à une implémentation sur une carte Nvidia Tesla K40C. On utilise une décomposition LDLt en double précision programmée sur GPU en batch computing [3] pour obtenir $\sqrt{\Gamma_{K,i}}$. L'inversion s'obtient en résolvant les équations $\sqrt{\Gamma_{K,i}}X = b_t$ pour $0 \leq t \leq i$ où b_t est le vecteur de la base canonique dont toutes les coordonnées sont nulles à l'exception de la t-ième qui vaut 1.

Sur *sagemath*[92] on fait :

- La concaténation des vecteurs G_K^i pour obtenir $\widetilde{\mathcal{G}}_i$,
- La suppression des redondances du vecteur $\widetilde{\mathcal{G}}_i$ pour obtenir $\mathcal{G}_i$,
- Le calcul et la diagonalisation de la matrice $\Psi_i = \mathbb{E}(\mathcal{G}_i\mathcal{G}_i^T)$.

Dans les tableaux de 3.1 à 3.6, on indique les résultats pour différents nombres de moduli ($5 \leq n \leq 12$) et différents nombres de distance de Hamming ($5 \leq i \leq 10$). Pour l'analyse, on a besoin du nombre de valeurs propres nulles (Nb VP $= 0$) et de la dimension de la matrice Ψ_i. On considère qu'une valeur inférieure à 10^{-10} est un zéro numérique. On donne aussi la valeur propre maximale (max VP) et la valeur propre minimale non nulle (Min VP$\neq$ 0). La dernière ligne de chaque tableau est le rapport $\dfrac{\text{Max VP}}{\text{Min VP}}$. Ce ratio permet de comparer l'évolution d'apparition des valeurs propres nulles entre les différentes tailles de base RNS. On peut voir alors l'éloignement de MinVP relativement à MaxVP et prévoir si des valeurs propres risquent d'apparaître.

Le Tableau 3.1 montre que 5 distances de Hamming ne sont pas suffisantes pour établir une relation forte entre les gaussiennes. Cependant, il est à noter que le rapport Max/Min entre plus grande et plus petite valeurs propres est relativement plus important pour RNS5, en comparaison avec RNS$\geq$ 6. En utilisant une information plus importante engendrée par 6 ou 7 distances de Hammings (cf. Tableau 3.2 et Tableau 3.3), il n'est toujours pas possible d'observer des valeurs propres nulles et donc toujours pas assez d'information pour établir une relation forte. En comparant les tableaux 3.1, 3.2 et 3.3, le ratio Max/Min est en croissance quasi-continue en fonction du nombre de distances de Hamming exploitées. Cette augmentation du rapport Max/Min est encore plus remarquable pour RNS6 et RNS5. Pour ce dernier, la plus petite valeur propre s'approche sensiblement de 0.

RNS	5	6	7	8	9
dimension	15	15	15	15	15
Nb VP $= 0$	0	0	0	0	0
Max VP	1.912182	1.67919	1.53871	1.53094	1.504195
Min VP $\neq 0$	0.372273	0.435199	0.479538	0.485067	0.496479
$\frac{\text{Max VP}}{\text{Min VP}}$	5.136505	3.858442	3.208734	3.156141	3.029725

Table 3.1 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 4$ (Clef de 5 bits).

RNS	5	6	7	8	9
dimension	31	31	31	31	31
Nb VP $= 0$	0	0	0	0	0
Max VP	2.33897	1.80384	1.5968	1.53576	1.51206
Min VP $\neq 0$	0.22111	0.40126	0.47274	0.48324	0.49308
$\frac{\text{Max VP}}{\text{Min VP}}$	10.57831	4.49544	3.37776	3.17805	3.06656

Table 3.2 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 5$ (Clef de 6 bits).

RNS	5	6	7	8	9
dimension	63	63	63	63	63
Nb VP $= 0$	0	0	0	0	0
Max VP	3.02268	1.99967	1.69389	1.55472	1.52124
Min VP $\neq 0$	0.092169	0.3359	0.45159	0.4767	0.49163
$\frac{\text{Max VP}}{\text{Min VP}}$	32.79497	5.95317	3.75095	3.26142	3.09428

Table 3.3 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 6$ (Clef de 7 bits).

A partir d'une information induite par 8 distances de Hamming (cf. Tableau 3.4 et Tableau 3.5), plusieurs valeurs propres nulles apparaissent pour RNS5 mais il faut 9 distances de Hamming pour avoir un peu plus de la moitié des valeurs propres nulles. Pour RNS5, à chaque distance de Hamming supplémentaire, le nombre de valeurs propres nulles augmente fortement mais pas suffisamment en comparaison avec la dimension de la matrice qui augmente exponentiellement. A partir de 10 distances de Hamming (cf. Tableau 3.6), RNS6 commence aussi à avoir des valeurs propres nulles. En se basant les résultats de RNS5 et RNS6, on peut conjecturer qu'il faudrait avoir une analyse sur 12 distances de Hamming afin d'avoir des valeurs propres nulles pour RNS7. Cependant, on remarque que l'augmentation de la dimension est telle qu'elle rendrait impossible l'exploitation des valeurs propres nulles fondée sur l'information induite par plus de distances de Hamming.

RNS	5	6	7	8	9
dimension	127	127	127	127	127
Nb VP $= 0$	4	0	0	0	0
Max VP	4.2469	2.3847	1.80301	1.59316	1.53039
Min VP $\neq 0$	0.00019	0.21543	0.40419	0.46608	0.488448
$\frac{\text{Max VP}}{\text{Min VP}}$	22352.11	11.0695	4.4608	3.4182	3.1332

Table 3.4 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 7$ (Clef de 8 bits).

RNS	5	6	7	8	9
dimension	511	511	511	511	511
Nb VP $= 0$	260	0	0	0	0
Max VP	5.9848	3.1228	2.0282	1.6508	1.5506
Min VP $\neq 0$	0.1979	0.0676	0.3339	0.4397	0.4796
$\frac{\text{Max VP}}{\text{Min VP}}$	30.2415	46.1953	6.0743	3.7544	3.2331

Table 3.5 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 8$ (Clef de 9 bits).

Pour RNS$\geq$ 8, on constate que la différence entre valeurs propres est sensiblement stable. En augmentant le nombre de distance de Hamming, il sera donc difficile de faire apparaître plus de valeurs propres nulles. Et celles-ci risquent d'être en nombre insuffisant par rapport à la dimension de la matrice pour pouvoir trouver une relation forte. Le Tableau 3.6 conforte ainsi une protection robuste de RNS5-RNS12 contre toute attaque forte fondée sur le modèle gaussien des distances de Hamming.

RNS	5	6	7	8	9	10	11	12
dimension	1023	1023	1023	1023	1023	1023	1023	1023
Nb VP $= 0$	772	100	0	0	0	0	0	0
Max VP	9.3158	4.3339	2.4378	1.7784	1.5869	1.5266	1.5112	1.504
Min VP $\neq 0$	1.0894	0.0026	0.1985	0.3986	0.4696	0.4893	0.4942	0.4982
$\frac{\text{Max VP}}{\text{Min VP}}$	8.5513	1666.88	12.2811	4.4616	3.3793	3.1200	3.0579	3.0189

Table 3.6 – Nombre de Valeurs propres nulles ($< 10^{-10}$) pour $i = 9$ (Clef de 10 bits).

Au-delà de cette première analyse, nous remarquons plusieurs faits notoires :

- Quand on utilise 10 Hamming d'information, on conforte le résultat de la Section 2.5.2.1 qui estimait que $n > 7$ était un bon nombre de moduli pour une protection efficace.
- Le premier Hamming $H_{0,K}$ a un statut particulier car c'est un poids de Hamming. Il sera commun à toutes les clefs. Si on supprime cette valeur dans le calcul, les résultats ne changent pas. On a seulement une valeur propre nulle en moins.

- Pour 10 distances de Hamming d'information, le cas RNS5 se place dans un cas défavorable à la protection. Il y a des relations fortes mais elles sont difficiles à établir. RNS5 se situe entre les deux cas des Figures 3.4a et 3.4b.

- RNS6 se situe dans la situation de la Figure 3.4a. C'est un cas défavorable mais les relations ne sont pas exploitables.

- Pour RNS$\geq$ 7, on se trouve dans le cas favorable à la protection (cf. Figure 3.4c).

Pour pouvoir établir des relations fortes entre les variables aléatoires, il faudrait atteindre une proportion non négligeable de valeurs propres nulles. Plus il y a de valeurs propres nulles, plus la relation sera forte entre les Hamming. Deux situations me paraissent exploitables :

- Imaginons que l'on se retrouve dans le cas parfait où le nombre de valeurs propres non nulles est égale à $i+1$, c'est-à-dire au nombre de Hammings, alors cela signifierait qu'il n'y a pas d'aléa lié à la clef. Dans ce cas, on change seulement de moyenne et de matrice de covariance.

- Une relaxation du problème précédent reviendrait à avoir un certain nombre de directions privilégiées (proche de $i+1$) avec un supplément de bruit faible. Dans ce dernier cas, on pourrait avoir un protocole de dé-bruitage puis établir une relation forte.

Ces deux cas figures exploitables n'ont pas été mis en evidence. On a montré que l'on ne peut pas réduire suffisamment l'espace de probabilité pour avoir une relation forte entre les Hamming. Le système de moduli aléatoires est donc résistant à une analyse qui établit une relation forte entre les variables aléatoires. Malgré le modèle gaussien, cette étude montre les limites de l'exploitation d'une dépendance forte. Cela justifie l'étude du Chapitre 2 et ce que l'on fait dans les sections suivantes sur la dépendance faible.

3.3 Une analyse forte de la dépendance faible : DPA square

On parle de dépendance faible d'une variable aléatoire X par rapport à Y lorsque les paramètres de la loi de X dépendent de Y. On parlera de dépendance faible dans notre étude lorsque la loi du vecteur $H_k^i = (H_{0,k}, ..., H_{i,k})^T$ dépend du secret.

Grâce au comportement gaussien des distances de Hamming annoncé dans la propriété γ de la Section 2.4.1, il n'est pas absurde de supposer que le vecteur H_K^i des distances de Hamming a une distribution normale multivariée. Avec cette hypothèse gaussienne, H_K^i

est complètement déterminé par le vecteur moyen et sa matrice de covariance. La Section 3.3.1 présente les bases mathématiques pour appliquer la DPA Square. La Section 3.3.2 rappelle la méthodologie utilisée et l'illustre sur un algorithme. La Section 3.3.3 présente les résultats numériques de l'analyse par DPA square.

3.3.1 Base théorique de la DPA Square

Alors que la DPA est une attaque utilisant le vecteur moyen de H_K^i sur les réalisations à K fixé, la DPA Square est une attaque sur sa matrice de covariance. La DPA square a un gros avantage sur la DPA parce qu'elle tient compte de l'information mutuelle donnée par les différentes distances de Hamming. Le nom de DPA Square est choisie au lieu de second-order DPA car cette dernière est déjà utilisée pour un autre concept de DPA dans [55, 75]. A notre connaissance, la second-order DPA est une notion différente qui utilise les différences de différences des distances de Hamming tandis que la DPA square prend en compte la différence des éléments de la matrice de covariance.

L'échantillon nécessaire pour réussir une DPA est imprévisible dans le cas des moduli aléatoires (cf. Figure 2.6). Ce ne sera pas le cas pour la DPA square qui donnera lieu à une interprétation plus intuitive dans la Section 3.3.3. On pourra donc quantifier asymptotiquement le nombre d'observation nécessaires pour une attaque. Cette contribution est possible grâce à la DPA square contrairement à la DPA qui est beaucoup moins concluante.

Supposons que $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ est gaussien, le vecteur des moyennes sera $\mathbb{E}(H_K^i)$ et notons $\sigma^2(H_{t,K})$ la variance de chaque coordonnée pour $0 \leq t \leq i$. Pour une clef fixée K, étudier la matrice de covariance de H_K^i est équivalent à étudier la matrice de covariance de $Y_K^i = (Y_{0,K}, ..., Y_{i,K})^T$ avec $Y_{t,K} = \dfrac{H_{t,K} - \mathbb{E}(H_{t,K})}{\sigma(H_{t,K})}$ pour $0 \leq t \leq i$. On note $\Lambda_{K,i} = \mathbb{E}\left(Y_K^i {Y_K^i}^T\right)$ qui donne

$$\Lambda_{K,i} = \begin{pmatrix} var(Y_{0,K}) & cov(Y_{0,K}, Y_{1,K}) & \dots & cov(Y_{0,K}, Y_{i,K}) \\ cov(Y_{1,K}, Y_{0,K}) & var(Y_{1,K}) & \dots & cov(Y_{1,K}, Y_{i,K}) \\ \vdots & \vdots & \ddots & \vdots \\ cov(Y_{i,K}, Y_{0,K}) & cov(Y_{i,K}, Y_{1,K}) & \dots & var(Y_{i,K}) \end{pmatrix}.$$

En utilisant un échantillon de taille S, $\Lambda_{K,i}$ est approximée par $\widetilde{\Lambda}_{K,i}$ et on calcule $\widetilde{\Lambda}_{K',i}$ associé à une clef K' :

$$\widetilde{\Lambda}_{K,i} = \frac{1}{S}\sum_{j=1}^{S} Y_K^i(C^j) Y_K^i(C^j)^T, \quad \text{et } \widetilde{\Lambda}_{K',i} = \frac{1}{S}\sum_{j=1}^{S} Y_{K'}^i(C^{j+S}) Y_{K'}^i(C^{j+S})^T.$$

A chaque lancement d'un $j$ème calcul, $Y_K^i(C^j)$ est une observation associée à la clef K et $Y_{K'}^i(C^j)$ est une simulation associée à la clef K'. Le terme $+S$ dans C^{j+S} exprime l'indépendance entre les configurations de moduli $\{C^j\}_{1\leq j\leq S}$ et $\{C^{j+S}\}_{1\leq j\leq S}$. C'est le résultat de l'indépendance de la suite complète $C = \{C^1, C^2, \ldots, C^S, C^{S+1}, \ldots, C^{2S}\}$.

On calcule la distance entre les deux matrices $\widetilde{\Lambda}_{K,i}$ et $\widetilde{\Lambda}_{K',i}$ en utilisant la norme de Frobenius $\|\cdot\|$. La norme de Frobenius d'une matrice carrée A de taille m est

$$\|A\| = \big(tr(A^T A)\big)^{1/2} = \sqrt{\sum_{0\leq l_1,l_2<m} |A_{l_1,l_2}|^2}.$$

Grâce au théorème central limite, on peut supposer que $\widetilde{\Lambda}_{K,i} = \Lambda_{K,i} + \delta_{K,i}$ et $\widetilde{\Lambda}_{K',i} = \Lambda_{K',i} + \delta_{K',i}$ où $\sqrt{S}\delta_{K,i}$ et $\sqrt{S}\delta_{K',i}$ sont les matrices qui convergent asymptotiquement vers des variables Gaussiennes et centrées avec des variances plus petites ou égales à 1. On définit alors la DPA square par

$$\text{DPA2} = 2\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2. \tag{3.14}$$

En posant $\Lambda_{K,i} = \{k_{l_1,l_2}\}_{0\leq l_1,l_2\leq i}$, $\Lambda_{K',i} = \{k'_{l_1,l_2}\}_{0\leq l_1,l_2\leq i}$, $\delta_{K,i} = \{\delta_{l_1,l_2}\}_{0\leq l_1,l_2\leq i}$ et $\delta'_{K,i} = \{\delta'_{l_1,l_2}\}_{0\leq l_1,l_2\leq i}$, on obtient

$$\begin{aligned}\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2 &= \sum_{0\leq l_1,l_2\leq i} (k_{l_1,l_2} - k'_{l_1,l_2} + \delta_{i,j} - \delta'_{i,j})^2 \\ &\leq 2\sum_{0\leq l_1,l_2\leq i} (k_{l_1,l_2} - k'_{l_1,l_2})^2 + (\delta_{l_1,l_2} - \delta'_{l_1,l_2})^2\end{aligned}$$

et comme $\sqrt{S}(\delta_{l_1,l_2} - \delta'_{l_1,l_2})$ a asymptotiquement une distribution normale G_{l_1,l_2} avec une variance plus petite ou égale à 2 alors, asymptotiquement,

$$\begin{aligned}\mathbb{E}\left[\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2\right] &\leq 2\sum_{0\leq l_1,l_2\leq i} (k_{l_1,l_2} - k'_{l_1,l_2})^2 + 2\mathbb{E}\left[\sum_{0\leq l_1,l_2\leq i} (\frac{\sqrt{2}}{\sqrt{S}} G_{l_1,l_2})^2\right] \\ &\leq 2\sum_{0\leq l_1,l_2\leq i} (k_{l_1,l_2} - k'_{l_1,l_2})^2 + \frac{4}{S}\sum_{0\leq l_1,l_2\leq i} \mathbb{E}\left[G_{l_1,l_2}^2\right]. \\ \mathbb{E}\left[\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2\right] &\leq 2\sum_{0\leq l_1,l_2\leq i} (k_{l_1,l_2} - k'_{l_1,l_2})^2 + \frac{4(i+1)^2}{S} \qquad (3.15) \\ &\leq 2\|\Lambda_{K,i} - \Lambda_{K',i}\|^2 + \frac{4(i+1)^2}{S}. \qquad (3.16)\end{aligned}$$

Cette dernière expression indique qu'on a besoin d'avoir une valeur de S suffisamment grande pour réussir une attaque et diminuer le terme d'erreur asymptotique $\dfrac{4(i+1)^2}{S}$

quand il est comparé à $2\|\Lambda_{K,i} - \Lambda_{K',i}\|^2$. Cette attaque est donc asymptotique.

3.3.2 Méthodologie

On rappelle la méthodologie indiquée dans la Section 2.4.2.1 qu'on adapte ici pour la DPA square. On note $K = \sum_{\ell=0}^{d-1} b_\ell 2^{d-1-\ell}$ la clef de d bits que l'on veut trouver. En supposant que nous connaissions les j premiers bits, on note $K'_j = \sum_{\ell=0}^{j-1} b_\ell 2^{d-1-\ell} + \sum_{\ell=j}^{d-1} 2^{d-1-\ell}$ une clef de d bits qui commence comme K et qui n'a que des uns après. On note C^ℓ, la configuration du ℓ-ième tirage de base RNS. A chaque étape i, $H^i_K(C^\ell)$ est une observation associée à la clef K et $H^i_{K'_j}(C^{\ell+S})$ est une simulation associée à K'_j. Dans le cas où S tirages indépendants seront faits au niveau de l'observation, le terme $+S$ dans $C^{\ell+S}$ exprime l'indépendance entre les configurations de moduli $\{C^\ell\}_{1\leq\ell\leq S}$ et $\{C^{\ell+S}\}_{1\leq\ell\leq S}$. C'est le résultat de l'indépendance de toute la suite $C = \{C^1, C^2, \ldots, C^S, C^{S+1}, \ldots, C^{2S}\}$.

En utilisant la discordance induite par $\left\{H_K(C^\ell)\right\}_{\ell=1}^S$ et par $\left\{H_{K'_0}(C^{\ell+S})\right\}_{\ell=1}^S$, on détermine la position j_1 du premier zéro dans le développement binaire de K. Ensuite, on fait la même chose pour K et K'_{j_1}, ce qui donne la position j_2 du deuxième zéro et ainsi de suite jusqu'à ce qu'on trouve K. Par exemple, pour $K = 11101101110_2$:

- On obtient $j_1 = 3$ de $K = 111\,0\,1101110_2$ et $K'_0 = 11111111111_2$.
- On obtient $j_2 = 6$ de $K = 111011\,0\,1110_2$ et $K'_3 = 11101111111_2$.
- On obtient $j_3 = 10$ de $K = 1110110111\,0\,_2$ et $K'_6 = 11101101111_2$.

Formellement on pose $\min(\emptyset) = d$ puis

$$j_0 = 0, \quad j_p = \min\left\{ i > j_{p-1} \,\middle|\, f_i\left(\left\{H^i_K(C^\ell), H^i_{K'_{j_p}}(C^{\ell+S})\right\}_{\ell=1}^S\right) > \mathcal{T}_i \right\} \tag{3.17}$$

et la clef est trouvée quand $j_p = d$, ce qui signifie que $K = K'_{j_{p-1}}$. $\mathcal{T}_i > 0$ est le seuil de distinction et f_i est un différenciateur. A partir de l'inéquation (3.15), On définit

- Le seuil de distinction $\mathcal{T}_i = \dfrac{4(i+1)^2}{S}$
- Le differentiateur $f_i\left(\left\{H^i_K(C^\ell), H^i_{K'_{j_p}}(C^{\ell+S})\right\}_{\ell=1}^S\right) = 2\|\Lambda_{K,i} - \Lambda_{K',i}\|^2$.

Dans le pire cas ($K = 2^{d-1}$), cette méthode requiert $d-1$ hypothèses au lieu 2^{d-1} hypothèses.

Pour avoir une attaque efficace basée sur la DPA Square, $2\|\Lambda_{K,i} - \Lambda_{K',i}\|^2$ doit être plus grand que $\dfrac{4(i+1)^2}{S}$. Comme on ne connaît pas la valeur exacte de $2\|\Lambda_{K,i} - \Lambda_{K',i}\|^2$,

on la remplace numériquement par $\mathrm{DPA2}(K, K', i) = 2\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2$. Puisque les clefs K et K' ont les mêmes j premiers bits alors $\left(\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\right)_{0 \leq l_1 < j, 0 \leq l_2 < j} = 0_{j \times j}$. De plus les termes de covariance $\left(\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\right)_{0 \leq l_1 \leq i, 0 \leq l_2 < j}$ et $\left(\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\right)_{0 \leq l_1 < j, 0 \leq l_2 \leq i}$ sont négligeables par rapport au terme de variance $\left(\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\right)_{j \leq l_1 \leq i, j \leq l_2 \leq i, l_1 = l_2}$ à cause de la randomisation des moduli.

$$\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i} \approx \begin{pmatrix} 0 & \approx 0 \\ \approx 0 & \left(\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\right)_{j \leq l_1 \leq i, j \leq l_2 \leq i} \end{pmatrix}$$

Le différenciateur numérique et le seuil numérique sont donc donnés respectivement par les termes $2\|\widetilde{\Lambda}_{K,i} - \widetilde{\Lambda}_{K',i}\|^2$ et $\dfrac{4(i+2-j)^2}{S}$. On résume la méthodologie de l'attaque avec l'Algorithme 4.

Algorithm 4 Attaque DPA2

Require: K une clef que l'on veut découvrir.
Ensure: K' qui vaut K si l'attaque réussie.

```
d =nombre de bits de K
K' = 2^d − 1 une clef avec seulement des 1 dans sa représentation binaire de taille d
i = 0
while i + 1 < d do
    shift = i + 1
    while i + 1 < d et DPA2(K, K', i) ≤ 4(i + 2 − shift)^2 / S do
        i = i + 1
    end while
    if i+1<d then
        K' = K' XOR 2^(d−(i+1))
    end if
end while
```

3.3.3 Résultats numériques de la DPA Square

La figure 3.1 illustre, pour exemple, une attaque sur la clef 0xdeeefbf7 où les bits 2, 7, 11, 15, 21 et 28 valent zéro. Cette figure montre qu'attaquer RNS5 nécessite $S \geq 4000$ pour avoir un saut de la DPA2 au-dessus de l'erreur estimée (en utilisant deux calculs indépendants par la méthode de Monte Carlo) et de la valeur asymptotique $\dfrac{4(i+1)^2}{S}$ où

$i+1$ est la taille du morceau de la clef attaquée.

En ce qui concerne RNS10, on a besoin de $S \geq 2500000$ pour réussir la DPA square que l'on illustre sur deux bits qui changent sur la Figure 3.2.

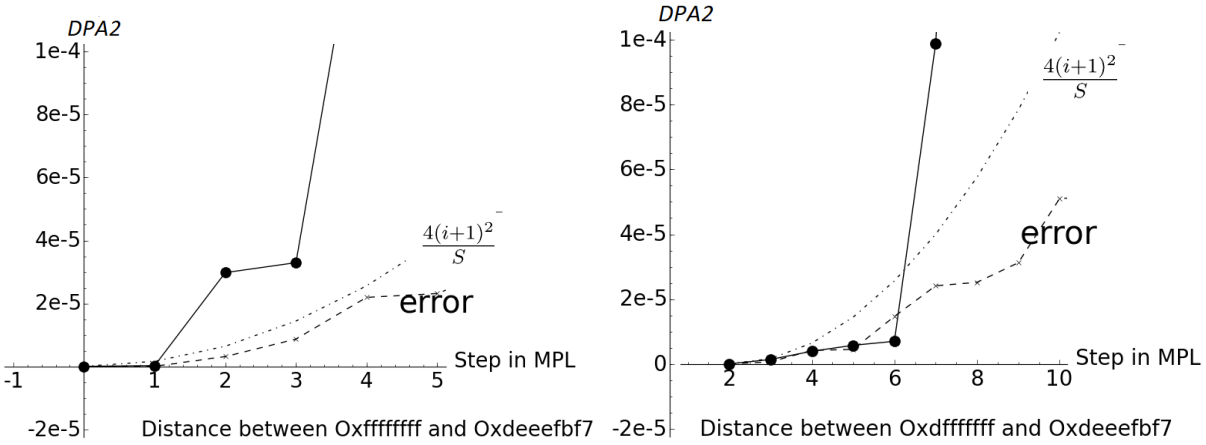


Figure 3.2 – DPA2 sur RNS10 sur une ECC112 avec $S = 2500000$: Chaque nouveau saut au dessus de $\frac{4(i+1)^2}{S}$ donne l'indice du bit qui est égal à zéro.

La Figure 3.3 montre l'évolution en moyenne du nombre d'observations S pour effectuer une attaque. On peut remarquer que le nombre d'échantillons pour réussir la DPA square par rapport au nombre de moduli est monotone et prévisible de l'ordre de $O\left(\frac{(2n)!}{n!^2}\right)$ qui est le nombre de combinaisons en RNSn. On montre donc le lien fort entre la taille de l'échantillon et le cardinal du nombre de configurations possibles. On fait un tirage avec remise donc il peut y avoir plusieurs cas qui ressortent. La probabilité qu'il y ait au moins trois fois la même configuration est approximativement de 8%. C'est pour cela que le nombre de tirages doit être supérieur à $\frac{(2n)!}{(n!)^2}$. Il reste clair que plus on augmente le nombre de moduli plus il devient difficile de retrouver la clef cible. Finalement, on montre les résultats sur une implémentation d'une courbe d'Edwards 25519 dans l'Annexe 3.B.

Une attaque DPA square utilisant la matrice de covariance est donc possible. La complexité d'une DPA square est la complexité d'une DPA appliquée au vecteur de taille $(i+1)^2$. La DPA square a l'avantage d'être plus prévisible en termes de traces nécessaires que la DPA. Cependant ce nombre de traces reste encore conséquent. Si un système utilise la méthode des moduli aléatoires, il faut encore de l'ordre de $\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$ traces pour réussir une attaque avec la DPA square.

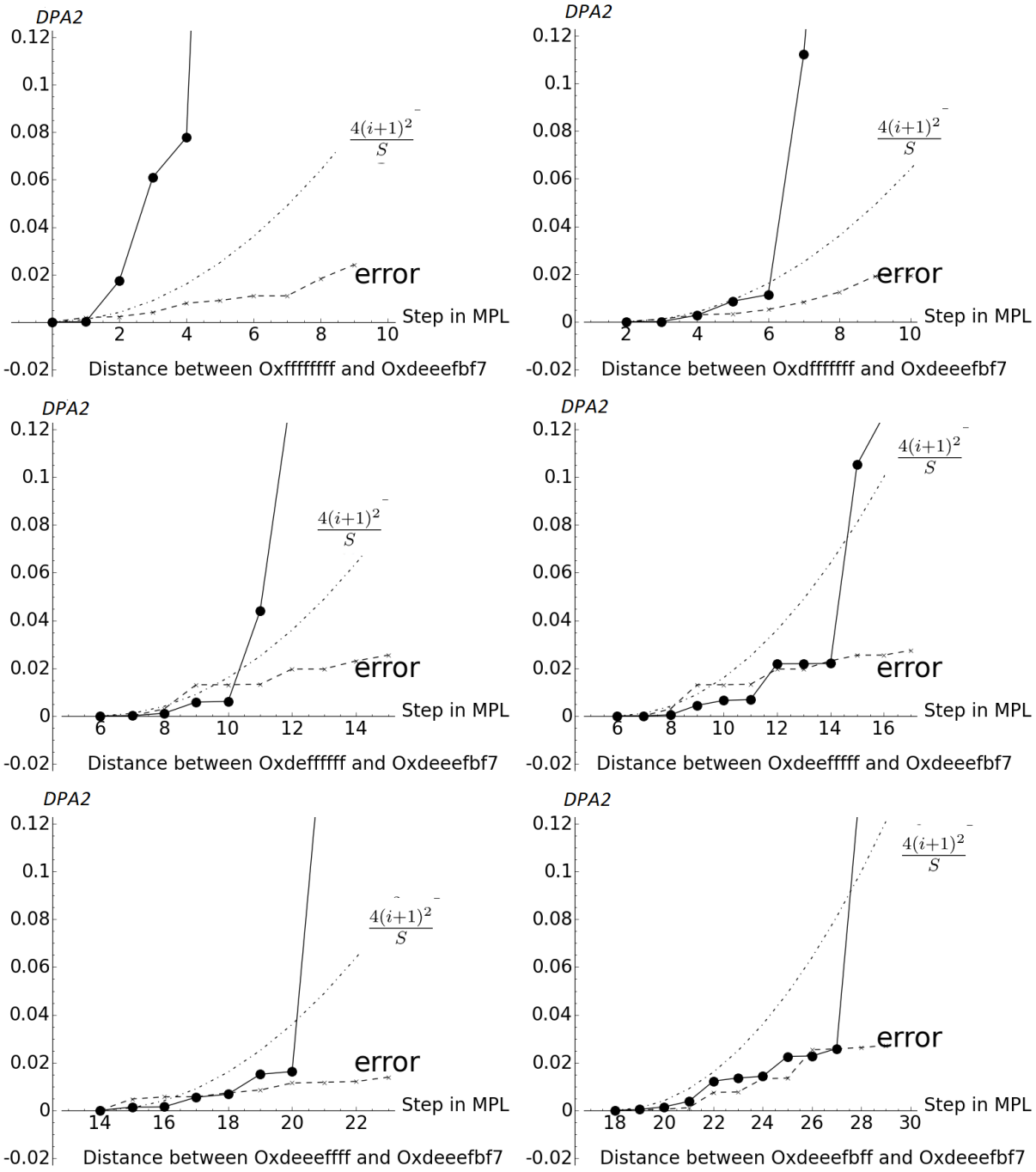


Figure 3.1 – DPA2 sur RNS5 sur une ECC112 avec $S = 4000$: Chaque nouveau saut au dessus de $\frac{4(i+1)^2}{S}$ donne l'indice du bit qui est égal à zéro.

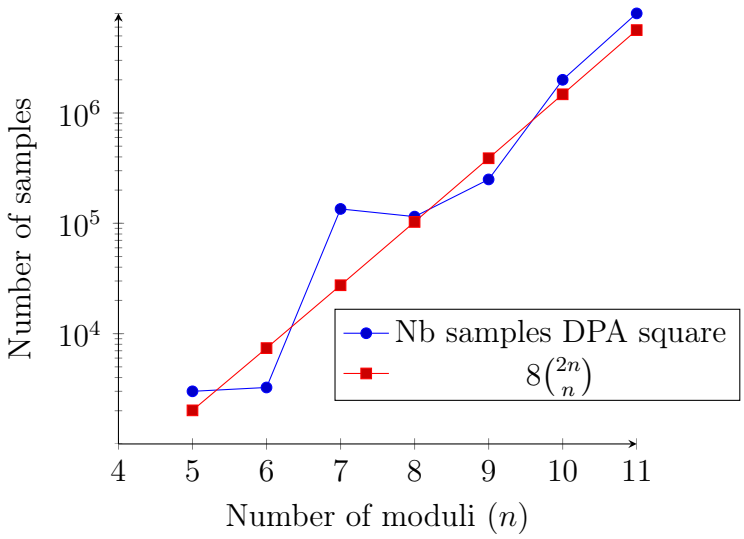


Figure 3.3 – Attaque avec DPA2 sur ECC112: Taille S de l'échantillon pour attaquer les 10 bits premiers bits de 0xdeeefbf7 on fonction du nombre de moduli

3.4 Une analyse faible de la dépendance faible : MLE conditionnel

Des sections précédentes, on peut conclure de la difficulté de retrouver la clef avec une seule trace. Dans Section 3.4.1, on propose de relaxer le problème ; plutôt que de chercher une solution, on cherchera à éliminer les solutions les moins probables. Après la présentation de cette relaxation, on indique que l'hypothèse gaussienne sur $H^i = (H_0, ..., H_i)^T$ est toujours insuffisante pour extraire de l'information. Il faut donc encore restreindre l'ensemble des états de $H^i = (H_0, ..., H_i)^T$ avec un conditionnement. On présente ce conditionnement avec le MLE associé dans la Section 3.4.1. On arrive alors à réduire l'ensemble des clefs candidates. Le décryptage de la machine Enigma par la Bomb est un exemple célèbre de cryptanalyse relaxée [23]. Un des premiers principes de la Bomb est d'éliminer les solutions les moins probables. Dans la Section 3.4.2, on apporte une base théorique associée à cette analyse. La Section 3.4.4.2 présente les résultats numériques.

3.4.1 Relaxation de l'analyse

Dans le cas d'un système cryptographique asymétrique, il arrive qu'on utilise la clef qu'une seule fois. Il faut donc se poser la question de la fragilité du système dans le cas où on ne possède qu'une seule trace ou très peu de traces. Une personne malveillante qui possède une seule trace de Hamming fournie par une clef secrète pourrait tout de même extraire le peu d'information laissée par cette trace.

Pour extraire cette information, on propose une stratégie utilisant les 50% plus mauvais scores donnés par une fonction de différenciation $f^{(k)}$ associée à une clef k.

Remarque : on parle plutôt d'enlever les scores les plus faibles plutôt que de garder les meilleurs scores car un début de clef enlevée ne revient plus alors qu'un début de clef gardée peut partir.

On construit deux suites de listes :

- La liste $\mathbb{T}_i^{before}$ des possibilités de clefs à l'étape i avant différenciation de taille 2^{i_0}.

- La liste $\mathbb{T}_i^{after}$ des possibilités de clefs à l'étape i aprés différenciation de taille 2^{i_0-1}.

L'entier $i_0 \geq 1$ correspond à la taille initiale de la relaxation. Formellement, on définit ces listes de la manière suivante :

On initialise avec 2^{i_0} entiers :

$$\mathbb{T}_{i_0-1}^{before} = \left\{2^{i_0}, \ldots, 2^{i_0+1}-1\right\} \equiv \{1\} \times \{0,1\}^{i_0} \tag{3.18}$$

On sélectionne 2^{i_0-1} entiers qui ont les meilleurs scores :

$$\mathbb{T}_i^{after} = \left\{k \in \mathbb{T}_{i-1}^{before} \,\middle|\, f^{(k)}\left(\left\{H_K\left(C^\ell\right)\right\}_{\ell=1}^{S}\right) > \mathcal{T}^{\mathbb{T}_{i-1}^{before}}\right\} \text{ donc } Card(\mathbb{T}_i^{after}) = 2^{i_0-1} \tag{3.19}$$

$$\equiv \left\{(b_0, \ldots, b_i) \in \mathbb{T}_{i-1}^{before} \,\middle|\, f^{(b_0,\ldots,b_i)}\left(\left\{H_K\left(C^\ell\right)\right\}_{\ell=1}^{S}\right) > \mathcal{T}^{\mathbb{T}_{i-1}^{before}}\right\}$$

On rajoute un bit aux entiers avec le meilleur score :

$$\mathbb{T}_i^{before} = \left\{2k, 2k+1 \,\middle|\, k \in \mathbb{T}_i^{after}\right\} \equiv \mathbb{T}_i^{after} \times \{0,1\} \text{ donc } Card(\mathbb{T}_i^{before}) = 2^{i_0} \tag{3.20}$$

$\mathcal{T}^{\mathbb{T}_{i-1}^{before}}$ est la médiane de $\left\{f^{(k)}\left(\left\{H_K\left(C^\ell\right)\right\}_{\ell=1}^{S}\right) \,\middle|\, k \in \mathbb{T}_{i-1}^{before}\right\}$ pour avoir 50% des meilleurs scores. On choisit 50% pour ne pas augmenter la complexité de l'algorithme à chaque étape i.

Il est important de remarquer que la valeur de i_0 est une valeur fixée qui donne la quantité de distances de Hamming qui vont être utilisées pour faire l'attaque relaxée. A chaque étape, on avance sur les bits de la clef en présupposant que les premiers bits de la clef cible sont dans les 50% que l'on garde. Si cela se passe bien on obtient à la fin un ensemble $\mathbb{T}_d^{after}$ de taille 2^{i_0-1} dans lequel se trouverait la clef cible de taille d. Le problème est de choisir un i_0 assez grand pour pouvoir extraire suffisamment d'information ($i_0+1 \geq 10$ d'après 2.5), mais pas trop grand non plus pour ne pas surcharger les calculs.

On résume ce type d'attaque dans l'Algorithme 5.

Algorithm 5 Attaque Relaxée sur 50% des meilleurs scores

Require: Une trace de Hamming sur une clef K de d bits.
Un entier i_0 tel que $1 \leq i_0 < d$
$i = i_0$
$\mathbb{T}_{i-1}^{before} = \{2^i, \ldots, 2^{i+1} - 1\}$
while $i \leq d$ **do**
Construction de $\mathbb{T}_i^{after}$, ensemble des entiers qui ont les 50% meilleurs scores parmi $\mathbb{T}_{i-1}^{before}$
Construction de $\mathbb{T}_i^{before}$ en rajoutant un bit aux éléments de $\mathbb{T}_i^{after}$.
i=i+1
end while
Ensure: Une liste $\mathbb{T}_d^{after}$ de 2^{i_0-1} éléments dans laquelle peut se trouver la clef K.

3.4.2 Analyse théorique: Gain de la relaxation

Cette partie se base sur le chapitre Statistical Properties of a Sample Average Approximations Estimators Section 5.3.1 du livre de A. Shapiro & al. : Lectures on stochastic programming: modeling and theory[88].

Les auteurs [88] considèrent le problème suivant :

$$\min_{x \in X} \{\phi(x) := \mathbb{E}\left[\Phi\left(x, \xi\right)\right]\} \tag{3.21}$$

qui est équivalent à un problème de maximum si on remplace $\phi(x)$ par $-\phi(x)$.

- X est un sous-ensemble fermé d'un espace $\mathbb{R}^n$. Dans notre cas cela correspond à l'ensemble des entiers de K de $\mathbb{T}_i^{before}$.

- ξ est un vecteur aléatoire dont la distribution P est définie sur $\Theta \subset \mathbb{R}^{i+1}$. Dans notre cas ξ est une suite de distance de Hamming considérées comme des variables aléatoires.

- $\Phi : X \times \Theta \to \mathbb{R}$. Dans notre cas Φ est la fonction de densité sur une variable multivariée gaussienne décrite en 3.24.

On suppose qu'on a un échantillon $\xi^1, \ldots, \xi^S$ de S réalisations du vecteur aléatoire ξ. On estime alors la valeur de $\phi(x)$ en faisant la moyenne de $\Phi(x, \xi^j), j = 1, \ldots, ..., S$. Cela conduit au problème du minimum d'une fonction approximée par la moyenne de l'échantillon (SAA : Sample Approximation Average).

$$\min_{x \in X} \left\{ \hat{\phi}_S(x) := \frac{1}{S} \sum_{j=1}^{S} \Phi\left(x, \xi^j\right) \right\} \tag{3.22}$$

Pour notre étude on voudrait avoir S le plus petit possible et éventuellement $S = 1$. Pour la suite on note

$$v^* = \min_{x \in X} \{\phi(x) := \mathbb{E}\left[\Phi\left(x, \xi\right)\right]\} \text{ et } \hat{v}_S = \min_{x \in X} \left\{ \hat{\phi}_S(x) := \frac{1}{S} \sum_{j=1}^{S} \Phi\left(x, \xi^j\right) \right\}.$$

Pour $\epsilon > 0$, on pose :

$$\mathscr{S}^\epsilon = \{x \in X : \phi(x) \leq v^* + \epsilon\}$$

et

$$\hat{\mathscr{S}}_S^\epsilon = \left\{ x \in X : \hat{\phi}_S(x) \leq \hat{v}_S + \epsilon \right\}.$$

- $\mathscr{S}^\epsilon$ est l'ensemble des clefs qui donne une valeur de ϕ comprise entre v^* et $v^* + \epsilon$. Il correspond à la relaxation du problème de minimum donné par (3.21).
- $\hat{\mathscr{S}}_S^\epsilon$ est l'ensemble des clefs qui donne une valeur de $\hat{\phi}_S$ comprise entre $\hat{v}_S$ et $\hat{v}_S + \epsilon$. Il correspond à la relaxation du problème de minimum donné par (3.22).

Pour les paramètres $\epsilon \geq 0$ et $\delta \in [0, \epsilon]$, on considère l'événement $\left\{ \hat{\mathscr{S}}_S^\delta \subset \mathscr{S}^\epsilon \right\}$. Cet événement signifie que toute solution δ-optimale du SAA (3.22) est une solution ϵ-optimal du vrai problème (3.21). En utilisant le Théorème 5.16 dans [88], on donne une minoration théorique de $P\left(\hat{\mathscr{S}}_S^\delta \subset \mathscr{S}^\epsilon\right)$. Dans notre cas, $P\left(\hat{\mathscr{S}}_S^\delta \subset \mathscr{S}^\epsilon\right)$ permettra d'évaluer la probabilité d'avoir un ensemble de clef dans l'ensemble des valeurs qui sont proches de l'optimum réel avec S réalisations.

Soit ϵ et δ deux nombres positifs. Pour une valeur de δ proche de $-\mathbb{E}\left[\Phi(u(x), \xi) - \Phi(x, \xi)\right]$:

$$1 - P\left(\hat{\mathscr{S}}_S^\delta \subset \mathscr{S}^\epsilon\right) \leq |X| e^{-S\nu(\delta, \epsilon)} \tag{3.23}$$

où

$$\nu(\delta, \epsilon) := \min_{x \in X \setminus \mathscr{S}^\epsilon} \frac{(\epsilon^* - \delta)^2}{2\sigma_x^2}$$

avec $\sigma_x^2 := \mathbb{V}ar\left[\Phi(u(x), \xi) - \Phi(x, \xi)\right]$, $\epsilon^* = \min_{x \in X \setminus \mathscr{S}^\epsilon} \phi(x) - v^*$ et une application $u : X \setminus \mathscr{S}^\epsilon \to X$. Dans notre exemple, u est l'application constante qui envoie une valeur de $X \setminus \mathscr{S}^\epsilon$ sur la clef cible.

Il est remarquable que la majoration (3.23) n'est pas asymptotique en fonction de S. Elle est donc vraie pour toute valeur de S, y compris pour $S = 1$. Elle est donc applicable lors d'une analyse fondée sur une seule trace. Mais le prix à payer est d'avoir une majoration conservative comme l'énoncent les auteurs de [88] p.183: "Although such estimates of the sample size typically are too conservative for a practical use...". La majoration

(3.23) n'est pas utilisable en pratique. Cela est amplifié dans notre cas lorsqu'on veut estimer la valeur de σ_x^2 pour H_t loin de $\mathbb{E}(H_t)$. L'espace d'état étant alors maigre pour une hypothèse gaussienne, les matrices de covariances sont mal conditionnées.

Pourtant la difficulté d'utiliser la majoration (3.23) en pratique n'empêche pas d'employer la relaxation $\mathscr{S}^\epsilon$ de (3.22) pour des attaques type template en faisant appel au MLE. On voit l'élaboration d'une telle stratégie dans la section suivante.

3.4.3 Analyse conditionnelle relaxée

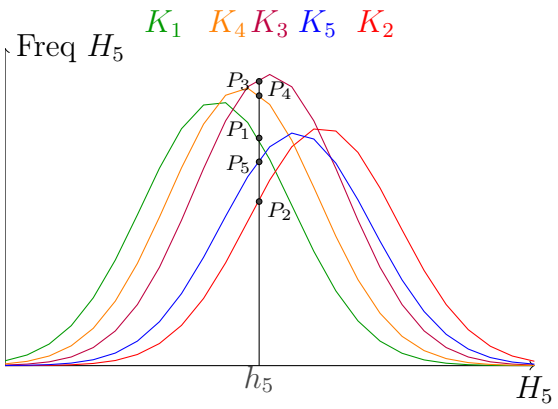
Dans la Section 3.4.1, on a introduit le principe de relaxation pour l'analyse d'une clef cible en introduisant $f^{(k)}$ comme différenciateur dans le cas général. Dans cette section, on précise le choix de ce différenciateur en utilisant une loi gaussienne conditionnelle.

Connaissant S réalisations de $H^i = (H_0, ..., H_i)^T$, on a vu dans la Section 2.5.1 du Chapitre 2 qu'on retrouve la clef en utilisant le modèle gaussien et une valeur de S suffisamment grande. Dans le cas où S est petit, on ne peut pas retrouver la clef. Autrement dit, les clefs retrouvées sont assimilables à un tirage aléatoire. La première idée pour contourner le problème serait d'ajouter une information donc de faire une analyse conditionnelle.

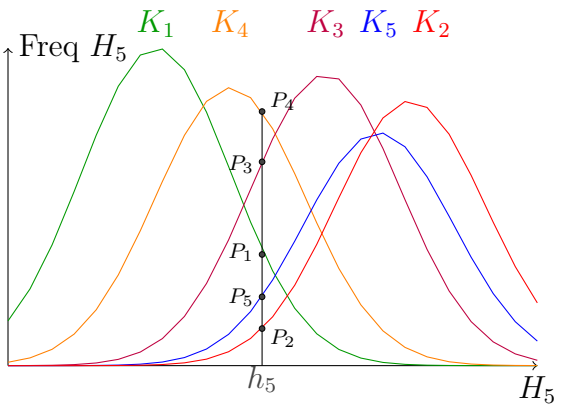
Comme représenté dans la Figure 3.1, alors que dans le cas non conditionné il est difficile de séparer les gaussiennes, rajouter un conditionnement "judicieux" écarte les centres des gaussiennes. On suppose que la clef cible/secrète est la clef K_4 et qu'on obtienne une réalisation avec $H_5 = h_5$ en faisant le calcul cryptographique correspondant à cette clef. Les scores calculés sont marqués par les points P_1, P_2, P_3, P_4 et P_5. On remarque qu'on a plus de chance d'avoir K_4 dans le cas conditionné "$H_0 = h_0$" plutôt que de dans le cas non conditionné.

Le mode de calcul des paramètres des distributions de la Figure 3.1 change selon le conditionnement. Par exemple pour le conditionnement $H_0 = h_0$, les paramètres des distributions gaussiennes H_{5,K_1}, H_{5,K_2}, H_{5,K_3}, H_{5,K_4} et H_{5,K_5} sont calculés en utilisant uniquement les réalisations vérifiant la condition $H_0 = h_0$. Cela revient à établir un dictionnaire contenant les paramètres de gaussiennes indicées par h_0 et les clefs K_1, K_2, K_3, K_4 et K_5. En supposant que K_4 est la clef cible, on peut voir (cf. Figure 3.1) que dans le cas d'une attaque non conditionnée, on ne peut pas la distinguer de K_3 alors qu'elle serait plus distinguable dans le cas conditionné.

Une solution est alors de construire un dictionnaire conditionné par un événement A sur une ou plusieurs composantes du vecteur $H^i = (H_0, ..., H_i)^T$. On se donne une fonction g_A qui dépend du type de conditionnement. La fonction g_A enlève la ou les



(a) Représentation de la fréquence de H_5 non conditionnée.



(b) Représentation de la fréquence de H_5 conditionnée par $H_0 = h_0$

Figure 3.1 – Distribution de H_5 pour cinq clefs différentes: Hypothèse gaussienne conditionnée vs. hypothèse gaussienne non conditionnée.

coordonnées qui interviennent dans le conditionnement:

$$\begin{aligned} g_A \ &: \mathbb{R}^{i+1} \to \mathbb{R}^{m+1} \\ &(x_0, ..., x_i)^T \to (x_{\ell_0}, ..., x_{\ell_m})^T \text{ avec } \ell_k \text{ strictement monotone sur } [\![0, i]\!]. \end{aligned}$$

Par exemple :

- Si $A =$"$H_t = h_t$" ou $A =$"$H_t < \mathbb{E}(H_t)$"
  alors $g_A((x_0, ..., x_i)^T) = (x_0, \ldots, x_{t-1}, x_{t+1}, \ldots, x_i)^T$.
- Si $A =$"$H_0 = h_0$ et $H_1 = h_1$"
  alors $g_A((x_0, ..., x_i)^T) = (x_2, x_3, \ldots, x_i)^T$.

Remarque : L'événement "$H_t < \mathbb{E}(H_t)$" enlève le caractère gaussien de la t-ième coordonnée; il faut donc éliminer la coordonnée x_t.

On construit un dictionnaire en pré-calculant le vecteur moyen et la matrice de covariance du vecteur $g_A(H^i) = g_A((H_0, \ldots, H_i)^T)$ pour un conditionnement A donné. On obtient le couple moyenne et matrice de covariance $(m_{A,k,i}, \Gamma_{A,k,i})$. Pour chaque triplet (A, k, i), on suppose que $g_A((H_0, \ldots, H_i)^T)$ a une distribution normale multivariée avec une densité conditionnée par A :

$$p_{k,i}\left(\mathbf{x}^i | A\right) = \frac{\exp\left(-\frac{(g_A(\mathbf{x}^i) - m_{A,k,i})^T \Gamma_{A,k,i}^{-1} (g_A(\mathbf{x}^i) - m_{A,k,i})}{2}\right)}{\left(\sqrt{2\pi}\right)^{i_A} \sqrt{\det(\Gamma_{A,k,i})}}, \tag{3.24}$$

où $\mathbf{x}^i = (x_0, ..., x_i)^T \in \mathbb{R}^{i+1}$ et i_A la dimension du vecteur $g_A(\mathbf{x}^i)$.

L'analyse MLE conditionné non relaxée se résume, comme dans la Section 2.5.1, pour $i = 0, 1, ..., d-1$ par :

- On calcule la valeur exacte[1] de $(m_{A,k,i}, \Gamma_{A,k,i})$ avec $\frac{(2n)!}{n!^2}$ combinaisons en RNSn, ou on estime $(m_{A,k,i}, \Gamma_{A,k,i})$ en utilisant moins de combinaisons simulées avec la méthode de Monte Carlo. C'est la phase d'apprentissage.
- On observe **une** réalisation $\mathbf{x}^i$ de $H^i = (H_0, ..., H_i)^T$.
- On choisi le secret $K = k$ qui maximise $p_{k,i}\left(g_A\left(\mathbf{x}^i\right)|A\right)$. C'est la phase d'estimation.

Posons

$$K^* = \arg\max_k \left\{p_{k,i}\left(g_A\left(\mathbf{x}^i\right)|A\right)\right\} = \left\{k \,|\forall k' \quad p_{k',i}\left(g_A\left(\mathbf{x}^i\right)|A\right) \leq p_{k,i}\left(g_A\left(\mathbf{x}^i\right)|A\right)\right\} \tag{3.25}$$

Pour $A = \oslash$ (analyse MLE vue dans la Section 2.5.1) ou $A =$ "$H_t = h_t$" ou $A =$ "$H_t < \mathbb{E}(H_t)$", cette méthode donne de très mauvais résultats. En effet, on retrouve des résultats similaires au hasard. Cela s'explique aussi à travers la Figure 3.1 où l'optimisation (3.25) donne de mauvais résultats dans le cas conditionné, bien que meilleur que dans le cas non conditionné.

Malgré le conditionnement, on n'arrive pas à améliorer sensiblement l'attaque MLE pour l'utiliser sur une seule trace ($S = 1$). Il faut donc se donner une perspective moins contraignante. Dans la suite, on choisit d'associer le MLE conditionné avec la relaxation introduite dans la Section 3.4.1. On montre qu'il est plus avantageux d'enlever l'ensemble des clefs candidates qui enregistrent les scores les plus faibles sur la base d'un différenciateur $f^{(k)}$ définie avec la densité gaussienne conditionnée donnée par

$$f^{(k)}\left(\mathbf{x}^i|A\right) = p_{k,i}\left(\mathbf{x}^i|A\right) = \frac{\exp\left(-\frac{(g_A(\mathbf{x}^i)-m_{A,k,i})^T\Gamma_{A,k,i}^{-1}(g_A(\mathbf{x}^i)-m_{A,k,i})}{2}\right)}{\left(\sqrt{2\pi}\right)^{i_A}\sqrt{\det(\Gamma_{A,k,i})}}, \tag{3.26}$$

Ainsi on complète la définition de $\mathbb{T}_i^{before}$ et $\mathbb{T}_i^{after}$ avec ce différenciateur et faire l'analyse relaxée.

3.4.4 Principaux Résultats numériques sur les 10 premiers Hammings

Dans cette section on présente une analyse relaxée uniquement sur les 10 premiers bits. Pour l'instant on n'a pas suffisamment cherché un bon conditionnement pour avoir une

[1] C'est le choix qui a été fait par défaut dans les figures et les tableaux suivants.

attaque efficace sur une clef entière. La Section 3.4.4.1 illustre la séparation des densités avec un conditionnement et la Section 3.4.4.2 montre le niveau d'information obtenu grâce à l'analyse relaxée.

3.4.4.1 Principaux résultats numériques sur la séparation des densités

Dans le cas de notre étude, les gaussiennes non conditionnées se superposent comme montré dans la Figure 3.2). Cela illustre davantage la nécessité d'utiliser de nombreuses traces ($S = \binom{2n}{n}$ où n est le nombre de moduli) pour réussir une attaque avec le MLE (Section 2.5.1 du Chapitre 2). Dans les cas où on conditionne les gaussiennes par "$H_0 = h_0$" (cf; Fig.3.3 et 3.4), on constate une séparation plus ou moins importante entre les densités marginales selon que h_0 obtenue par la clef cible est plus ou moins éloignée de la moyenne $\mathbb{E}(H_0)$.

On a représenté sur les figures précédentes uniquement les lois marginales des H_i. L'étalement ne risque pas d'être suffisant. Il faut donc utiliser le vecteur complet $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ pour avoir des résultats probants. En effet la séparation est plus importante avec les lois jointes qui sont utilisées dans le MLE relaxé.

3.4.4.2 Principaux résultats sur la probabilité de succès de l'analyse MLE relaxée

Dans la suite, on montre que les 10 premiers bits à deviner se trouve dans la liste $\mathbb{T}_{10}^{after}$ avec une probabilité nettement meilleure qu'un choix aléatoire. On définit une chance de succès quand l'entier k (clef) de taille 10 représenté en binaire par $(b_0, ..., b_9)$ obtient un score supérieur à la médiane $\mathcal{T}^{\mathbb{T}_9^{before}}$ des résultats pour toutes les clefs de 10 bits, i.e. $f^{(k)}\left(\left\{H_K\left(C^\ell\right)\right\}_{\ell=1}^{S}\right) > \mathcal{T}^{\mathbb{T}_9^{before}}$. La première Figure 3.5 montre le gain que l'on peut obtenir quand on passe d'un conditionnement faible ("$H_0 \leq \mathbb{E}(H_0)$") à un conditionnement plus fort ("$H_0 = h_0$"). On rappelle les résultats dans le cas où il n'y a pas de conditionnement ("$A = \oslash$"). La deuxième Figure 3.6 distingue la qualité du conditionnement suivant les valeurs de "$H_t = h_t$". Pour obtenir ces figures, on tire aléatoirement (Sauf le cas RNS6 qui est exhaustif) 924 configurations de moduli pour les 512 suites de 10 bits commençant par 1. Ce qui donne des échantillons de taille $924 \times 512 = 473088$. On obtient donc le pourcentage de réussite avec un faible intervalle de confiance (de l'ordre de 0.3% avec 95% de chance) pour RNS$>$ 6. Le dictionnaire des couples moyenne et matrice de covariance contenant $(m_{A,k,i}, \Gamma_{A,k,i})$ pour calculer le MLE conditionné a été construit en amont afin de définir $f^{(k)}$.

Les résultats en RNS5 ne sont pas représentés car il n'y a pas assez de valeurs pour construire une matrice de covariance inversible. Le conditionnement réduit considérablement l'ensemble d'état donc il est trop fort pour garder l'hypothèse gaussienne. En RNS6,

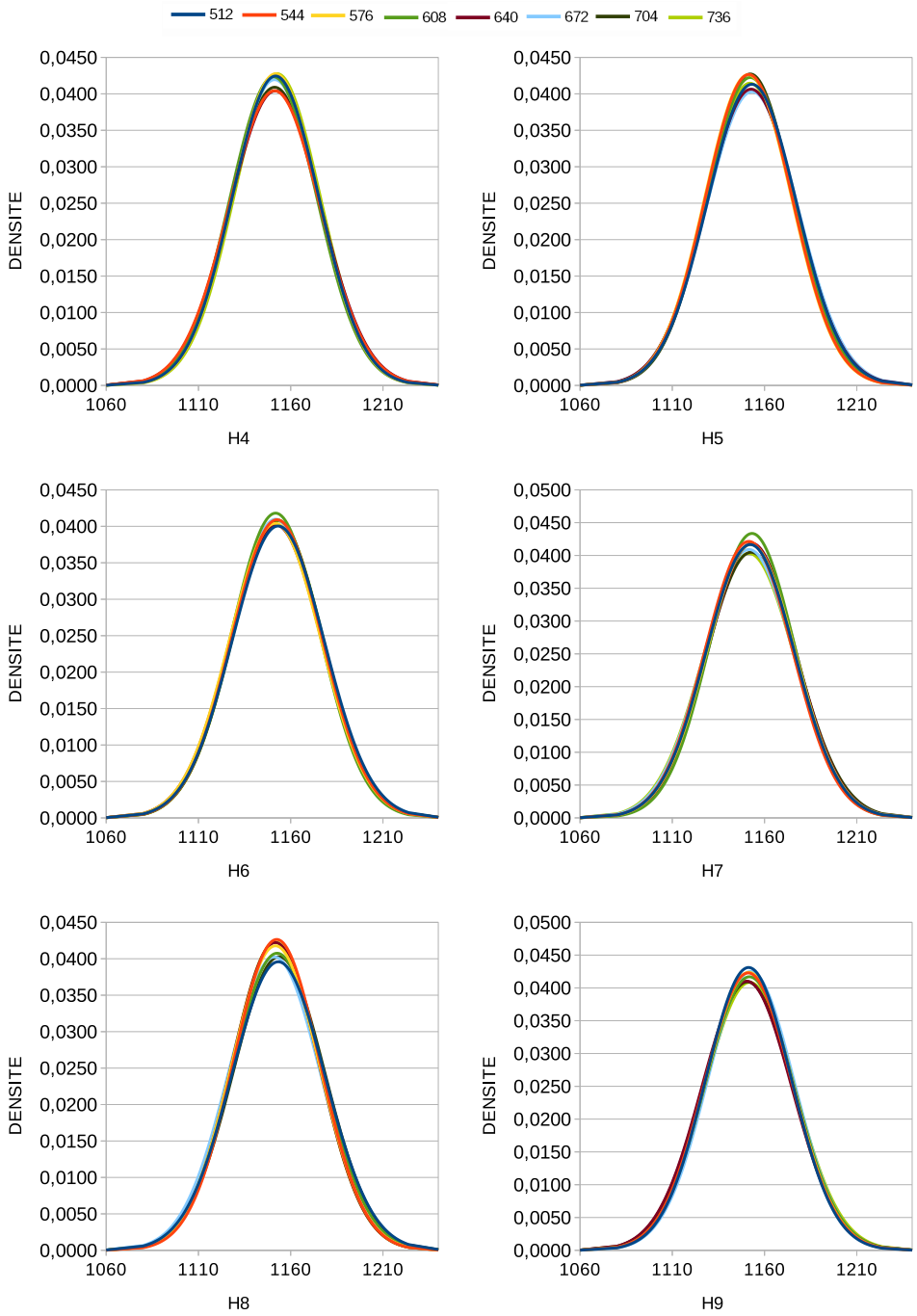


Figure 3.2 – Densité de H_i avec l'hypothèse gaussienne non conditionnée pour différentes valeurs de clef de 10 bits. RNS6

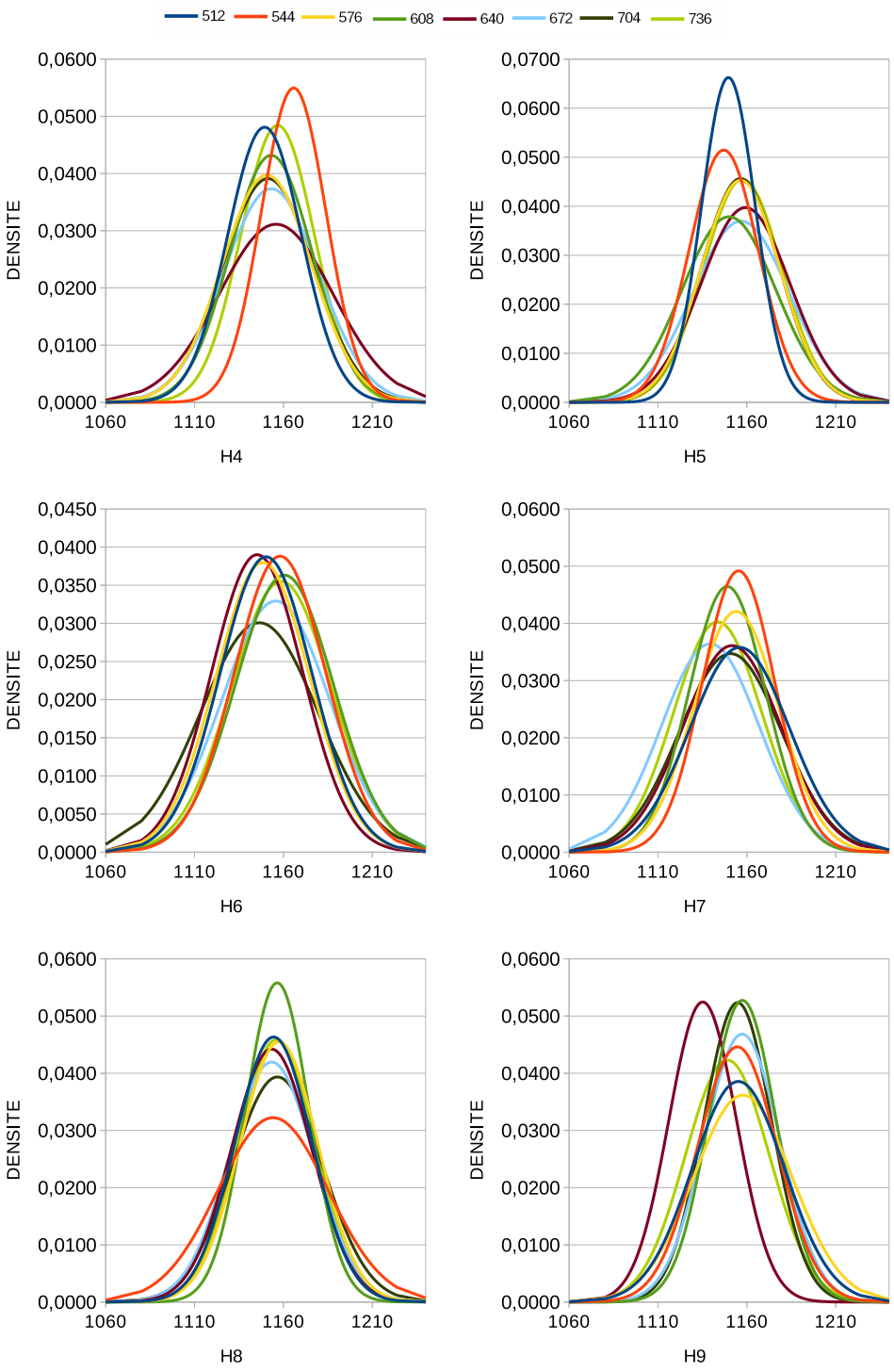


Figure 3.3 – Densité de H_i avec l'hypothèse gaussienne conditionnée par $H_0 = 1151$ proche de $\mathbb{E}(H_0) \approx 1151,26$. RNS6 pour différentes valeurs de clef de 10 bits

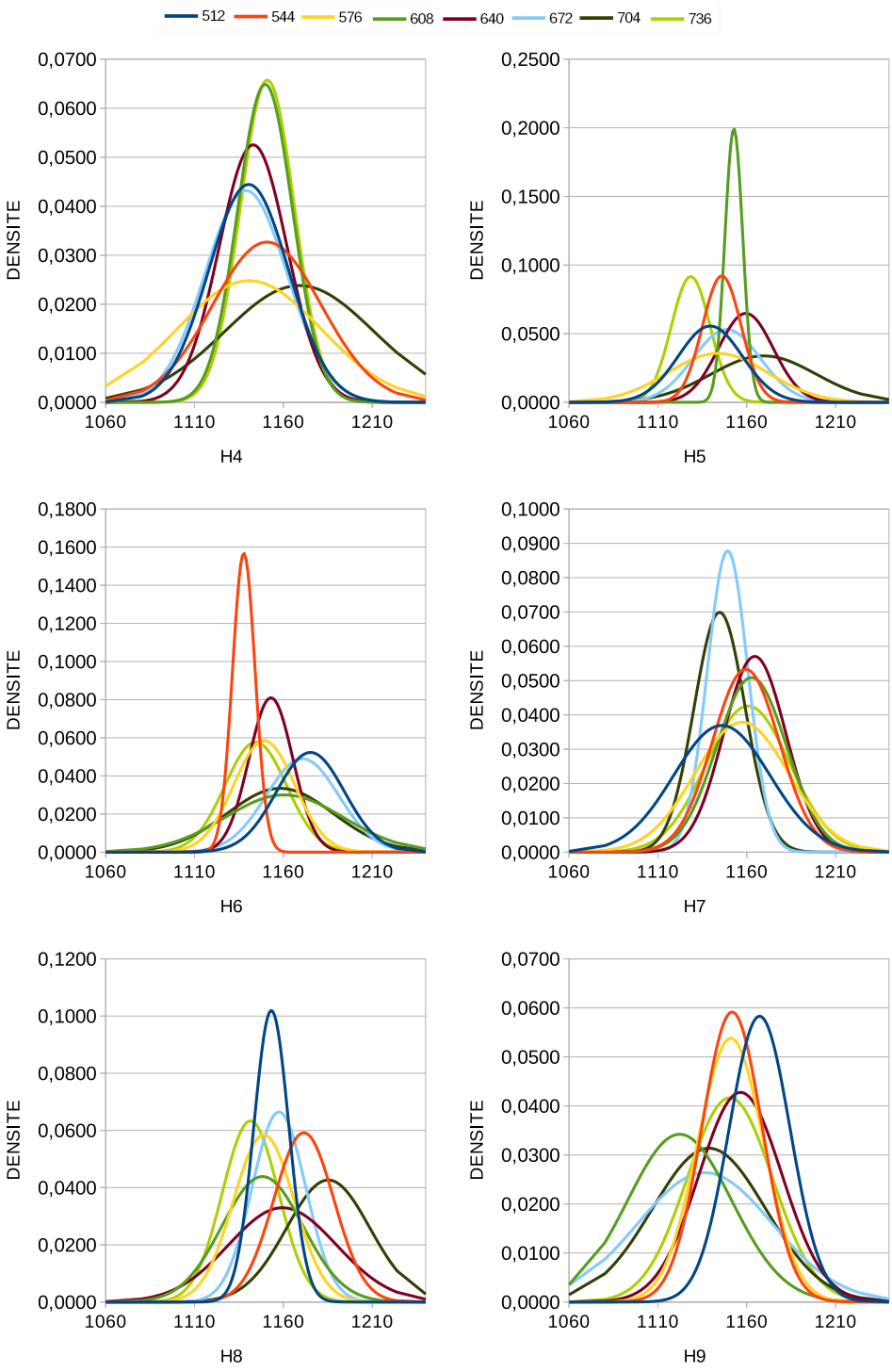


Figure 3.4 – Densité de H_i avec l'hypothèse gaussienne conditionnée par $H_0 = 1100$ loin de $\mathbb{E}(H_0) \approx 1151,26$. RNS6 pour différentes valeurs de clef de 10 bits

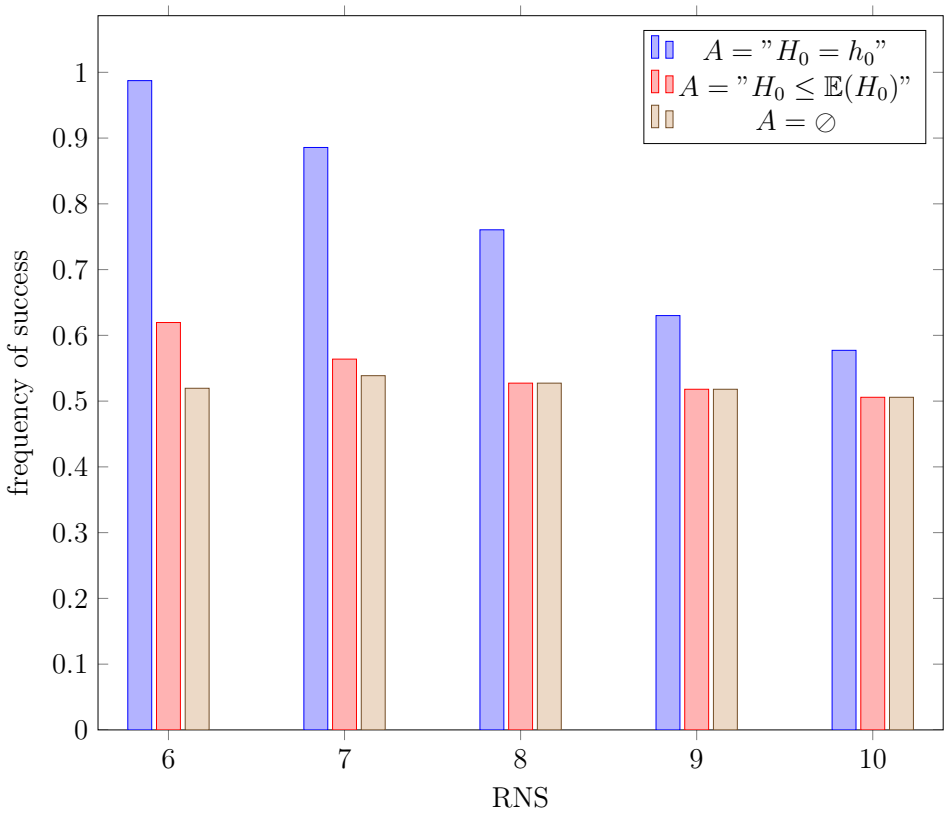


Figure 3.5 – Fréquence d'apparition de la clef dans la moitié de l'ensemble des clefs de 10 bits avec le meilleur score en utilisant le critère de la densité gaussienne conditionnée par $H_0 = h_0$ et $H_0 \leq \mathbb{E}(H_0)$. $i = 9$, Clef de 10 bits.

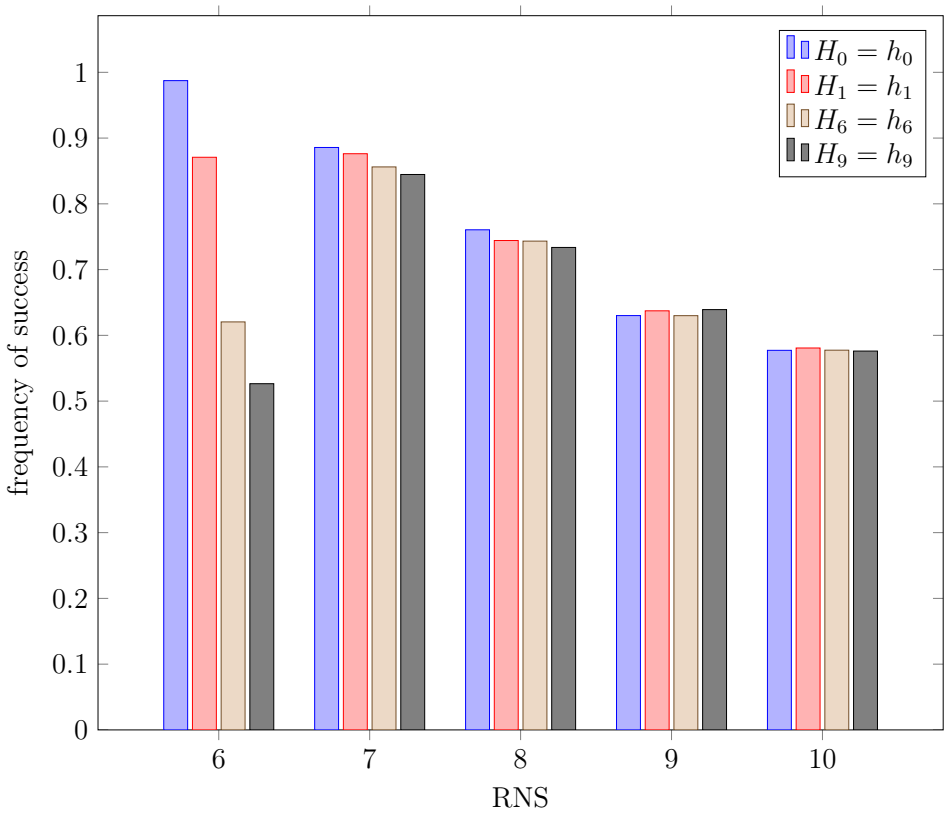


Figure 3.6 – Fréquence d'apparition de la clef dans la moitié de l'ensemble des clefs avec le meilleur score en utilisant le critère de la densité gaussienne conditionnée par $H_t = h_t$ ($t \in \{0, 1, 6, 9\}$), $i = 9$, Clef de 10 bits.

le nombre de configurations de moduli de 924 est faible. Puisque la taille de l'ensemble d'état de H_t est croissant en fonction de t, on remarque que les possibilités de conditionnement par $H_t = h_t$ sont plus importantes pour t grand. Par voie de conséquence, on diminue le nombre de scénarios engendrant chaque état et on s'éloigne de l'hypothèse gaussienne. On augmente alors les cas où les matrices ne sont pas bien conditionnées. Cette situation se retrouve dans une moindre mesure en RNS7 où il y a 3432 configurations. La même situation devient marginale pour RNS$>$ 7 (Plus de 12870 configurations).

Les Figures 3.5 et3.6 montrent qu'on peut extraire de l'information en utilisant une seule trace du moment qu'on possède un dictionnaire $(m_{H_t=h_t,k,i}, \Gamma_{H_t=h_t,k,i})$ moyenne et matrice de covariance complet. Évidemment, plus on prend de moduli plus l'information se délie dans l'aléa. Le fait de conditionner par "$H_t < \mathbb{E}(H_t)$" permet de faire apparaître un certain niveau d'information pour RNS6 et RNS7 mais cela reste négligeable par rapport au conditionnement par "$H_t = h_t$". Le meilleur conditionnement est "$H_0 = h_0$" pour RNS$\leq$ 7 sinon les résultats sont équivalents pour RNS$\geq$ 8. Les chances de succès sont cependant inférieures à 0.9 pour RNS$>$ 6, ce qui rend à peu prés impossible une attaque complète même sur une ECC de 112 bits ou alors il faudrait une valeur de $i_0 + 1$ beaucoup plus grande que 10. Seul le cas RNS6 est vraiment dangereux avec le conditionnement "$H_0 = h_0$" car on se rapproche d'une probabilité de 0.99 de réussite avec $i_0 + 1 = 10$ bits d'analyse.

Étant donné qu'on possède une densité de probabilité conditionnée par différentes distances de Hammings H_t. On s'est penché sur une amélioration de l'information en cumulant les densités de probabilité. Dans la figure 3.7, on utilise un nouveau différenciateur $f_m^{(k)}$ avec une démarche semblable à la Figure 3.6. Le différenciateur $f_m^{(k)}$ est donné par:

$$f_m^{(k)}(\mathbf{x}^i) = \sum_{t=0}^{m} p_{k,i}\left(\mathbf{x}^i \,|H_t = h_t\right) \text{ avec } m \leq i. \tag{3.27}$$

Pour RNS6, on retrouve, le problème de conditionnement vu à la Figure 3.6. Comme les configurations de moduli sont encore assez peu nombreuses en RNS7, on ne remarque pas de nette amélioration. Cependant les résultats deviennent meilleurs seulement pour RNS8 et RNS9. On peut donc améliorer les résultats avec cette méthode mais elle trouve assez rapidement ses limites.

3.4.5 Résultats annexes sur les 10 premiers Hamming

Les résultats précédents font apparaître des questions sur la structure du vecteur des distances de Hamming. Suite à la Section 3.4.4, on approfondit ici l'analyse des résultats utilisant les 10 premiers Hamming. La probabilité de succès introduite précédemment n'est pas homogène dans les deux cas présentés dans cette section :

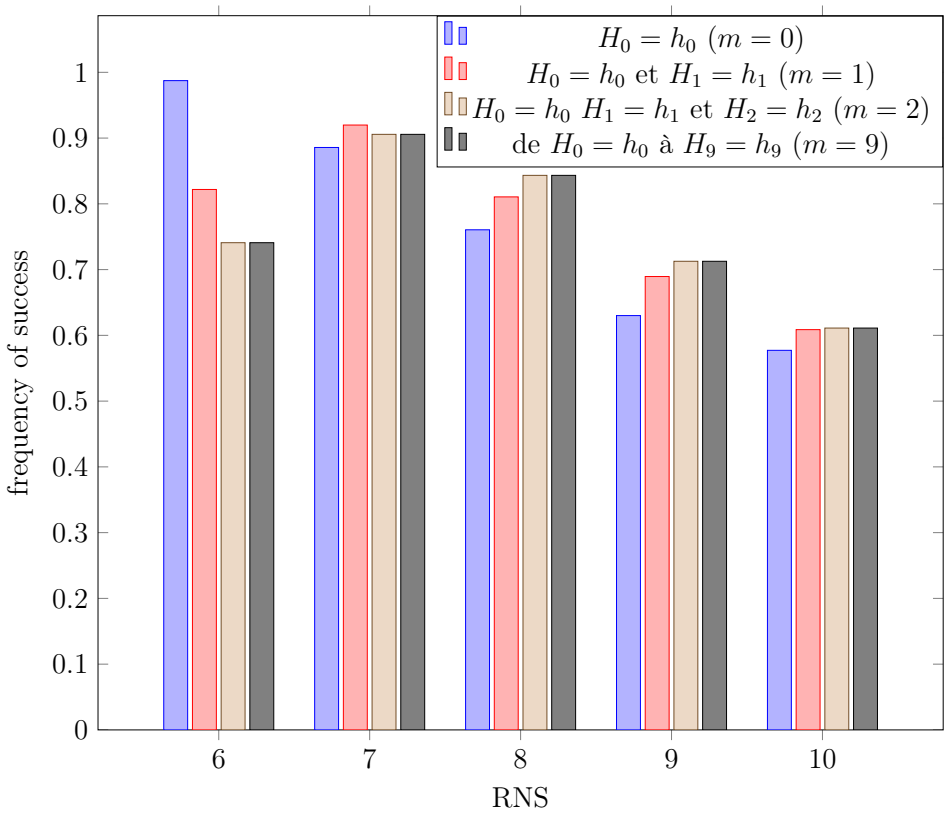


Figure 3.7 – Fréquence d'apparition de la clef dans la moitié de l'ensemble des clefs avec le meilleur score utilisant le cumul de densité gaussienne conditionnée par $H_t = h_t$. $i = 9$, Clef de 10 bits.

$j+1$ bits	2	3	4	5	6	7	8	9	10
Tirage Aléatoire	1	1	1	0.999	0.999	0.996	0.938	0.750	0.5
RNS6	1	1	0.999	0.999	0.999	0.998	0.995	0.991	0.987
RNS7	1	0.999	0.999	0.997	0.989	0.972	0.946	0.915	0.886
RNS8	1	0.999	0.997	0.989	0.971	0.932	0.878	0.815	0.760
RNS9	1	0.999	0.996	0.981	0.945	0.881	0.794	0.703	0.630
RNS10	1	0.999	0.995	0.978	0.936	0.862	0.762	0.658	0.577

Table 3.1 – Pourcentage de réussite des bits trouvés sur les 50% meilleurs scores donnés par un MLE conditionné par $H_0 = h_0$ sur $i+1 = 10$ bits. Résultat sur 300×512 clefs de $i+1 = 10$ bits

- La probabilité de succès peut être plus importante sur l'ensemble des débuts de clef, dans la Section 3.4.5.1.
- L'efficacité du conditionnement varie selon qu'on se trouve au centre ou dans les queues de distribution d'une distance de Hamming dans la Section 3.4.5.2. Et par voie de conséquence la probabilité de succès y est plus importante.

3.4.5.1 Fréquence de succès suivant la taille du début de clef

Soit X_j la variable aléatoire qui compte le nombre de clef avec le mauvais choix de bit, i.e. le nombre de clef qui ne commencent pas avec les bons j premiers bits. Connaître la probabilité que les $j+1$ premiers bits sont parmi les 50% meilleurs scores, revient à chercher

$$P(X_j < 2^{i-1}) = 1 - P(X_j \geq 2^{i-1}) = 1 - P(X_j = 2^{i-1}) = 1 - \frac{\binom{2^i - \frac{2^i}{2^j}}{2^{i-1}}}{\binom{2^i}{2^{i-1}}}. \tag{3.28}$$

sachant que l'on fait un tirage de 2^{i-1} clefs. La preuve de l'égalité (3.28) est dans l'Annexe 3.C.

Pour une valeur de i fixée, la fonction $j \longrightarrow P(X_j < 2^{i-1})$ est décroissante jusqu'à $\frac{1}{2}$. A travers cette monotonie, un tirage aléatoire des bits montre que les débuts de clefs se classent mieux que la clef complète. On s'attend donc à de meilleurs résultats avec le MLE conditionnel relaxé appliqué sur les débuts de clefs. On compare avec les résultats obtenus pour différents RNS et différentes valeurs de j.

En examinant le Tableau 3.1 et la Figure 3.8, on note que la probabilité de succès de la méthode chute en fonction du nombre de bits. Cependant cela chute moins rapidement que le résultat d'un tirage aléatoire des bits. On remarque aussi qu'on peut obtenir des valeurs inférieures au tirage aléatoire, par exemple dans le cas RNS10 et 9 bits. En effet,

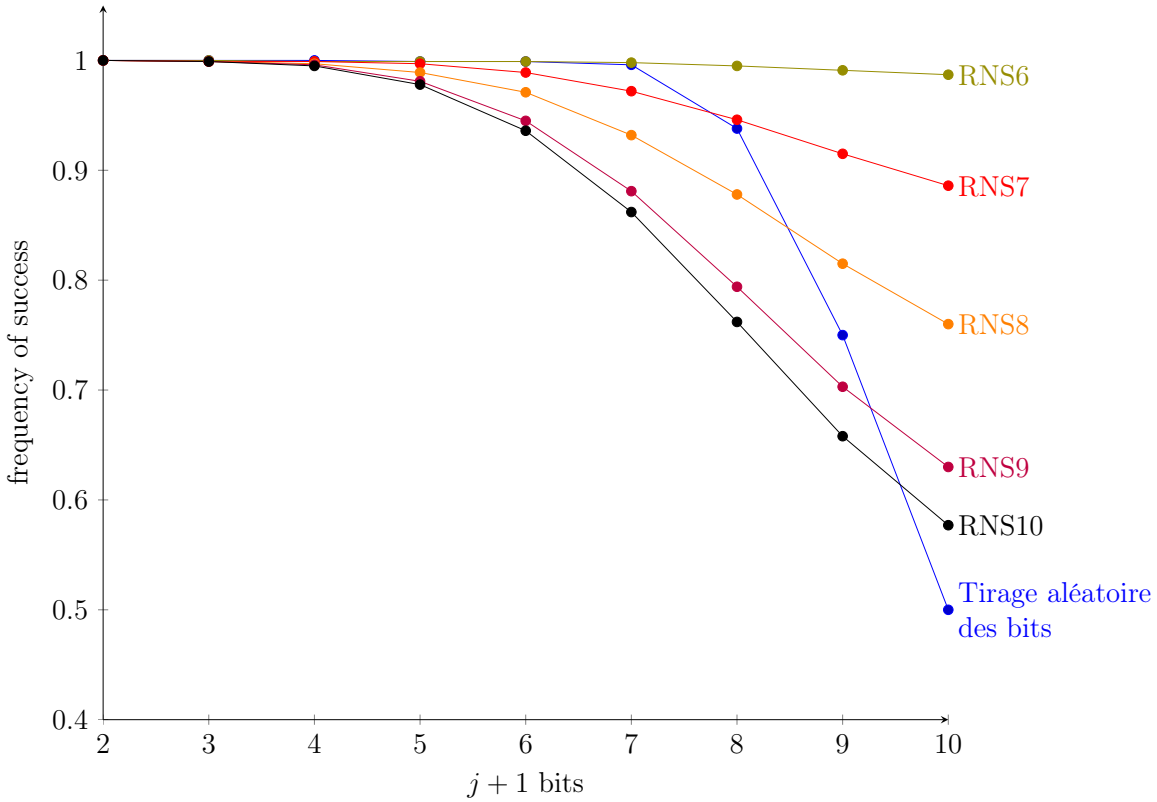


Figure 3.8 – Pourcentage de réussite des bits trouvés sur les 50% meilleurs scores donnés par un MLE conditionné par $H_0 = h_0$ relaxé sur $i_0 + 1 = 10$ bits. Résultat sur 300 tirages aléatoires de configuration de moduli et 512 clefs de $i_0 + 1 = 10$ bits

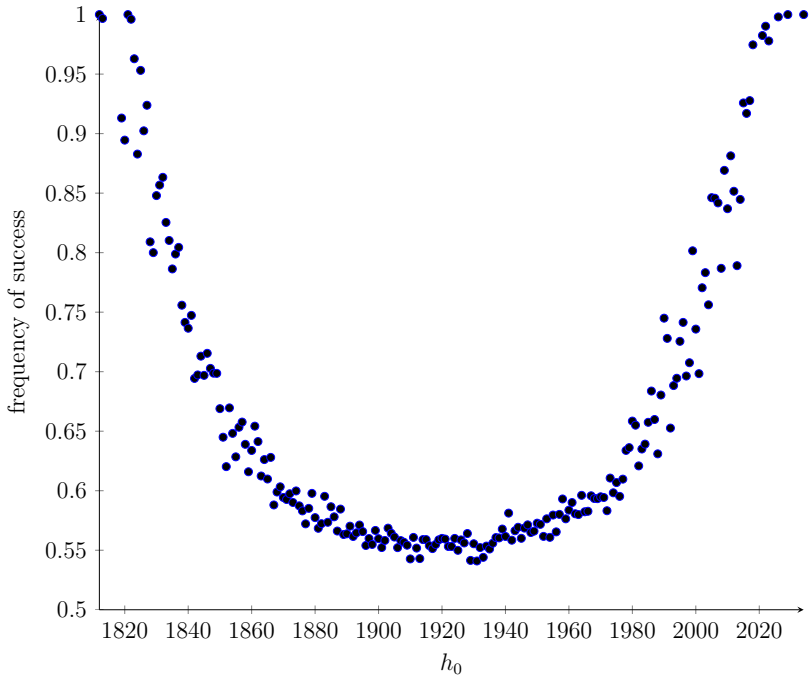


Figure 3.9 – Fréquence d'apparition de la clef dans la moitié des plus probables en utilisant le MLE conditionné par $H_0 = h_0$ en fonction de h_0 avec le critère de la densité gaussienne. Au moins 512 clefs testées par valeur de h_0. Clef de 10 bits sur RNS10.

très souvent une clef K classée parmi les 50% plus mauvais scores se retrouve dans le même quantile d'une autre clef qui a les mêmes premiers bits que K.

On remarque donc que le MLE conditionné donne plus d'information sur les bits précédents alors qu'on n'utilise qu'une seule trace d'observation. Seulement, une attaque sur un début de clef n'est pas intéressante car la différence avec un tirage aléatoire des bits diminue. Il y a même des cas où le tirage aléatoire est meilleur que l'analyse relaxée sur un début de clef. On est donc à l'optimum de l'efficacité quand on utilise la totalité des $i + 1$ bits dans l'analyse relaxée.

3.4.5.2 Fréquence de succès en fonction du conditionnement "$H_t = h_t$"

On a vu dans la Section 3.4.4.1 que la séparation est plus ou moins efficace selon qu'on se trouve plus ou moins loin du centre de la distribution (cf. Fig. 3.3 et 3.4). La Figure 3.9, qui représente la probabilité de succés en fonction du conditionnement, confirme un succès plus important lorsque h_0 est loin de $\mathbb{E}(H_0)$. L'information induite par un événement rare (sur les queues de distribution) est plus conséquente donc plus pertinente pour une attaque; il est plus facile de retourner à l'information ciblée. On retrouve le même type de résultat quand on conditionne par d'autres distances de Hamming. (cf. Annexe 3.D).

Compte tenu de la Figure 3.9, il serait justifié de faire une attaque conditionnée par la

$P\left(\lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert \leq \alpha\right)$	$\frac{\alpha}{\sigma(H_{t,K})}$	$P\left(\exists t \leq i, \lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert \geq \alpha\right)$	Probabilité minimale approximative de succès
68%	1.	0.979	0.57
80%	1.28	0.893	0.58
90%	1.645	0.651	0.6
95%	1.96	0.401	0.63

Table 3.2 – Probabilité d'obtenir une distance de Hamming en dehors de l'intervalle de confiance pour $i + 1 = 10$. Fréquence d'apparition de la clef dans la moitié des plus probables en utilisant le MLE conditionné par $H_t = h_t$ avec le critère de la densité gaussienne si h_t est en dehors de l'intervalle de confiance. Clef de 10 bits sur RNS10.

valeur de Hamming la plus éloignée de la moyenne. On approxime la probabilité d'avoir une valeur de distance de Hamming loin de la moyenne sur la suite $(H_{0,K}, ..., H_{i,K})$. On cherche, pour $\alpha \in \mathbb{R}^{+*}$

$$P\left(\exists t \leq i, \lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert \geq \alpha\right) = 1 - P\left(\forall t \leq i, \lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert < \alpha\right) \tag{3.29}$$

Si on suppose que les distances de Hamming sont des gaussiennes indépendantes alors

$$P\left(\exists t \leq i, \lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert \geq \alpha\right) = 1 - \prod_{t=0}^{i} P\left(\lvert H_{t,K} - \mathbb{E}\left(H_{t,K}\right)\rvert \leq \alpha\right). \tag{3.30}$$

On voit sur la Table 3.2 qu'il faudrait une valeur de i plus grande pour une attaque efficace. La probabilité de trouver une distance de Hamming suffisamment loin de la moyenne est trop petite pour être exploitable. Même si on se retrouve en dehors de 95% des valeurs, on arrive péniblement à une fréquence minimum d'apparition de la clef de 0.63.

3.5 Conclusion

Dans le cadre des moduli aléatoires, on a explicité l'hypothèse gaussienne avec différents aspects :

a) L'hypothése gaussienne sur le vecteur complet des distances de Hammings (Clefs et étapes comprises) $(H_K^i, K = 2^i, ..., 2^{i+1} - 1) := \left({H_{2^i}^i}^T | {H_{2^i+1}^i}^T | \dots | {H_{2^{i+1}-1}^i}^T\right)^T$.

b) L'hypothèse gaussienne sur les distances de Hamming produit par chaque clef $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ avec une analyse forte (recherche du cas le plus favorable).

c) L'hypothèse gaussienne sur $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ conditionné avec une analyse forte.

d) L'hypothèse gaussienne sur $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ conditionné avec une analyse relaxée. On élimine les 50% cas les moins probables.

On a montré que la méthode des moduli aléatoires reste suffisamment résistante pour RNS>6 quelle que soit l'hypothèse retenue. Dans le cas a), on a montré que l'on restait dans un cadre pseudo aléatoire, on ne trouve pas de relation forte entre les Hammings car on ne réduit pas suffisamment l'espace de probabilité. Dans le cas b), il faut beaucoup de traces pour réussir à retourner à la clef cible et on retrouve un résultat proche de ce qui a été montré dans la Section 2.5 du Chapitre 2; le nombre d'observations doit être de l'ordre de $O\left(\frac{(2n)!}{n!^2}\right)$. Le cas c) donne des résultats équivalents au cas b). Dans le cas d), même si on montre qu'il peut y avoir une information dans une seule trace d'observation, celle-ci n'est pas suffisante avec le conditionnement "$H_t = h_t$" pour retourner à la clef cible avec les paramètres utilisés ($i_0 + 1 = 10$) sauf pour $RNS \leq 6$. Il serait d'ailleurs intéressant d'évaluer complètement la dangerosité du cas RNS6. On remarque donc que le cas RNS7 est le cas limite sur lequel le nombre de configurations possibles de moduli est tout juste suffisant pour créer un aléa acceptable sur les distances de Hamming dans le cas où on ne fait qu'une seule observation. Le cas RNS7 est à la limite de notre analyse car il faudrait prendre en compte le problème des matrices de covariance mal conditionnées. C'est ce qui a été entrepris dans la Section 4.5 du Chapitre 4 suivant, où on améliore la précision dans la résolution de système avec le Cuppen Divide & Conquer. On pourrait alors affiner la recherche d'un conditionnement qui retournerait complétement le secret.

Annexes

3.A Intersection d'aléatoire généré

Soit X une variable aléatoire:

$$X : (\Omega, \mathcal{F}, P) \to (E_t, \mathcal{E}_t)$$

- $(\Omega, \mathcal{F}, P)$ est l'espace de probabilité.
- E est l'espace d'état de X et $\mathcal{E}$ est la tribu associée.

Pour tout $A \in \mathcal{B}$, l'image inverse est définie par $X^{-1}(A) = \{\omega \in \Omega, X(\omega) \in A\}$. $\sigma(X)$, représente la tribu engendrée par la variable aléatoire X, i.e. la plus petite tribu rendant X mesurable.

Afin d'éclaircir notre propos, on montre comment l'information sur une variable aléatoire peut donner une information sur une autre variable avec un exemple fini discret. Soit X_1 et X_2 deux variables aléatoires associées à un tirage dans $\Omega = \{1, 2, 3\}$.

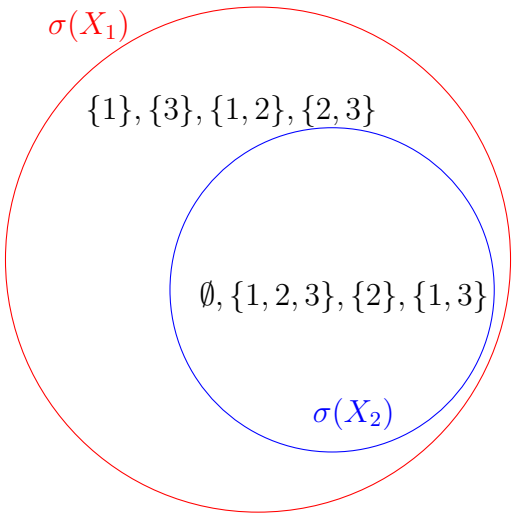
$$X_1 : (\Omega, \mathcal{F}, P) \to (E_1, \mathcal{E}_1) \tag{3.31}$$

$X_1(1) = a$, $X_1(2) = b$, $X_1(3) = c$, les valeurs que prennent X_1 donc
$E_1 = \{a, b, c\}$ donc
$\mathcal{E}_1 = \mathcal{P}(E_1) = \{\oslash, E_1, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}\}$, la tribu discrète.

$$X_2 : (\Omega, \mathcal{F}, P_2) \to (E_2, \mathcal{E}_2) \tag{3.32}$$

$X_2(1) = a$, $X_2(2) = b$ et $X_2(3) = a$ donc
$E_2 = \{a, b\}$ donc
$\mathcal{E}_2 = \mathcal{P}(E_2) = \{\oslash, \{a, b\}, \{a\}, \{b\}\}$
On peut évaluer les tribus inverses engendrées par les deux variables X_1 et X_2.

$$\sigma(X_1) = \{(X_1)^{-1}(A), A \in \mathcal{E}_1\} = \mathcal{P}(\Omega) \tag{3.33}$$
$$\sigma(X_2) = \{(X_2)^{-1}(A), A \in \mathcal{E}_2\} = \{\emptyset, \{1, 2, 3\}, \{2\}, \{1, 3\}\} \subset \sigma(X_1) \tag{3.34}$$



$$\sigma(X_1) \bigcap \sigma(X_2) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} = \sigma(X_2) \tag{3.35}$$

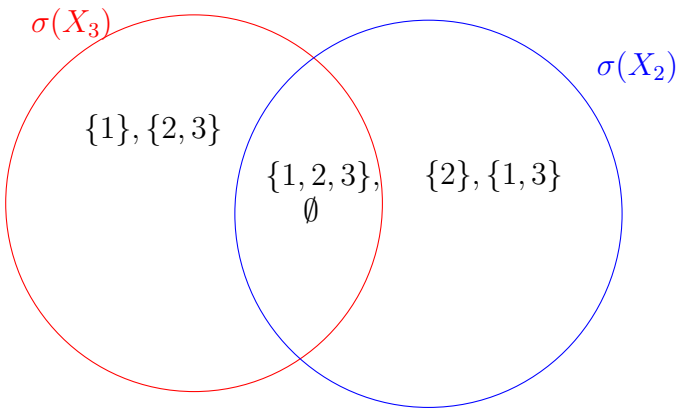
$\sigma(X_2) \subset \sigma(X_1)$ donc l'aléa généré par X_2 est contenu dans l'aléa généré par X_1. Si X_1 et X_2 font parti d'une même expérience aléatoire, la connaissance des réalisations de X_1 donne une connaissance des réalisations de X_2

Maintenant prenons une autre variable aléatoire X_3 du même type que X_2 mais avec la définition suivante :

$$X_3 : (\Omega, \mathcal{F}, P) \to (E_3, \mathcal{E}_3) \tag{3.36}$$

$X_3(1) = a$, $X_3(2) = b$ et $X_3(3) = b$ donc
$E_3 = \{a, b\}$ donc
$\mathcal{E}_3 = \mathcal{P}(E_3) = \{\oslash, \{a, b\}, \{a\}, \{b\}\}$

$$\sigma(X_3) = \{(X_3)^{-1}(A), A \in \mathcal{E}_3\} = \{\emptyset, \{1, 2, 3\}, \{1\}, \{2, 3\}\} \tag{3.37}$$



Dans ce cas, $\sigma(X_2) \bigcap \sigma(X_3)$ se réduit au cas trivial.
On peut construire un cas de figure où deux variables ont des tribus engendrées qui n'ont pas d'intersection.

3.B DPA square sur courbe d'Edwards 25519

.

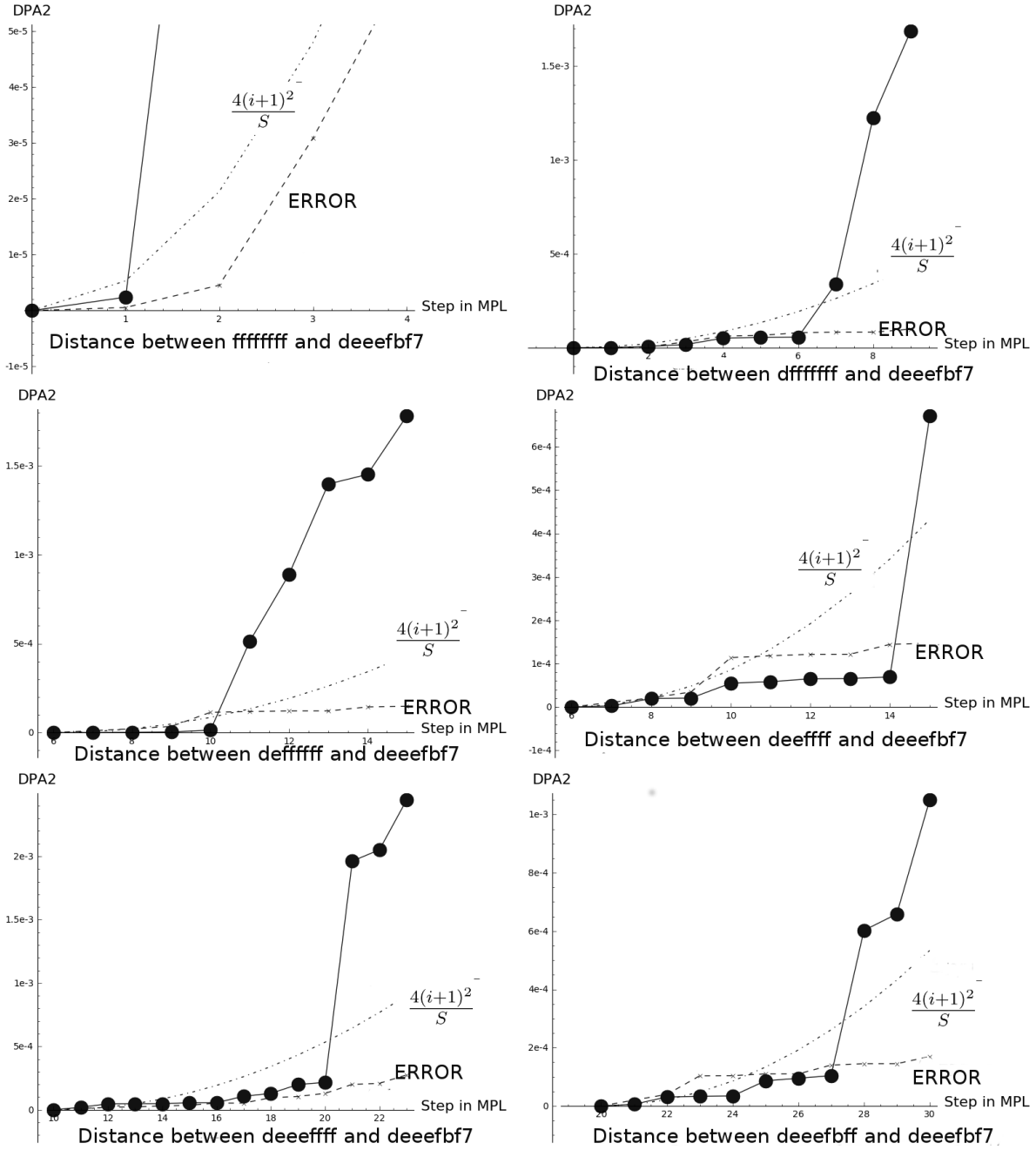


Figure 3.B.1 – DPA2 sur RNS9 sur ECC Edwards 25519 de 255 bits avec $S = 750000$: Chaque nouveau saut au dessus de $\frac{4(i+1)^2}{S}$ donne l'indice du bit qui est égal à zéro.

3.C Probabilité d'apparition de bits corrects avec un tirage uniforme

En classant les clefs, on a pas forcément la clef complète dans les 50% premiers scores mais les bits précédents se classeront peut-être mieux. Faisons le calcul pour connaître la probabilité d'avoir les j premiers bits parmi des clefs de $i+1$ bits. On se donne une procédure qui tire 50% des clefs aléatoirement et sans remise parmi l'ensemble des clefs. On veut calculer la probabilité que l'ensemble des clefs candidates ne contiennent pas celle avec la bonne suite des $j+1$ premiers bits de la clef cible. Il y a $\mathcal{N} = 2^i$ possibilités de clefs (toujours parce qu'une clef commence par 1). Sans perte de généralité, on considère que le problème revient à tirer sans remise $\frac{\mathcal{N}}{2} = 2^{i-1}$ boules dans une urne qui contient $\mathcal{N} = 2^i = n_1 + n_2$ boules.

$n_1 = \frac{\mathcal{N}}{2^j} = \frac{2^i}{2^j}$ est le nombre de clef avec le bon choix de bits. (i.e. le nombre de boule avec la bonne couleur "blanche").

$n_2 = \mathcal{N} - n_1 = 2^i - \frac{2^i}{2^j}$ correspond à toutes les autres possibilités.

Soit X_j la variable aléatoire qui compte le nombre de clef avec le mauvais choix de bit, i.e. le nombre de clef qui ne commencent pas avec les bons j premiers bits. Connaître la probabilité que les $j+1$ premiers bits sont parmi les 50% meilleurs scores, revient à chercher :

$$P\left(X_j < \frac{\mathcal{N}}{2}\right) = 1 - P\left(X_j \geq \frac{\mathcal{N}}{2}\right) = 1 - P\left(X = \frac{\mathcal{N}}{2}\right) = 1 - \frac{\binom{n_2}{\frac{\mathcal{N}}{2}}}{\binom{\mathcal{N}}{\frac{\mathcal{N}}{2}}} \tag{3.38}$$

$$P(X_j < 2^{i-1}) = 1 - P(X_j \geq 2^{i-1}) = 1 - P(X_j = 2^{i-1}) = 1 - \frac{\binom{n_2}{2^{i-1}}}{\binom{2^i}{2^{i-1}}} \tag{3.39}$$

$$P(X_j < 2^{i-1}) = 1 - \frac{\binom{2^i - \frac{2^i}{2^j}}{2^{i-1}}}{\binom{2^i}{2^{i-1}}}. \tag{3.40}$$

sachant que l'on fait un tirage de $\frac{\mathcal{N}}{2}$ clefs. En particulier on retrouve bien que pour $j = i$,

$$P(X_i < 2^{i-1}) = 1 - \frac{\binom{2^i - \frac{2^i}{2^i}}{2^{i-1}}}{\binom{2^i}{2^{i-1}}} = 1 - \frac{\binom{2^i - 1}{2^{i-1}}}{\binom{2^i}{2^{i-1}}} = \frac{1}{2}.$$

3.D Fréquence de succès en fonction du conditionnement, Hypothèse gaussienne

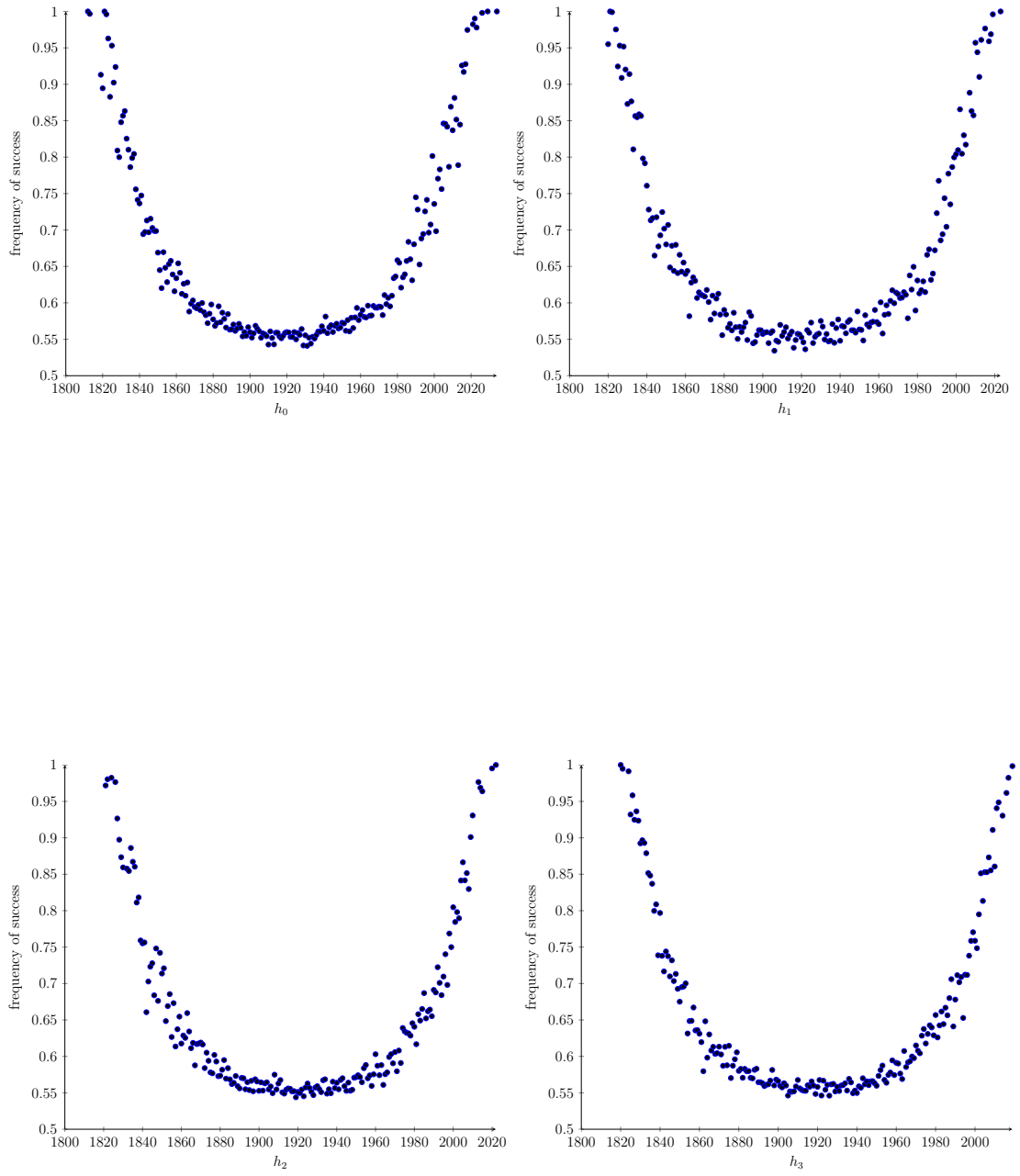


Figure 3.D.1 – Fréquence d'apparition de la clef dans la moitié des plus probables en utilisant le MLE conditionné par $H_t = h_t$ en fonction de h_t avec le critère de la densité gaussienne. Clef de 10 bits sur RNS10.

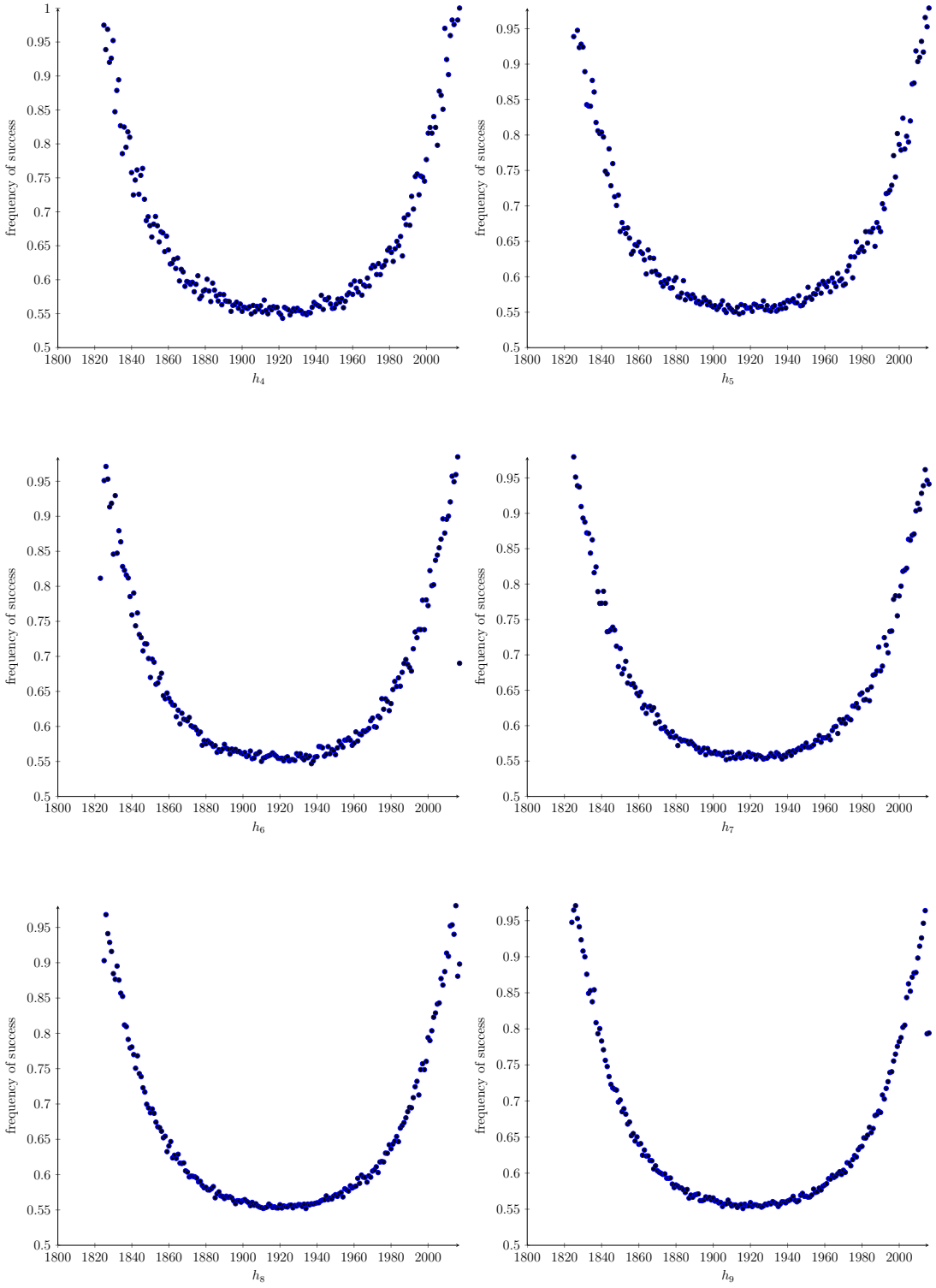


Figure 3.D.2 – Fréquence d'apparition de la clef dans la moitié des plus probables en utilisant le MLE conditionné par $H_t = h_t$ en fonction de h_t avec le critère de la densité gaussienne. Clef de 10 bits sur RNS10.

Chapter 4

Calcul des inversions de matrice de covariance en batch computing

Sommaire

4.1 Introduction

Dans les Sections 2.5 du Chapitre 2, 3.2.2 et 3.4.3 du Chapitre 3, on a eu besoin de faire un dictionnaire contenant des inversions faites avec des résolutions de systèmes, pour un ensemble conséquent de clefs. Dans ce chapitre, on justifie d'abord brièvement les choix algorithmiques faits pour les problèmes de résolution de système linéaire de la forme $AX = Y$ sur un grand nombre de matrices symétriques définies positives. Vu la quantité de calcul, il est nécessaire de paralléliser pour pouvoir obtenir des résultats rapidement en ayant le plus de paramétrage possible pour avoir des statistiques fiables. Un travail conséquent de programmation C/CUDA [80] est à l'origine de l'automatisation de l'analyse grâce à une implémentation sur une carte Nvidia Tesla K40C. On peut ainsi faire le calcul des vecteurs moyens et des matrices de covariances pour de nombreuses clefs. On parlera de batch computing lorsqu'on fait de nombreux calculs de petites tailles, c'est-à-dire ici de nombreuses matrices ou vecteurs de petite taille.

Plusieurs raisons nous ont amené à ne pas utiliser de blas pour cette étude. La parallélisation sur GPU sert pour de nombreuses opérations : chargement des traces, calculs des moyennes et matrices de covariances, factorisations des matrices, calculs de valeurs et vecteurs propres, résolutions de système pour obtenir les inversions de matrices, calcul de la racines carré des matrices de covariances, calcul du MLE, évaluation d'un maximum, relaxation de l'analyse du maximum. Tous ces calculs nécessitent une homogénéisation de l'espace mémoire pour être utilisés de façon efficace. Le contrôle de chaque partie des codes, qui font ces différentes opérations, nécessite de faire varier à minima les structures mémorielles.

Les blas développés par Nvidia [81] pour le problème de valeurs propres sont adaptés à des matrices de grande taille. La stratégie de parallélisation pour faire des calculs sur des matrices de grande taille n'est pas la même que pour le calcul sur de nombreuses matrices de petites tailles. De plus, la gestion de la mémoire serait différente si on utilisait les routines blas, par exemple en concaténant les matrices de petites tailles pour utiliser les blas proposés par Nvidia.

La communauté scientifique se retrouve de plus en plus confrontée à la résolution de nombreux problèmes de petites tailles indépendants les uns des autres. Les GPU sont de plus en plus efficaces, économiques, robustes, faciles d'installation et d'utilisation. Il existe donc un besoin croissant d'une approche efficace pour développer des codes pour des petits problèmes de matrice. On trouve dans la documentation de Nvidia, des blas pour résoudre des problèmes de batch parallélisation sur des opérations simples comme la résolution de systèmes triangulaires ou des produits de matrices [81]. Nvidia propose un exemple de factorisation de Cholesky en batch computing [82]. La bibliothèque MAGMA propose aussi un batch pour la factorisation de Cholesky [69]. Azzam Haidar, GTC de Nvidia, a présenté des travaux sur la factorisation LU en batch computing [49] et

sur la factorisations de Cholesky [4]. Suite à ces constatations, on lui a demandé, en octobre 2018, s'il existait du batch computing pour le problème des valeurs propres sur des matrices de petites tailles. Il nous a répondu qu'on n'avait pas encore développé de routine pour ce type de problème.

Pour la résolution de systèmes linéaire en batch, on a utilisé dans les chapitres 2 et 3 une factorisation LDLt développée par Lokmane Abbas-Turki et Stef Graillat [3]. La factorisation de LDLt ressemble à la factorisation de Cholesky. Cette dernière nommée d'après André-Louis Cholesky, consiste, pour une matrice symétrique définie positive A, à déterminer une matrice triangulaire inférieure L telle que : $A = LL^T$. La décomposition LDLt permet d'éviter l'utilisation des racines carrées qui peut être une source potentielle de problèmes en calcul numérique. Elle se calcule de la façon suivante :

$$A = LDL^T$$

où D est une matrice diagonale, et L une matrice triangulaire inférieure avec des 1 sur sa diagonale. On justifie le choix de LDLt en le comparant avec la décomposition de Householder associée avec des Parallel Cyclic Reductions (PCR) [97, 44, 3]. On préférera la factorisation LDLt à celle de Cholesky car on sait que cette méthode est plus précise et plus rapide car elle n'utilise pas la fonction racine carré contrairement à la décomposition de Cholesky. On fait remarquer que Nvidia propose un exemple de factorisation de Cholesky en batch computing [82]. La bibliothèque MAGMA propose aussi un batch pour la factorisation de Cholesky [69].

Suite à l'étude sur l'Estimateur Maximum de Vraisemblance (MLE : Maximum Likelihood Estimator) conditionnel de la Section 3.4.3 du chapitre 3, certaines matrices de covariance se sont avérées mal conditionnées. Le choix du LDLt en batch n'est donc pas le meilleur choix pour réaliser cette analyse pour certain type de matrice. Il nous est apparu donc intéressant d'améliorer un code de diagonalisation de matrice Cuppen Divide & Conquer de [3] pour l'adapter au problème de l'analyse des moduli aléatoires. Le code proposé est basé sur le principe de "Diviser pour Régner" ou "Divide & Conquer" en anglais. On préfère l'appellation anglaise qui exprime l'idée de la deuxième étape de l'algorithme qui consiste à rassembler (Conquer) ce qui a été divisé. Ce code s'intéresse à la résolution du problème des vecteurs et valeurs propres d'une matrice tridiagonale symétrique dans le cas où la sous-diagonale ne contient pas de zéro ($b_i \neq 0$). C'est-à-dire résoudre :

$$\Gamma = \begin{bmatrix} a_1 & b_1 & & 0 \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{bmatrix}, \qquad \Gamma x = \lambda x. \tag{4.1}$$

en décomposant la matrice de la manière suivante

$$\Gamma = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \rho u u^T \tag{4.2}$$

où T_1 et T_2 sont deux sous matrices et ρ est égal à l'un des b_i, et u est un vecteur résultant de cette décomposition.

Comme la référence [3] ne traitait que des valeurs b_i non nuls, on propose une généralisation dans le cas où il existe des zéros sur la sous-diagonale. Ensuite, on montre deux optimisations en temps de calcul qui s'adaptent bien au Divide & Conquer en batch computing. La première optimisation traite de la meilleure façon de faire un produit de matrices en batch computing. La seconde adapte le Merge Path proposé dans [46] pour fusionner des tableaux triés des valeurs propres dans les étapes du Divide & Conquer.

Au delà de la généralisation et l'optimisation de ce code, on s'est penché sur l'amélioration de la précision. Cette amélioration est basée sur une étude fine de la déflation en simple précision. Une déflation apparaît quand deux valeurs numériques sont très proches au niveau de la précision flottante. Dans ce cas, il faut gérer l'approximation par une approche différente. Le premier cas de déflation usuellement étudiée apparaît lorsque deux valeurs propres successives sont très proches numériquement (cf.[35, 47]). On le nomme déflation de Type I. Lors de la procédure du Divide & Conquer, on obtient un vecteur v en calculant

$$vv^T = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} uu^T \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \tag{4.3}$$

où Q_1 et Q_2 sont les matrices des vecteurs propres de T_1 et T_2. Le deuxième cas de déflation étudié dans la littérature se produit quand un élément du vecteur v est numériquement proche de zéro (cf.[35, 47]), cas qu'on nomme Type III.

Dans notre étude, on introduit un autre cas qui traite d'un autre genre de déflation qui n'a pas été prévu par les théories développées sur le Divide & Conquer utilisé dans le cadre numérique. Ce nouveau type de déflation et la réorganisation algorithmique que cela implique apporte une vraie amélioration de la précision. On le nomme cas de Type II. Sur l'ensemble de matrices étudiées cela divise l'erreur maximale d'un facteur 10. Et cela diminue aussi l'erreur d'un facteur 1000 sur certains cas où celle-ci est particulièrement importante.

Suite à de nouveaux choix algorithmiques et l'introduction de la déflation de Type II, on valorise cet apport sur des matrices de taille inférieure à 32. Cependant ce travail doit être étendu à des tailles plus grandes.

Ce chapitre s'organise comme suit

- La Section 4.2 justifie les choix algorithmiques qui ont été fait dans les Chapitres 2 et 3, notamment l'utilisation de la factorisation LDLt en batch Computing versus

Householder+PCR.

Suite à l'étude de la Section 3.4.3 du chapitre 3, on améliore un code de Cuppen [31] qui utilise le principe du Divide & Conquer pour la diagonalisation de matrice tridiagonale symétrique. On améliore ainsi l'erreur pour utiliser la résolution de système linéaire dans le MLE conditionné dans le cas de matrice mal conditionnée.

- Dans la section 4.3, on rappelle brièvement le principe du Divide & Conquer et on généralise le code de [3] au cas où il y a des zéros sur la sous-diagonale.
- La Section 4.4.1 explique comment faire un produit de matrice efficace en temps d'exécution en batch computing sur GPU utilisable dans le Divide & Conquer. Et on propose une méthode de fusion de tableaux triés dans la Section 4.4.2.
- Et pour finir, on indique dans la Section 4.5 une nouvelle façon de gérer les erreurs sur le Cuppen Divide & Conquer. Ces erreurs proviennent de l'approximation flottante en simple précision avec un nouveau type de déflation.

Ce travail a été fait en collaboration avec Babacar Diallo docteur en mathématiques appliquées (2019) à l'université Paris-Saclay, co-encadré par Lokmane Abbas-Turki et Stéphane Crépey.

4.2 LDLt versus Householder

4.2.1 Décomposition LDLt

La résolution du système linéaire $AX = Y$ avec A une matrice symétrique définie positive est divisée en deux étapes :

1. la factorisation de la matrice A sous la forme $A = LDL^T$,
2. la résolution de $LZ = Y$ avec $DL^TX = Z$ où L^t est la transposée de L.

La matrice D est diagonale et la matrice L triangulaire inférieure avec des 1 sur sa diagonale. La factorisation est effectuée grâce aux expressions suivantes :

$$\begin{aligned} A = LDL^t, \quad D_{j,j} &= A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2 D_{k,k}, \\ L_{i,j} = \frac{1}{D_{j,j}} &\left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} D_{k,k} \right) \quad \text{si } i > j. \end{aligned} \tag{4.4}$$

Parce qu'elle évite le calcul des racines carrées, la factorisation LDLt est généralement considérée comme une meilleure alternative à la décomposition de Cholesky. Les deux

méthodes partagent la même stabilité pour des matrices définies positives symétriques A. De plus, lorsque le caractère défini positif est numériquement remis en question, il convient d'éviter l'utilisation des décompositions LDLt ou de Cholesky. Ces deux méthodes partagent le même ordre de complexité et la même occupation de l'espace mémoire. En raison de toutes ces similitudes et du fait que la littérature autour de la décomposition de Cholesky est plus grande que celle de LDLt, nous ne distinguerons pas les références de chaque méthode.

La propriété de stabilité de LDLt et Cholesky est importante. En réalité, lorsque la matrice n'est pas numériquement singulière, ces deux méthodes sont si stables qu'elles ne nécessitent pas de pivotement comme ceux effectués pour la décomposition LU [94]. Éviter les pivotements présente un grand avantage pour la mise en œuvre du GPU car il réduit les communications entre les threads. Par ailleurs, LDLt et Cholesky sont les méthodes de factorisation les plus efficaces avec une complexité en $O(n^3/6)$ où $n \times n$ est la taille de la matrice. Ce sont aussi celles qui utilisent le moins l'espace mémoire car elles n'impliquent que $n(n+1)/2$ valeurs. Une fois la factorisation LDLt effectuée, la résolution de $LZ = Y$ puis de $DL^tX = Z$ sont simples. Ces résolutions sont d'une complexité quadratique par rapport à n.

En fait, pour traiter une grande matrice, on doit la diviser en morceaux plus petits et répartir les calculs sur les blocs de threads. En revanche, avec un grand nombre de petites matrices, il suffit de distribuer directement les matrices sur les différents blocs. De plus, diverses techniques développées pour des matrices de taille supérieures à 64 ne sont pas efficaces pour un grand nombre de matrices de taille inférieure à 64.

4.2.2 Décomposition de Householder

De façon analogue à la méthode basée sur LDLt, on peut résoudre le système linéaire $AX = Y$, avec A symétrique, en deux étapes :

- La décomposition tridiagonale de Householder $A = QUQ^t$ où Q est orthogonale et U est tridiagonale symétrique.

- La Parallel Cyclic Reductions (PCR) associée au problème $UZ = Q^tY$ qui permet de récupérer X grâce à $X = QZ$.

La référence [97] explique comment faire une PCR avec des matrices dont les tailles sont des puissances de 2. Dans une carte Nvidia, les threads sont synchronisés par bloc de 32 threads (warp), si plusieurs warps accèdent en même temps à la mémoire partagée, il y a un conflit de ban. Dans [44], on propose une implémentation du PCR avec un décalage de bit pour éviter les conflits de ban pendant le calcul GPU. Et [3] propose une PCR pour toute taille de matrice. Lorsque le système linéaire est symétrique, la

tridiagonalisation de Householder est généralement utilisée comme première étape d'un algorithme de diagonalisation qui pourrait utiliser d'autres méthodes comme la factorisation QR, la méthode de bissection, Multiple Relatively Robust Representations ou Divide & Conquer. On se réfère à [36] pour une comparaison séquentielle entre ces algorithmes en fonction de la vitesse et de la précision. On utilise la tridiagonalisation de Householder avec la PCR et on vérifie l'erreur résiduelle avant de rechercher une diagonalisation du système. Cette procédure est justifiée par le fait que la PCR est moins complexe que n'importe lequel des algorithmes cités ci-dessus avec un ratio théorique égal au moins à $n/\log_2(n)$ en faveur de la PCR. De plus, comme déjà montré dans [97], la PCR est assez stable pour les matrices définies symétriques et positives et est adaptée aux architectures parallèles comme les GPUs. Pour plus de détail sur la méthode de Householder + PCR, on renvoie le lecteur à [3].

Pour les tests, on a besoin de construire un ensemble de matrices symétriques définies positives proches de la singularité. Pour cela, on introduit les matrices Ξ_ρ décrites ci-dessous. Il est assez simple de montrer que les matrices Ξ_ρ du type

$$\Xi_\rho = \begin{pmatrix} 1 & \rho & \cdots & \rho & \rho \\ \rho & 1 & \rho & & \vdots \\ \rho & \ddots & \ddots & \ddots & \rho \\ \vdots & & \rho & 1 & \rho \\ \rho & \rho & \cdots & \rho & 1 \end{pmatrix} \quad \text{with } 0 < \rho < 1 \tag{4.5}$$

sont définies positive en décomposant $\Xi_\rho = \rho J + (1-\rho)I$ où J est la matrice ne contenant que des 1 et I la matrice identité. Parce que ces matrices sont très structurées, on préfére utiliser une version randomisée donnée par $A = Q_{\Xi_\rho} R Q'_{\Xi_\rho}$, où R est une matrice aléatoire tridiagonale symétrique à diagonale dominante et Q_{Ξ_ρ} est la matrice orthogonale résultant de la tridiagoonalisation de Householder de Ξ_ρ. Les composantes de R sont définies à l'aide de variables aléatoires uniformes et de la multiplication des éléments diagonaux par le facteur approprié pour que R domine diagonalement.

On mesure l'erreur en résolvant le système $AX = Y$ en prenant
$Y = (0, 1, 2, 3, \ldots, sizeof(A) - 1)^T$. Si $\widetilde{X}$ est la solution numérique, l'erreur est

$$Erreur = \|A\widetilde{X} - Y\|_2$$

On voit sur la Figure 4.1 que l'erreur est bien meilleure avec la méthode LDLt pour des cas qui ne sont pas proches de la singularité. Ensuite si on joue sur la diagonale dominante pour avoir des matrices proche de la singularité, on remarque sur la Figure 4.2 que l'erreur explose dans la majorité des cas quand on utilise la méthode de Householder. La méthode

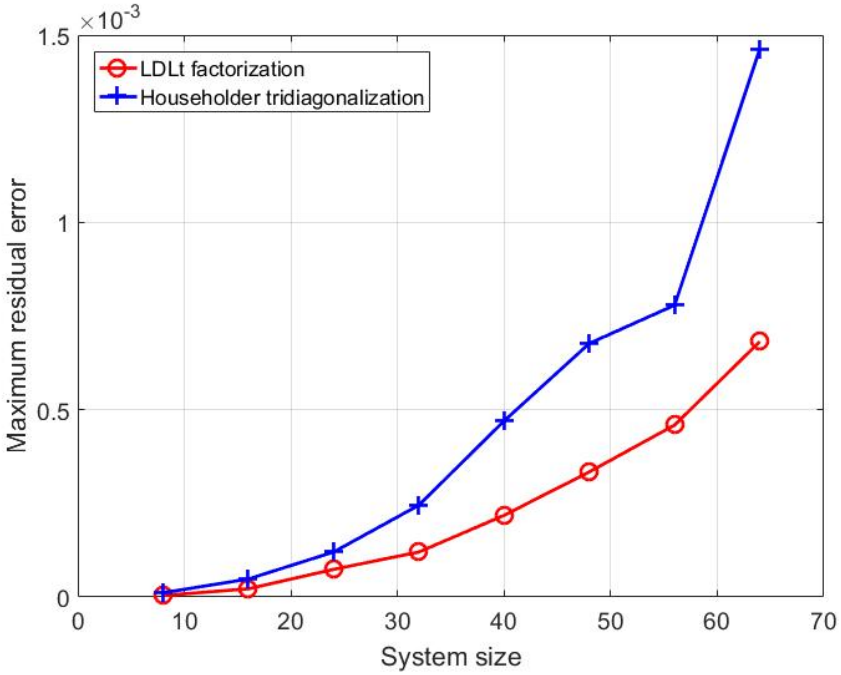


Figure 4.1 – Residual error, comparison between LDLt and Householder for different size of matrices

LDLt est plus stable.

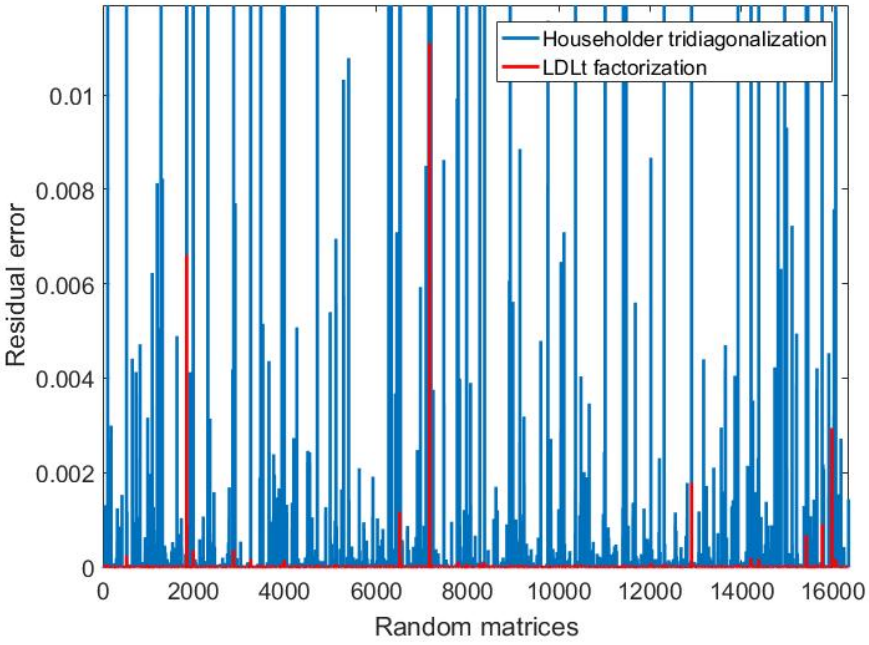


Figure 4.2 – Residual error, comparison between LDLt and Householder, Matrices closed to singularity of size 32

On voit sur la Figure 4.2 que la LDLt donne la meilleure erreur résiduelle maximale. C'est donc la méthode qu'on utilise pour résoudre le problème du MLE dans la Section 2.5. La LDLt sert aussi à calculer les résultats de l'Équation (3.6) dans l'analyse forte étudiée dans la Section 3.2.1. Mais l'hypothèse gaussienne conditionnée (cf. Section 3.4) fait apparaître des matrices mal conditionnées; en particulier les matrices de covariance calculées avec un conditionnement sur les distances de Hamming loin de la moyenne. On s'est donc intéressé avec Babacar Diallo à améliorer un code plus robuste de diagonalisation des matrices tridiagonales développé par Lokmane Abbas-Turki et Stef Graillat [3].

4.3 Cuppen Divide and Conquer pour GPU

In [35, p.216], on trouve une présentation détaillée de l'algorithme de Divide & Conquer pour le problème des valeurs propres. On va utiliser la version GPU de cet algorithme développé par [3]. Cette adaptation pour GPU est dédiée à un grand nombre de petites matrices.

On rappelle les étapes importantes de cette méthode :

1. Diviser le problème de diagonalisation en deux sous-problèmes de diagonalisation avec une factorisation diagonale connue.
2. Résoudre l'équation (4.17) séculaire qui donne les valeurs propres de la matrice.
3. Utilisez le théorème de Löwner (voir [66] ou [35]) pour la stabilité de la procédure globale.
4. Effectuer une multiplication matricielle pour assembler les sous-problèmes diagonalisés et terminer le problème de diagonalisation (Conquer).

4.3.1 Principe Divide & Conquer

Dans cette partie, on s'intéresse à la résolution du problème des vecteurs et valeurs propres d'une matrice tridiagonale symétrique.

$$\Gamma = \begin{bmatrix} a_1 & b_1 & & 0 \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{bmatrix}, \qquad \Gamma x = \lambda x. \tag{4.6}$$

En 1980, Cuppen propose un algorithme [32] fondé sur le principe du Divide & Conquer. Dans la situation présente, il y a une matrice tridiagonale Γ que l'on partitionne

en deux matrices tridiagonales plus petites. On résout le problème des valeurs propres sur ces deux nouvelles matrices puis on recombine les solutions des petites matrices pour avoir la solution finale du problème aux valeurs propres de Γ.

On fait la partie Divide (Diviser) de la manière suivante :

$$\text{en posant } \Gamma = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \rho u u^T, \tag{4.7}$$

T_1 et T_2 sont deux matrices tridiagonales, $\rho = b_m$ avec $m \approx \frac{n}{2}$ et u est un vecteur de taille n tel que $u^T = (\overbrace{0, \ldots, 0, 1}^{m}, \overbrace{1, 0, \ldots, 0}^{n-m})$.

Exemple

$$\Gamma = \begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & 2 & 3 & 0 \\ 0 & 3 & 4 & 6 \\ 0 & 0 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & -1 & 0 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 6 & 8 \end{bmatrix} + 3(0,1,1,0) \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 5 & 0 & 0 \\ 5 & -1 & 0 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 6 & 8 \end{bmatrix} + 3 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Dans le cas où il y a des zéros sur les sous-diagonales, comme par exemple,

$$\Gamma = \begin{bmatrix} a_1 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & a_2 & b_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_2 & a_3 & b_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_3 & a_4 & b_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & b_4 & a_5 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} & a_6 & b_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & b_6 & a_7 & b_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_7 & a_8 \end{bmatrix}$$

on décompose la matrice en plusieurs sous-matrices tridiagonales symétriques sur lesquelles on appliquera le Divide & Conquer. Dans l'exemple précédent (**??**), il y a trois sous-matrices : $[a_1]$ de taille 1, Γ_1 de taille 4 et Γ_2 de taille 3. La premiére étape du Divide &

Conquer donnera donc

$$\Gamma = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & \Gamma_1 & 0 \\ 0 & 0 & \Gamma_2 \end{bmatrix} = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & \begin{bmatrix} T_{1,1} & 0 \\ 0 & T_{1,2} \end{bmatrix} + b_3 u_1 u_1^T & 0 \\ 0 & 0 & \begin{bmatrix} a_6 & 0 \\ 0 & T_{2,2} \end{bmatrix} + b_6 u_2 u_2^T \end{bmatrix}. \tag{4.8}$$

On peut alors faire la suite du Cuppen Divide & Conquer sur les sous-matrices

$$\Gamma_1 = \begin{bmatrix} T_{1,1} & 0 \\ 0 & T_{1,2} \end{bmatrix} + b_3 u_1 u_1^T \text{ et } \Gamma_2 = \begin{bmatrix} a_6 & 0 \\ 0 & T_{2,2} \end{bmatrix} + b_6 u_2 u_2^T.$$

Plus généralement avant de commencer le Divide & Conquer, on décompose la matrice Γ en sous-matrices tridiagonales Γ_1, Γ_2,...,Γ_t telles que chacune de ces sous matrices ne possède pas de zéro sur leur sous-diagonale respective. Les sous-matrices peuvent avoir des tailles complétement différentes.

$$.\Gamma = \begin{bmatrix} \Gamma_1 & 0 & 0 & 0 \\ 0 & \Gamma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \Gamma_t \end{bmatrix} \tag{4.9}$$

On décompose chaque sous-matrice comme on l'a décrit ci-dessus dans l'équation 4.7.

$$\Gamma = \begin{bmatrix} \begin{bmatrix} T_{1,1} & 0 \\ 0 & T_{1,2} \end{bmatrix} + \rho_1 u_1 u_1^T & 0 & 0 & 0 \\ 0 & \begin{bmatrix} T_{2,1} & 0 \\ 0 & T_{2,2} \end{bmatrix} + \rho_2 u_2 u_2^T & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \begin{bmatrix} T_{t,1} & 0 \\ 0 & T_{t,2} \end{bmatrix} + \rho_t u_t u_t^T \end{bmatrix} \tag{4.10}$$

et on résout le problème aux valeurs propres de chaque sous-matrice $T_{i,1}$ et $T_{i,2}$ pour $1 \leq i \leq n$.

4.3.2 Conquer : Valeurs propres et Vecteurs propres

On peut donc maintenant résoudre le problème aux valeurs propres de T_1 et T_2 telles que

$$T_1 = Q_1 D_1 Q_1^T \tag{4.11}$$
$$T_2 = Q_2 D_2 Q_2^T. \tag{4.12}$$

D_1 et D_2 sont des matrices diagonales contenant respectivement les valeurs propres de T_1 et T_2. Q_1 et Q_2 sont des matrices orthogonales composées respectivement des vecteurs propres de T_1 et T_2.

Exemple :

$$D_1 \approx \begin{bmatrix} -5.10 & 0 \\ 0 & 5.10 \end{bmatrix} \qquad D_2 \approx \begin{bmatrix} -2.45 & 0 \\ 0 & 11.45 \end{bmatrix}$$

$$Q_1 \approx \begin{bmatrix} 1 & -1.22 \\ 1 & 0.82 \end{bmatrix} \qquad Q_2 \approx \begin{bmatrix} 1 & -0.57 \\ 1 & 1.74 \end{bmatrix}$$

L'équation (4.7) devient alors

$$\begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} \Gamma \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} + \rho v v^T \tag{4.13}$$

avec

$$v v^T = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} u u^T \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \tag{4.14}$$

donc

$$v = \begin{bmatrix} \text{dernière colonne de } Q_1 \\ \text{1er colonne de } Q_2 \end{bmatrix}. \tag{4.15}$$

Résoudre le problème $\Gamma x = \lambda x$ revient alors à résoudre

$$(D + \rho v v^T)x = \lambda x \text{ avec } D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}. \tag{4.16}$$

En ordonnant dans l'ordre croissant, les valeurs propres $d_1, d_2, \ldots, d_n$ de D qui sont les éléments de sa diagonale, on peut montrer [11, 35] que les valeurs propres $\lambda_1, \lambda_2, \ldots, \lambda_n$ de Γ sont les solutions de l'équation en λ, dite séculaire :

$$\textbf{équation séculaire : } \sum_{i=1}^{n} \frac{v_i^2}{d_i - \lambda} + \frac{1}{\rho} = 0. \tag{4.17}$$

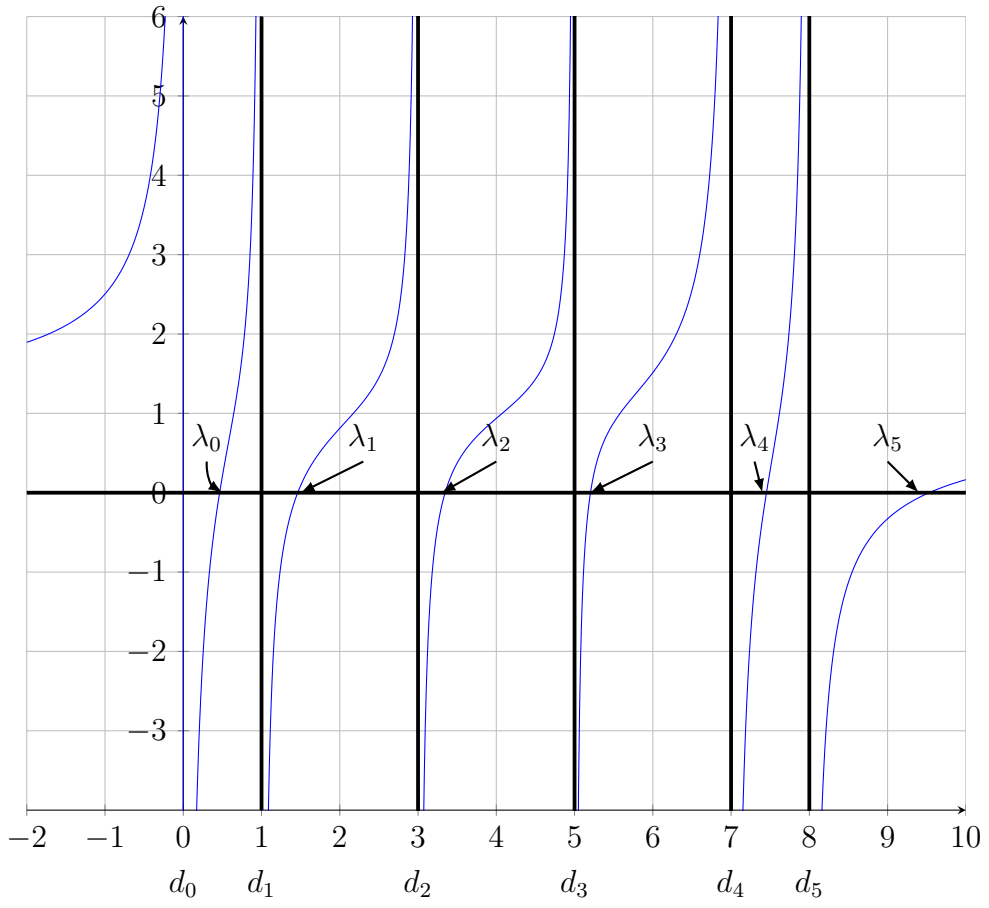


Figure 4.1 – Représentation de la résolution de l'équation séculaire (4.17).

La référence [65] fournit un bon résumé des différentes méthodes pour résoudre l'équation (4.17) illustrée par la Figure 4.1. Elle propose aussi une procédure hybride dont les performances rivalisent avec le schéma de Gragg [45] qui a une convergence cubique.

Quand une valeur propre λ solution de (4.17) est trouvée, le vecteur propre V_λ de $\begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} + \rho v v^t$ est calculé par

$$V_\lambda = \left(\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} - \lambda I \right)^{-1} \widetilde{u} \tag{4.18}$$

où le vecteur $\widetilde{u}$ est défini dans [40] et dans [47] grâce au théorème de Löwner :

$$\widetilde{u}_k^2 = \frac{1}{|\rho|} \frac{\prod_{j=1}^{n} |d_k - \lambda_j|}{\prod_{j=1, j\neq k}^{n} |d_k - d_j|}. \tag{4.19}$$

Le remplacement de v par $\widetilde{u}$ est très important pour assurer la stabilité et la robustesse de l'algorithme, en particulier lorsque certaines valeurs propres sont presque égales.

Supposons maintenant que tous les vecteurs propres $W = (V_\lambda)_{\lambda \text{ VP de } U}$ sont connus. Pour assembler (Conquer), on a besoin de calculer les vecteurs propres de Γ en utilisant le produit

$$\begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} W. \tag{4.20}$$

Cette dernière étape est la plus lourde numériquement de tout l'algorithme. C'est pour cela que l'on traite tout particulièrement du produit de matrice dans la Section 4.4.1 dans le cas d'un calcul en parallèle.

Un problème de stabilité va se poser lorsque v_i ou $d_i - d_j$ disparaissent numériquement ($i \neq j$). On utilise une rotation dite de Jacobi [47] sur les vecteurs associés aux valeurs propres d_i et d_j pour gérer la déflation $d_i - d_j \approx 0$. Dans le cas où $v_i \approx 0$, on remplace le vecteur propre par un vecteur canonique [47]. Les déflations sont déjà présentées dans les références citées ci-dessus, mais on montre dans la Section 4.5 qu'il faut les gérer avec plus de subtilité pour améliorer les erreurs.

Avant de passer à une présentation d'une optimisation de la précision, on propose deux optimisations algorithmiques dans le cadre du batch computing et l'utilisation du Divide & Conquer pour la diagonalisation des matrices tridiagonales, l'une sur la multiplication de matrice (cf. Section 4.4.1) pour faire le calcul (4.20) et le tri fusion parallélisé dans la Section 4.4.2 pour trier les valeurs propres $d_1, d_2, ..., d_n$.

4.4 Optimisation en temps d'execution

4.4.1 Produit de matrices en batch computing

Les matrices sont enregistrées de façon à ce que les valeurs soient alignées. Une matrice carrée $A \in \mathcal{M}_d(\mathbb{R})$ de taille d s'écrivant :

$$\begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,d-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,d-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d-1,0} & a_{d-1,1} & \dots & a_{d-1,d-1} \end{bmatrix}$$

s'enregistre en alignant les lignes :

$$A_{lig} = [a_{0,0}, a_{0,1}, \dots, a_{0,d-1}, a_{1,0}, a_{1,1}, \dots, a_{1,d-1}, \dots, a_{d-1,0}, a_{d-1,1}, \dots, a_{d-1,d-1}]$$

ou en alignant les colonnes :

$$A_{col} = [a_{0,0}, a_{1,0}, \dots, a_{d-1,0}, a_{0,1}, a_{1,1}, \dots, a_{d-1,1}, \dots a_{0,d-1}, a_{1,d-1}, \dots, a_{d-1,d-1}]$$

Dans le programme Divide & Conquer développé par [3], d threads traitent une matrice de taille d. On considére qu'un thread calcule d valeurs de chaque ligne du produit de matrices (4.20)

Sachant que les matrices sont de petites dimensions, les méthodes de produit par bloc du genre Strassens sont inintéressantes ici. Quatre options sont offertes pour faire un produit de matrice AB où A et B sont des matrices carrés flottantes :

- Les deux matrices sont enregistrées sous la forme A_{lig} et B_{lig}. On parlera d'un produit ligne-colonne (row-column).
- La matrice A est enregistrée sous la forme A_{col} et B sous la forme B_{lig}. On parlera d'un produit colonne-colonne (column-column).
- La matrice A est enregistrée sous la forme A_{col} et B sous la forme B_{col}. On parlera d'un produit colonne-ligne (column-row).
- La matrice A est enregistrée sous la forme A_{ligne} et B sous la forme B_{col}. On parlera d'un produit ligne-ligne (row-row).

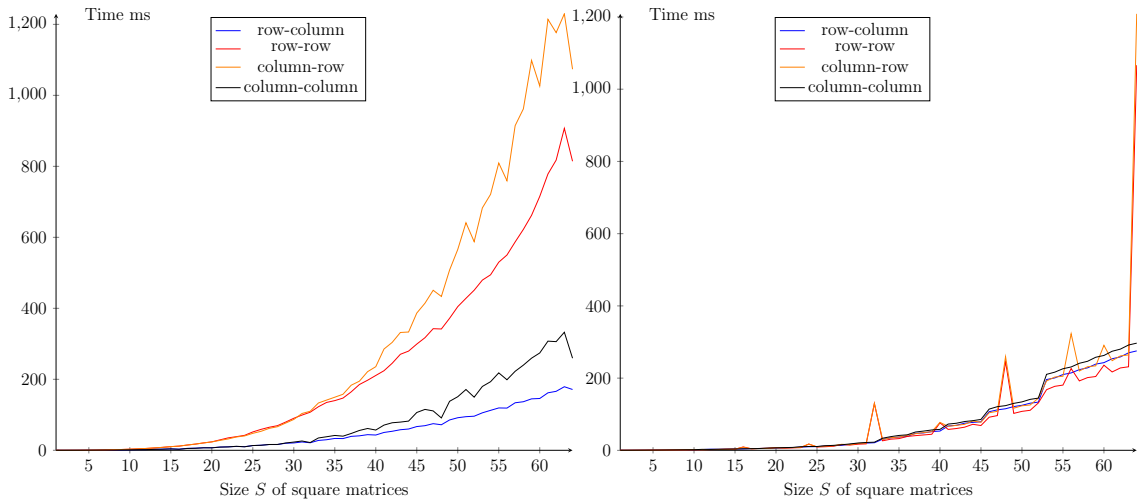
Le code est dans l'annexe 4.B. Il est écrit pour faire le produit ligne-colonne dans la shared memory. On peut tester les temps de calculs sur les autres cas en décommentant et en commentant les lignes adéquates dans la partie du code 4.1. Le calcul final n'est

```
for(i=0;i<n;i++){
        sum=0;
        for(j=0;j<n;j++){
                sum+=A[i*n+j]*B[j*n+tidx];//row-column calcul
                //of value of indice i,tidx
                //sum+=A[j*n+i]*B[j*n+tidx];//column-column
                //sum+=A[j*n+i]*B[tidx*n+j];//column-row
                //sum+=A[i*n+j]*B[tidx*n+j];//row-row
        }
        C[i*n+tidx]=sum;
}
```

Figure 4.1 – Extract of Code Cuda/C product of matrix. tidx is a number of a thread

pas correct pour les autres cas mais on ne s'intéresse ici qu'au temps de calcul. tidx est un numéro de thread.

Sur la Figure 4.2a, on voit que les meilleurs temps de calcul sont obtenus pour le cas ligne-colonne alors que les accès successifs sur la deuxième boucle ne sont pas contigus sur la matrice B. On peut expliquer les performances du cas ligne-colonne car tous les threads d'un même bloc accèdent à des valeurs contiguës de B. On retrouve d'ailleurs des résultats équivalents pour le cas colonne-colonne. Ce dernier cas est un peu moins bon car les accès successifs sur A ne sont pas contigus dans la deuxième boucle. On peut améliorer les résultats pour les cas ligne-ligne et colonne-ligne si on passe à un calcul sur la Shared Memory mais on aperçoit alors des conflits de ban pour certaines tailles de matrices. Dans tous les cas, il n'y a pas de gain notable quand on passe à la Shared memory par rapport au cas ligne-colonne. On améliore donc la complexité en temps d'exécution en utilisant le produit ligne-colonne dans le produit (4.20).



(a) Computation with global memory. (b) Computation with shared memory.

Figure 4.2 – Average time for products of matrices with batch computing.

4.4.2 Tri fusion parallèle

Pour trier les valeurs propres, on implémente la fusion de deux tableaux triés présentée dans " GPU Merge Path - A GPU Merging Algorithm " par O. Green, R. McColl et D. A. Bader [46]. Pour l'algorithme du Merge Path, on procéde comme suit. Soient A et B deux tableaux ordonnés (ordre croissant), on veut les fusionner dans un tableau trié M. La fusion de A et B est basée sur un chemin qui commence dans le coin supérieur gauche de la grille $|A| \times |B|$ et arrive dans le coin inférieur droit. L'algorithme de fusion séquentiel est donné par l'Algorithme 6 et un exemple visuel du Merge Path est fourni dans la Figure 4.3.

Chaque point de la grille a une coordonnée $(i, j) \in [\![0, |A|]\!] \times [\![0, |B|]\!]$. Le Merge Path commence au point $(i, j) = (0, 0)$ dans le coin supérieur gauche de la grille. Si $A[i] < B[j]$ le chemin descend sinon il va à droite. Le tableau $[\![0, |A| - 1]\!] \times [\![0, |B| - 1]\!]$ de valeurs booléennes $A[i] < B[j]$ n'est pas important dans l'algorithme. Cependant, cela montre clairement que le chemin de fusion est une frontière entre les 1 et les 0.

Pour paralléliser l'algorithme, la grille doit être étendue à la taille $\max(|A|, |B|) \times \max(|A|, |B|)$. On note respectivement K_0 et P_0 le point bas et le point haut d'une des diagonales ascendantes Δ_k donc $\Delta_k = (K_0 P_0)$. Sur GPU, chaque thread $k \in [\![0, |A| + |B| - 1]\!]$ est responsable d'au moins une diagonale. On trouve l'intersection du Merge Path avec la diagonale Δ_k avec une recherche dichotomique décrite succinctement dans l'Algorithme 7.

Pour un $\Delta = [KP]$ dans l'ensemble des diagonales, chaque diagonale est gérée par un thread parmi les $n = |A| + |B|$ threads. K et P sont les points haut et bas de la diagonale. Les coordonnées du point Q de la grille est défini comme le point le plus proche

Algorithm 6 Sequential Merge

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$

```
procedure SEQUENTIAL MERGE(A, B, M)
    j = 0 and i = 0
    while i < |A| and j < |B| do
        if A[i] < B[j] then
            M[i+j]=A[i]                          ▷ The path goes down
            i = i + 1
        else
            M[i+j]=B[j]
            j = j + 1                            ▷ The path goes right
        end if
    end while
    if i == |A| then          ▷ The path is at some bottom-node, then goes right
        for t = j to |B| − 1 do
            M[|A| + t] = B[t]
        end for
    else                        ▷ The path is at some right-node, then goes down
        for t = i to |A| − 1 do
            M[|B| + t] = A[t]
        end for
    end if
end procedure
```

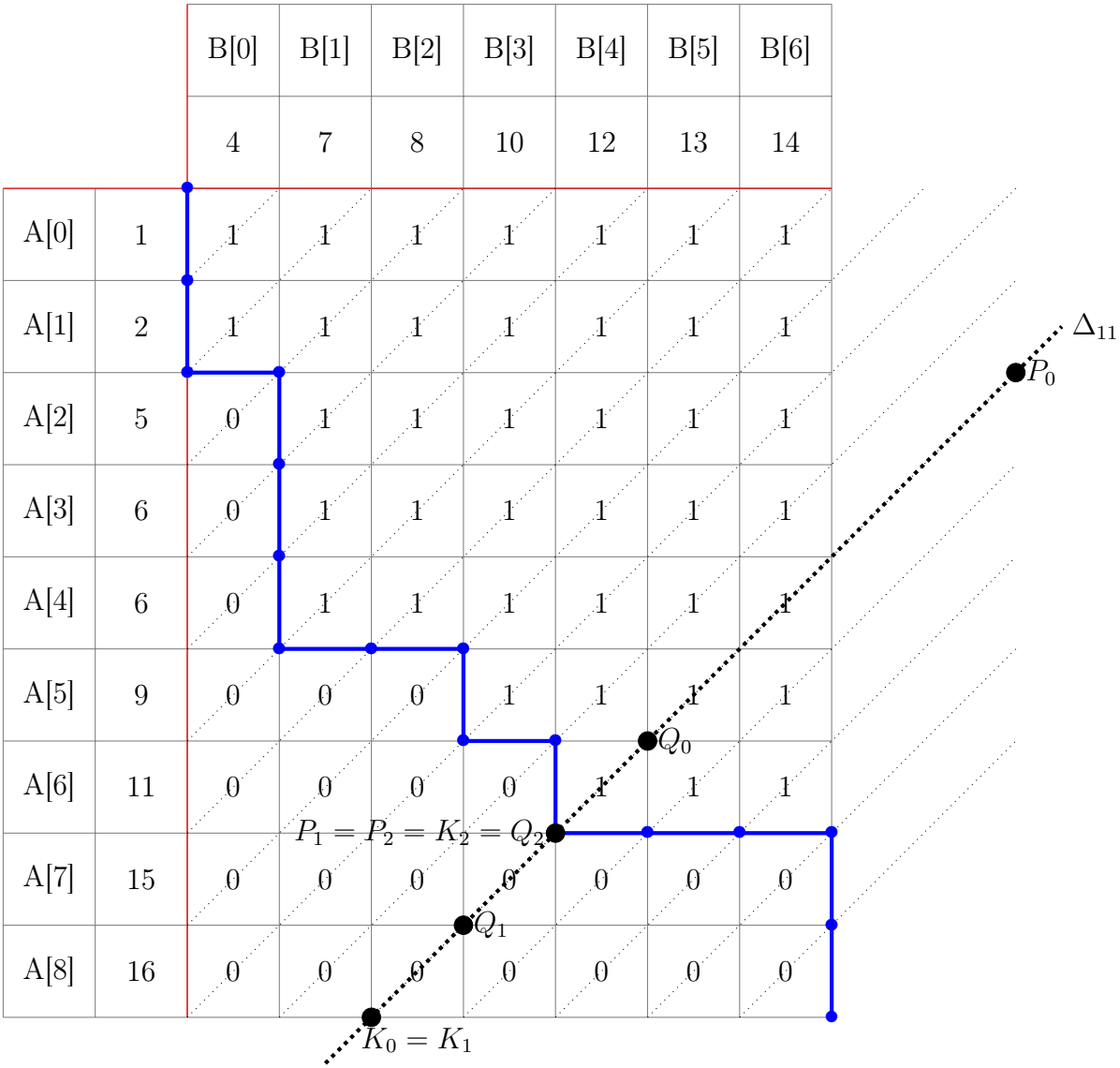


Figure 4.3 – An example of Merge Path procedure

inférieurement du milieu de $[KP]$. Si le point Q est strictement au dessus du Merge Path ($A[Q_y] \leq B[Q_x - 1]$), le point haut de la diagonale P devient égal à $P = (Q_x - 1, Q_y + 1)$. Si le point Q est strictement en dessous du Merge Path ($A[Q_y - 1] > B[Q_x]$)), le point bas de la diagonale K est égal à $K = (Q_x - 1, Q_y - 1)$. Si le point est en dessous et au dessus du Merge Path, on a trouvé l'intersection de la diagonale Δ_k avec le Merge Path. On connait alors la valeur de $M[thread\ index]$. Pour l'implémentation, [46] propose

Algorithm 7 Parallelized Merge Path summary (Indices of n threads are 0 to $|A|+|B|-1$)

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  for each thread **in parallel do**
    Definition of diagonal $\Delta = [KP]$ of the grid
    while True **do**
      Computation of $Q \in \Delta$, "middle" of $[KP]$ in the grid.
      if Q is under or in the Merge Path **then**
        if Q is above or in the Merge Path **then**
          Fill $M[thread\ index]$ ▷ Merge in M
          Break
        else
          $K = (Q_x + 1, Q_y - 1)$ ▷ is the new lower point of the diagonal.
        end if
      else
        $P = (Q_x - 1, Q_y + 1)$ ▷ is the new upper point of the diagonal.
      end if
    end while
  end for

une version condensée de l'Algorithme 8. La version de [46] n'explicite pas le calcul des coordonnées des points Q, K et P. Le début de l'Algorithme 8 distingue les diagonales coupant le bas du tableau (Lower Diagonal) de celles coupant la partie gauche du tableau (Upper Diagonal). Cependant l'algorithme 8 n'est pas implémentable dans l'état car pour fonctionner, certaines conditions nécessitent d'être considérées comme fausses ou vraies alors qu'il y a des sorties de tableau. On propose donc l'algorithme 9 pour résoudre le problème.

Algorithm 8 Parallelized Merge Path Detail(Indices of n threads are 0 to $|A|+|B|-1$)

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  for each thread k **in parallel do**
    if $k > |A|$ **then** ▷ Lower Diagonals
      $K = (k - |A|, |A|)$ ▷ Low point of diagonal
      $P = (|A|, k - |A|)$ ▷ High point of diagonal
    else ▷ Upper Diagonals
      $K = (0, k)$ ▷ Low point of diagonal
      $P = (k, 0)$ ▷ High point of diagonal
    end if
    while True **do**
      $offset = abs(K_y - P_y)/2$
      $Q = (K_x + offset, K_y - offset)$
      if $A[Q_y] > B[Q_x - 1])$ **then**
        ▷ Condition to be under the merge path
        if $A[Q_y - 1] \leq B[Q_x]$ **then**
          ▷ Condition to be above the merge path
          if $A[Q_y] \leq B[Q_x])$ **then**
            $M[k] = A[Q_y]$ ▷ Merge in M
          else
            $M[k] = B[Q_x]$
          end if
          Break
        else
          $K = (Q_x + 1, Q_y - 1)$ ▷ is the new lower point of the diagonal.
        end if
      else
        $P = (Q_x - 1, Q_y + 1)$ ▷ is the new upper point of the diagonal.
      end if
    end while
  end for

Algorithm 9 Parallelized Merge Path (Indices of n threads are 0 to $|A| + |B| - 1$)

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  for each thread k **in parallel do**
    if $k > |A|$ **then**
      $K = (k - |A|, |A|)$ ▷ Low point of diagonal
      $P = (|A|, k - |A|)$ ▷ High point of diagonal
    else
      $K = (0, k)$ ▷ Low point of diagonal
      $P = (k, 0)$ ▷ High point of diagonal
    end if
    while True **do**
      $offset = abs(K_y - P_y)/2$
      $Q = (K_x + offset, K_y - offset)$
      if $Q_y \geq 0$ and $Q_x \leq B$ and ($Q_y = |A|$ or $Q_x = 0$ or $A[Q_y] > B[Q_x - 1]$) **then**
        ▷ Condition to be under the merge path
        if $Q_x = |B|$ or $Q_y = 0$ or $A[Q_y - 1] \leq B[Q_x]$ **then**
          ▷ Condition to be above the merge path
          if $Q_y < |A|$ and ($Q_x = |B|$ or $A[Q_y] \leq B[Q_x]$) **then**
            $M[k] = A[Q_y]$ ▷ Merge in M
          else
            $M[k] = B[Q_x]$
          end if
          Break
        else
          $K = (Q_x + 1, Q_y - 1)$ ▷ is the new lower point of the diagonal.
        end if
      else
        $P = (Q_x - 1, Q_y + 1)$ ▷ is the new upper point of the diagonal.
      end if
    end while
  end for

On considère dans l'algorithme 9 et dans la suite qu'un argument booléen est évalué de gauche à droite. Par conséquent, le booléen p ET q est automatiquement Faux si p = Faux sans évaluation de q. p OU q est automatiquement Vrai si p = Vrai sans évaluation de q. Ce type d'évaluation est utilisé avec le langage python et C. Il est utile dans un saut conditionnel "IF ... ELSE ..." pour gérer les sorties de tableaux possibles qui ne correspondent pas à une erreur algorithmique. La diagonale $[KP]$ qui initialise l'algorithme est donnée par

$$K = \begin{cases} (0, k) & \textbf{if } k \in [\![0, |A|]\!] \\ (k - |A|, |A|) & \textbf{if } k \in [\![|A| + 1, |A| + |B| - 1]\!] \end{cases}$$

$$P = \begin{cases} (k,0) & \textbf{if } k \in [\![0, |A|]\!] \\ (|A|, k-|A|) & \textbf{if } k \in [\![|A|+1, |A|+|B|-1]\!] \end{cases}$$

En résumé, dans l'algorithme 9, on a rajouté les précisions ci-dessous qui ne sont pas forcément indiquées dans [46] pour permettre son implémentation :

1. Le point Q est un milieu approximatif dans la recherche dichotomique.

   Soit T le milieu de $[KP]$, $Q = \begin{cases} T \text{ si } T \text{ appartient à la grille} \\ T'(T_x - 0.5; T_y - 0.5) \text{ sinon} \end{cases}$

   Q est donc le point de la grille le plus proche inférieurement du milieu de $[KP]$. Il appartient à la grille et à $[KP]$.

2. La suite des points K et P de la recherche dichotomique est indiquée dans l'Algorithme 9 et on les visualise sur la Figure 4.3 .

3. Dans $offset = abs(K_y - P_y)/2$, la division est euclidienne.

4. Les conditions $A[Q_y] > B[Q_x - 1]$, $A[Q_y] \leq B[Q_x]$ et $A[Q_y] \leq B[Q_x]$ sont considérées comme fausses s'il y a une sortie de tableau. On a ajouté des conditions qui évitent une sortie de tableau en utilisant l'évaluation des booléens de gauche à droite.

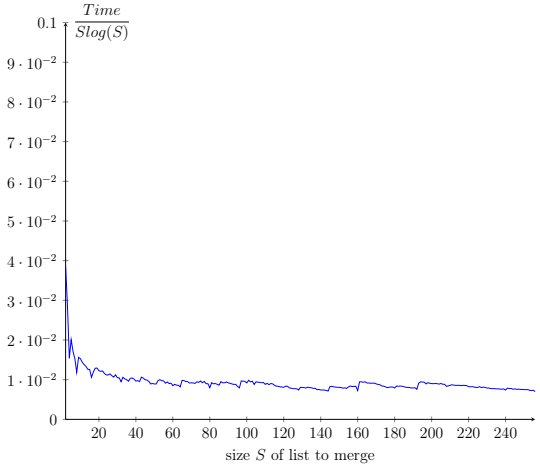
5. On a remplacé une partie de l'algorithme de [46] par le morceau ci-dessous. Cette partie permet de fusionner A et B dans M sans utiliser de tableaux intermédiaires dans la shared memory ou la global memory de la carte Nvidia.

   ...
   if $Q_x < |A|$ and ($Q_y = |B|$ or $A[Q_x] \leq B[Q_y]$) **then**
   $M[k] = A[Q_x]$
   else
   $M[k] = B[Q_y]$
   end if
   ...

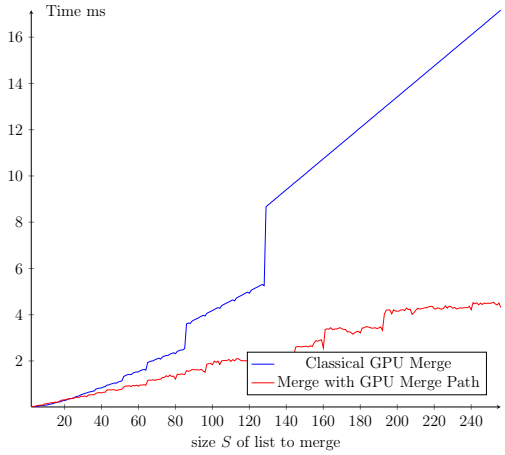
6. Dans la recherche dichotomique discrète de l'Algorithme 9 on utilise le morceau suivant :

   ```
      ....
    else
      K = (Q_x + 1, Q_y - 1)
    end if
else
      P = (Q_x - 1, Q_y + 1)
 end if
    ....
   ```

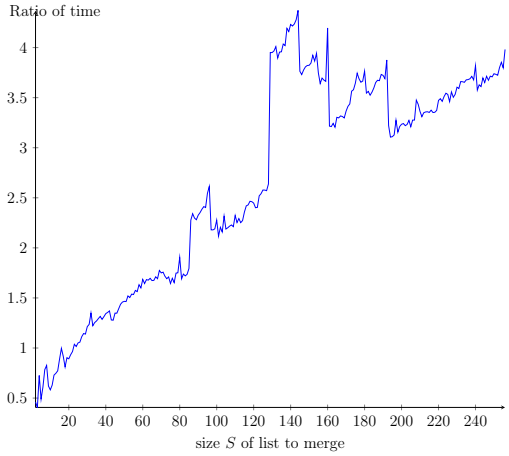
   qui signifie que, si Q est au-dessus du Merge Path, on cherche un point du Merge Path sur l'intervalle $[K; Q[$ semi-ouvert de la grille sinon on le cherche sur $]Q; P]$.



(a) Fusion with Merge Path. Illustration of time complexity $O(n \log n)$. 16384 arrays to merge with 16384 blocks.



(b) Comparison between Classical Merge and Merge Path. 16384 arrays to merge with 16384 blocks.



(c) Ratio between classical merge and Merge Path. 16384 arrays to merge with 16384 blocks.

Les tests sont faits sur 16384 tableaux de tailles n égales à fusionner avec 16384 blocs. On utilise $2n$ threads pour chaque blocs. On montre dans la Figure 4.4a que le calcul de la fusion se fait en $O(nlog(n))$. Les figures 4.4b et 4.4c montre bien l'intérêt du Merge Path par rapport à une fusion classique. Le ratio de temps gagné peut aller jusqu'à 4 par rapport à une fusion classique. La différence essentielle entre la fusion classique et le Merge Path est le nombre de threads utilisés pour chaque couple de tableaux de taille n à fusionner. Pour la fusion classique, on utilise un seul thread par couple alors que l'on utilise n threads pour le Merge Path, ce qui est adéquat pour le Cuppen Divide & Conquer en GPU.

4.4.3 Ajout des optimisations dans l'ancien code Cuppen Divide & Conquer

L'ancien code est suffisamment optimisé pour des matrices de taille inférieure à 32. Pour des matrices de taille supérieure à 32, on ajoute les optimisations pour éviter les conflits de ban. On voit sur la Figure 4.5 que les optimisations améliorent des temps de calculs pour des matrices de taille supérieure à 52.

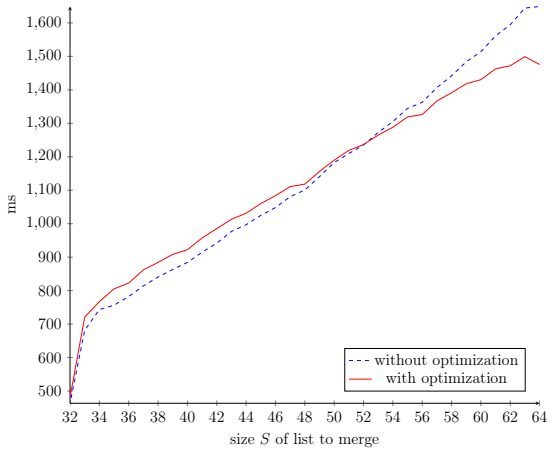


Figure 4.5 – Speed up Comparaison beetween old Batch Divide & Conquer with and without optimizations.

4.5 Gestion des déflations dans le Batch Cuppen Divide & Conquer

Dans le cas où $d_i \approx d_j$ (déflation de Type I), on procède à une transformation sur la matrice des vecteurs propres en utilisant une rotation plane dite de Jacobi qui est décrite dans [47]. Dans le cas où $v_i \approx 0$ (Déflation de Type III), on remplace le vecteur propre par un vecteur canonique. On montre dans cette section l'intérêt de ces déflations sur la précision des calculs. Et surtout on propose un nouveau type de déflation qui apporte une véritable amélioration de la précision.

On résume l'algorithme de [3] avec les points suivants :

A chaque étape du Divide & Conquer :

1. Faire les rotations de Jacobi nécessaires dans le cas où il y a des valeurs propres égales numériquement $d_i \approx d_j$ pour $i \neq j$.

2. Tri des valeurs propres d_i.

3. Résolution de l'équation séculaire pour obtenir les valeurs propres λ_i.

4. Utilisation du théorème de Löwner et gestion du cas $v_i \approx 0$.

5. Calcul des déflations.

6. Calcul des vecteurs propres.

7. Multiplication de matrices pour l'assemblage.

En testant cet algorithme, on s'est aperçu que la précision de calcul s'améliore en gérant les cas où les anciennes valeurs propres d_i sont égales (du point de vue numérique) aux nouvelles valeurs propres λ_i. Pour pouvoir gérer ces nouveaux cas que l'on appelle Type II, on transforme l'algorithme de la manière suivante :

A chaque étape du Divide & Conquer :

1. Tri des valeurs propres d_i.

2. Résolution de l'équation séculaire pour obtenir les valeurs propres λ_i et conservation des anciennes valeurs propres d_i.

3. Calculs des déflations dans le cas $d_i \approx d_{i+1}$ (Type I).

4. Calculs des déflations dans les cas $\begin{cases} d_i \approx \lambda_{i-1} \\ \quad \text{ou} \\ d_i \approx \lambda_i \\ \quad \text{ou} \\ d_i \approx \lambda_{i+1} \end{cases}$ (Type II)

5. Calcul des déflations dans le cas $v_i \approx 0$.(Type III)

6. Calcul des vecteurs propres avec le théorème de Löwner.

7. Multiplication de matrices pour l'assemblage.

Les déflations de Type II se gèrent avec une rotation de Jacobi de la même manière que les déflations de Type I.

4.5.1 Résultats

Le code original se trouve à l'adresse internet [1]. Le nouveau code se trouve sur la page [2]. Pour étudier les résultats, on construit des matrices aléatoires tridiagonales symétriques. Les valeurs prises ne contiennent pas de zéros sur les sous-diagonales. Les valeurs de la matrice sont inférieures à 10 en valeur absolue. On note Γ la matrice

i	d_i	λ_i	$\lvert d_5 - d_6 \rvert$	Erreur Avec déflation	Erreur Sans déflation
5	5.2230172157	5.2237019539			
6	5.2314786911	5.7325143814	$8.46 \cdot 10^{-3}$	$1.28 \cdot 10^{-3}$	$1.90 \cdot 10^{-5}$
7	6.0241780281	6.0714206696			

Table 4.1 – Comparaison de l'erreur avec et sans déflation. Taille de matrice 16.

i	d_i	λ_i	$\lvert d_0 - d_1 \rvert$	Erreur Avec déflation	Erreur Sans déflation
0	3.9350430965	3.1608371735			
1	3.9350435734	3.9350440502	$4.76 \cdot 10^{-7}$	$9.2 \cdot 10^{-5}$	4.32
2	5.9429497719	4.1755232811			

Table 4.2 – Cas $d_i \approx d_{i+1}$. Comparaison entre ancien et nouveau code. Taille de matrice 16.

tridiagonale. $\widetilde{D}$ est la matrice diagonale des valeurs propres obtenue numériquement et $\widetilde{Q}$ la matrice des vecteurs propres associées. L'erreur de calcul est évaluée en calculant :

$$erreur(\Gamma) = \max_{0 \leq i,j < n} \left| \left(\Gamma - \widetilde{Q}\widetilde{D}\widetilde{Q}^T \right)_{i,j} \right|$$

On parlera d'erreur maximale pour l'erreur maximale sur l'ensemble des matrices du tirage aléatoire.

Pour faire une déflation de Type I ($d_i \approx d_{i+1}$), il faut un choix de seuil pour le zero numérique. On considére que l'on fait une déflation si

$$|d_i - d_j| < C\epsilon\|\Gamma\| \text{ où } \epsilon \text{ est l'erreur flottante [47].}$$

La difficulté est de trouver la valeur de C. Dans les tableaux suivants qui illustrent cette recherche, on met en bleu la meilleure erreur et en rouge la pire. Le vert indique une égalité numérique. Le seuil ne doit pas être trop élevé (Ici $8.5 \cdot 10^{-3}$ cf. Tableau 4.1). On peut avoir une erreur plus grande avec déflation que sans déflation. Mais il convient surtout que ce seuil ne soit pas être trop petit (ici $4. \cdot 10^{-7}$ cf. tableau 4.2) car si la déflation n'est pas faite, l'erreur risque de s'accroitre considérablement.

Dans les cas de déflation de Type I , le nouveau code peut améliorer l'erreur comme dans l'exemple du Tableau 4.3.
Mais il peut y avoir des cas où cette erreur est un peu moins bonne avec le nouveau code (cf. Tab 4.4). Pour l'ensemble des matrices où il y a une déflation de Type I, la moyenne de l'erreur est divisée par 9 et l'erreur maximale est divisée par 28 avec le nouveau code.

i	d_i	λ_i	$\lvert d_1 - d_2 \rvert$	Erreur Ancien Code	Erreur Nouveau Code
1	8.8155546188	8.2229280472			
2	8.8158578873	8.8156385422	$8.39 \cdot 10^{-5}$	$1.04 \cdot 10^{-3}$	$1.40 \cdot 10^{-4}$
3	12.9924802780	12.3674755096			

Table 4.3 – Cas $d_i \approx d_{i+1}$. Comparaison entre ancien et nouveau code. Taille de matrice 16.

i	d_i	λ_i	$\lvert d_0 - d_1 \rvert$	Erreur Ancien Code	Erreur Nouveau Code
0	3.9350430965	3.1608371735			
1	3.9350435734	3.9350440502	$4.76 \cdot 10^{-7}$	$9.2 \cdot 10^{-5}$	$4.94 \cdot 10^{-4}$
2	5.9429497719	4.1755232811			

Table 4.4 – Cas $d_i \approx d_{i+1}$. Comparaison entre ancien et nouveau code. Taille de matrice 16.

Il y a donc une amélioration avec le nouveau code pour les matrices avec une grande erreur comme l'illustre la Figure 4.1 pour les matrices subissant une déflation de Type I. Le nouveau code est plus stable, l'écart type de l'erreur est divisée par 20.

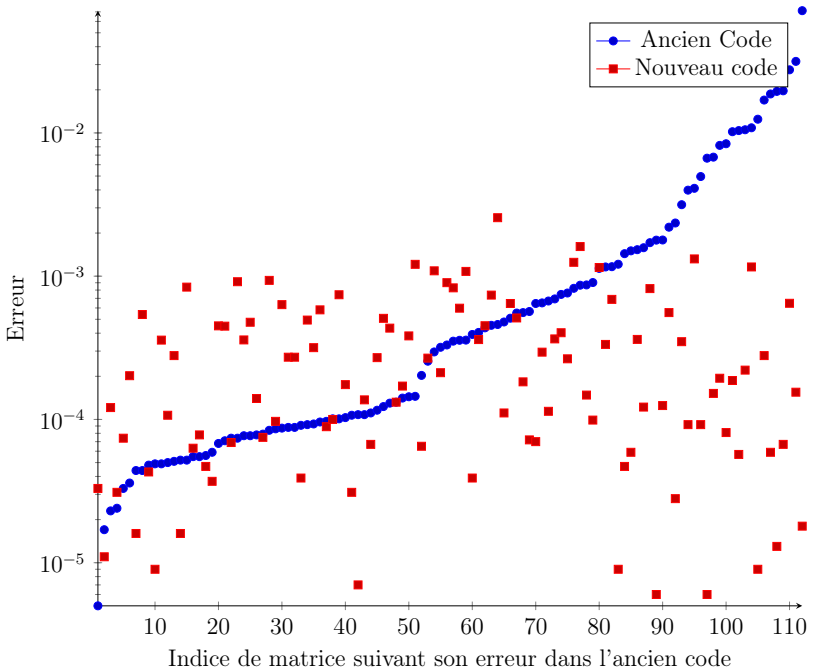


Figure 4.1 – Comparaison de l'erreur dans le cas d'une déflation de Type I entre ancien et nouveau code. Sur un ensemble de matrices de taille 16 indicées par ordre croissant de l'erreur dans l'ancien code.

De plus, on voit sur le tableau (cf. Tableau 4.5), qui donne l'erreur maximale sur toutes les matrices diagonalisées en batch computing, que cette fois l'erreur maximale est divisée par 10. On améliore aussi l'erreur moyenne.

Taille de matrice	4	8	16	32
Erreur max Ancien Code	0.006803	0.052962	0.071346	0.267637
Erreur max Nouveau Code	0.000622	0.001246	0.009996	0.060
Erreur moyenne Ancien Code	$1.13 \cdot 10^{-5}$	$3.8 \cdot 10^{-4}$	$8.51 \cdot 10^{-3}$	$2.88 \cdot 10^{-2}$
Erreur moyenne Nouveau Code	$8.15 \cdot 10^{-6}$	$3.66 \cdot 10^{-5}$	$6.73 \cdot 10^{-4}$	$1.18 \cdot 10^{-2}$

Table 4.5 – Comparaison de l'erreur maximale et moyenne sur l'ancien et le nouveau code.

Le nouveau code améliore donc l'erreur maximale sur les matrices de tailles 4, 8, 16 et 32 grâce à l'ajout de la déflation de Type II et une réorganisation de l'algorithme. Pour appliquer la déflation de Type II, on choisit trois constantes C_1, C_2 et C_3 telles que

$$|d_i - \lambda_i| < C_1\epsilon\|\Gamma\|, \; |d_i - \lambda_{i+1}| < C_2\epsilon\|\Gamma\| \text{ et } |d_i - \lambda_{i-1}| < C_3\epsilon\|\Gamma\|$$

On a cherché petit à petit les seuils C_1, C_2 et C_3 pour améliorer l'erreur sur ces tailles de matrices mais cela devient de plus en plus difficile de trouver les bons paramètres pour des tailles supérieure à 32. On peut voir avec plus de précision comment s'améliore l'erreur sur le Tableau 4.6 pour les tailles 32 et dans l'Annexe 4.A pour les autres tailles.

Erreur	Ancien Code	Nouveau Code
erreur <1.	100.00 %	100.00 %
erreur <0.1	99.98 %	100.00%
erreur <0.01	88.25 %	99.56 %
erreur <0.001	25.48 %	35.67 %
erreur <0.0001	0.68 %	0.74 %

Table 4.6 – Pourcentage de matrice de taille 32 qui ont une erreur inférieure à un certain seuil.

Ces tableaux montrent que les choix algorithmiques vont dans la bonne direction. Pour chaque type de déflation, il faut trouver les seuils adaptés à la taille des matrices. A partir des matrices de taille 64, les grandes erreurs viennent de matrices avec des doubles ou triples déflations successives, voire plus.

Si une nouvelle valeur propre est très proche d'une ancienne valeur propre, comme on le montre sur la Figure 4.2, la déflation de Type II peut être appliquée et on va voir de quelle manière elle améliore la précision. On montre aussi l'intérêt de la déflation de Type II sur trois exemples avec les trois cas où une déflation de ce type est applicable.

Dans les très rares cas où la déflation de Type II n'améliore pas l'erreur avec les seuils choisis, l'erreur reste très proche de l'ancienne erreur (au maximum d'un facteur 1.08 et pour une erreur de l'ordre de 10^{-3}). Sur les exemples suivants (Tableaux 4.7, 4.8, 4.9), l'erreur est améliorée grâce au Type II.

i	d_i	λ_i	$\lvert d_1 - \lambda_1 \rvert$	Erreur Ancien Code	Erreur Nouveau Code
0	3.605526	3.5633531			
1	3.610064	3.6100631	$9.53 \cdot 10^{-7}$	$1.48 \cdot 10^{-4}$	$5.4 \cdot 10^{-5}$
2	5.3898721	5.2089825			

Table 4.7 – Cas $d_i \approx \lambda_i$. Comparaison entre ancien et nouveau code. Taille de matrice 16

i	d_i	λ_i	$\lvert d_4 - \lambda_3 \rvert$	Erreur Ancien Code	Erreur Nouveau Code
2	3.1267593	3.4910524			
3	4.3632765	4.3694711	$9.53 \cdot 10^{-7}$	$3.376 \cdot 10^{-3}$	$4.6 \cdot 10^{-5}$
4	4.369472	4.8753777			

Table 4.8 – Cas $d_i \approx \lambda_{i-1}$. Comparaison entre ancien et nouveau code. Taille de matrice 16

i	d_i	λ_i	$\lvert d_{13} - \lambda_{14} \rvert$	Erreur Ancien Code	Erreur Nouveau Code
12	8.9452028	8.7770767			
13	9.8264313	9.7585649	$9.53 \cdot 10^{-7}$	1.568×10^{-3}	$1.8 \cdot 10^{-5}$
14	9.831624	9.8264322			

Table 4.9 – Cas $d_i \approx i + 1$. Comparaison entre ancien et nouveau code. Taille de matrice 16.

Pour faire une déflation de Type II efficace, la différence entre l'ancienne et la nouvelle valeur propre est numériquement égale à 9.53×10^{-7}. Cette valeur est à la limite de la précision flottante entre deux successeurs flottants pour les nombres considérés ici :

$$succ(x) - x = 2^{-23+ufp(x)}$$

où $ufp(x) = \lfloor \log_2(x) \rfloor$ est le bit de poids fort de x (par exemple $ufp(11) = 3$).

- Pour la Table 4.7, d_1 est le troisième successeur de λ_1, $\lvert d_1 - \lambda_1 \rvert = 2^{-23+ufp(\lambda_1)+2}$.
- Pour la Table 4.8, d_4 est le troisième successeur de λ_3, $\lvert d_4 - \lambda_3 \rvert = 2^{-23+ufp(\lambda_3)+2}$.

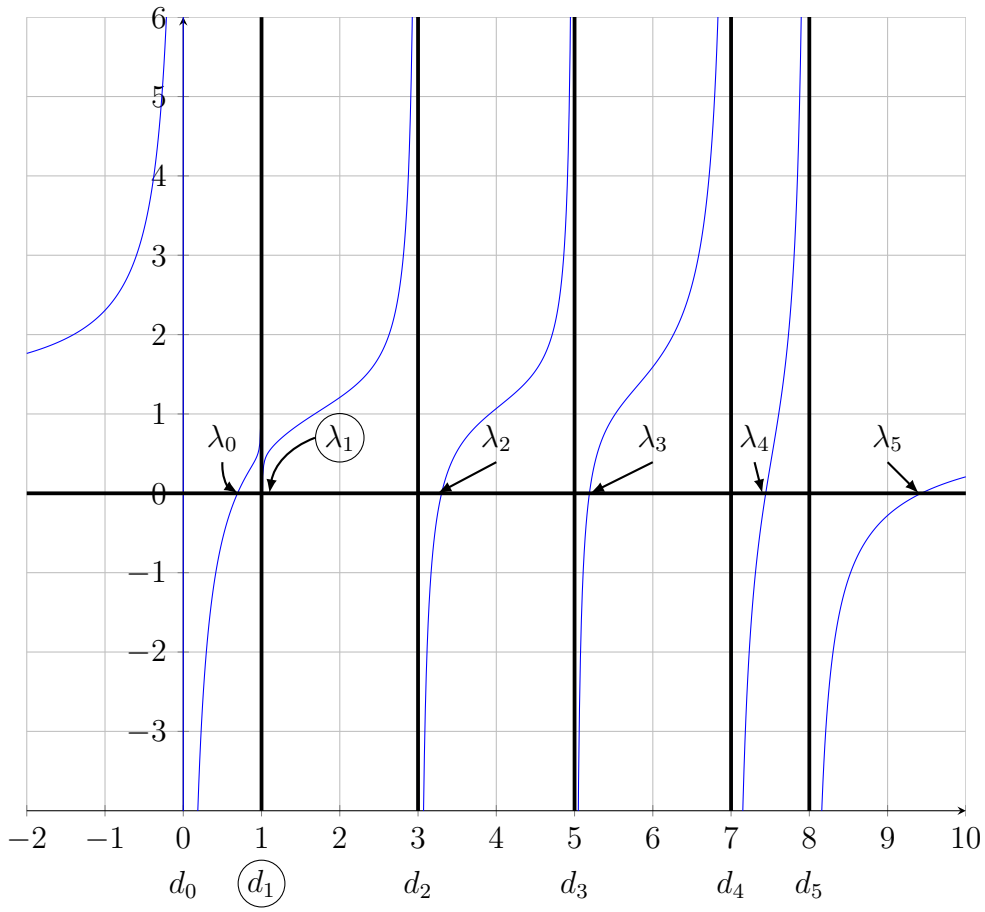


Figure 4.2 – Représentation de la résolution de l'équation séculaire (4.17). Cas où une nouvelle valeur propre est proche d'une ancienne valeur propre ($d_1 \approx \lambda_1$)

- Pour la Table 4.9, λ_{14} est le successeur de d_{13}, $|d_{13} - \lambda_{14}| = 2^{-23+ufp(d_{13})}$.

Il faut donc régler les paramètres C_1, C_2 et C_3 afin que $|d_i - \lambda_j|$ soit proche de l'erreur flottante pour appliquer la déflation de Type II.

On se demande si les mêmes résultats auraient pu être obtenus sans introduire la déflation de Type II. Sur les Tableaux 4.10, 4.12 et 4.11, on a mis l'erreur sur chaque type de matrice pour l'ancien code et pour chaque cas de déflation de Type I, II et III que l'on impose avec un seuil adapté. On a rajouté l'erreur maximale obtenue sur toutes les diagonalisations faites pendant le batch computing quand on imposait une déflation d'un certain type.

Dans le Tableau 4.10, on remarque que $d_1 \approx \lambda_1$. L'ancien code ne fait pas de déflation et donne une erreur raisonnable de $1.48 \cdot 10^{-4}$ et si on impose une déflation de Type III dans le nouveau code, on obtient une erreur du même ordre. Si on impose un Type I, on obtient la plus mauvaise erreur ce qui était prévisible car la distance entre d_0 et d_1 est grande et le seuil choisi donne une erreur maximale très mauvaise (de l'ordre de 10^{-1}). La meilleure erreur est obtenue avec le Type II, que ce soit pour la matrice considérée ou pour l'erreur maximale. On obtient le même genre de résultat dans le cas $d_i \approx \lambda_{i+1}$ (Tableau 4.11).

i	d_i	λ_i	Erreur Ancien Code	Type I	Type II	Type III
0	3.605526	3.5633531				
1	3.610064	3.6100631	$1.48 \cdot 10^{-4}$	$8.2 \cdot 10^{-3}$	$5.4 \cdot 10^{-5}$	$1.46 \cdot 10^{-4}$
2	5.3898721	5.2089825				
Erreur max			$7.13 \cdot 10^{-2}$	$2.4 \cdot 10^{-1}$	$8.3 \cdot 10^{-3}$	$3.6 \cdot 10^{-2}$
Type Déflation pour matrice avec Erreur max			Pas de déflation	Type I	Pas de déflation	Pas de déflation

Table 4.10 – Cas $d_i \approx \lambda_i$. Comparaison entre les types. Taille de matrice 16.

i	d_i	λ_i	Erreur Ancien Code	Type I	Type II	Type III
12	8.9452028	8.7770767				
13	9.8264313	9.7585649	$1.568 \cdot 10^{-3}$	$1.15 \cdot 10^{-2}$	$1.8 \cdot 10^{-5}$	$1.54 \cdot 10^{-3}$
14	9.831624	9.8264322				
Erreur max			$7.13 \cdot 10^{-2}$	$2.4 \cdot 10^{-1}$	$8.2 \cdot 10^{-3}$	$3.6 \cdot 10^{-2}$
Type Déflation pour matrice avec Erreur max			Pas de déflation	Type I	Pas de déflation	Pas de déflation

Table 4.11 – Cas $d_i \approx \lambda_{i+1}$. Comparaison entre les types. Taille de matrice 16.

Dans le cas $d_i \approx \lambda_{i-1}$, on trouve un cas un peu différent où le Type III donne la plus

mauvaise erreur mais la déflation de Type I donne la plus mauvaise erreur maximale. Le meilleur résultat est obtenu avec le Type II.

i	d_i	λ_i	Erreur Ancien Code	Type I	Type II	Type III
2	3.1267593	3.4910524				
3	4.3632765	4.3694711	$3.376 \cdot 10^{-3}$	$6.7 \cdot 10^{-3}$	$4.6 \cdot 10^{-5}$	$1.6 \cdot 10^{-2}$
4	4.369472	4.8753777				
		Erreur max	$7.13 \cdot 10^{-2}$	$2.4 \cdot 10^{-1}$	$8.2 \cdot 10^{-3}$	$3.6 \cdot 10^{-2}$
		Type Déflation pour matrice avec Erreur max	Pas de déflation	Type I	Pas de déflation	Pas de déflation

Table 4.12 – Cas $d_i \approx \lambda_{i-1}$. Comparaison entre les types. Taille de matrice 16.

Dans tous les cas, le nouveau code avec le Type II donne une erreur maximale meilleure qu'avec l'ancien code. Mais surtout, si on utilise un autre Type de déflation I ou III, c'est toujours la déflation de Type II qui donnera la meilleure erreur. L'avantage du Type II par rapport à l'ancien code se voit sur la Figure 4.3. Dans cette figure, on trie les matrices suivant leur erreur dans l'ancien code et on indique l'erreur avec le type II dans le nouveau code. Lorsqu'il y a des cas de déflation de Type II sur le nouveau code, l'erreur s'est améliorée dans 98.95% des cas par rapport à l'ancien code. Les rares cas contraires n'augmente l'erreur que d'un facteur 1.08 au maximum.

En conclusion, pour les petites matrices (de taille inférieure ou égale à 32), on affine les résultats sur les matrices traitées par le Type III ou qui ne subissent pas de déflation. On constate aussi que plus l'erreur est mauvaise sur l'ancien code mieux le nouveau code améliore l'erreur grâce à la déflation de Type II. Le Type II impose donc une erreur plutôt stable.

Explorons le cas des matrices de taille 64. On se retrouve alors avec des déflations multiples comme on peut le voir sur le Tableau 4.13 et cela entraîne des conflits qui sont difficiles à gérer. On illustre le problème avec le tableau 4.13 où l'on met en couleur les cas une ancienne valeur propre est égale à une nouvelle pour une potentielle déflation de Type II. Le vert met en évidence $v_i \approx 0$ pour une déflation de Type III potentielle. Dans cet exemple, il y a potentiellement 5 déflations de Type II et 4 déflations de Type III. En selectionnant manuellement, on trouve une solution convenable qui donne une erreur de $4.7 \cdot 10^{-3}$. Il a fallu faire un choix entre les deux types de déflations, et ce choix est délicat. Les seuils choisis dans un cas pour améliorer l'erreur sur une matrice peuvent être mauvais sur une autre matrice. De plus on remarque qu'il ne faut pas toujours imposer la déflation de Type II pour avoir une erreur correcte.

La déflation de Type I seule est insuffisante car on reste avec une erreur déraisonnable (10.62). L'ajout du Type III est insuffisant (Erreur=6.92). Si on hiérarchise dans l'ordre Type I, Type III, puis Type II, on commence à avoir quelque chose de raisonnable (0.13).

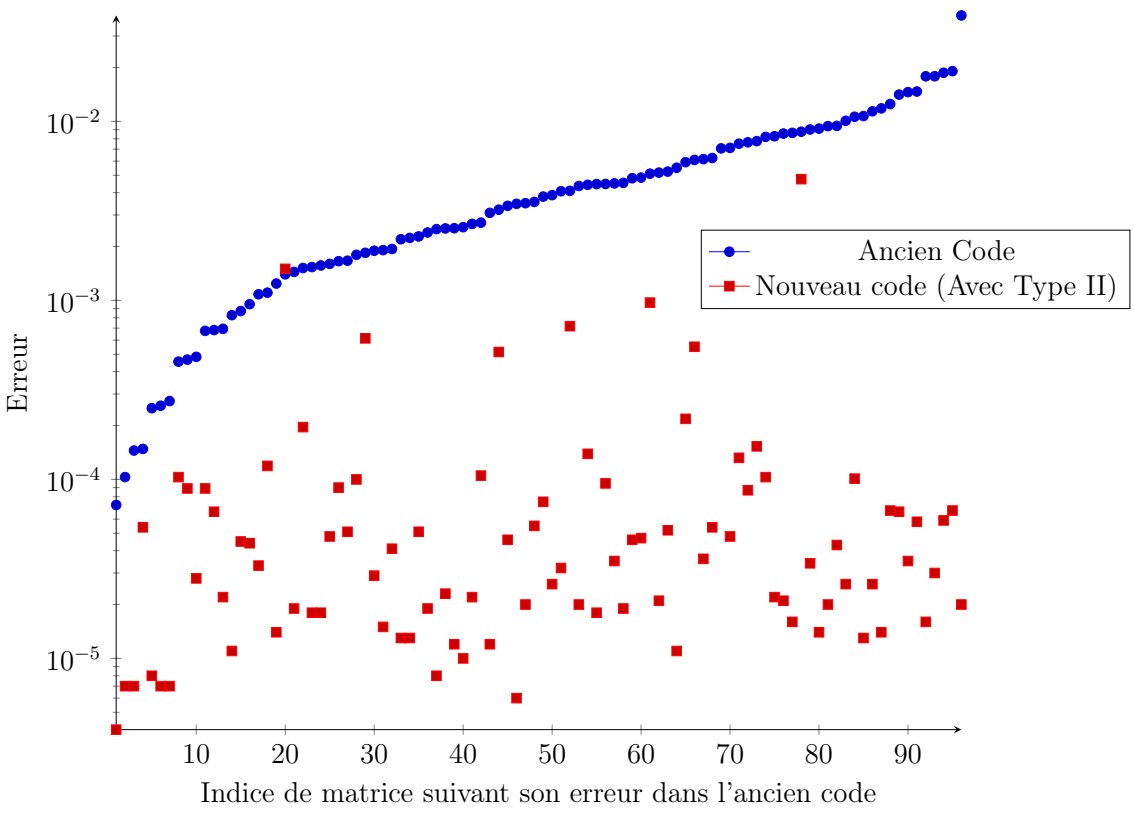


Figure 4.3 – , Comparaison de l'erreur dans le cas d'une déflation de Type II entre ancien et nouveau code. Sur un ensemble de matrices de taille 16 indicées par ordre croissant de l'erreur dans l'ancien code.

C'est seulement avec une observation de la répartition des déflations qu'on obtient une erreur correcte ($4.7 \cdot 10^{-3}$).

i	d_i	λ_i	v_i	$\lvert d_i - \lambda_i \rvert$	$\lvert d_i - \lambda_{i-1} \rvert$	Solution
57	10.7511615753	11.0909166336	$6.78 \cdot 10^{-2}$			Pas de déflation
58	11.0909175873	11.1754264832	$6.32 \cdot 10^{-11}$		$9.53 \cdot 10^{-7}$	Type II
59	11.1754274368	11.2149152756	$9.05 \cdot 10^{-8}$		$9.53 \cdot 10^{-7}$	Pas de déflation
60	11.4380741119	11.4380750656	$3.98 \cdot 10^{-6}$	$9.53 \cdot 10^{-7}$		Pas de déflation
61	11.5277595520	11.5277605057	$9.72 \cdot 10^{-8}$	$9.53 \cdot 10^{-7}$		Type III
62	11.8270521164	11.8270626068	$1.43 \cdot 10^{-3}$			Pas de déflation
63	12.1843700409	12.1843709946	$2.85 \cdot 10^{-9}$	$9.53 \cdot 10^{-7}$		Type III
					Erreur pour cette solution	$4.7 \cdot 10^{-3}$

Erreur sans déflation	10.62
Erreur avec seulement Type I	10.62
Erreur avec seulement Type I et Type III	6.92
Hiérarchie Type I, Type III puis Type II	0.13
Erreur avec Type sélectionné manuellement	$4.7 \cdot 10^{-3}$

Table 4.13 – Taille de matrice 64, illustration de la problématique des déflations. Présentation d'une matrice dont la meilleure solution trouvée donne une erreur $4.7 \cdot 10^{-3}$ mais l'erreur maximale est 8.00

On n'a pas trouvé jusqu'à présent de solution satisfaisante pour automatiser le type de déflation à utiliser. Il y a aussi le problème de la hiérarchisation des types. La solution choisie pour les matrices de taille inférieure à 32 est de hiérarchiser les trois Types dans l'ordre Type I, Type II puis Type III, mais l'exemple du Tableau 4.13 interroge sur ce choix pour les matrices de taille 64 car les cas de Type II ne sont pas prioritaires. On pense que plus la taille augmente, plus on se retrouve avec des déflations de Type III.

4.5.2 Conclusion et perspectives

Dans la perspective d'une amélioration du code de [3], on donne une méthode pour généraliser le Divide & Conquer au cas où des zéros seraient sur la sous-diagonale. On a montré comment choisir la meilleure méthode de multiplication de matrices en batch computing (cf. Section 4.4.1) et on a remplacé le tri des valeurs propres par un tri plus efficace, le tri fusion parallélisé avec le Merge Path (cf. Section 4.4.2). Pour améliorer la précision de calcul, une étude détaillée montre que la déflation sur l'égalité numérique de deux valeurs propres successives n'est pas suffisante pour améliorer la précision de calcul. Il faut aussi comparer les anciennes et les nouvelles valeurs propres lors des étapes du Divide & Conquer. On a, dans ce cas, une véritable amélioration de l'erreur sur les matrices qui ne subissent pas de déflation ou qui subissent une déflation de type III dans l'ancien code. Un autre problème commence à apparaître sur des matrices de taille

supérieure à 32. L'étude sur la robustesse du Divide & Conquer pour la diagonalisation des matrices tridiagonales n'est pas terminée. Le réglage des seuils reste difficile à faire. Des déflations multiples à gérer apparaissent, mais les difficultés majeures sont les conflits entre les différentes déflations qui posent de véritables problèmes algorithmiques.

Annexes

4.A Matrices pour Divide & Conquer : Evaluation de l'erreur

Les tableaux suivants indiquent le pourcentage de matrices qui ont une erreur inférieure à certain seuil.

Erreur	Ancien Code	Nouveau Code
erreur <1.	100.00 %	100.00 %
erreur <0.1	100.00 %	100.00 %
erreur <0.01	99.69 %	100.00%
erreur <0.001	99.98 %	100.00 %
erreur <0.0001	97.70 %	99.77 %

Table 4.A.1 – Pourcentage de matrice de taille 4 qui ont une erreur inférieure à un certain seuil.

Erreur	Ancien Code	Nouveau Code
erreur <1.	100.00 %	100.00 %
erreur <0.1	100.00 %	100.00 %
erreur <0.01	99.69 %	100.00%
erreur <0.001	99.51 %	99.98 %
erreur <0.0001	97.85 %	98.81 %

Table 4.A.2 – Pourcentage de matrice de taille 8 qui ont une erreur inférieure à un certain seuil.

Erreur	Ancien Code	Nouveau Code
erreur <1.	100.00 %	100.00 %
erreur <0.1	100.00 %	100.00 %
erreur <0.01	95.68 %	100.00%
erreur <0.001	71.39 %	97.48 %
erreur <0.0001	48.62 %	76.33 %

Table 4.A.3 – Pourcentage de matrice de taille 16 qui ont une erreur inférieure à un certain seuil.

Erreur	Ancien Code	Nouveau Code
erreur <1.	100.00 %	100.00 %
erreur <0.1	99.98 %	100.00%
erreur <0.01	88.25 %	99.56 %
erreur <0.001	25.48 %	35.67 %
erreur <0.0001	0.68 %	0.74 %

Table 4.A.4 – Pourcentage de matrice de taille 32 qui ont une erreur inférieure à un certain seuil.

4.B Produit de matrice batch computing

```
#include <stdio.h>
#include "timer.h"

__global__ void mulMatrice(float *a,float *b,float *c, int n){
  int nt,i,j;
   int tidx = threadIdx.x%n;
int Qt = (threadIdx.x-tidx)/n;
int gb_index_x = Qt + blockIdx.x*(blockDim.x/n);
float sum;
nt=Qt*3*n*n;
extern __shared__ float sAds[];
       float* A=&sAds[nt];
       float* B=&A[n*n];
       float* C=&B[n*n];
for(i=0;i<n;i++){
  A[i*n+tidx]=a[gb_index_x*n*n+i*n+tidx];

  B[i*n+tidx]=b[gb_index_x*n*n+i*n+tidx];

}

__syncthreads();
//if(gb_index_x==100){printf(" A A[%d]=%f ",tidx,b[gb_index_x*n*n+tidx]);}// b[7*n+ti
// if(gb_index_x==100){printf("A[%d]=%f B[%d]=%f\n",tidx,A[7*n+tidx],tidx,B[tidx]);}

for(i=0;i<n;i++){
  sum=0;
```

```
  for(j=0;j<n;j++){
    sum+=A[i*n+j]*B[j*n+tidx];//ligneColonne
    //sum+=A[j*n+i]*B[j*n+tidx];//ColonneColonne
    //sum+=A[j*n+i]*B[tidx*n+j];//Colonneligne
    //sum+=A[i*n+j]*B[tidx*n+j];//LigneLigne
  }
  C[i*n+tidx]=sum;
}

__syncthreads();
for(i=0;i<n;i++){
  c[gb_index_x*n*n+i*n+tidx]=C[i*n+tidx];
}
}
__global__ void remplissage(float *a,float *b,int n){
  int tidx = threadIdx.x%n,i;
int Qt = (threadIdx.x-tidx)/n;
int gb_index_x = Qt + blockIdx.x*(blockDim.x/n);
for(i=0;i<n;i++){
  a[gb_index_x*n*n+i*n+tidx]=threadIdx.x+blockIdx.x;
  b[gb_index_x*n*n+i*n+tidx]=float(threadIdx.x)-float(blockIdx.x);
}

}
int  main(int argc,const char * argv[]){

  int i, j, k;
unsigned long long total;
// The rank of the matrix
int Dim =atoi(argv[1]);
// The minimum number of threads per block
int minTB=1024/Dim;
while(Dim*Dim*minTB*3*4>49152){
  minTB=minTB-1;
}

// The number of blocks
int NB = 16384;//minTB;
```

```
// The number of matrices to invert
int size = NB*minTB;
// Output test  number
int Num =  6;
  // The matrix
float *AGPU,*BGPU,*CGPU;
float TimerV,sommeTemps=0; // GPU timer instructions
cudaEvent_t start, stop; // GPU timer instructions
cudaEventCreate(&start); // GPU timer instructions
cudaEventCreate(&stop); // GPU timer instructions

  cudaMalloc(&AGPU, size*Dim*Dim*sizeof(float));
cudaMalloc(&BGPU, size*Dim*Dim*sizeof(float));
cudaMalloc(&CGPU, size*Dim*Dim*sizeof(float));

remplissage<<<NB,Dim*minTB>>>(AGPU,BGPU,Dim);
cudaDeviceSynchronize();
printf("\n\n\n");

  cudaEventRecord(start,0);
  mulMatrice<<<NB,minTB*Dim,Dim*Dim*minTB*3*sizeof(float)>>>(AGPU,BGPU,CGPU,Dim);
  cudaEventRecord(stop,0);
  cudaEventSynchronize(stop);
  cudaEventElapsedTime(&TimerV,start, stop);

printf("Execution time of the Mul: %f ms\n", TimerV);

fprintf(stderr,"%d %f\n ",atoi(argv[1]),TimerV);
cudaFree(AGPU);
cudaFree(BGPU);
cudaFree(CGPU);
return 0;
}
```

Chapter 5

Conclusions et perspectives

5.1 Conclusions

Pour plus de clarté, on rappelle que l'on définit :

- Analyse forte, une analyse qui permet de retrouver une clef cible.
- Analyse faible, une analyse où on élimine des clefs candidates.

Le but de cette thèse est essentiellement la compréhension du comportement de l'aléa des distances de Hamming produites par un système cryptographique de type ECC qui utilise une représentation RNS et la méthode des moduli aléatoires. Trouver les outils qui donnent une telle compréhension fut un long parcours. Une première frustration s'est déclarée quand on s'est aperçu que les tests statistiques n'offraient qu'une compréhension partielle d'un tel système. Suite à cela, on a précisé l'objectif de la thèse : la protection contre les attaques par canaux auxiliaires. On montre la résistance du RNS randomisé vis-à-vis des attaques classiques telles que la DPA, CPA, la DPA du second ordre et la MIA. Cependant une compréhension plus complète a été apportée en utilisant l'hypothèse gaussienne sur les vecteurs de Hamming et le MLE. Il faut de l'ordre de $\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$ traces d'observations pour revenir à la clef cible/secrète dans le cadre d'une ECC avec une analyse forte pour une base RNS de n moduli de 32 bits. Il reste à savoir si des moduli plus grands renforceraient l'efficacité, ou des plus petits la diminueraient. Pour l'instant, l'analyse forte de l'hypothèse gaussienne a montré que des moduli de 16 bits ne changeaient pas les résultats. Il faut toujours de l'ordre de $\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$ traces d'observations pour revenir à la clef cible/secrète avec des moduli de 16 bits.

Vu la quantité de calculs, pour obtenir les résultats précédents, on a parallélisé les calculs pour pouvoir obtenir des résultats rapidement en ayant le plus de paramétrage possible pour avoir des statistiques fiables. Un travail conséquent de programmation C/CUDA [80] est à l'origine de l'automatisation de l'analyse. Cette parallélisation a

offert de nombreux calculs exhaustifs utilisés pour calculer les moyennes et les matrices de covariances des vecteurs de Hamming. Ainsi les dictionnaires utilisés pour l'analyse forte et l'analyse faible sont exactes et non approximatifs.

Nombre de critiques nous ont fait remarquer qu'il y a de nombreuses situations où on n'a très peu de traces d'observation dans un cryptosystéme asymétrique, voire une seule. Au vu des résultats obtenus avec l'analyse MLE et l'hypothèse gaussienne, on a exploré cette hypothèse en partant de l'hypothèse la plus générale puis on a augmenté les contraintes pour savoir si l'information d'une seule trace suffirait pour revenir à la clef cible. En posant $H_K^i = (H_{0,K}, ..., H_{i,K})^T$ le vecteur de Hamming pour une clef K, on a investigué :

- L'hypothèse forte : le vecteur concaténé des vecteurs de Hamming $\left({H_{K_1}^i}^T | {H_{K_2}^i}^T | \dots | {H_{K_n}^i}^T\right)^T$ est gaussien.
- L'hypothèse faible : le vecteur H_K^i est gaussien.
- L'hypothèse faible conditionnée : Le vecteur H_K^i conditionné par une condition A est gaussien.
- L'hypothèse faible conditionnée avec une analyse faible : le vecteur H_K^i conditionné par une condition A est gaussien et on ne cherche plus à remonter à la clef cible mais seulement à éliminer les clefs les moins favorables.

Cela étant dit, le RNS randomisé ressort comme très résistant face à toutes ces manières de l'attaquer. Tout au moins pour un ensemble de plus de 7 moduli de 32 bits, faire ressortir de l'information d'une trace d'observation reste difficile avec les méthodes qu'on a développées. Il y aussi la possibilité de chercher un meilleur conditionnement pour pouvoir faire une analyse faible efficace qui remonte à minima un ensemble restreint de clefs favorables voire jusqu'à la clef cible complète.

L'hypothèse gaussienne conditionnée a généré des blocages. On a eu besoin de trouver une résolution de système suffisamment stable pour compléter cette étude. On s'est donc penché sur l'amélioration du code Cuppen Divide & Conquer [3] de diagonalisation en batch computing pour des petites matrices. Au-delà de la généralisation et de l'optimisation de ce code avec un produit de matrices efficace et l'utilisation du Merge Path, on améliore la précision. Cette amélioration est basée sur une étude fine de la déflation en simple précision. Une déflation apparait quand deux valeurs numériques sont très proches au niveau de la précision flottante. Dans notre étude on introduit un cas qui traite d'une déflation qui n'a pas été prévue par les théories développées sur le Divide & Conquer utilisé dans le cadre numérique. Ce nouveau type de déflation, et la réorganisation algorithmique que cela implique, apporte une vraie amélioration de la précision. Sur l'ensemble de matrices étudiées, cela divise l'erreur maximale d'un facteur 10. Et

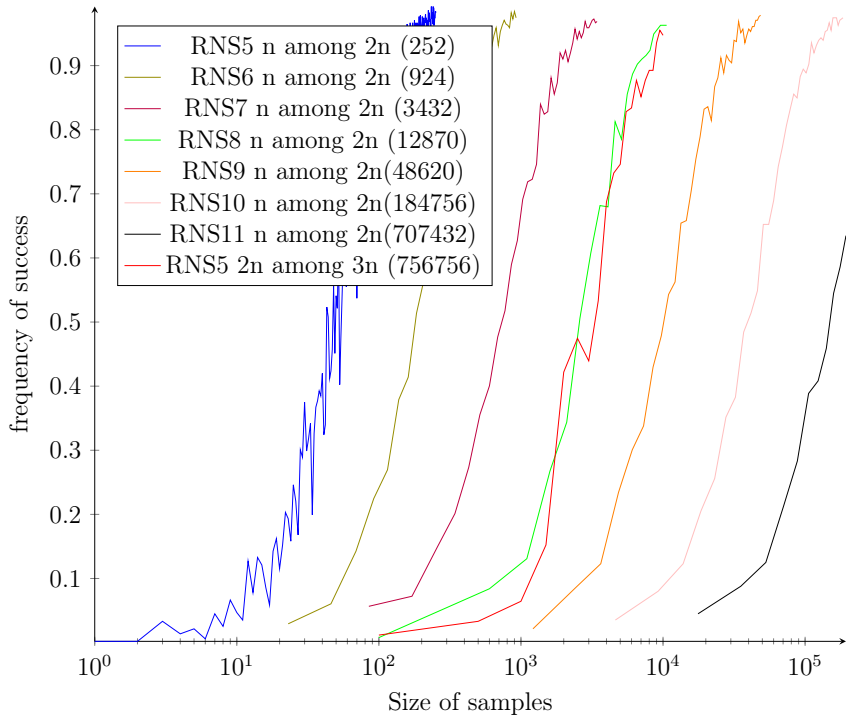


Figure 5.1 – Frequency of succes with strong analysis and MLE for drawing n among 2n for different size of moduli and RNS5 with 2n among 3n drawing. Number of moduli configurations in parenthesis.

cela diminue aussi l'erreur d'un facteur 1000 sur certains cas où l'erreur est particulièrement importante. Cette optimisation n'a pas été finalisée pour des matrices supérieures à 32. Cependant ce code est utilisable pour affiner la recherche d'un conditionnement qui approcherait le secret dans le cas de l'hypothèse gaussienne conditionnée avec l'analyse faible puisqu'on se limite à des tailles de matrices inférieures à 16.

5.2 Perspectives

Lors de mon séjour de recherche en Australie au laboratoire d'informatique de l'université de Wollongong, Thomas Plantard se demandait si le facteur de Montgomery impliqué dans la multiplication modulaire n'avait pas une influence notable sur la qualité de l'aléa. Les résultats obtenus n'ont pas été encore suffisamment exploités pour donner une conclusion pertinente. Suite à cette question, on a essayé un autre type de tirage que n parmi $2n$ comme par exemple un tirage $2n$ parmi $3n$. La complexité d'un tirage du type $2n$ parmi $3n$, plus sophistiqué, est négligeable par rapport au calcul ECC. On a un début de réponse avec la Figure 5.1. On peut voir sur la figure 5.1 qu'un tirage de $2n$ parmi $3n$ diminue le pourcentage de succès du MLE mais on remarque aussi qu'un système avec 11 moduli (tirage n parmi $2n$) possède une meilleure résistance à l'analyse forte alors que le nombre

de combinaisons est inférieur au cas 5 moduli avec un tirage $2n$ parmi $3n$. Le tirage $2n$ parmi $3n$ pour RNS5 est équivalent au tirage n parmi $2n$ pour RNS8 du point de vue de l'analyse forte. Il serait donc plus intéressant de prendre plus de moduli plutôt que d'augmenter la randomisation si on veut une protection équivalente. Si la complexité du calcul limite le nombre de moduli, un type de tirage convenable offrirait une alternative pour obtenir la sécurité voulue.

La méthode de randomisation étudiée dans cette thèse pourrait être aussi comparée, surtout du point de vue de la complexité de calcul et de l'occupation de mémoire, à celle de [43] qui randomise à chaque étape du calcul dans le MPL. Au vu de l'importance des calculs nécessaires pour faire leur randomisation, les auteurs de [43] se limitent à un RNS4 avec des moduli de grande taille (50 bits) alors qu'on s'est limité à 32 bits dans notre étude pour avoir le maximum de facilité d'implémentation. Ils tabulent tous les calculs liés à cette randomisation pour limiter la complexité en temps de calcul mais augmentent l'occupation de mémoire. L'avantage de la randomisation étudiée dans cette thèse repose sur sa souplesse d'utilisation comparée à [43], surtout sur le nombre de moduli. Notre implémentation est plus intuitive car il n'y a pas de changement de base en cours de calcul et il est très facile de changer les moduli ou d'augmenter leur nombre, mais offre-t-elle une protection aussi efficace que [43] ?

Dans cette thèse, on utilise des moduli de 32 bits les plus grands possibles. On a peu parlé de la variabilité du type de base utilisable car rien de concluant n'est vraiment ressorti de ce coté-là quand une tentative d'analyse sur 5 moduli a été faite. Dans le Tableau 5.1, on montre une analyse forte avec le MLE sur différents types de moduli. A toute première vue, le type de moduli n'a pas d'influence sur la randomisation des Hammings dans le cas de l'hypothèse gaussienne. Cependant cela demanderait un approfondissement en faisant plus de simulation. Et pour aller au bout, il faudrait faire l'analyse forte et faible sur chaque type de moduli.

Type de moduli :

- non creux=autant de 1 que de 0.
- creux=beaucoup de 1.

Type de choix :

- $2^{32} - \epsilon$= beaucoup de 1 en poids fort.
- aléa=Choix aléatoire des $2 \times 5 = 10$ moduli participant à la randomisation du RNS5.

Type moduli	Taille	Type de choix	Attaque réussie	9 et 10 bits trouvés
Non creux	≤ 32	aléa	62.89%	77.53%
creux	=32	$2^{32} - \epsilon$	62.30%	74.60%
non creux	=32	$2^{32} - \epsilon$	59.57%	73.82%
quelconque	=27	aléa	58.98%	72.85%
non creux	=32	aléa	52.73%	68.75%
quelconque	≤ 32	aléa	62.50 %	75.78%
quelconque	$= 32$	aléa	54.29%	69.53%

Table 5.1 – MLE et analyse forte sur différents types de moduli. ECC 112, RNS5, 100 traces d'observations.

Pour compléter cette étude, il serait intéressant de confirmer si la méthode des moduli aléatoires est résistante à une attaque utilisant des réseaux de neurones comme celle présentée par Yaro et Benger dans [96]. En général, on utilise les réseaux de neurones pour résoudre des problèmes de non-linéarité. Et d'ailleurs on remarque qu'il y a moins de linéarité sur les queues de distributions des Hammings. Cependant on a montré que l'hypothèse gaussienne conditionnée avec l'analyse faible offre les meilleurs résultats quand on se place dans les queues de distributions.

On a maintenant l'outil de résolution de système pour des petites matrices mal conditionnées. Désormais on sait que l'analyse faible avec une hypothèse gaussienne conditionnée donne de l'information même si on ne possède que peu de traces d'observation. Ces deux outils constituent les composantes pour chercher un conditionnement plus sophistiqué qui approcherait le secret de plus en plus efficacement avec une seule trace d'observation. Serait ce une direction pour faire apparaître une vulnérabilité de la méthode des moduli aléatoires ?

Appendix A

Tests du NIST sur des distances de Hamming lors d'une ECC

A.1 100000 calculs

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is KawamuraBinaryECCRNS5_S112_n100000_powdb7c2abf62e35e7628dfac6561c5
100000 calculations on [N]G where N is the order of G thus 112 Hamming Distance
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 22  18  13   8  10   5   3   8   8   5  0.000145     94/100  *  Frequency
  5  12   3   5   8  12  14  12  19  10  0.011791    100/100     BlockFrequency
 19  12  15   8  13   5  11   5   6   6  0.014550     93/100  *  CumulativeSums
 20  19  11  10   5   6   6   9   7   7  0.002203     94/100  *  CumulativeSums
 43  21   9   6   5   5   2   2   4   3  0.000000 *   87/100  *  Runs
 53  20  10   6   6   1   3   0   1   0  0.000000 *   87/100  *  LongestRun
 10   8   9   6  14   9  10  11  12  11  0.883171    100/100     Rank
 29   7  14  12  12   5   8   4   6   3  0.000000 *   93/100  *  FFT
 95   4   1   0   0   0   0   0   0   0  0.000000 *   29/100  *  NonOverlappingTemplate
 28  13  10  14   5   9   6  12   3   0  0.000000 *   94/100  *  NonOverlappingTemplate
 23  15  12   4   7   5  14   6   9   5  0.000157     98/100     NonOverlappingTemplate
 10  10  14  12  11   8   8   9  13   5  0.699313    100/100     NonOverlappingTemplate
 59  10   8   8   7   4   1   1   1   1  0.000000 *   72/100  *  NonOverlappingTemplate
 11  11  12   9   8  13   6  12   9   9  0.897763     99/100     NonOverlappingTemplate
 65  12   9   6   2   1   1   1   1   2  0.000000 *   67/100  *  NonOverlappingTemplate
```

46	17	6	5	4	5	1	8	4	4	0.000000	*	90/100	*	NonOverlappingTemplate
96	1	1	1	0	1	0	0	0	0	0.000000	*	27/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	*	NonOverlappingTemplate
21	16	8	12	14	7	6	4	5	7	0.001112		97/100		NonOverlappingTemplate
4	6	9	9	6	11	7	15	13	20	0.010988		100/100		NonOverlappingTemplate
15	14	8	8	14	9	9	12	5	6	0.262249		99/100		NonOverlappingTemplate
20	11	9	16	7	13	8	5	5	6	0.007160		96/100		NonOverlappingTemplate
18	18	17	13	5	7	3	4	7	8	0.000216		96/100		NonOverlappingTemplate
13	10	7	10	15	7	15	9	8	6	0.366918		99/100		NonOverlappingTemplate
83	3	3	3	3	0	4	1	0	0	0.000000	*	44/100	*	NonOverlappingTemplate
23	15	8	6	7	10	13	4	6	8	0.000700		95/100	*	NonOverlappingTemplate
22	20	11	6	9	9	5	5	7	6	0.000097	*	99/100		NonOverlappingTemplate
5	6	7	3	11	12	19	10	13	14	0.012650		99/100		NonOverlappingTemplate
18	16	16	8	11	3	10	7	7	4	0.003712		96/100		NonOverlappingTemplate
21	19	8	9	14	2	6	9	9	3	0.000051	*	97/100		NonOverlappingTemplate
15	6	17	16	5	5	3	11	14	8	0.003447		97/100		NonOverlappingTemplate
32	17	13	7	9	5	8	3	2	4	0.000000	*	90/100	*	NonOverlappingTemplate
33	14	16	8	6	7	5	5	5	1	0.000000	*	95/100	*	NonOverlappingTemplate
23	14	7	5	11	11	6	9	7	7	0.002374		96/100		NonOverlappingTemplate
10	10	14	9	11	7	6	11	9	13	0.798139		99/100		NonOverlappingTemplate
84	9	2	3	0	1	0	1	0	0	0.000000	*	46/100	*	NonOverlappingTemplate
23	20	10	9	8	10	3	7	9	1	0.000004	*	94/100	*	NonOverlappingTemplate
72	12	7	1	3	1	0	1	3	0	0.000000	*	68/100	*	NonOverlappingTemplate
50	17	4	5	5	5	3	4	4	3	0.000000	*	80/100	*	NonOverlappingTemplate
17	20	17	13	11	4	7	7	2	2	0.000012	*	97/100		NonOverlappingTemplate
14	13	7	6	7	8	13	6	16	10	0.191687		98/100		NonOverlappingTemplate
43	19	11	9	3	5	0	6	1	3	0.000000	*	93/100	*	NonOverlappingTemplate
10	9	17	13	12	13	7	8	7	4	0.162606		97/100		NonOverlappingTemplate
77	14	4	3	2	0	0	0	0	0	0.000000	*	56/100	*	NonOverlappingTemplate
2	8	8	6	14	16	9	12	12	13	0.071177		100/100		NonOverlappingTemplate
6	7	19	14	6	8	14	6	8	12	0.032923		98/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	4/100	*	NonOverlappingTemplate
56	14	5	10	5	3	2	2	3	0	0.000000	*	86/100	*	NonOverlappingTemplate
6	9	10	2	10	9	14	12	14	14	0.145326		99/100		NonOverlappingTemplate
94	5	0	1	0	0	0	0	0	0	0.000000	*	23/100	*	NonOverlappingTemplate
13	14	7	4	11	9	10	10	13	9	0.514124		99/100		NonOverlappingTemplate
23	17	8	14	11	8	5	6	3	5	0.000043	*	97/100		NonOverlappingTemplate
21	12	12	14	9	7	10	5	7	3	0.004629		97/100		NonOverlappingTemplate

95	4	0	1	0	0	0	0	0	0	0.000000	*	15/100	*	NonOverlappingTemplate
30	9	7	9	7	7	8	7	8	8	0.000001	*	94/100	*	NonOverlappingTemplate
13	10	10	5	10	15	7	10	15	5	0.224821		100/100		NonOverlappingTemplate
47	15	9	9	8	3	4	2	2	1	0.000000	*	88/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	2/100	*	NonOverlappingTemplate
20	10	9	8	14	5	10	12	8	4	0.025193		96/100		NonOverlappingTemplate
16	17	16	12	8	10	6	5	5	5	0.008879		99/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
15	13	12	12	6	7	9	11	7	8	0.514124		98/100		NonOverlappingTemplate
97	2	1	0	0	0	0	0	0	0	0.000000	*	22/100	*	NonOverlappingTemplate
92	5	2	0	0	0	1	0	0	0	0.000000	*	25/100	*	NonOverlappingTemplate
51	14	12	7	3	5	5	1	2	0	0.000000	*	84/100	*	NonOverlappingTemplate
70	13	6	3	4	1	0	1	1	1	0.000000	*	68/100	*	NonOverlappingTemplate
72	19	2	2	2	0	1	1	1	0	0.000000	*	47/100	*	NonOverlappingTemplate
98	1	0	1	0	0	0	0	0	0	0.000000	*	24/100	*	NonOverlappingTemplate
19	10	5	8	17	9	11	8	10	3	0.010988		99/100		NonOverlappingTemplate
16	16	7	14	14	8	6	8	5	6	0.037566		96/100		NonOverlappingTemplate
12	6	17	8	10	12	14	8	7	6	0.202268		99/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	3/100	*	NonOverlappingTemplate
66	12	13	2	4	2	0	1	0	0	0.000000	*	70/100	*	NonOverlappingTemplate
27	13	15	13	8	8	4	4	3	5	0.000000	*	98/100		NonOverlappingTemplate
11	5	8	11	10	11	10	13	11	10	0.897763		97/100		NonOverlappingTemplate
15	17	10	8	10	14	3	8	12	3	0.017912		98/100		NonOverlappingTemplate
7	11	13	11	12	10	12	9	9	6	0.867692		100/100		NonOverlappingTemplate
99	0	0	1	0	0	0	0	0	0	0.000000	*	5/100	*	NonOverlappingTemplate
17	13	8	10	11	5	11	10	8	7	0.334538		99/100		NonOverlappingTemplate
16	10	9	8	12	9	8	11	8	9	0.779188		95/100	*	NonOverlappingTemplate
47	13	9	8	5	3	7	4	3	1	0.000000	*	92/100	*	NonOverlappingTemplate
16	11	11	8	15	11	8	8	6	6	0.289667		97/100		NonOverlappingTemplate
95	4	1	0	0	0	0	0	0	0	0.000000	*	29/100	*	NonOverlappingTemplate
59	11	4	6	5	8	3	1	1	2	0.000000	*	73/100	*	NonOverlappingTemplate
15	11	12	14	11	12	9	5	6	5	0.224821		98/100		NonOverlappingTemplate
67	16	8	3	0	2	1	3	0	0	0.000000	*	75/100	*	NonOverlappingTemplate
50	21	11	4	7	3	2	1	1	0	0.000000	*	77/100	*	NonOverlappingTemplate
19	15	11	14	10	8	13	6	3	1	0.000883		99/100		NonOverlappingTemplate
33	15	10	6	13	9	5	5	4	0	0.000000	*	94/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	*	NonOverlappingTemplate
67	14	7	6	2	3	0	0	0	1	0.000000	*	69/100	*	NonOverlappingTemplate

17	11	12	7	9	9	9	7	11	8	0.534146		97/100		NonOverlappingTemplate
96	3	1	0	0	0	0	0	0	0	0.000000	*	24/100	*	NonOverlappingTemplate
19	17	11	9	16	4	5	9	7	3	0.000700		95/100	*	NonOverlappingTemplate
9	13	8	12	9	12	11	6	9	11	0.897763		100/100		NonOverlappingTemplate
13	14	14	5	11	8	3	12	7	13	0.115387		100/100		NonOverlappingTemplate
4	8	10	10	8	5	6	12	21	16	0.003447		100/100		NonOverlappingTemplate
29	13	16	9	9	9	4	4	3	4	0.000000	*	92/100	*	NonOverlappingTemplate
74	11	5	4	2	0	2	0	2	0	0.000000	*	59/100	*	NonOverlappingTemplate
18	16	4	9	3	14	13	4	9	10	0.003201		99/100		NonOverlappingTemplate
33	14	14	9	7	8	4	5	3	3	0.000000	*	97/100		NonOverlappingTemplate
98	1	1	0	0	0	0	0	0	0	0.000000	*	10/100	*	NonOverlappingTemplate
19	12	7	12	5	8	13	6	10	8	0.075719		94/100	*	NonOverlappingTemplate
12	5	14	9	9	9	5	14	9	14	0.304126		97/100		NonOverlappingTemplate
32	16	12	6	8	8	6	6	5	1	0.000000	*	95/100	*	NonOverlappingTemplate
47	14	9	8	7	5	4	2	2	2	0.000000	*	89/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
19	10	14	14	11	11	5	8	5	3	0.009535		97/100		NonOverlappingTemplate
35	16	9	8	4	7	5	9	4	3	0.000000	*	94/100	*	NonOverlappingTemplate
29	19	13	8	13	6	2	2	5	3	0.000000	*	96/100		NonOverlappingTemplate
4	8	8	8	7	7	14	14	15	15	0.096578		99/100		NonOverlappingTemplate
68	13	8	4	3	2	2	0	0	0	0.000000	*	66/100	*	NonOverlappingTemplate
28	18	15	6	6	5	8	8	4	2	0.000000	*	93/100	*	NonOverlappingTemplate
67	12	6	3	5	3	1	1	1	1	0.000000	*	64/100	*	NonOverlappingTemplate
23	12	17	14	6	7	5	7	4	5	0.000043	*	94/100	*	NonOverlappingTemplate
84	8	2	3	0	1	0	1	1	0	0.000000	*	37/100	*	NonOverlappingTemplate
85	5	4	5	0	0	0	0	1	0	0.000000	*	49/100	*	NonOverlappingTemplate
61	16	7	9	4	2	0	1	0	0	0.000000	*	75/100	*	NonOverlappingTemplate
47	19	8	9	4	2	0	9	1	1	0.000000	*	79/100	*	NonOverlappingTemplate
7	13	8	10	20	8	9	8	10	7	0.122325		99/100		NonOverlappingTemplate
82	7	5	1	1	1	1	0	1	1	0.000000	*	45/100	*	NonOverlappingTemplate
24	17	11	16	10	7	5	2	3	5	0.000001	*	92/100	*	NonOverlappingTemplate
13	16	14	5	15	7	11	6	5	8	0.055361		98/100		NonOverlappingTemplate
21	7	12	15	5	8	12	10	3	7	0.002971		97/100		NonOverlappingTemplate
47	16	14	3	7	3	4	2	1	3	0.000000	*	76/100	*	NonOverlappingTemplate
34	19	11	6	5	9	3	5	3	5	0.000000	*	90/100	*	NonOverlappingTemplate
12	12	11	14	7	10	10	7	10	7	0.816537		98/100		NonOverlappingTemplate
98	1	1	0	0	0	0	0	0	0	0.000000	*	21/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	4/100	*	NonOverlappingTemplate

98	0	1	0	1	0	0	0	0	0	0.000000	*	14/100	* NonOverlappingTemplate
98	1	1	0	0	0	0	0	0	0	0.000000	*	15/100	* NonOverlappingTemplate
19	10	13	9	13	5	12	4	8	7	0.037566		97/100	NonOverlappingTemplate
86	6	4	2	0	1	0	1	0	0	0.000000	*	51/100	* NonOverlappingTemplate
93	3	3	0	0	1	0	0	0	0	0.000000	*	24/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
12	11	12	11	7	10	8	10	8	11	0.971699		98/100	NonOverlappingTemplate
7	7	6	11	9	16	14	16	3	11	0.042808		100/100	NonOverlappingTemplate
44	12	14	5	10	2	5	3	1	4	0.000000	*	84/100	* NonOverlappingTemplate
7	5	14	10	8	12	9	11	11	13	0.637119		100/100	NonOverlappingTemplate
8	8	18	10	9	10	8	8	9	12	0.474986		100/100	NonOverlappingTemplate
93	5	0	1	0	0	0	0	0	1	0.000000	*	42/100	* NonOverlappingTemplate
19	13	16	7	9	8	7	10	5	6	0.025193		96/100	NonOverlappingTemplate
6	7	11	9	4	14	13	8	14	14	0.191687		99/100	NonOverlappingTemplate
75	13	6	0	2	0	2	1	0	1	0.000000	*	58/100	* NonOverlappingTemplate
34	10	12	3	15	9	3	6	4	4	0.000000	*	92/100	* NonOverlappingTemplate
71	13	8	3	1	1	0	2	1	0	0.000000	*	58/100	* NonOverlappingTemplate
6	11	6	12	12	10	8	9	17	9	0.383827		100/100	NonOverlappingTemplate
7	10	5	14	14	13	7	12	10	8	0.419021		99/100	NonOverlappingTemplate
68	10	9	4	3	1	1	2	1	1	0.000000	*	63/100	* NonOverlappingTemplate
50	13	10	7	9	5	4	2	0	0	0.000000	*	86/100	* NonOverlappingTemplate
28	10	12	8	11	10	6	2	6	7	0.000002	*	94/100	* NonOverlappingTemplate
18	9	8	7	11	11	6	7	14	9	0.202268		98/100	NonOverlappingTemplate
30	17	14	12	9	4	5	2	2	5	0.000000	*	97/100	NonOverlappingTemplate
27	19	12	10	7	7	9	3	5	1	0.000000	*	93/100	* NonOverlappingTemplate
7	16	9	8	8	8	12	8	11	13	0.574903		99/100	NonOverlappingTemplate
16	11	11	8	15	11	8	8	6	6	0.289667		97/100	NonOverlappingTemplate
59	9	9	9	3	6	1	2	1	1	0.000000	*	86/100	* OverlappingTemplate
11	7	12	14	10	11	10	5	11	9	0.759756		99/100	Universal
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* ApproximateEntropy
3	3	8	6	5	3	8	6	3	5	0.616305		49/50	RandomExcursions
2	7	7	5	3	4	11	4	2	5	0.137282		49/50	RandomExcursions
5	5	6	4	8	3	5	6	2	6	0.816537		48/50	RandomExcursions
4	5	3	9	5	5	3	6	5	5	0.816537		50/50	RandomExcursions
7	4	3	6	8	2	4	7	5	4	0.657933		49/50	RandomExcursions
4	4	4	6	3	7	7	4	7	4	0.883171		49/50	RandomExcursions
5	7	3	5	7	5	6	4	4	4	0.955835		50/50	RandomExcursions
5	4	4	6	7	6	3	6	4	5	0.971699		49/50	RandomExcursions

```
  8   8   5   8   4   4   6   1   2   4  0.262249     49/50      RandomExcursionsVariant
  9   7   6   5   3   5   3   7   1   4  0.350485     50/50      RandomExcursionsVariant
 11   7   3   5   3   4   4   3   3   7  0.191687     50/50      RandomExcursionsVariant
  9  11   3   4   2   4   4   4   4   5  0.122325     49/50      RandomExcursionsVariant
 10   8   9   2   3   4   2   2   5   5  0.058984     49/50      RandomExcursionsVariant
  9  11   6   2   4   7   2   3   1   5  0.023545     49/50      RandomExcursionsVariant
  8  13   2   2   7   4   2   4   4   4  0.010237     48/50      RandomExcursionsVariant
  7   8   8   2   1   4   3   4   5   8  0.191687     49/50      RandomExcursionsVariant
  6   6   3   5   9   1   5   3   6   6  0.455937     49/50      RandomExcursionsVariant
  4   4   8   4   3   6   5   8   6   2  0.616305     49/50      RandomExcursionsVariant
  2   3   8   3   9   4   2   7   7   5  0.213309     50/50      RandomExcursionsVariant
  1   5   7   4   2   7  10   4   3   7  0.137282     50/50      RandomExcursionsVariant
  1   7   2   4   8   7   5   4   7   5  0.383827     50/50      RandomExcursionsVariant
  2   4   3   7   6   6   5   5   4   8  0.739918     50/50      RandomExcursionsVariant
  2   1   7   6   5   5   9   4   7   4  0.319084     50/50      RandomExcursionsVariant
  4   3   5   5   7   4   2   4   6  10  0.419021     50/50      RandomExcursionsVariant
  3   3   7   5   6   1   5   4   9   7  0.350485     50/50      RandomExcursionsVariant
  4   3   3   9   3   5   3   7   7   6  0.494392     50/50      RandomExcursionsVariant
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
  4  14  10  10  13   8  10  12  11   8  0.595549     99/100     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 47 for a sample size = 50 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

A.2 1000000 calculs

```
--------------------------------------------------------------------------------
```

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

---

generator is KawamuraBinaryECCRNS5_S112_n1000000_powdb7c2abf62e35e7628dfac6561c5

1000000 calculations on [N]G where N is the order of G thus 112 Hamming Distance

---

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE		PROPORTION		STATISTICAL TEST
83	5	6	0	3	1	1	0	1	0	0.000000	*	39/100	*	Frequency
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	BlockFrequency
81	9	3	2	1	2	0	2	0	0	0.000000	*	44/100	*	CumulativeSums
83	7	4	2	1	2	0	0	1	0	0.000000	*	43/100	*	CumulativeSums
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	Runs
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	LongestRun
8	10	11	15	9	6	7	10	14	10	0.616305		100/100		Rank
98	0	0	0	0	2	0	0	0	0	0.000000	*	11/100	*	FFT
18	20	13	11	9	13	9	3	1	3	0.000034	*	99/100		NonOverlappingTemplate
8	15	13	5	7	11	9	16	8	8	0.224821		99/100		NonOverlappingTemplate
99	1	0	0	0	0	0	0	0	0	0.000000	*	12/100	*	NonOverlappingTemplate
97	2	0	1	0	0	0	0	0	0	0.000000	*	8/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
99	1	0	0	0	0	0	0	0	0	0.000000	*	12/100	*	NonOverlappingTemplate
18	16	14	13	8	7	5	6	9	4	0.010237		98/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	3/100	*	NonOverlappingTemplate
69	20	4	3	1	2	1	0	0	0	0.000000	*	62/100	*	NonOverlappingTemplate
12	9	18	9	9	10	11	10	9	3	0.202268		97/100		NonOverlappingTemplate
32	12	9	7	12	12	5	5	4	2	0.000000	*	93/100	*	NonOverlappingTemplate
77	10	3	2	2	3	1	0	0	2	0.000000	*	53/100	*	NonOverlappingTemplate
18	10	13	13	3	8	7	13	8	7	0.055361		96/100		NonOverlappingTemplate
95	2	3	0	0	0	0	0	0	0	0.000000	*	25/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
30	19	11	9	9	3	6	3	5	5	0.000000	*	92/100	*	NonOverlappingTemplate
98	1	0	0	0	1	0	0	0	0	0.000000	*	17/100	*	NonOverlappingTemplate

15	16	17	7	6	7	13	7	5	7	0.020548	97/100	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
96	2	1	1	0	0	0	0	0	0	0.000000 *	17/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
26	15	9	9	6	10	9	4	9	3	0.000014 *	95/100 *	NonOverlappingTemplate
33	18	18	7	6	6	4	2	3	3	0.000000 *	95/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
99	1	0	0	0	0	0	0	0	0	0.000000 *	1/100 *	NonOverlappingTemplate
38	12	15	5	11	5	2	3	3	6	0.000000 *	90/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
35	13	17	9	5	6	4	3	2	6	0.000000 *	91/100 *	NonOverlappingTemplate
12	10	11	9	8	7	13	8	14	8	0.816537	100/100	NonOverlappingTemplate
11	20	18	6	7	6	9	8	10	5	0.004981	99/100	NonOverlappingTemplate
77	12	5	3	0	2	0	0	1	0	0.000000 *	60/100 *	NonOverlappingTemplate
21	18	9	11	7	8	11	6	6	3	0.000883	96/100	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
46	13	10	5	3	9	6	6	1	1	0.000000 *	88/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	2/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
31	13	14	12	6	5	5	7	5	2	0.000000 *	91/100 *	NonOverlappingTemplate
21	5	15	14	6	11	7	9	7	5	0.003201	97/100	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
85	3	3	2	3	2	1	0	1	0	0.000000 *	46/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	1/100 *	NonOverlappingTemplate
27	12	9	7	6	10	7	8	3	11	0.000016 *	98/100	NonOverlappingTemplate
89	6	1	3	0	1	0	0	0	0	0.000000 *	33/100 *	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
47	12	12	9	10	5	1	2	2	0	0.000000 *	83/100 *	NonOverlappingTemplate
21	11	13	10	11	9	6	6	8	5	0.021999	97/100	NonOverlappingTemplate

100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
83	5	3	5	1	0	1	1	1	0	0.000000	*	43/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
91	5	1	0	1	1	0	0	1	0	0.000000	*	40/100	*	NonOverlappingTemplate
71	12	3	6	6	1	1	0	0	0	0.000000	*	66/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
12	9	14	15	13	9	6	15	4	3	0.032923		99/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
17	12	10	9	15	9	4	12	5	7	0.080519		99/100		NonOverlappingTemplate
89	7	1	1	0	2	0	0	0	0	0.000000	*	43/100	*	NonOverlappingTemplate
37	18	14	4	7	5	3	4	4	4	0.000000	*	92/100	*	NonOverlappingTemplate
18	20	13	11	9	13	9	3	1	3	0.000034	*	99/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
57	17	6	7	4	2	5	1	1	0	0.000000	*	75/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
54	18	9	3	5	4	3	2	1	1	0.000000	*	82/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	*	NonOverlappingTemplate
54	11	13	5	4	3	2	4	2	2	0.000000	*	82/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
20	11	10	10	13	8	10	8	6	4	0.048716		98/100		NonOverlappingTemplate
12	11	13	13	12	5	7	9	10	8	0.678686		97/100		NonOverlappingTemplate
95	3	1	0	1	0	0	0	0	0	0.000000	*	29/100	*	NonOverlappingTemplate
16	17	14	9	10	8	7	6	6	7	0.075719		100/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
90	6	1	2	1	0	0	0	0	0	0.000000	*	32/100	*	NonOverlappingTemplate
57	19	12	3	3	1	1	2	2	0	0.000000	*	72/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
54	19	10	4	4	1	4	0	4	0	0.000000	*	82/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	6/100	*	NonOverlappingTemplate
13	14	14	10	15	5	8	6	11	4	0.096578		97/100		NonOverlappingTemplate

98	1	0	1	0	0	0	0	0	0	0.000000	*	18/100	*	NonOverlappingTemplate
45	15	16	5	9	3	2	2	2	1	0.000000	*	86/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	*	NonOverlappingTemplate
41	15	9	9	6	8	2	3	4	3	0.000000	*	87/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
26	13	15	8	13	4	7	4	5	5	0.000002	*	92/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
92	4	1	1	1	1	0	0	0	0	0.000000	*	30/100	*	NonOverlappingTemplate
25	10	13	8	11	6	5	10	7	5	0.000253		97/100		NonOverlappingTemplate
26	19	9	14	10	6	5	6	4	1	0.000000	*	92/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
35	16	13	12	6	2	4	5	4	3	0.000000	*	88/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
50	17	4	9	4	4	3	2	4	3	0.000000	*	87/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
82	11	5	2	0	0	0	0	0	0	0.000000	*	46/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
98	2	0	0	0	0	0	0	0	0	0.000000	*	11/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	2/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
51	10	9	5	5	8	4	4	1	3	0.000000	*	83/100	*	NonOverlappingTemplate
14	11	18	13	6	9	9	1	12	7	0.016717		96/100		NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
21	16	6	11	11	7	4	11	9	4	0.002203		96/100		NonOverlappingTemplate
36	19	5	15	1	5	8	4	2	5	0.000000	*	93/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	*	NonOverlappingTemplate

100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
63	13	13	3	1	6	1	0	0	0	0.000000	*	74/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	1/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* NonOverlappingTemplate
34	14	12	9	7	9	4	2	5	4	0.000000	*	87/100	* NonOverlappingTemplate
24	13	18	8	10	5	8	5	5	4	0.000013	*	96/100	NonOverlappingTemplate
37	18	14	4	7	5	3	4	4	4	0.000000	*	92/100	* NonOverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* OverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* Universal
100	0	0	0	0	0	0	0	0	0	0.000000	*	0/100	* ApproximateEntropy
8	8	8	4	17	5	5	2	5	5	0.001919		67/67	RandomExcursions
11	5	4	3	7	11	8	7	5	6	0.222869		67/67	RandomExcursions
10	3	5	10	3	9	7	6	6	8	0.287306		66/67	RandomExcursions
7	8	8	6	12	7	8	6	2	3	0.186566		67/67	RandomExcursions
12	8	5	6	5	8	7	9	2	5	0.204076		66/67	RandomExcursions
4	6	5	12	6	6	4	7	9	8	0.364146		67/67	RandomExcursions
6	9	5	9	2	5	4	8	9	10	0.242986		67/67	RandomExcursions
11	6	5	9	6	7	4	6	8	5	0.585209		67/67	RandomExcursions
3	13	3	7	4	5	7	4	9	12	0.015065		66/67	RandomExcursionsVariant
5	7	5	4	5	6	11	12	4	8	0.170294		66/67	RandomExcursionsVariant
6	5	2	8	5	9	8	10	4	10	0.222869		65/67	RandomExcursionsVariant
8	4	3	2	7	12	8	11	5	7	0.051391		66/67	RandomExcursionsVariant
6	6	5	6	6	6	9	14	4	5	0.155209		65/67	RandomExcursionsVariant
5	9	7	5	4	8	10	9	6	4	0.517442		65/67	RandomExcursionsVariant
9	5	6	12	5	8	7	5	4	6	0.392456		63/67	RandomExcursionsVariant
9	7	6	12	5	6	6	2	11	3	0.063482		65/67	RandomExcursionsVariant
11	8	12	4	6	6	4	5	5	6	0.186566		65/67	RandomExcursionsVariant
9	7	9	8	8	5	2	10	3	6	0.242986		66/67	RandomExcursionsVariant
10	4	13	6	6	6	5	8	2	7	0.086458		66/67	RandomExcursionsVariant
8	12	4	7	3	5	10	5	9	4	0.116519		67/67	RandomExcursionsVariant
8	3	10	6	8	7	10	5	8	2	0.222869		67/67	RandomExcursionsVariant
5	8	9	7	6	7	8	6	8	3	0.788728		67/67	RandomExcursionsVariant
5	9	3	8	8	3	12	5	9	5	0.128379		67/67	RandomExcursionsVariant
4	4	12	5	6	5	4	9	13	5	0.033288		67/67	RandomExcursionsVariant

```
  4   4   9   9   8   1   8   8  10   6  0.155209     67/67      RandomExcursionsVariant
  5   5   5   8   7   9   7   4   4  13  0.186566     67/67      RandomExcursionsVariant
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Serial
 10   7   7  14  12  15  10   8   7  10  0.574903    100/100     LinearComplexity
```

- -

The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 63 for a sample size = 67 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.

- -

Références

[1] Abbas-Turki, L., and Graillat, S. CUDA/C Cuppen Divide and Conquer, Batch Computing. `http://www.lpsm.paris/pageperso/abbasturki/soft.htm`, 2016. 118

[2] Abbas-Turki, L., Graillat, S., Courtois, J., and Babacar, D. CUDA/C Cuppen Divide and Conquer, Batch Computing. `https://www.dropbox.com/sh/w70nv7b38ka5yuh/AABobvT-5NAhtyrX-G6aFhwja?dl=0`, 2020. 118

[3] Abbas-Turki, L. A., and Graillat, S. Resolving small random symmetric linear systems on graphics processing units. *The Journal of Supercomputing 73*, 4 (2017), 1360–1386. 13, 14, 56, 95, 96, 97, 98, 99, 101, 107, 117, 127, 134

[4] Abdelfattah, A., Haidar, A., Tomov, S., and Dongarra, J. Performance tuning and optimization techniques of fixed and variable size batched cholesky factorization on gpus. *Procedia Computer Science 80* (2016), 119–130. 95

[5] Ambrose, J. A., Pettenghi, H., and Sousa, L. DARNS: a randomized multi-modulo RNS architecture for double-and-add in ECC to prevent power analysis side channel attacks. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)* (2013), IEEE, pp. 620–625. 19

[6] American Bankers Association, et al. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). *ANSI X9* (2005), 62–1998. 7, 8, 21

[7] Andrew, R., et al. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. *NIST Special Publication 800-22rev1a* (2010). 8, 24

[8] Antao, S., Bajard, J.-C., and Sousa, L. Elliptic curve point multiplication on GPUs. In *ASAP 2010-21st IEEE International Conference on Application-specific Systems, Architectures and Processors* (2010), IEEE, pp. 192–199. 5

[9] Antão, S., Bajard, J.-C., and Sousa, L. RNS-based elliptic curve point multiplication for massive parallel architectures. *The Computer Journal 55*, 5 (2011), 629–647. 17, 18

[10] Aranha, D. F., Fouque, P.-A., Gérard, B., Kammerer, J.-G., Tibouchi, M., and Zapalowicz, J.-C. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In *International Conference on the Theory and Application of Cryptology and Information Security* (2014), Springer, pp. 262–281. 14, 21, 37

[11] Arbenz, P., Kressner, D., and Zürich, D. Lecture notes on solving large scale eigenvalue problems. *D-MATH, EHT Zurich 2* (2012). 104

[12] Bajard, J.-C., Didier, L.-S., and Kornerup, P. Modular multiplication and base extensions in residue number systems. In *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001* (2001), IEEE, pp. 59–65. 19, 20, 40

[13] Bajard, J. C., and Hördegen, H. Pseudo-random generator based on chinese remainder theorem. In *Mathematics for Signal and Information Processing* (2009), vol. 7444, International Society for Optics and Photonics, p. 74440Q. 5

[14] Bajard, J.-C., Imbert, L., Liardet, P.-Y., and Teglia, Y. Leak resistant arithmetic. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2004), Springer, pp. 62–75. 5, 18, 20, 21

[15] Bajard, J.-C., Imbert, L., and Plantard, T. Modular number systems: Beyond the Mersenne family. In *International Workshop on Selected Areas in Cryptography* (2004), Springer, pp. 159–169. 4

[16] Bajard, J.-C., and Plantard, T. RNS bases and conversions. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XIV* (2004), vol. 5559, International Society for Optics and Photonics, pp. 60–70. 22, 39, 40

[17] Barthe, G., Dupressoir, F., Fouque, P.-A., Grégoire, B., and Zapalowicz, J.-C. Synthesis of fault attacks on cryptographic implementations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 1016–1027. 14

[18] Batina, L., Chmielewski, Ł., Papachristodoulou, L., Schwabe, P., and Tunstall, M. Online template attacks. In *International Conference in Cryptology in India* (2014), Springer, pp. 21–36. 3, 17, 18

[19] Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F.-X., and Veyrat-Charvillon, N. Mutual information analysis: a comprehensive study. *Journal of Cryptology 24*, 2 (2011), 269–291. 14, 17, 18, 23, 33

[20] Bauer, A., Jaulmes, E., Prouff, E., and Wild, J. Horizontal and vertical side-channel attacks against secure RSA implementations. In *Topics in Cryptology – CT-RSA 2013* (Berlin, Heidelberg, 2013), E. Dawson, Ed., Springer Berlin Heidelberg, pp. 1–17. 14

[21] Benhamouda, F., Chevalier, C., Thillard, A., and Vergnaud, D. Easing Coppersmith methods using analytic combinatorics: applications to public-key cryptography with weak pseudorandomness. In *IACR International Workshop on Public Key Cryptography* (2016), Springer, pp. 36–66. 16, 18

[22] Bernstein, D. J., and Lange, T. Faster addition and doubling on elliptic curves. In *Advances in cryptology—ASIACRYPT 2007* (2007), Springer, pp. 29–50. 8, 17, 19, 24, 35, 39

[23] Borowska, A., and Rzeszutko, E. The cryptanalysis of the enigma cipher. the plugboard and the cryptologic bomb. *Computer Science 15* (2014). 66

[24] Brier, E., Clavier, C., and Olivier, F. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2004), Springer, pp. 16–29. 3, 17, 18, 23

[25] Carrol, L. *Alice Through the Looking-Glass*. Oxford Companion to English Literature 5th Ed, 1986, 1871. 2

[26] Certicom Research. Version 1.0 and 2.0: Standards for efficient cryptography recommended elliptic curve domain parameters. `https://www.secg.org/`, 2000. 8, 12, 17, 19, 21, 24, 35, 39

[27] Chari, S., Rao, J. R., and Rohatgi, P. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2002), Springer, pp. 13–28. 17, 18, 33

[28] Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., and Verneuil, V. Horizontal correlation analysis on exponentiation. In *Information and Communications Security* (Berlin, Heidelberg, 2010), M. Soriano, S. Qing, and J. López, Eds., Springer Berlin Heidelberg, pp. 46–61. 14

[29] Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and Vercauteren, F. *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman and Hall/CRC, 2005. 8, 21

[30] Courtois, J., Abbas-Turki, L., and Bajard, J.-C. Resilience of randomized rns arithmetic with respect to side-channel leaks of cryptographic computation. *IEEE Transactions on Computers 68*, 12 (2019), 1720–1730. 16

[31] Cuppen, J. J. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik 36*, 2 (1980), 177–195. 13, 97

[32] Cuppen, J. J. M. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik 36*, 2 (Jun 1980), 177–195. 101

[33] David, H. A., and Nagaraja, H. N. Order statistics. *Encyclopedia of Statistical Sciences* (2004). 11

[34] DeGroot, M. H., and Schervish, M. J. *Probability and statistics*. Pearson Education, 2012. 32

[35] Demmel, J. W. *Applied numerical linear algebra*, vol. 56. SIAM, 1997. 96, 101, 104

[36] Demmel, J. W., Marques, O. A., Parlett, B. N., and Vömel, C. Performance and accuracy of lapack's symmetric tridiagonal eigensolvers. *SIAM Journal on Scientific Computing 30*, 3 (2008), 1508–1526. 99

[37] Didier, L.-S., Dosso, F.-Y., El Mrabet, N., Marrez, J., and Véron, P. Randomization of Arithmetic over Polynomial Modular Number System. In *26th IEEE International Symposium on Computer Arithmetic* (Kyoto, Japan, June 2019). to appear. 4

[38] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE transactions on Information Theory 22*, 6 (1976), 644–654. 2

[39] Dugardin, M., Papachristodoulou, L., Najm, Z., Batina, L., Danger, J.-L., and Guilley, S. Dismantling real-world ECC with horizontal and vertical template attacks. In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (2016), Springer, pp. 88–108. 3, 17, 18

[40] Eisenstat, S. A stable algorithm for the rank-1 modification of the symmetric eigenproblem. *Computer Science Dept. Report YALEU/DCS/RR-916, Yale University* (1992). 106

[41] Fan, J., and Verbauwhede, I. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security : From Theory to Applications*. Springer, 2012, pp. 265–282. 17, 18

[42] Fournaris, A. P., Klaoudatos, N., Sklavos, N., and Koulamas, C. Fault and Power Analysis Attack Resistant RNS Based Edwards Curve Point Multiplication. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems* (New York, NY, USA, 2015), CS2 '15, ACM, pp. 43:43–43:46. 5, 14

[43] Fournaris, A. P., Papachristodoulou, L., and Sklavos, N. Secure and Efficient RNS software implementation for Elliptic Curve Cryptography. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2017), IEEE, pp. 86–93. 5, 19, 136

[44] Goddeke, D., and Strzodka, R. Cyclic reduction tridiagonal solvers on gpus applied to mixed-precision multigrid. *IEEE Transactions on Parallel and Distributed Systems 22*, 1 (2010), 22–32. 95, 98

[45] Gragg, W., Thornton, J. R., and Warner, D. D. Parallel divide and conquer algorithms for the symmetric tridiagonal eigen problem and bidiagonal singular value problem. *Modeling and Simulation 23* (1993), 49–49. 105

[46] Green, O., McColl, R., and Bader, D. A. Gpu merge path : a gpu merging algorithm. In *Proceedings of the 26th ACM international conference on Supercomputing* (2012), ACM, pp. 331–340. 96, 109, 112, 115

[47] Gu, M., and Eisenstat, S. C. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM Journal on Matrix Analysis and Applications 16*, 1 (1995), 172–191. 96, 106, 117, 119

[48] Guillermin, N. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over $\mathbb{F}_p$. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2010), Springer, pp. 48–64. 5, 17, 18, 19

[49] Haidar, A., Dong, T., Luszczek, P., Tomov, S., and Dongarra, J. Batched matrix computations on hardware accelerators based on gpus. *The International Journal of High Performance Computing Applications 29*, 2 (2015), 193–208. 94

[50] Hankerson, D., Menezes, A. J., and Vanstone, S. Guide to elliptic curve cryptography. *Computing Reviews 46*, 1 (2005), 13. 8, 21

[51] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, J. A. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st USENIX Security Symposium* (2012), pp. 205–220. 16, 18

[52] Izu, T., and Takagi, T. A fast parallel elliptic curve multiplication resistant against side channel attacks. In *International Workshop on Public Key Cryptography* (2002), Springer, pp. 280–296. 8, 21

[53] Jacod, J., and Protter, P. *Probability essentials*. Springer Science & Business Media, 2012. 26, 27, 40, 49, 50

[54] Joye, M. Smart-card implementation of elliptic curve cryptography and DPA-type attacks. In *Smart card research and advanced applications VI*. Springer, 2004, pp. 115–125. 17, 18

[55] Joye, M., Paillier, P., and Schoenmakers, B. On second-order differential power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2005), Springer, pp. 293–308. 60

[56] Joye, M., and Yen, S.-M. The montgomery powering ladder. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2002), Springer, pp. 291–302. 6

[57] Kawamura, S., Koike, M., Sano, F., and Shimbo, A. Cox-rower architecture for fast parallel montgomery multiplication. *International Conference on the Theory and Applications of Cryptographic Techniques* (2000), 523–538. 40

[58] Koblitz, N. Elliptic curve cryptosystems. *Mathematics of computation 48*, 177 (1987), 203–209. 2

[59] Kocher, P. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology* (1996), CRYPTO '96. 3

[60] Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Annual International Cryptology Conference* (1999), CRYPTO '99, Springer, pp. 388–397. 3

[61] Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. Introduction to differential power analysis. *Journal of Cryptographic Engineering 1*, 1 (2011), 5–27. 17, 18, 29

[62] L'ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. An object-oriented random-number package with many long streams and substreams. *Operations research 50*, 6 (2002), 1073–1075. 26

[63] Lesavourey, A., Negre, C., and Plantard, T. Efficient leak resistant modular exponentiation in RNS. In *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)* (2017), IEEE, pp. 156–163. 5, 19

[64] Levin, D. A., and Peres, Y. *Markov chains and mixing times*, vol. 107. American Mathematical Soc., 2017. 25

[65] Li, R.-C. Solving secular equations stably and efficiently. Tech. rep., California University Berkeley Dept of Electrical Engineering and Computer Science, 1993. 105

[66] Loevner, C. Über monotone matrixfunctionen. *Mathematische Zeitschrift 38* (1934), 177–216. 101

[67] Lund, R. An intermediate course in probability. *Journal of the American Statistical Association 92*, 439 (1997), 1225–1226. 50

[68] Maghrebi, H., Portigliatti, T., and Prouff, E. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering* (2016), Springer, pp. 3–26. 4

[69] MAGMA. Cholesky Batch computing. `https://icl.cs.utk.edu/projectsfiles/magma/doxygen/group__group__posv__batched.html`. 94, 95

[70] Mangard, S., Oswald, E., and Popp, T. *Power analysis attacks: Revealing the secrets of smart cards*, vol. 31. Springer Science & Business Media, 2008. 16, 18, 28, 29

[71] Martínez, V. G., Encinas, L. H., and Ávila, C. S. A survey of the elliptic curve integrated encryption scheme. *ratio 80*, 1024 (2010), 160–223. 7, 21

[72] Medwed, M., and Herbst, C. Randomizing the montgomery multiplication to repel template attacks on multiplicative masking. *COSADE, Lecture Notes in Computer Science.* (2010). 3

[73] Meloni, N. New point addition formulae for ECC applications. In *International Workshop on the Arithmetic of Finite Fields* (2007), Springer, pp. 189–201. 8, 21, 22

[74] Merkle, R. C. Secure communications over insecure channels. *Communications of the ACM 21*, 4 (1978), 294–299. 2

[75] Messerges, T. S. Using second-order power analysis to attack DPA resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2000), Springer, pp. 238–251. 14, 17, 18, 32, 60

[76] Miller, V. S. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques* (1985), Springer, pp. 417–426. 2

[77] Montgomery, P. L. Modular multiplication without trial division. *Mathematics of computation 44*, 170 (1985), 519–521. 19, 21

[78] Montgomery, P. L. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation 48*, 177 (1987), 243–264. 8

[79] NTT Information, Sharing Platform Laboratories, NTT Corporation. PSEC-KEM Specification version 2.2, 2008. 8, 21

[80] NVIDIA Corporation. Cuda/C programming guide. `https://docs.nvidia.com/cuda/`, 2019. 13, 56, 94, 133

[81] NVIDIA Corporation. Documentation CUDA Toolkit v8. 0.61, cuBLAS. `https://docs.nvidia.com/cuda/cublas/`, 2019. 94

[82] NVIDIA Corporation. Documentation CUDA Toolkit v8. 0.61, cuSOLVER. `https://docs.nvidia.com/cuda/cusolver/index.html`, 2019. 94, 95

[83] Perin, G., Imbert, L., Torres, L., and Maurine, P. Attacking randomized exponentiations using unsupervised learning. In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (2014), Springer, pp. 144–160. 4, 5, 19, 24

[84] Perin, G., Imbert, L., Torres, L., and Maurine, P. Practical Analysis of RSA Countermeasures Against Side-Channel Electromagnetic Attacks. In *Smart Card Research and Advanced Applications* (2014), A. Francillon and P. Rohatgi, Eds., Springer International Publishing, pp. 200–215. 5

[85] Poussier, R., Zhou, Y., and Standaert, F.-X. A systematic approach to the side-channel analysis of ECC implementations with worst-case horizontal attacks. In *International Conference on Cryptographic Hardware and Embedded Systems* (2017), Springer, pp. 534–554. 14

[86] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126. 2

[87] Scholz, F. Maximum likelihood estimation. *Encyclopedia of statistical sciences* (2004). 33

[88] Shapiro, A., Dentcheva, D., and Ruszczyński, A. *Lectures on stochastic programming: modeling and theory*. SIAM, 2009. 68, 69

[89] Shenoy, A., and Kumaresan, R. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers 38*, 2 (1989), 292–297. 40

[90] Stottinger, M., Yao, G., and Cheung, R. C. Zero collision attack and its countermeasures on Residue Number System multipliers. *Proceedings of the 14th International Symposium on Integrated Circuits, ISIC 2014* (02 2015), 30–33. 5

[91] Szabó, N., and Tanaka, R. *Residue arithmetic and its applications to computer technology*. McGraw-Hill series in information processing and computers. McGraw-Hill, 1967. 5

[92] The Sage Developers. SageMath, the Sage Mathematics Software System (Version 6.5). `https://www.sagemath.org`, 2015. 56

[93] Van, V., and Van Valen, L. A new evolutionary law. *Evolutionary theory* (1973). 3

[94] Vetterling, W. T., Press, W. H., Teukolsky, S. A., and Flannery, B. P. *Numerical Recipes Example Book (C++): The Art of Scientific Computing*. Cambridge University Press, 2002. 98

[95] Yao, G. X., Stöttinger, M., Cheung, R. C., and Huss, S. A. Side Channel Attacks and Their Low Overhead Countermeasures on Residue Number System Multipliers. In *Emerging Technology and Architecture for Big-data Analytics*. Springer, 2017, pp. 137–158. 5, 19, 38

[96] Yarom, Y., and Benger, N. Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack. *IACR Cryptology ePrint Archive 2014* (2014), 140. 137

[97] Zhang, Y., Cohen, J., and Owens, J. D. Fast tridiagonal solvers on the gpu. *ACM Sigplan Notices 45*, 5 (2010), 127–136. 95, 98, 99

Étude des fuites d'implémentations de cryptosystème en arithmétique RNS randomisée

Résumé

On parlera d'analyse forte pour une analyse qui permet de retrouver la clef d'un système cryptographique et d'une analyse faible dans le cas où on élimine des clefs candidates. Le but de cette thèse est essentiellement la compréhension du comportement de l'aléa des distances de Hamming produites par un système cryptographique de type ECC (Elliptic Curve for Ccryptography) quand on utilise une représentation RNS (Residue Number System) avec la méthode des moduli aléatoires.

Le Chapitre 2 introduit les différentes notions nécessaires à la compréhension de ce document. Il introduit brièvement l'algorithme de multiplication modulaire (Algorithme de Montgomery pour RNS) qui a inspiré la méthode des moduli aléatoires. Puis il décrit l'algorithme qui génère les séquences de distances de Hamming nécessaires à notre analyse. Ensuite il montre quel niveau de résistance apporte la méthode des moduli aléatoires contre différentes attaques classiques comme DPA (Diferential Power Analysis), CPA (Correlation Power Analysis), DPA du second ordre et MIA (Mutual Information Analysis). On apporte une compréhension de la distribution des distances de Hamming considérées comme des variables aléatoires. Suite à cela, on ajoute l'hypothèse gaussienne sur les distances de Hamming. On utilise alors le MLE (Maxumum Likelihood Estimator) et une analyse forte comme pour faire des Template Attacks pour avoir une compréhension fine du niveau d'aléa apporté par la méthode des moduli aléatoires.

Le Chapitre 3 est une suite naturelle des conclusions apportées par le Chapitre 2 sur l'hypothèse gaussienne. Si des attaques sont possibles avec le MLE, c'est qu'il existe sans doute des relations fortes entre les distances de Hamming considérées comme des variables aléatoires. La Section 3.2 cherche à quantifier ce niveau de dépendance des distances de Hamming. Ensuite, en restant dans l'hypothèse gaussienne, on remarquera qu'il est possible de construire une type de DPA qu'on a appelé DPA square reposant sur la covariance au lieu de la moyenne comme dans la DPA classique. Mais cela reste très gourmand en traces d'observation d'autant que dans de nombreux protocoles utilisant une ECC, on utilise une clef qu'une seule fois. La Section 3.4 s'efforce de montrer qu'il existe de l'information sur peu de traces de distances de Hamming malgré la randomisation des moduli. Pour cela, on fait un MLE par un conditionnement sur l'une des distances de Hamming avec une analyse faible.

Le dernier Chapitre 4 commence par introduire brièvement les choix algorithmiques

qui ont été faits pour résoudre les problèmes d'inversion de matrices de covariance (symétriques définies positives) de la Section 2.5 et l'analyse des relations fortes entre les Hamming dans la Section 3.2. On utilise ici des outils de Graphics Processing Unit (GPU) sur un très grand nombre de matrices de petites tailles. On parlera de Batch Computing. La méthode LDLt présentée au début de ce chapitre s'est avérée insuffisante pour résoudre complètement le problème du MLE conditionné présenté dans la Section 3.4. On présente le travail sur l'amélioration d'un code de diagonalisation de matrice tridiagonale utilisant le principe de Diviser pour Régner (Divide & Conquer) développé par Lokmane Abbas-Turki et Stéphane Graillat. On présente une généralisation de ce code, des optimisations en temps de calcul et une amélioration de la robustesse des calculs en simple précision pour des matrices de taille inférieure à 32.

Mots-clés: RNS, moduli randomisation, Monte Carlo, ECC, canaux auxiliaires, DPA, CPA, MIA, Template Attack, fuite d'information, poids de Hamming, distances de Hamming, Estimateur maximal de vraisemblance, GPU, Cuppen Divide and Conquer, hypothèse gaussienne.

Leak study of randomized RNS arithmetic cryptosystem implementations

Abstract

We will speak of strong analysis for an analysis that recovers the key of a cryptosystem. We define a weak analysis in the case where candidate keys are eliminated. The goal of this thesis is to understand the randomness behavior of Hamming distances produced by an ECC (Elliptic Curve for Cryptography) cryptographic system when using a RNS (Residue Number System) representation with the random moduli method.

Chapter 2 introduces the different concepts for understanding this document. It briefly introduces the modular multiplication algorithm (Montgomery algorithm for RNS) which inspired the method of random moduli. We describe the algorithm which generates the Hamming distance sequences necessary for our analysis. Then we shows what level of resistance brings the method of random moduli against different classic attacks like DPA (Differrential Power Analysis), CPA (Correlation Power Analysis), DPA of the second order and MIA (Mutual Information Analysis). We provide an understanding of Hamming distances distribution. Following this, we add the Gaussian hypothesis on Hamming distances. We use MLE (Maximum Likelihood Estimator) and a strong analysis to have a better understanding of the level of random brought by the method of random moduli.

The Chapter 3 is a natural continuation of the conclusions brought by the Chapter 2 on the Gaussian hypothesis. If attacks are possible with the MLE, it is because there are perhaps strong relationships between the Hamming distances considered as random variables. Section 3.2 seeks to quantify this level of dependence on Hamming distances. Then, staying in the Gaussian hypothesis, we will notice that it is possible to construct a type of DPA that we called DPA square based on covariance instead of the average as in classical DPA. But that remains very greedy in traces of observation especially since in many protocols using an ECC, one uses a key only once. Section 3.4 strives to show that there is information on few traces of Hamming distances despite the randomization of the moduli. For this, we do an MLE by conditioning on one of the Hamming distances with a weak analysis.

The last Chapter 4 begins by briefly introducing the algorithmic choices that have been made to solve the inversion problem of covariance matrices (symmetric definite positive) of Section 2.5 and the analysis of strong relationships between Hamming in Section 3.2. We use here Graphics Processing Unit (GPU) tools on a very large number of small

size matrices. We talk about Batch Computing. The LDLt method presented at the beginning of this chapter proved to be insufficient to completely solve the problem of conditioned MLE presented in Section 3.4. We present work on the improvement of a diagonalization code of a tridiagonal matrix using the principle of Divide & Conquer developed by Lokmane Abbas-Turki and Stéphane Graillat. We present a generalization of this code, optimizations in computation time and an accuracy improvement of computations in simple precision for matrices of size lower than 32.

Mots-clés: RNS, moduli randomization, Monte Carlo, ECC, side channel, DPA, CPA, MIA, Template Attack, information leakage, Hamming weight, Hamming distance, Maximum Likelihood Estimator, GPU, Cuppen Divide and Conquer, Gaussian hypothesis.