Etudes cryptographiques et statistiques de signaux compromettants

Yanis Linge

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques Appliquées**

Arrêté ministériel : 7 août 2006

Présentée par

Yanis Linge

Thèse dirigée par **Sophie Lambert-Lacroix**
et encadrée par **Cécile Dumas**

préparée au sein **du CESTI du CEA/LETI et du TIMC**
et de **l'École Doctorale Mathématiques, Sciences et Techniques de l'Ingénieur, Informatique**

Études cryptographiques et statistiques de signaux compromettants

Thèse soutenue publiquement le **22 Novembre 2013**,
devant le jury composé de :

M. Jean-Guillaume Dumas
Professeur à l'université de Grenoble, Président
M. Gilles Ducharme
Professeur à l'université de Montpellier 2, Rapporteur
M. Emmanuel Prouff
Ingénieur à l'ANSSI, Rapporteur
M. Louis Goubin
Professeur à l'université Versailles St-Quentin-en-Yvelines, Examinateur
Mme. Thanh Ha Le
Ingénieur à Morpho, Examinateur
Mme. Gwenaëlle Martinet
Ingénieur à la DGA , Examinateur
Mme. Cécile Dumas
Ingénieur-Chercheur au CEA-Leti Minatec, Encadrante
Mme. Sophie Lambert-Lacroix
Professeur des Universités à l'université de Grenoble, Directeur de thèse

Remerciements

Les remerciements sont toujours un exercice difficile. Il est évident que je vais oublier de citer certaines personnes et je m'en excuse par avance.

Je souhaiterais tout d'abord remercier Sophie Lambert-Lacroix d'avoir dirigé cette thèse et de m'avoir initié au monde des statistiques. Je tiens aussi remercier l'ensemble des membres du jury qui ont su braver les éléments pour venir assister à ma soutenance. Et en particulier Gilles Ducharme et Emmanuel Prouff pour le temps qu'ils ont bien voulu consacré pour rapporter mon travail. Merci aussi à Jean-Guillaume Dumas qui m'a donné la confiance nécessaire pour effectuer ce doctorat.

Pour effectuer cette thèse, j'ai eu la chance d'être intégré au CESTI-LETI. J'ai toujours eu le sentiment d'être "comme à la maison", cela grâce aux différents membres de cette équipe. Côtoyer chaque jour des experts en sécurité des composants m'a permis d'aiguiser mes connaissances et des les enrichir. Je tiens tout particulièrement à remercier Elisabeth Crochon pour l'accueil qu'elle m'a réservé dans l'équipe, ainsi que Jessy Clédière pour les nombreuses discutions que nous avons pu avoir tant au niveau professionnel que personnel. Merci aussi à Manu Pezzin pour tous ces bons moments passé ensemble au CEA, car oui, un lieu de travail est aussi un lieu de vie.

Ces trois années de thèse ont été ponctuées de haut et de bas. Chaque creux a pu être surmonté grâce à mes proches et ma famille. Anne-Julie m'a permis de relativiser tous les problèmes inhérent à un doctorat. Merci à mes amis triathlètes du GUC pour tous les entrainements que nous avons partagé et qui m'ont permis de m'échapper et de voir autre chose que les sciences. Et en particulier, Chloé, ma coéquipière préférée avec laquelle nous monterons peut être enfin ensemble "sur la boîte". Bien évidement, toute ma gratitude va à Véronique avec qui nous partageons tant de choses depuis plus de 12 ans et qui a hérité de la lourde tâche de relire mes différents écrits. Mes remerciements vont aussi à ma petite sœre avec qui nous avons partagé tant de bons moments, à ma maman qui m'a toujours soutenu et à mon papa qui a relu ce manuscrit et qui a activement participé à la préparation de la soutenance en y intégrant Gaston. Merci aussi à tous ceux qui été présent au Sappey pour partager la fin de cette aventure.

Mais cette thèse n'aurait jamais vu le jour sans Cécile Dumas qui m'a fait confiance et qui a su m'accompagner, me guider, m'épauler, me faire rebondir ...

Enfin, je voudrais remercier Diana, qui partage ma vie depuis 10 ans, pour tout le bonheur que nous partageons. Elle a su m'entourer de toute son attention et son amour, me redonner du courage lorsque les difficultés semblaient s'accumuler (et me nourrir durant la rédaction de ce manuscrit). Après ces trois années, une nouvelle vie commence pour nous...

À mon premier ami
Alex et à Rib4e.
Le passé et le futur

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des acronymes

AES *Advanced Encryption Standard*
ANSSI *Agence Nationale de la Sécurité des Systèmes d'Information*
API *Application Programming Interface*
CPA *Correlation Power Analysis*
CPU *Central Processing Unit*
DES *Data Encryption Standard*
DPA *Differential Power Analysis*
GPU *Graphics Processing Unit*
GPGPU *General Programming on Graphic Processing Units*
HSA *Heterogeneous System Architecture*
hUMA *Heterogeneous Uniform Memory Access*
LUT *Look-Up Table*
MIA *Mutual Information Analysis*
MIC *Maximal Information Coefficent*
MICA *Maximal Information Coefficent Analysis*
NIST *National Institute of Standard and Technology*
OpenCL *Open Computing Language*
RFID *Radio Frequency IDentification*
RSA *Rivest Shamir Adleman*
SPA *Simple Power Analysis*
WEP *Wired Equivalence Privacy*

Avant-propos

Ce manuscrit présente les travaux de recherche que j'ai effectués au sein du CESTI [1] du CEA-LETI. La cryptographie classique se propose de fournir des algorithmes reposant sur la difficulté à résoudre certains problèmes mathématiques. Elle s'appuie sur ceux-ci afin d'exposer des preuves de sécurité pour les algorithmes proposés du point de vue de la théorie de la complexité. Depuis le milieu des années 90, de nouvelles formes d'attaques contre les algorithmes cryptographiques ont vu le jour, notamment dans le domaine des cartes à puce. Elles n'utilisent plus les faiblesses mathématiques de l'algorithme lui-même, mais les vulnérabilités du composant sous-jacent et la façon dont l'algorithme est implémenté. Les attaques par observations ont été fortement développées jusqu'à aujourd'hui.

Mon travail de recherche a été orienté autour de trois grands axes : l'implémentation efficace des attaques par observations, l'étude de nouveaux distingueurs et enfin la recherche d'attaques ne nécessitant ni la connaissance du texte clair ni celle du texte chiffré. Ce mémoire est divisé en cinq chapitres. Le premier est un chapitre d'introduction à la cryptologie, nous y présenterons notamment trois algorithmes classiques de la cryptographie : le RSA, le DES et l'AES qui sont aussi couramment utilisés dans les applications carte à puce. Le second a pour but d'introduire les attaques par observations. Dans le troisième chapitre, nous proposons l'utilisation de GPGPU dans le cadre des attaques par observations afin d'obtenir des implémentations plus performantes. Le quatrième expose une méthode statistique récente qui permet de comparer deux variables, et montre comment l'utiliser pour monter une attaque par observations. Nous proposerons dans le dernier chapitre une nouvelle attaque par observations permettant de retrouver une clé secrète sans connaître ni l'entrée ni la sortie de l'algorithme attaqué. Finalement, nous conclurons ce manuscrit en rappelant les résultats obtenus ainsi que quelques perspectives de recherches qui ont été soulevées lors de cette rédaction.

1. *Centre d'Evaluation de la Sécurité des Technologies de l'Information*

Chapitre 1

Introduction

DANS ce premier chapitre, nous présenterons quelques bases de la cryptologie. Nous allons rappeler qu'il est important que la sécurité des algorithmes de chiffrement repose sur les clés utilisées et non sur l'algorithme lui-même. Nous présenterons de manière succincte la cryptographie ainsi que la cryptanalyse, avant de décrire trois algorithmes classiques de la cryptographie.

1.1 Introduction

La cryptologie, du grec Kruptos (caché) et Logos (discours), signifie la science du secret. La cryptologie est une science millénaire, sans doute née en même temps que le langage. En effet, la capacité pour un groupe de personnes, à communiquer sans que leurs ennemis puissent intercepter des informations sensibles, peut décider de la physionomie d'un conflit. Lors de la Seconde Guerre mondiale, par exemple, les alliés ont été capables de briser le secret des communications allemandes grâce au travail acharné d'une équipe de cryptologues dirigée par Alan Turing. La réussite de cette équipe a été un des tournants de cette guerre.

L'avènement de l'informatique et de l'internet a vu exploser les techniques de cryptologie. Aujourd'hui, chacun utilise quotidiennement un grand nombre d'algorithmes cryptographiques sans en avoir conscience.

L'étude de la cryptologie s'articule autour de quatre grands piliers :

— La confidentialité : qui permet de communiquer un message dont la lecture ne sera possible que par un petit nombre d'entités choisies.
— L'authenticité : qui permet de s'assurer qu'un individu est bien l'auteur d'un message.
— L'intégrité : qui permet de s'assurer que le message n'a pas été modifié sans autorisation ou par erreur.
— La non-répudiation : qui permet de prouver que le message a bien été envoyé et/ou reçu.

Classiquement la cryptologie se divise en deux grandes branches : la **cryptographie** et la **cryptanalyse**.

La **cryptographie**, ou écriture secrète, consiste à élaborer des méthodes permettant d'assurer au moins l'un des quatre piliers précédents. Elle se divise en deux grandes familles d'algorithmes de chiffrement : les algorithmes symétriques et les algorithmes asymétriques.

La cryptographie symétrique, encore appelée cryptographie à clé secrète, est la première à avoir vu le jour. Son principe est d'utiliser une clé commune pour les opérations de chiffrement et de déchiffrement. La sécurité de la communication est alors assurée par le secret de cette clé. Ces algorithmes sont capables de chiffrer et de déchiffrer une quantité très importante de données en peu de temps. L'avantage de tels algorithmes est le rapport entre la taille des clés utilisées (moins de 256 bits généralement) et la robustesse des algorithmes. Le défaut principal de la cryptographie symétrique est la gestion des clés. En effet, il faut que les deux parties qui souhaitent communiquer conviennent d'une clé commune. Aujourd'hui, chacun utilise quotidiennement un grand nombre de communications chiffrées (téléphone cellulaire, distributeur de billets, badge d'accès, etc.). Il n'est donc plus possible de rencontrer la personne avec laquelle on veut communiquer afin de s'accorder sur une clé commune, comme le faisait par exemple les présidents des États Unis et de l'U.R.S.S durant la Guerre Froide pour le fameux *Téléphone rouge* [Sin99]. Il a donc fallu inventer une nouvelle famille d'algorithmes de chiffrement afin de pouvoir contourner ce problème.

C'est en 1976 que Withfield Diffie et Martin Hellman [DH76] ont introduit le concept de cryptographie asymétrique ou cryptographie à clé publique. Malheureusement, leur article ne proposait pas réellement d'algorithme et il a fallu attendre 1978 et l'invention du *Rivest Shamir Adleman* (RSA) [RSA78] pour voir émerger cette nouvelle famille d'algo-

rithmes. Les algorithmes à clé publique utilisent deux clés différentes, l'une pour chiffrer, l'autre pour déchiffrer et sont basés sur des problèmes mathématiques réputés difficiles à résoudre (la factorisation pour le RSA et le logarithme discret pour El-Gamal [Gam84] pour ne citer qu'eux). Pour chiffrer un message que l'on souhaite envoyer à un correspondant, il suffit de connaître sa clé publique qui servira à chiffrer le message. Dès lors, le message est illisible même avec la connaissance de cette clé. Lorsque le correspondant reçoit un message chiffré, il utilise sa clé privée, connue de lui seul, afin de le déchiffrer. La cryptographie à clé publique souffre de deux défauts majeurs. D'une part, il est difficile d'être sûr que l'on utilise bien la clé publique de notre correspondant et non celle d'une personne mal intentionnée et d'autre part, ces algorithmes sont plutôt lents et leur sécurité nécessite l'utilisation de clés de taille très importante (la recommandation de l'*Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI)[1] en 2010 est d'utiliser des clés de taille supérieure ou égale à 2048 bits pour l'algorithme RSA). L'application la plus courante de ces algorithmes est ce que l'on appelle le protocole d'échange de clé. Il permet à deux interlocuteurs de se mettre d'accord sur une clé commune afin de pouvoir utiliser un algorithme de chiffrement à clé privée pour pouvoir échanger plus de données.

D'une manière générale, il est très important que la sécurité des algorithmes de chiffrement repose bien sur la clé secrète et non sur le secret de l'algorithme. Ce principe a été introduit par Kerckhoffs dès la fin du XIX^e siècle [Ker83a][Ker83b]. Lors de la Seconde Guerre Mondiale, les allemands et leurs alliés utilisaient pour chiffrer leur communication la machine *Enigma*. La sécurité de cette machine était basée principalement sur le secret de l'algorithme. Alan Turing et les alliés ont été capables de briser le chiffrement des communications allemandes grâce, en partie, à l'obtention de machines *Enigma* et à l'étude de l'algorithme secret utilisé par ces machines [Sin99]. Le non-respect de cette règle a été la perte du protocole *Wired Equivalence Privacy* (ou WEP) utilisé il y a encore quelques années pour chiffrer des communications WiFi [Man05]. Les algorithmes de chiffrement standard actuels sont tous publics. Ainsi chacun est libre de les tester et d'essayer de les attaquer. La force de ces algorithmes est d'avoir été testés par un grand nombre de personnes.

La seconde branche de la cryptologie, la **cryptanalyse**, vise à analyser les méthodes proposées par la cryptographie en vue de retrouver les clés secrètes utilisées ou à défaut d'être capable de retrouver un texte clair à partir d'un texte chiffré. On distingue deux types de cryptanalyse : la cryptanalyse mathématique et la cryptanalyse physique.

La cryptanalyse mathématique étudie la résistance des algorithmes en s'intéressant plus particulièrement à leurs propriétés mathématiques. Cette partie a été la première à avoir été explorée et développée.

La cryptanalyse dite physique, quant à elle, étudie plus particulièrement les faiblesses liées aux composants sur lesquels sont implantés les algorithmes cryptographiques.

Il existe de nombreux composants contenant des algorithmes cryptographiques, les cartes à puce, bien sûr, mais aussi les puces RFID, certains processeurs de nos ordinateurs, etc. Pour le lecteur désireux d'en savoir un peu plus sur la cryptographie, les ouvrages [Dub98, FS03, DRTV07] permettent d'approfondir le sujet.

Avant de continuer plus avant, il nous faut décrire le RSA, un algorithme asymétrique ainsi que deux algorithmes symétriques, le *Data Encryption Standard* (DES) et l'*Advanced Encryption Standard* (AES).

1. L'ANSSI assure la mission d'autorité nationale en matière de sécurité des systèmes d'information.

1.2 Description du RSA

La construction des clés RSA peut être décrite de la manière suivante :
— Choisir deux nombres p et q premiers, leur produit pq sera noté n.
— Calculer l'indicatrice d'Euler de n : $\varphi(n) = (p-1) \times (q-1)$.
— Choisir e un entier premier avec $\varphi(n)$, e sera appelé l'exposant de chiffrement.
— Trouver un entier d tel que $d.e \equiv 1 \bmod \varphi(n)$. [2]

Le couple (n, e) sera la clé publique tandis que le couple (n, d) sera la clé privée.

On a maintenant à notre disposition une clé publique et une clé privée. Soit X un entier plus petit que n, qui représente le message que l'on souhaite chiffrer. Le message chiffré, noté C, sera calculé grâce à :

$$C \equiv X^e \bmod \text{n}.$$

Pour déchiffrer le message, qui sera un entier plus petit que n, il suffit au propriétaire de la clé secrète de calculer :

$$C^d \equiv X^{ed} \bmod \text{n} \equiv X \bmod \text{n}.$$

Cet algorithme est sûr si l'on considère que les nombres p et q ont une taille suffisante (au moins 1 024 bits chacun) et qu'ils sont *bien choisis* [KKLM09]. En effet, il est très difficile de déterminer $\varphi(n)$ lorsque l'on ne connaît pas les nombres p et q, c'est-à-dire lorsque l'on n'est pas capable de factoriser n. Cet algorithme est très utilisé dans la cryptographie moderne pour partager des clés.

1.3 Description du DES

Nous présentons ici les notations usuelles du DES qui seront utilisées dans ce document et qui dérivent directement des notations du *National Institute of Standard and Technology* (NIST) [oS77]. La Figure 1.1 présente le schéma de principe de l'algorithme de chiffrement DES. L'entrée de l'algorithme est un message à chiffrer M de 8 octets, soit 64 bits, et la sortie un chiffré de taille identique à l'entrée. Le DES est utilisé pour chiffrer et déchiffrer un message avec une même clé k, dans ce deuxième cas on le note $DES^{-1}(M, k)$.

La clé est également de 8 octets mais en pratique seulement 56 bits sont utilisés. En 2013, la recherche exhaustive des 2^{56} valeurs possibles de clés est possible depuis longtemps. Il est donc possible de *casser le DES en force brute*. Pour pallier cette faiblesse évidente, tous les responsables d'applications utilisant le DES ont mis à jour leurs spécifications pour passer en triple-DES à la fin des années 90. Le triple-DES utilise trois clés $k1$, $k2$ et $k3$ et correspond à un enchaînement d'un DES, d'un DES^{-1} et d'un dernier DES, soit pour le chiffrement de M : $DES(DES^{-1}(DES(M, k1), k2), k3)$. On peut choisir trois clés différentes, mais il existe une attaque par force brute sur deux des clés basée sur l'attaque de la rencontre au milieu [vOW96, Luc98], aussi choisit-on souvent une configuration triple-DES avec seulement deux clés différentes telles que $k1 = k3 \neq k2$.

Le DES commence par une permutation initiale et se termine par une permutation finale qui est l'inverse de la permutation initiale. Le quantum de l'information de l'algorithme est le bit, valeur 0 ou 1. Les permutations ont pour but de mélanger les bits.

Suite à cette première permutation, 16 rondes sont appliquées suivant un schéma de Feistel (voir Figure 1.1). L'entrée d'une ronde prend les registres internes de 32 bits L_i et R_i et en sortie les registres L_{i+1} et R_{i+1}. A chaque ronde, une sous-clé k_i de

2. Le théorème de Bachet-Bezout permet de prouver l'existence d'un tel inverse

48 bits est utilisée. Ces sous-clés sont générées à partir de la clé k par des opérations linéaires, principalement les permutations $PC1$ et $PC2$ [3] et des décalages à gauche. La génération des sous-clés n'est pas illustrée dans ce manuscrit, mais on peut noter une forte redondance des bits de clés entre les différentes sous-clés k_i. La fonction f est le cœur des 16 rondes du DES, celle-ci prend en entrée un registre de 32 bits, le registre R_i, et donne 32 bits en sortie. La Figure 1.2 schématise cette fonction. La fonction commence par une expansion de R_i en un registre de 48 bits : 16 bits sont dupliqués. Un OU exclusif est alors réalisé avec la sous-clé correspondante. Le OU exclusif entre deux bits est défini par la Table 1.1.

$\oplus$	0	1
0	0	1
1	1	0

Tableau 1.1 – OU exclusif

Puis les boites-S du DES sont appliquées. Ces boites-S, au nombre de 8, prennent en entrée 6 bits du registre, et donnent en sortie 4 bits. Le morcelage du résultat du OU exclusif de la donnée et de la sous-clé en bloc de 6 bits est un point important du DES. Ce morcelage a comme origine une raison technique d'implémentation : une boite-S est implémentée par un tableau (en logiciel) ou par une *Look-Up Table* (LUT) (en câblé) et la mémorisation de cet élément doit avoir une taille raisonnable, 64 entrées pour une LUT ou un tableau ; une taille acceptable dans les années 70. À la sortie des boites-S, 32 bits sont disponibles et sont permutés par P. Un OU exclusif de la sortie de la fonction f avec le registre L est réalisé en fin de ronde.

Notons que les boites-S sont les seuls éléments non linéaires de l'algorithme DES. La façon dont elles ont été choisies n'est donc pas anodine comme l'a expliqué Don Coppersmith dans [Cop94]. Le DES et le DES^{-1} sont identiques, à cela près que les sous-clés sont prises en sens inverse.

3. PC comme "Permuted Choice", les deux choix sont 1 et 2.

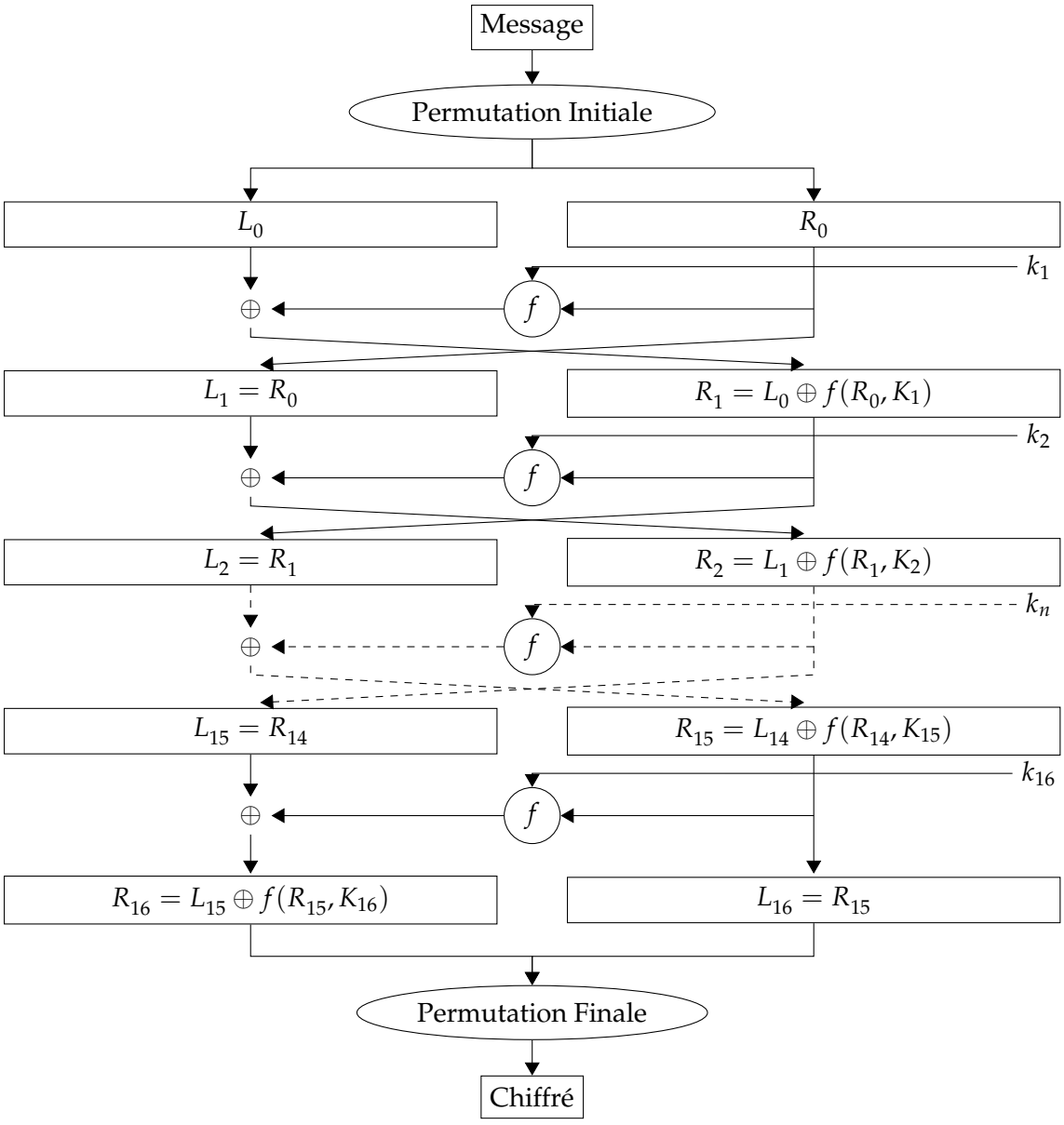


FIGURE 1.1 – Algorithme DES.

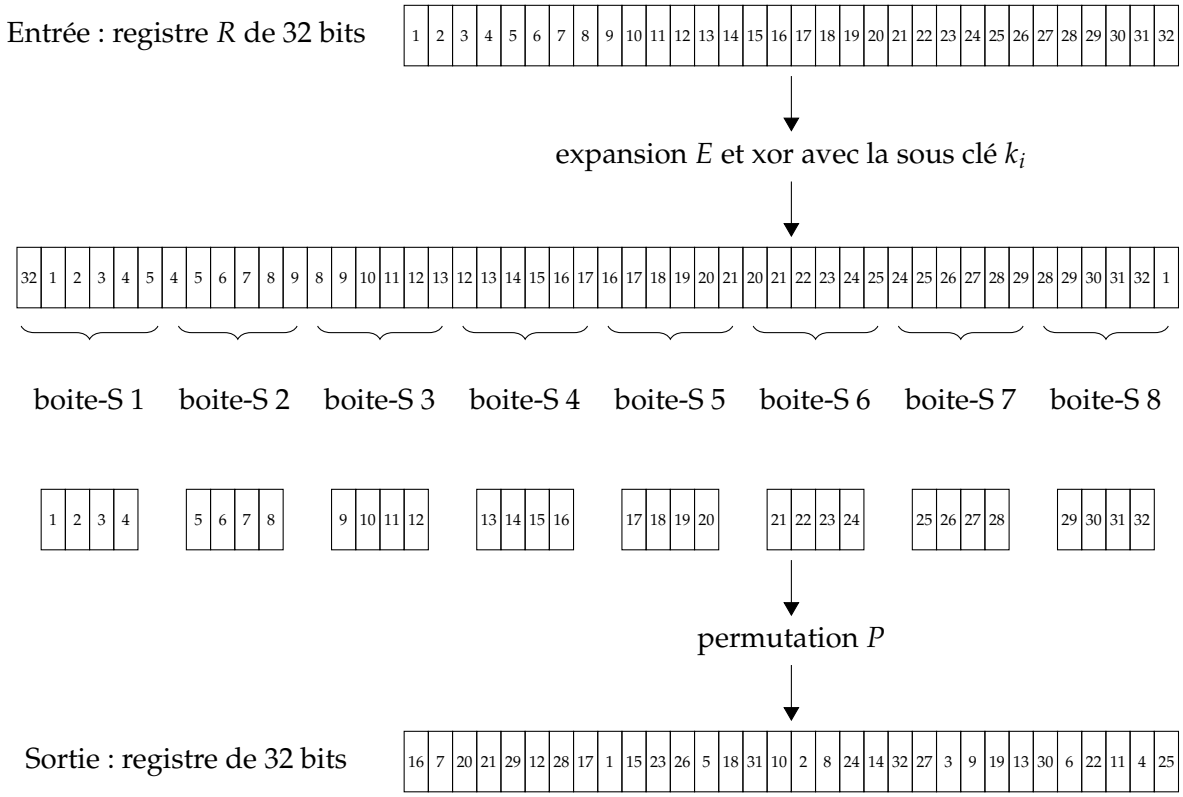


FIGURE 1.2 – Schématisation de la fonction f du DES avec numérotation des bits des registres.

1.4 Description de l'AES

L'AES [oS01] est un algorithme de chiffrement symétrique. Cet algorithme prend en entrée un message de 128 bits et une clé de 128, 192 ou 256 bits. Le processus de chiffrement se décompose en rondes. Le nombre de rondes varie selon la taille des clés (10 pour 128 bits, 12 pour 192 et 14 pour 256). On distingue essentiellement deux types de données utilisées dans l'AES, la clé de ronde, dérivée de la clé, et l'état qui correspond au résultat temporaire du chiffrement. Le quantum de l'information de l'AES est l'octet, valeurs dans $\{0, \cdots, 255\}$. Ces données utilisées dans l'algorithme sont vues sous forme de matrices d'octets de 4 lignes et 4 colonnes. L'élément de la i-ème ligne, j-ème colonne d'une donnée E sera noté $E^{(i,j)}$ avec $i, j \in \{0, 1, 2, 3\}$. Chaque élément d'un état est vu comme un élément de $GF(2^8) = \mathbb{Z}/2\mathbb{Z}[X]/P(X)$ où $P(X) = X^8 + X^4 + X^3 + X + 1$. Cinq grandes fonctions composent l'AES, la fonction AddRoundKey, le ShiftRow, le SubBytes, MixColumns et enfin le KeySchedule.

La fonction AddRoundKey

Cette étape consiste à faire un OU exclusif bit à bit entre la clé de ronde et l'état.

Le ShiftRow

Dans le ShiftRow, on effectue une permutation circulaire sur chaque ligne de l'état. Cette permutation peut être décrite par la formule suivante où i est le numéro de la ligne concernée et E l'état auquel on applique le ShiftRow (Figure 1.3) :

$$ShiftRow(E^{(i,j)}) = E^{(i,(j-i) \bmod 4))}$$

FIGURE 1.3 – ShiftRow

Le SubBytes

C'est une fonction non linéaire, elle est l'un des rouages les plus essentiels de l'AES. Elle est décrite soit par une table (comme les boites-S du DES), soit par une expression matricielle, ou si l'on considère les octets comme des éléments de GF_{2^8} par la fonction :

$$SubBytes(a) = a^{-1} \text{ si } a \neq 0, 0 \text{ sinon.}$$

Le MixColumns

Le MixColumns est une fonction qui permet de lier les éléments d'une même colonne d'un état. C'est une multiplication entre l'état et la matrice suivante :

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

De manière plus générale, si l'on considère chaque colonne comme des polynômes, $P(X)$, de GF_{2^8} on peut décrire cette opération par :

$$MixColumns(P(X)) = P(X) \times (3X^3 + X^2 + X + 2) \bmod X^4 + 1$$

Le KeySchedule

Le KeySchedule est la fonction qui permet d'obtenir les clés de rondes. Dans cette fonction on regarde chaque colonne de la clé K comme un mot, on a alors $K = W_0W_1W_2W_3$. Pour calculer le mot W_i on utilise les mots W_{i-1} et W_{i-4} et on a :

$$W_i^j = SB(W_{i-4}^j) \oplus W_{i-1}^{j-1} \text{ si } j > 0,$$

et

$$W_i^j = SB(W_{i-4}^j) \oplus W_{i-1}^{j-1} \oplus RCON^{(j,i/4)} \text{ sinon,}$$

avec RCON la matrice suivante :

$$\begin{pmatrix} 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1b & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{pmatrix}$$

On définit

$$K_i = W_{4.i}W_{4.i+1}W_{4.i+2}W_{4.i+3}$$

où K_i est la clé de la i-ème ronde.

L'algorithme de l'AES

Maintenant que l'on connaît chaque fonction de l'AES, on peut décrire l'algorithme dans son ensemble. Soit M le message à chiffrer et K la clé. L'entrée de chaque ronde est l'état à la fin de la ronde précédente.

Ronde 0

Lors de cette ronde, on fait AddRoundKey(M,K).

FIGURE 1.4 – Première Ronde

Ronde 1 à antépénultième

Soit C_n l'état à l'entrée de la n-ème ronde. On effectue les opérations suivantes :

$$X_n = SubBytes(C_n)$$

$$Y_n = ShiftRow(X_n)$$

$$Z_n = MixColumns(Y_n)$$
$$C_{n+1} = AddRoundKey(Z_n, K_n)$$

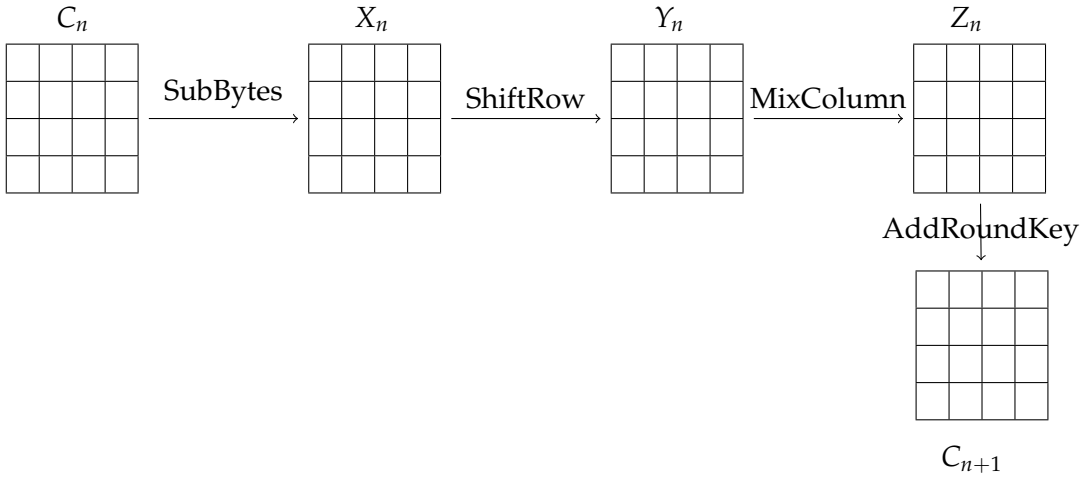


FIGURE 1.5 – Ronde 1 à 8

Dernière ronde

La dernière ronde est identique à la précédente, mais sans le MixColumns (Figure 1.6)

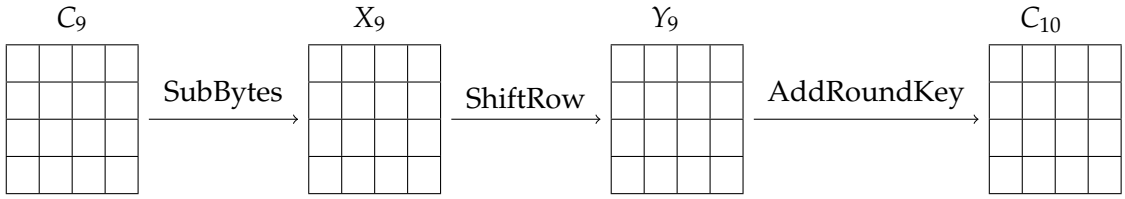


FIGURE 1.6 – Dernière Ronde

Dans la suite de ce manuscrit, nous nous intéresserons à la cryptanalyse physique et plus particulièrement aux attaques par observations. Nous présenterons des résultats liés aux deux algorithmes symétriques que nous avons présentés précédemment. Il est néanmoins possible d'étendre ce qui va suivre à d'autres algorithmes, y compris des algorithmes à clé publique comme le RSA.

Chapitre 2

Attaques par observations

DANS le chapitre précédent, nous avons soulevé le problème de la gestion des clés. Le développement des cartes à puce a permis de proposer une solution à ce problème. En effet, elles disposent d'une protection naturelle contre la lecture des données qu'elles contiennent. De nombreuses attaques ont été développées pour accéder à ces données de manière illicite. Le cadre de cette thèse a été les attaques par observations qui sont des attaques non-invasives permettant de retrouver des données secrètes manipulées par un composant comme une carte à puce. Dans ce chapitre, nous présenterons l'attaque originale proposée par Kocher [KJJ99] puis les différentes améliorations proposées au cours des dernières années.

2.1 Introduction

Jusqu'au milieu des années 90, la sécurité des algorithmes cryptographiques implantés sur un composant était considérée comme équivalente à celle de l'algorithme. Un composant était alors étudié comme une boite noire. Or, la façon dont est implémenté un algorithme est très importante. En 1996, Paul Kocher a introduit les premières attaques par observations [KJJ99] et a ainsi montré le rôle primordial de l'implémentation dans la sécurité d'un composant. L'idée de Kocher était d'utiliser la consommation électrique d'une carte à puce en vue de retrouver des informations secrètes. De nos jours, on utilise aussi d'autres émanations du composant, comme le rayonnement électromagnétique [AARR02, GMO01, QS01]. Dans la suite, ces émanations seront appelées *fuites*.

La Figure 2.1 représente la consommation électrique d'un composant lorsque l'on utilise le *DES* pour chiffrer une donnée. On peut y observer 16 motifs qui semblent beaucoup se ressembler et qui représentent les 16 itérations composant le DES. La Figure 2.2 est issue du DPAContestV2 [Par10] et représente la consommation électrique du composant lorsqu'il chiffre des données grâce à un AES. On observe ici 10 motifs qui sont proches et représentent les 10 rondes de l'AES.

La mesure des émanations d'un composant peut permettre de reconnaître quel algorithme est utilisé par le composant. Ces émanations sont liées aux opérations que le composant effectue mais aussi aux données qu'il manipule.

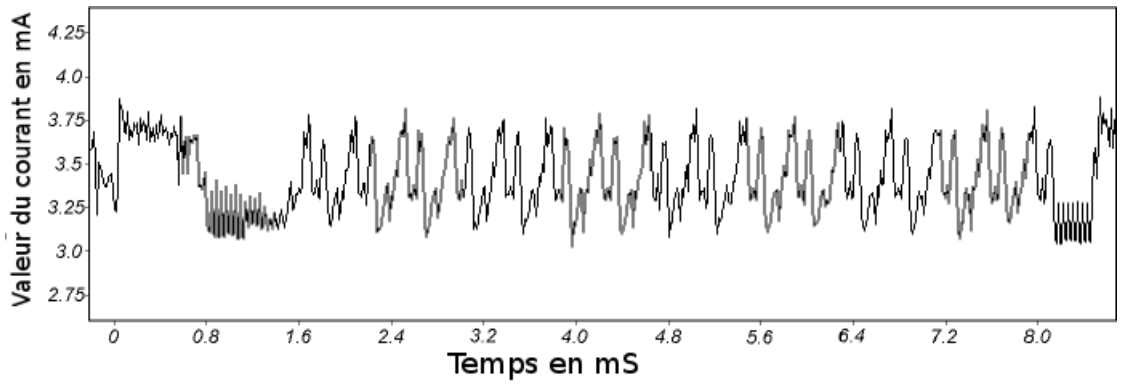


FIGURE 2.1 – Consommation électrique en mA d'un composant lors d'un chiffrement DES, extrait de l'article [KJJ99].

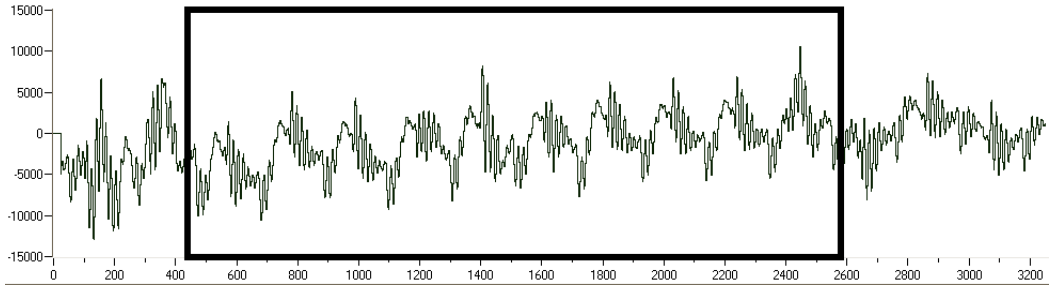


FIGURE 2.2 – Consommation électrique d'un composant lors d'un chiffrement AES [Par10].

2.2 Attaques par simple observation

Une carte à puce ne peut pas manipuler la clé secrète toute entière car elle est bien trop grande. Actuellement, la plupart des processeurs embarqués sur les cartes à puce permet de manipuler des objets codés sur 8, 16 ou 32 bits. Cette particularité, liée au composant, permet à un attaquant d'utiliser la méthode *Diviser pour mieux régner*, c'est-à-dire, dans notre cas, essayer de retrouver la clé par morceaux. Si l'on prend l'exemple d'un algorithme de chiffrement symétrique standard comme l'AES, les clés utilisées à chacune des 10 rondes sont composées de 128 bits, et il n'est pas possible d'énumérer les 2^{128} clés possibles. En revanche, si l'on divise ces clés en 16 morceaux de 8 bits chacun, il n'y a plus que $16 \times 2^8 = 4\,096$ clés à énumérer ce qui ne pose plus aucun problème.

La façon dont sont implémentés les algorithmes peut offrir à un attaquant la possibilité d'utiliser la méthode *Diviser pour mieux régner*. Ainsi, Kocher décrit la *Simple Power Analysis* (SPA), une méthode permettant de retrouver bit à bit la clé utilisée dans un algorithme RSA.

Le problème est que lorsque que l'on va chercher à implémenter un algorithme RSA, on va vouloir faire en sorte qu'il soit le plus efficace possible et pour cela, essayer d'optimiser le calcul de l'exponentiation modulaire. Dans le cas de Kocher, cette opération était implémentée en utilisant l'algorithme dit exponentiation binaire *Left-to-Right* [Knu81]. Dans cet algorithme, on commence par écrire e sous sa représentation binaire :

$$e = \sum_{i=l-1}^{0} a_i 2^i.$$

Dans cette représentation, on sait que $a_{l-1} = 1$ et que pour tout $i \geq l$, $a_i = 0$. Le calcul de M^e peut être donné par :

$$M^e = \prod_{i=0}^{l-1} \left(M^{2^i}\right)^{a_i} = (((\ldots(M^{a_{l-1}})^2 \times M^{a_{l-2}})^2 \ldots \times M^{a_i})^2 \ldots \times M^{a_1})^2 \times M^{a_0}.$$

L'algorithme d'exponentiation modulaire qui en résulte est décrit dans l'Algorithme 1.

Dans cette algorithme on observe que lorsque qu'un bit de clé est nul, on a un calcul de moins à effectuer que pour un bit de clé valant 1. Si l'émanation du composant est différente entre l'opération *multiplication* et l'opération *mise au carré* on sera capable de distinguer si le composant est en train de traiter un bit nul ou un bit non nul. La Figure 2.3 illustre ce phénomène : elle montre les résultats qu'avait obtenus Kocher en étudiant les fuites d'un algorithme RSA implémenté sous cette forme. Il est aisé, même à l'œil nu, de lire la clé utilisée par le composant, bit à bit. En particulier, on pourra retrouver un motif commun dans la consommation électrique du composant lorsque l'algorithme effectue une mise au carré et un autre motif pour la multiplication. Comme l'Algorithme 1 le laissait présager, pour traiter un bit valant 1, le composant a besoin de plus de temps car il a deux opérations à effectuer alors qu'il n'en fait qu'une pour un bit à 0, c'est pour cela que l'on observe un second pic suivant le motif commun lorsque le bit de clé manipulé vaut 1.

Il est possible de se prémunir de cette analyse en ajoutant des *calculs fantômes* [CJ01] dans les algorithmes afin que la carte à puce effectue dans tous les cas le même nombre de cycles. Néanmoins on peut adapter la SPA pour obtenir des attaques efficaces même sur des implémentations bénéficiant de ce genre de protection.

La SPA est l'une des attaques par canaux cachés la plus simple et met en exergue l'existence d'un lien entre les émanations du composant à un instant donné et les opérations effectuées à cet instant.

algorithme 1 Algorithme de l'exponentiation binaire *Left-to-Right*

1: **Algorithme** POWMOD(M, e, n)
2: $C = M$
3: **pour** $i \in \{l-2, \cdots, 0\}$ **faire**
4: $C = C \times C$ mod n ▷ *Mise au carré.*
5: **si** $a_i = 1$ **alors**
6: $C = C \times M$ mod n ▷ *Multiplication.*
7: **fin si**
8: **fin pour**
9: **retourner** C
10: **fin Algorithme**

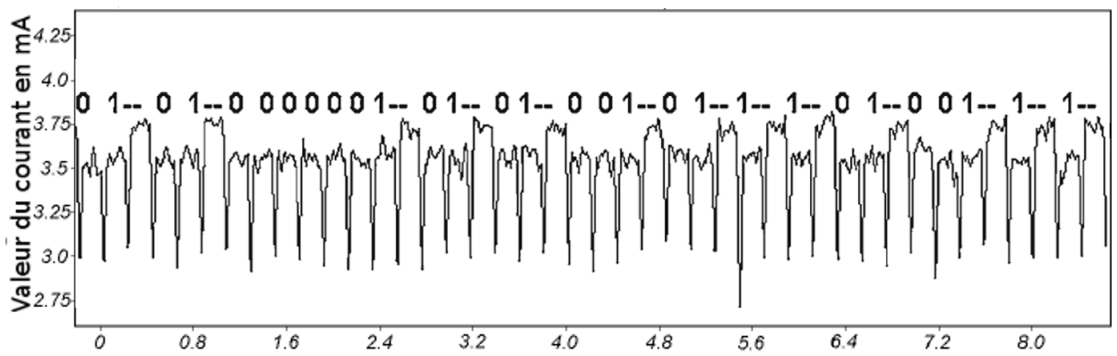


FIGURE 2.3 – Exemple d'une attaque SPA sur l'algorithme d'exponentiation binaire. En abscisse le temps en ms et en ordonnée la consommation électrique en mA [KJJ99]

2.3 Attaques statistiques par observations

La seconde attaque proposée par Kocher, appelée *Differential Power Analysis* (DPA), est plus complexe et fait appel à des notions de statistique. Cette attaque permet d'obtenir des résultats même lorsque la SPA ne permet pas d'identifier d'informations dans les émanations du composant que l'on cible. L'idée est non plus de regarder une unique courbe à l'œil nu, mais plusieurs et d'en retirer de l'information grâce à des méthodes statistiques.

De même que pour la SPA, cette attaque vise une opération intermédiaire de l'algorithme qui n'implique qu'une petite partie de la clé secrète. Pour cette analyse, l'attaquant va procéder en plusieurs phases. Dans un premier temps, il va demander au composant visé d'effectuer des opérations cryptographiques et enregistrer les émanations qui en résultent. Parallèlement, il construit pour chaque valeur de clés possibles un modèle représentant la fuite supposée en sortie d'une fonction intermédiaire choisie par l'attaquant. Dans [KJJ99], Kocher propose d'utiliser un modèle monobit en sortie des boites-S du DES. Enfin, l'attaquant étudie les relations possibles entre les différents modèles et les émanations issues du composant. Dans le cas de Kocher, il suffit de faire la différence entre deux moyennes comme nous le présenterons dans la Section 2.3.2. La fuite étant liée à la clé secrète, le modèle qui sera le plus lié aux émanations sera celui obtenu avec la clé secrète.

Dans cette section nous allons présenter les différentes méthodes statistiques mises en œuvre pour étudier le lien entre l'émanation et les modèles. Pour cela, il est important d'introduire quelques notations avant de poursuivre notre discussion.

2.3.1 Notations

On notera $g : \mathcal{X} \times \mathcal{K} \rightarrow \mathcal{Z}$ la fonction intermédiaire visée par l'attaque. $\mathcal{K} = \{0, \cdots, \mathbf{k}-1\}$ est l'ensemble des valeurs possibles de la clé et $k^\star$ est la valeur de la clé secrète. $\mathcal{X} = \{0, \cdots, \mathbf{x}-1\}$ est l'ensemble des entrées possibles de g et $\mathcal{Z} = \{0, \cdots, \mathbf{z}-1\}$ celui de la sortie de g. Soit X une variable aléatoire qui représente l'entrée de la fonction intermédiaire, et Z la variable aléatoire représentant le résultat de cette fonction.

La fonction intermédiaire g est connue par l'attaquant, déterministe, et dépend d'un paramètre $k \in \mathcal{K}$. Elle peut être très diverse : une permutation, une addition modulo 256, une fonction non-linéaire, etc. Finalement, on notera $L(Z) = L(g(X,k)) = \Phi(X,k)$, la variable aléatoire représentant la fuite générée par le calcul de Z. La fonction Φ est inconnue et non-déterministe, car elle représente une mesure physique.

Le but d'une attaque par observations est d'estimer la clé secrète $k^\star$. On dispose pour cela de n réalisations notées $l_i = \phi(x_i, k^\star)$, $i = 1, \cdots, n$ de $L(Z) = \Phi(X, k^\star)$. On considère que ces réalisations proviennent de variables aléatoires indépendantes. Ces réalisations sont obtenues à partir de mesures physiques fortement bruitées. D'autre part, on modélise par la fonction $m(X,k)$, la fuite pour une clé $k \in \mathcal{K}$. Notons que les valeurs de la fonction $m(X,k)$ sont discrètes. Plus cette modélisation sera proche de Φ, meilleure sera notre estimation de $k^\star$.

Lors d'une attaque par observation, on doit choisir une fonction qui représente la manière dont fuit le composant en fonction des variables manipulées. On notera cette fonction φ.

Les bits des variables manipulées sont stockés dans les composants grâce à la présence ou non de courant dans les transistors. Lorsque l'on change la valeur d'une variable, ces transistors se déchargent ou se chargent afin de stocker la valeur voulue. Le *poids de Hamming* (Π) est une fonction qui prend en entrée une suite de bits et qui renvoie

le nombre de bits non-nuls, c'est-à-dire le nombre de transistors à charger pour stocker la suite de bits. Dans le cadre d'une attaque par observations, la fuite est souvent liée au poids de Hamming des données manipulées en raison de la façon dont elles sont stockées, c'est pourquoi ce modèle de fuite est tant utilisé. La variable $m(X,k)$ est calculée à partir de la variable X, de l'algorithme et de la manière dont les données manipulées laisseront fuir de l'information φ. Par exemple, si l'on considère que le modèle de fuite est le poids de Hamming, on aura $m(X,k) = \varphi(g(X,k)) = \Pi(g(X,k))$. On notera $\mathcal{M}$ l'ensemble des valeurs possibles du modèle.

On dispose de n réalisations $m(x_i,k)$ pour différentes clés à tester dans un ensemble $\mathcal{K}$. À partir de ces données, l'attaquant va chercher à déterminer une mesure de *dépendance* entre les variables $L(Z)$ et $m(X,k)$. On retiendra la (ou les) clé(s) qui maximise(nt) cette mesure de *dépendance* comme une estimation de $k^\star$. Les attaques par observations visent à mesurer la dépendance entre une variable continue, les émanations d'un composant lors de calcul cryptographique et une variable discrète, un modèle de fuite pour chaque sous clé possible.

2.3.2 Différence de moyennes

L'attaque originelle proposée par Kocher dans [KJJ99] est basée sur la différence de moyennes. Kocher propose d'attaquer la valeur d'un bit de la sortie d'une boite-S du DES lors de la première ronde. Ainsi, les variables aléatoires $m(X,k), k \in \mathcal{K}$, n'ont ici que deux valeurs possibles, 0 et 1. L'attaquant souhaite estimer la différence $E[L|m(X,k) = 1] - E[L|m(X,k) = 0]$. Pour cela, il calcule :

$$\hat{\Delta}_k = \frac{\sum_{i:m(x_i,k)=1} l_i}{\#\{i : m(x_i,k) = 1\}} - \frac{\sum_{i:m(x_i,k)=0} l_i}{\#\{i : m(x_i,k) = 0\}} = \frac{\sum_{i=1}^{n} m(x_i,k) l_i}{\sum_{i=1}^{n} m(x_i,k)} - \frac{\sum_{i=1}^{n}(1 - m(x_i,k) l_i)}{\sum_{i=1}^{n}(1 - m(x_i,k))}.$$

Lorsque $m(X,k)$ et $L(Z)$ sont indépendantes, $E[L|m(X,k) = 1] = E[L|m(X,k) = 0]$. Lorsque Δ_k est proche de 0, $m(X,k)$ et $L(Z)$ sont considérées comme étant "indépendantes". En revanche, lorsque la valeur de Δ_k est élevée, il est probable qu'il existe une relation de dépendance entre les deux variables qui sont considérées. Ainsi, une estimation de la bonne clé sera donnée par :

$$\hat{k^*} = \underset{k \in \mathcal{K}}{argmax}(|\hat{\Delta}_k|).$$

La variable aléatoire modélisant la fuite ne peut prendre ici que deux valeurs, ce qui est extrêmement restrictif. En effet, les données calculées sont le plus souvent constituées de plusieurs bits. Le modèle proposé est donc très éloigné de Φ. L'une des premières améliorations proposées vise à permettre un plus vaste choix pour les valeurs du modèle. De plus, le nombre de réalisations nécessaires pour pouvoir découvrir une relation entre le modèle correspondant à la bonne clé $m(X,k^\star)$ et la fuite $L(Z)$ est important dans le contexte des attaques par canaux cachés. Enfin, la recherche académique concernant les attaques par observations, a été longtemps axée autour de la généricité des relations entre les deux variables aléatoires considérées (linéaire, d'ordre, fonctionnelle, etc.) [1]. Les mesures de dépendance proposées pour améliorer l'attaque de Kocher peuvent être divisées en deux grandes familles. La première est la détection de structure de corrélation entre les observations ; la seconde est basée sur les lois de probabilités sous-jacentes aux observations.

1. Ici la généricité est le fait d'être capable de retrouver la clé secrète avec le moins d'information possible quant à la forme de la fuite du composant

2.3.3 Méthodes basées sur la corrélation

2.3.3.1 Correlation Power Analysis : corrélation linéaire estimée grâce au *coefficient de Pearson*

Lorsque l'on parle de *dépendance* entre deux variables aléatoires, la première mesure qui vient à l'esprit est celle donnée par le coefficient de corrélation linéaire entre $m(x_\cdot, k)$ et $l_\cdot$. Ce coefficient est défini par :

$$\rho_k = \frac{\frac{1}{n}\sum_{i=1}^{n}(l_i - \bar{l})(m(x_i,k) - \bar{m_k})}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(l_i - \bar{l})^2 \frac{1}{n}\sum_{i=1}^{n}(m(x_i,k) - \bar{m_k})^2}}. \tag{2.1}$$

où $\bar{l}$ est la moyenne empirique associée aux l_i et $\bar{m_k}$ celle associée aux $m(x_i, k)$. $\rho_k = 1$ ou $\rho_k = -1$ suggère une relation fonctionnelle linéaire forte entre le modèle considéré et la fuite. ρ_k^2 mesure la proximité des n points à la droite de régression dont le signe de la pente est donné par le signe de ρ_k. La bonne clé sera donc estimée comme étant la clé pour laquelle on aura, en valeur absolue, le *coefficient de Pearson* le plus élevé.

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(|\rho_k|).$$

La *Correlation Power Analysis* (CPA) [BCO04] est actuellement l'attaque par observations la plus populaire. Cette popularité est principalement due au fait que les modèles utilisés le plus couramment sont souvent effectivement proches de la fuite à une fonction linéaire près. Comme montré dans l'article [LCC+06], l'utilisation de la CPA avec un modèle monobit revient à la DPA de Kocher. Elle est également très résistante au bruit présent dans les réalisations de la variable $L(Z)$ que l'on utilise. En effet, le bruit aura tendance à être moyenné par le calcul du coefficient de Pearson. Ainsi, si l'on augmente le nombre de réalisations dont on dispose, on va diminuer de plus en plus la contribution du bruit. Néanmoins, afin de prévenir cette attaque, les fabricants de cartes à puce essayent de mettre en œuvre des contre-mesures visant à casser cette linéarité. Il est donc important pour un attaquant d'avoir à sa disposition d'autres tests statistiques lui permettant d'identifier des relations entre $m(X, k)$ et $L(Z)$ même lorsque ces relations ne sont pas linéaires.

2.3.3.2 Corrélation des rangs de Spearman

Outre le fait que la relation entre la fuite et le modèle correspondant à la bonne clé peut exister sans être linéaire, il est important de noter la nature des variables aléatoires considérées. $L(Z)$ est continue alors que $m(X, k)$ est discrète. Bien sûr, $m(X, k)$ est discrète car le modèle que l'on considère n'est pas assez précis. Néanmoins, d'une manière générale, le coefficient de corrélation linéaire n'est guère recommandé pour comparer des variables de natures différentes. L'utilisation de la corrélation des rangs semble être une bonne alternative car elle permet d'étudier les relations entre une variable discrète et une variable continue, mais également de détecter non seulement les relations linéaires, mais aussi les relations d'ordre. Le coefficient de corrélation des rangs ou *coefficient de Spearman*, est calculé en remplaçant dans l'Équation (2.1) l_i et $m(x_i, k)$ par leur rang. Le rang d'une observation est donné par sa position lorsque l'on a trié par ordre croissant toutes les observations. Si l'on considère les données $(3, 7), (2, 9), (7, 10)$ les rangs associés seront : $(2, 1), (1, 2), (3, 3)$. Lorsque plusieurs observations ont la même valeur, on remplace cette valeur par leur rang moyen. Le *coefficient de Spearman* s'interpréte de la même façon que le *coefficient de Pearson*. Une proximité de 1 traduit une relation positive

(variation dans le même sens) entre les deux variables considérées, alors qu'une proximité de -1 traduit une relation négative (variation en sens inverse). Enfin une proximité de 0 implique une indépendance entre les variables.

La bonne clé sera donc estimée, comme précédemment, comme étant la clé pour laquelle le *coefficient de Spearman* [BGLR08] est le plus élevé en valeur absolue.

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(|r_k|).$$

La corrélation des rangs ne repose pas sur l'hypothèse d'une relation linéaire entre les données mais simplement sur une relation d'ordre. De plus, elle permet de comparer deux variables aléatoires qui ne sont pas de même nature. Malheureusement, en présence de nombreux ex æquo, la corrélation des rangs n'est pas toujours très adaptée. Or, dans les attaques par observations, que ce soit pour la variable représentant le modèle ou pour la variable représentant la fuite, les ex æquo sont très nombreux.

2.3.3.3 *Tau de Kendall*

Le *tau de Kendall* [Ken38] permet de déceler des relations d'ordre entre deux variables aléatoires, y compris de nature différente. Le tau de Kendall est la différence estimée entre la probabilité que les données observées aient le même rang pour les deux variables et la probabilité que les données observées aient des rangs différents. Le principe de ce tau est différent des précédents. On commence par trier une des deux séries d'observations et les valeurs des rangs de la seconde série sont mises en regard de la première. Une fois la première série triée, on ne s'intéresse plus qu'à la seconde. On remplace la valeur de chaque observation par le nombre de valeurs suivantes qui lui sont supérieures, auquel on retranche le nombre de valeurs suivantes qui lui sont inférieures.

Si l'on considère par exemple les couples suivant $(9,4),(0,3),(1,1),(2,2)$, on commence par ordonner ces couples suivant la première série et l'on obtient $(0,3),(1,1),(2,2),(9,4)$. Maintenant, on ne regarde plus que la seconde série. On attribue à 3 le score de $1-2=-1$, à 1 celui de $2-0=2$, à 2 celui de $1-0=1$ et enfin à 4 le score de $0-0=0$. On obtient ainsi une nouvelle série $-1,2,1,0$. La somme S des valeurs de cette série nous donne des indications sur les relations entre les deux variables considérées au départ. En effet, si $S = n(n-1)/2$, S sera la somme des n premiers entiers naturels. On en déduit que l'ordre est totalement respecté. De même si $S = -n(n-1)/2$, l'ordre est parfaitement inversé. Si en revanche S est nulle, on aura indépendance entre les deux variables étudiées.

Partant de S, on définit le *tau de Kendall* comme une mesure d'indépendance. Il s'exprime de la manière suivante :

$$\tau = \frac{2S}{n(n-1)}.$$

Le *tau de Kendall* est compris entre -1 et 1 et s'interprète de la même manière que le coefficient des rangs ou celui de Pearson. Plus il est proche de 1, plus on est certain qu'il existe une corrélation positive entre nos données ; plus il est proche de -1, plus on peut supposer qu'il existe une corrélation négative. Lorsque le *tau de Kendall* est proche de zéro, l'existence d'une liaison monotone entre nos données sera peu probable.

La clé secrète utilisée par le composant sera donc estimée par [BGLR08] :

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(|\tau_k|).$$

Le *tau de Kendall* et le *coefficient de Spearman* permettent de détecter un grand nombre de relation. Néanmoins, lorsque l'on est en présence de nombreux ex æquo, il est recommandé d'utiliser le coefficient gamma.

2.3.3.4 *Coefficient gamma*

Le *coefficient gamma* [BGLR08], comme le *tau de Kendall*, permet de détecter des relations ordinales entre deux variables. Il se calcule en étudiant la concordance ou la discordance entre deux paires. On dit qu'une paire, (U_1, V_1) et (U_2, V_2), est concordante lorsque $U_2 - U_1$ et $V_2 - V_1$ sont de même signe. Respectivement, on dira qu'une paire et discordante lorsque les deux différences n'ont pas le même signe. Si les deux différences sont nulles, la paire est considérée comme concordante. Si l'une est nulle mais pas l'autre, la paire n'est ni concordante ni discordante. On note N_c le nombre de paires concordantes dans nos observations et N_d le nombre de paires discordantes. Il est intéressant de noter que, si certaines paires ne sont ni concordantes ni discordantes, $n \neq N_c + N_d$.

Le coefficient gamma est calculé de la manière suivante :

$$\Gamma = \frac{N_c + N_d}{N_c - N_d}.$$

Lorsque le coefficient gamma sera proche de 0, on considérera que les deux variables sont indépendantes. En revanche, plus la valeur de ce coefficient sera proche de 1 ou de -1 plus la relation entre les variables sera forte. La clé secrète sera donc estimée par :

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(|\Gamma_k|).$$

Le *coefficient de Spearman*, le *tau de Kendall* et le *coefficient gamma* permettent de détecter des relations ordinales entre les variables. Néanmoins, ils ne permettent pas de détecter des relations fonctionnelles qui ne sont pas des relations d'ordre (par exemple : une relation elliptique, sinusoïdale ou encore des combinaisons de relations, ect). Dans la Section 2.3.4, nous allons présenter d'autres méthodes permettant de détecter de nombreuses relations y compris non-ordinales.

2.3.4 Méthodes basées sur des lois de probabilité

2.3.4.1 Mutual Information Analysis (MIA) : mesure des dépendances existantes entre deux variables grâce à l'*information mutuelle*

Avant de présenter l'*information mutuelle* [GBTP08], il est nécessaire de faire quelques rappels concernant la théorie de l'information.

L'entropie de Shannon est une manière de mesurer la quantité d'information contenue dans une variable aléatoire. On note $\mathbb{P}[E]$ la probabilité de l'événement E. L'entropie d'une variable aléatoire discrète U dans l'espace $\mathcal{U}$ est définie par :

$$H(U) = -\sum_{u \in \mathcal{U}} \mathbb{P}[U = u] log_2(\mathbb{P}[U = u]).$$

L'entropie d'une variable aléatoire est définie de manière similaire. L'entropie est maximale lorsque la loi de la variable est uniforme. L'*information mutuelle* entre deux variables aléatoires U et V est définie de la manière suivante :

$$I(U; V) = H(U) + H(V) - H(U, V).$$

Dans le cadre des attaques par canaux cachés, on cherchera à estimer l'information mutuelle entre la variable aléatoire continue $m(X,k)$ et la variable continue $L(Z)$. Afin d'alléger les notations, nous noterons $L(Z) = L$ et $m(X,k) = M_k$. L'information mutuelle entre nos deux variables sera définie par :

$$I_k(L;M_k) = \sum_{m\in\mathcal{M}} \mathbb{P}[M_k = m] \int_{l\in\mathcal{L}} f_{L|M_k=m}(l) log_2 \left(\frac{f_{L|M_k=m}(l)}{f_L(l)}\right) dl.$$

où f_L est la densité de probabilité de $L(Z)$ et $f_{L|M_k=m}$ est la densité de probabilité conditionnelle de $L(Z)$ sachant $m(X,k) = m$. La quantité $I_k(L;M_k)$ est toujours positive ou nulle. Si $I_k(L;M_k)$ est nulle, $L(Z)$ et $m(X,k)$ sont indépendantes. La valeur de l'*information mutuelle* est toujours inférieure à l'entropie de chacune des variables étudiées avec égalité si, et seulement si, l'une des variables est une fonction déterministe de l'autre. Plus l'information mutuelle sera forte, plus $L(Z)$ et $m(X,k)$ seront liées.

En pratique, pour une attaque par observations, on cherchera $k \in \mathcal{K}$ qui maximisera l'information mutuelle $I_k(L;M_k)$. La clé secrète sera donc estimée par :

$$\hat{k^\star} = \underset{k\in\mathcal{K}}{argmax}(I_k(L;M_k)).$$

Il est en général difficile de calculer l'*information mutuelle* entre $L(Z)$ et $m(X,k)$ car la loi de la variable $L(Z)$ n'est pas connue. On peut aussi remarquer que :

$$\int_{l\in\mathcal{L}} f_{L|M_k=m}(l) log_2 \left(\frac{f_{L|M_k=m}(l)}{f_L(l)}\right) dl = E_{L|M_k=m}\left[log_2\left(\frac{f_{L|M_k=m}(L)}{f_L(L)}\right)\right].$$

où $E_{L|M_k=m}$ signifie que l'espérance est calculée sous la loi de $L(Z)$ sachant que $m(X,k) = m$. Cette espérance peut être estimée par :

$$\sum_{l_i\in L(Z);m(x_i,k)=m} log_2\left(\frac{\hat{f}_{L|M_k=m}(l_i)}{\hat{f}_L(l_i)}\right).$$

où $\hat{f}_{L|M_k=m}$ et $\hat{f}_L$ sont des estimateurs de $f_{L|M_k=m}$ et f_L.

Le choix des estimateurs $\hat{f}_{L|M_k=m}$ et $\hat{f}_L$ est crucial. Dans la suite, nous allons présenter les estimateurs les plus utilisés dans les attaques par canaux cachés. Dans le cadre des attaques par observations, on utilise le plus souvent des approches non paramétriques. Ainsi les estimateurs les plus utilisés sont les estimateurs à base de noyaux. Afin de présenter cette méthode, on se place dans un contexte simple où l'on a accès à n observations d'une variable aléatoire continue U. On notera $u_i, i \in \{1,\cdots n\}$ ces observations. Un estimateur de la densité de probabilité associée à U est donné par :

$$\hat{f}_U(u) = \frac{1}{nh}\sum_i^n K(\frac{u-u_i}{h}).$$

où K est un noyau et h une fenêtre qui doit être choisie avec soin. La valeur optimale de h est celle qui minimise l'AMISE (Asymptotic Mean Integrated Squared Error) qui dépend de la variable aléatoire dont on souhaite estimer la densité de probabilité. Le noyau, quant à lui, est une fonction réelle telle que $\int_{-\infty}^{\infty} K(u)du = 1$ et $K(u) = K(-u)$. Il existe de nombreux noyaux. Le choix du noyau dépend essentiellement de la forme de la variable dont on souhaite estimer la densité de probabilité. Dans le cadre d'une attaque par observations, on utilise très souvent le noyau gaussien [BGP+11] défini par :

$$K(u) = \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2}u^2).$$

Dans [VCcXS09] les auteurs proposent d'utiliser comme largeur de fenêtre :

$$h = 1.06 \ min\left(\hat{\sigma}, \frac{IQR}{1.34}\right) \ n^{-\frac{1}{5}},$$

où $\hat{\sigma}(U)$ est l'écart type empirique et $IQR(U)$ représente l'écart interquartile empirique. [2] Un autre estimateur très populaire dans le cadre des attaques par observations est basé sur les histogrammes. Le noyau permettant d'obtenir cette estimateur est donné par :

$$K(u) = \mathbb{1}_{]-\frac{1}{2};\frac{1}{2}[}(u)$$

où $\mathbb{1}$ est la fonction indicatrice. [3] Dans [GBTP08], les auteurs proposent d'utiliser simplement un histogramme en découpant les ensembles $L(\mathcal{Z})$ et $m(\mathcal{X}, k)$ en classes de même taille et dont le nombre sera égal au nombre de valeurs différentes possibles pour $m(X, k)$. D'autres estimateurs ont été proposés, comme dans [Ven10] où l'auteur a étudié les résultats obtenus grâce aux B-splines. L'auteur de [Ven10] montre que l'utilisation de B-splines permet d'avoir une bonne estimation des densités de probabilité y compris lorsque les données sont bruitées.

L'*information mutuelle* permet d'obtenir des attaques extrêmement puissantes car aucune supposition sur la relation entre $m(X, k^\star)$ et $L(Z)$ n'est nécessaire. Néanmoins, en pratique, ces attaques sont difficiles à mettre en œuvre car il faut bien choisir les estimateurs utilisés ainsi que la taille de la fenêtre. La capacité à bien estimer une densité de probabilité est un facteur très important dans la réussite de telles attaques. Dans le cadre des attaques par observations, les réalisations de la variable $L(Z)$ dont on dispose sont obtenues à partir de mesures physiques et peuvent être fortement bruitées. Or l'information mutuelle n'est pas une méthode très résistante au bruit. En effet, lorsque les réalisations de la variable $L(Z)$ que l'on considère sont très bruitées, la variance de la loi $L(Z)$ va augmenter, alors que celle de la loi $m(X, k^\star)$ va rester la même. Ainsi, contrairement à la corrélation linéaire, l'information mutuelle ne moyenne pas le bruit mais y est sensible. Au sens de la théorie de l'information, l'information mutuelle est la meilleure méthode possible [PR10], mais en pratique, les données dont on dispose ne se prêtent pas très bien à cette méthode en particulier, car les réalisations de $L(Z)$ sont très bruitées.

2.3.4.2 *Statistique de Kolmogorov-Smirnov*

En considérant l'*information mutuelle*, on peut s'apercevoir qu'elle peut s'interpréter comme la moyenne (sur le modèle $m(X, k)$) des distances de Kullback-Leibler [4] entre la distribution de $L(Z)$ et la distribution de $L(Z)$ sachant $m(X, k) = m$. L'idée de l'approche de Kolmogorov-Smirnov est de remplacer cette distance par une distance plus simple (L_∞) et d'utiliser des fonctions de répartition en lieu et place des estimateurs de densité de probabilité. On notera D_k^{KS} la distance entre la fonction de répartition de $L(Z)$ et celle de $L(Z)$ sachant $m(X, k) = m$:

$$D_k^{KS} = \sum_{m \in \mathcal{M}} \mathbb{P}[M_k = m] \ \underset{l}{sup} \ |F_{L|m_k=m}(l) - F_L(l)| \tag{2.2}$$

où F_L est la fonction de répartition associée à $L(Z)$ et $F_{L|M_k=m}$ celle associée à $L(Z)$ sachant $m(X, k) = m$. Finalement, la statistique de Kolmogorov-Smirnov, $\hat{D_k^{KS}}$, est obtenue

2. Si l'on note Q_2 la médiane de nos observations, Q_1 la médiane de nos observations inférieures à Q_2 et Q_3 la médiane de nos observations supérieures à Q_2. L'écart interquartile empirique est donné par : $IQR = Q_3 - Q_1$

3. $\mathbb{1}_E(u) = 1$ si $u \in E$ et 0 sinon .

4. $d(f_1, f_2) = \int_{-\infty}^{\infty} f_1(x) ln\left(\frac{f_1(x)}{f_2(x)}\right) dx$

en estimant les fonctions de répartition par leur version empirique dans l'Équation (2.2). Lorsque les deux variables que l'on observe sont indépendantes, la valeur de la statistique de Kolmogorov-Smirnov est proche de 0. Ainsi, dans le cas où k est la bonne clé, on s'attend à ce que cette distance soit grande. La clé secrète sera estimée grâce à [WOM11] :

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(\hat{D_k^{KS}}).$$

La *statistique de Kolmogorov-Smirnov* ne nécessite pas l'estimation de densités de probabilité puisque l'on utilise la fonction de répartition empirique. Cette fonction est en fait une manière d'estimer la fonction de répartition en utilisant un histogramme avec un choix particulier de fenêtre. Il est intéressant de noter que l'on peut tout à fait substituer à la distance L_∞ une autre distance comme, par exemple, la distance L_1 ou la distance du χ^2. Ainsi, en utilisant le carré de la distance euclidienne, on obtient le critère de Cramér-von Mises. Dans [VCcXS09], les auteurs ont montré que les résultats obtenus grâce au critère de Cramér-von Mises sont très proches de ceux obtenus avec la statistique de Kolmogorov-Smirnov.

2.4 Résultats expérimentaux

Dans cette section, nous allons présenter les résultats que nous avons obtenus en appliquant les différentes méthodes introduites dans les paragraphes précédents. Pour cela, nous avons utilisé 1 000 courbes de consommation de courant en fonction du temps, proposées dans la version 1 de DPAContest [Par08]. Nous avons choisi de prendre comme fonction g les quatre bits issus de la première boite-S après le OU exclusif entre les registres R et L de la dernière ronde de l'algorithme DES [oS77] et comme fonction de fuite, φ le poids de Hamming. Comme six bits de clé interviennent dans le calcul de g, il y a 64 valeurs possibles pour la clé secrète, $k^\star$. Nous avons donc besoin de comparer les résultats des différentes méthodes pour 64 modèles différents. Les Figures 2.4 à 2.9 représentent les résultats que nous avons obtenus. Nous présentons ici uniquement la dernière des 16 rondes du DES car c'est celle durant laquelle le composant va calculer la fonction g que nous avons choisie. Chaque courbe représente le résultat obtenu pour un modèle. On constate que pour chaque méthode, une des courbes se détache. Celle-ci représente la clé secrète utilisée par le composant.

La Figure 2.4 représente la valeur du coefficient de Pearson. La puissance du coefficient semble faible même lorsque l'on considère le modèle correspondant à la bonne clé. Néanmoins, on constate que la différence entre les valeurs du coefficient de Pearson pour la bonne clé et pour les autres clés est suffisamment significative pour pouvoir conclure.

Sur les Figures 2.5, 2.6 et 2.7, nous avons présenté les résultats des autres méthodes basées sur la corrélation. Ces résultats sont très proches de ceux obtenus grâce à la corrélation linéaire, car la relation liant le modèle correspondant à la clé secrète et les données considérées est plutôt linéaire. À certains instants, la relation entre le modèle correspondant à la clé secrète et les données est positive, à d'autres elle est négative. De plus, on peut observer qu'aux instants où le coefficient de Pearson est maximal pour le modèle correspondant à la bonne clé, d'autres modèles semblent reliés de manière linéaire aux données considérées, même si cette relation est moins forte. On appelle ce phénomène *pic harmonique* et il est dû à la fonction cryptographique que l'on a choisie.

La Figure 2.8 présente les résultats obtenus grâce à l'information mutuelle que l'on a estimée avec un noyau gaussien en utilisant la fenêtre proposée dans [VCcXS09]. Ici, les résultats issus du modèle correspondant à la bonne clé se détachent davantage. De plus,

le niveau de bruit est bien plus faible et l'on n'observe pas de variations aussi importantes pour les mauvaises clés. Néanmoins, cette méthode a été bien plus compliquée à mettre en œuvre que les précédentes et un mauvais choix de noyaux et/ou de largeur de fenêtres entraîne l'absence de résultats concluants.

La Figure 2.9 présente les résultats obtenus en utilisant la statistique de Kolmogorov-Smirnov. On observe, comme pour les méthodes basées sur la corrélation, le phénomène de *pic harmonique* même s'il est bien moins prononcé. Même si l'échelle n'est pas identique, car le niveau de bruit est bien plus important, le modèle correspondant à la clé secrète ressort moins que pour l'information mutuelle.

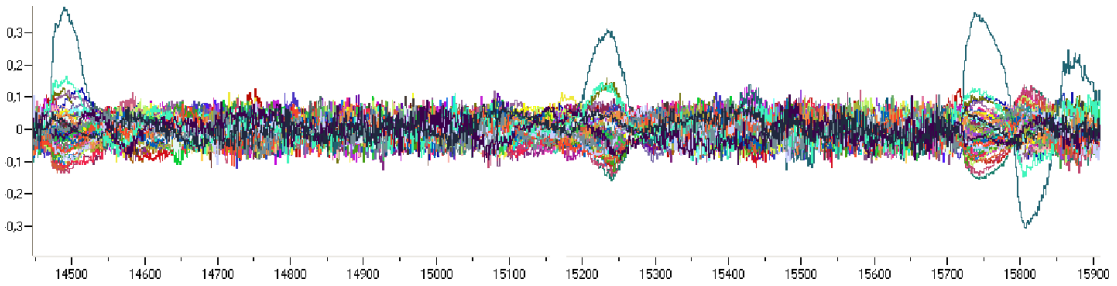


FIGURE 2.4 – En abscisse le temps en milliseconde, en ordonnée la valeur de la *corrélation linéaire*.

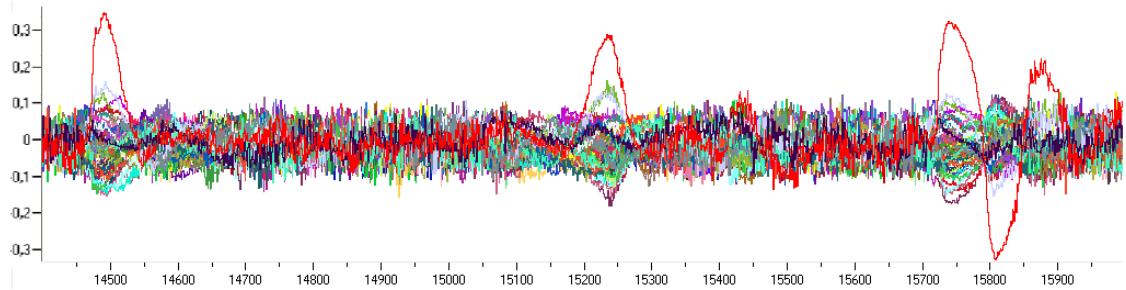


FIGURE 2.5 – En abscisse le temps en milliseconde, en ordonnée la valeur de la *corrélation des rangs*.

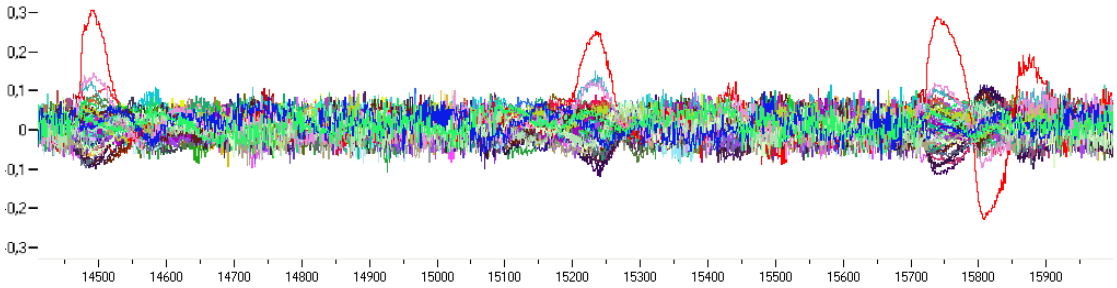


FIGURE 2.6 – En abscisse le temps en milliseconde, en ordonnée la valeur du *tau de Kendall*.

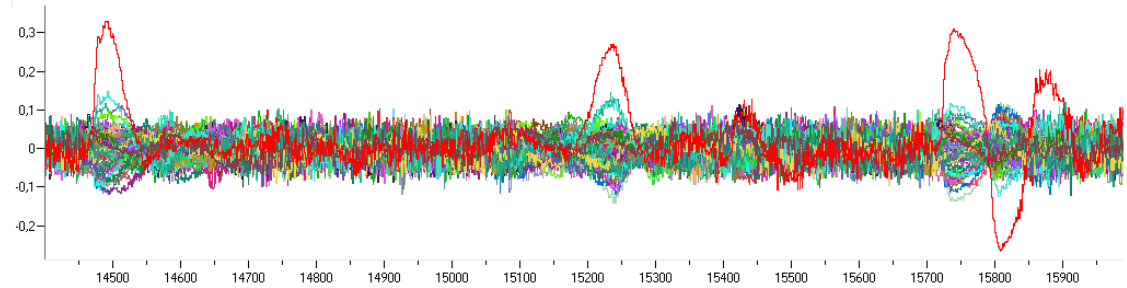


FIGURE 2.7 – En abscisse le temps en milliseconde, en ordonnée la valeur du *coefficient gamma*.

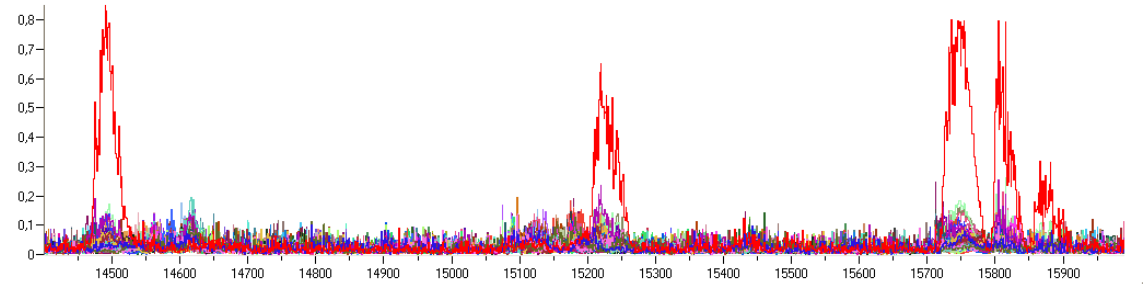


FIGURE 2.8 – En abscisse le temps en milliseconde, en ordonnée la valeur de *l'information mutuelle* obtenue grâce à un noyau gaussien et la fenêtre proposée dans [VCcXS09].

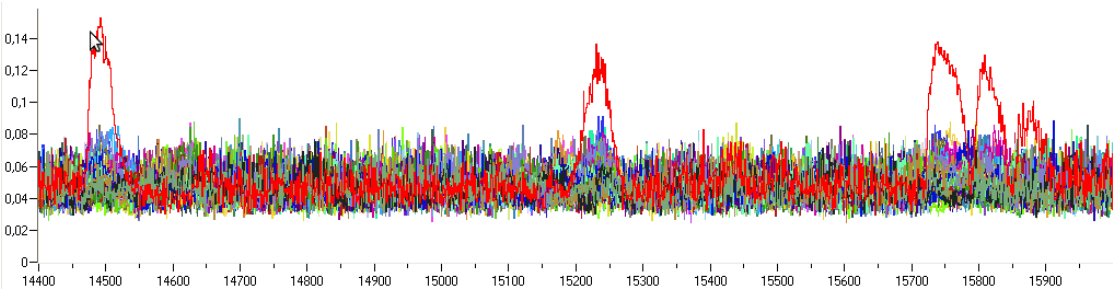


FIGURE 2.9 – En abscisse le temps en milliseconde, en ordonnée la valeur de la statistique de *Kolmogorov-Smirnov*.

2.5 Conclusion

Nous avons présenté les attaques par observations classiques. Ces attaques sont aujourd'hui couramment utilisées en cryptanalyse. Elles présentent deux étapes majeures, l'acquisition de la fuite par l'attaquant et le traitement des données obtenues. Nous avons exposé ici les différentes méthodes statistiques fréquemment utilisées pour traiter ces données. D'une manière générale, il est important d'utiliser différentes méthodes et de ne pas conclure à l'impossibilité d'une attaque après l'échec d'un seul test. En effet, chacune de ces méthodes présente des avantages et des inconvénients. À l'origine, Kocher a proposé d'utiliser la différence des moyennes. Le modèle qu'il proposait alors était très simple. Le principal atout de cette méthode venait de sa simplicité. Il est en effet très facile de calculer la différence des moyennes. En revanche l'utilisation d'un modèle monobit est très restrictif et ne sied généralement pas à des composants cryptographiques modernes. Le *coefficient de Pearson* est lui aussi facile à calculer et offre la possibilité d'utiliser des modèles de fuites plus complexes et ainsi de mieux représenter la fuite attendue. Lorsque le modèle de fuite est bien choisi, le *coefficient de Pearson* donne d'excellents résultats dans le cadre des attaques par observations. Il est intéressant de noter que la DPA avec un modèle multi-bits est équivalente à la CPA [LCC+06]. Malheureusement, il ne permet pas de détecter des relations non linéaires entre deux variables.

Le *coefficient de Spearman*, le *tau de Kendall* et le *coefficient gamma* détectent quant à eux des relations d'ordre. Bien sûr, cette plus grande capacité de détection entraîne une plus grande complexité de calcul mais elle autorise l'attaquant à être moins précis dans le choix de son modèle de faute. Dans le contexte des attaques par canaux cachés, il est très courant d'avoir un grand nombre d'ex æquo. Il est plutôt recommandé d'utiliser le *coefficient gamma*. Ces méthodes reposent sur la structure de corrélation qui peut exister entre les deux variables étudiées.

Les méthodes qui sont basées sur les lois de probabilité sous-jacentes aux variables considérées sont plus compliquées à mettre en œuvre mais peuvent se révéler très efficaces car elles permettent de détecter des relations sans a priori sur leurs formes. L'*information mutuelle* est la première de ces méthodes à avoir été utilisée dans le cadre des attaques par observations. Même si elle peut se montrer très efficace, sa mise en pratique est souvent difficile. Le *test de Kolmogorov-Smirnov* permet, en changeant la distance utilisée, d'avoir une très large famille de tests permettant de détecter des relations entre les variables considérées. La statistique de Kolmogorov-Smirnov est plus facile à mettre en œuvre que l'information mutuelle car elle repose sur la fonction de répartition estimée de manière empirique. Il n'y a donc pas de choix d'estimateur à faire.

Le *coefficient de Pearson* reste aujourd'hui le test principalement utilisé dans les attaques par observations car le choix du modèle de fuite, fait par l'attaquant, est souvent judicieux. Néanmoins si ce modèle de fuite est mal choisi, le *coefficient de Pearson* peut être mis en défaut.

Dans la suite, nous allons présenter des implémentations parallèles de ces attaques par observations.

Chapitre 3

Implémentations parallèles des attaques par observations classiques

Les contre-mesures, comme l'ajout de bruit, mises en œuvre par les fabricants de cartes à puce augmentent le nombre de traces requis pour rendre les attaques par observations effectives. Ainsi en 2005 on considérait déjà des attaques avec un million de traces [MPO05] et en 2011 il était conseillé de tenir compte d'au moins cent millions de traces [MPL+11]. La taille des signaux elle aussi est de plus en plus importante. Les courbes proposées pour la quatrième version de DPAContest sont composées de 435 002 points et ne représentent que la première ronde d'un AES logiciel [Par13]. Cet accroissement est dû principalement au fait que les appareils de mesure sont de plus en plus performant. Enfin dans le cadre des attaques par observations visant un algorithme asymétrique, on retrouve la clé morceau par morceau. Comme la taille des clés augmente, la quantité de calcul à effectuer afin de retrouver la clé secrète augmente. Il est donc important d'améliorer le temps nécessaire à la réalisation des attaques par observations.

Les attaques par observations sont le plus souvent composées de calculs simples mais que l'on effectue sur un grand nombre de données. En effet, un attaquant doit évaluer les relations possibles entre différents modèles (un pour chaque clé possible) et les fuites qu'il a obtenues en étudiant le composant. En outre, cette évaluation doit se faire à chaque instant indépendamment les uns des autres. Afin de mener à bien une attaque par observations, l'attaquant va donc devoir appliquer un même algorithme à différentes données ce qui correspond bien au modèle requis pour une implémentation efficace sur *General Programming on Graphic Processing Units* (GPGPU).

Nous présenterons, dans un premier temps les GPGPU, puis une *Application Programming Interface* (API) permettant l'utilisation de GPGPU, l'OpenCL. Nous expliquerons ensuite comment obtenir des implémentations sur GPGPU d'attaques par observations classiques. Le but de ce chapitre n'est pas de proposer les meilleures implémentations possibles, mais bien de montrer leur faisabilité et de permettre au lecteur d'appréhender les concepts nécessaires à l'implémentation sur GPGPU d'attaques par observations. Pour des implémentations plus optimisées en CUDA, le lecteur pourra consulter [BLR11, Hud].

3.1 GPGPU

Depuis quelques années, les processeurs graphiques sont devenus de plus en plus génériques afin de satisfaire les contraintes de plus en plus larges des opérations de traitement graphique. Ce gain de généricité a entrainé le développement du GPGPU. Le GPGPU permet d'utiliser le processeur graphique pour effectuer les calculs qui sont classiquement faits par le *Central Processing Unit* (CPU), *via* un environnement spécifique.

En 1972, Michael Flynn a proposé une taxinomie des architectures des ordinateurs dans [Fly72]. La Figure 3.1 présente cette classification qui dépend de la quantité de données à traiter et de celle des instructions à effectuer.

	Instruction unique	Instructions multiples
Donnée unique	SISD	MISD
Données multiples	SIMD	MIMD

Tableau 3.1 – Taxinomie de Flynn.

— Le modèle SISD correspond à un ordinateur séquentiel sans parallélisme (ordinateur de von Neumann, ...).
— Le modèle SIMD correspond à un ordinateur bénéficiant d'un parallélisme dans la mémoire (processeur vectoriel, ...).
— Le modèle MISD est l'architecture la moins utilisée et il existe peu d'implémentation de ce modèle (vérification de redondance dans les systèmes critiques).
— Le modèle MIMD est l'architecture parallèle la plus couramment utilisée (processeur multicœurs, ...).

Le GPGPU a une structure SIMD : il permet d'effectuer une même instruction sur de nombreuses données. Cette architecture est contraignante car elle n'est pas adaptée au traitement d'instructions conditionnelles. En effet, on a alors à traiter deux instructions différentes. Cependant, lorsque l'on respecte bien cette structure, les gains de performance sont significatifs.

Les environnement permettant d'utiliser le GPGPU sont dépendants du matériel utilisé. En effet, chaque constructeur fournit son propre *framework* (CUDA pour NVidia, Stream pour ATI). Le *Khronos Group* constitué d'acteurs majeurs du domaine, comme AMD/ATI, NVidia, Intel, ARM, Texas Instruments ou encore STMicroelectronics, propose un environnement qui vise à unifier ces différent *frameworks*. Cet environnement de programmation, *Open Computing Language* (OpenCL) [Gro10], permet d'utiliser des unités de calculs hétérogènes. Même si la norme permet d'utiliser bien d'autres unités de calcul, on ne s'intéressera qu'à l'utilisation de GPU. Bien que les spécifications existent, il n'y a pas à notre connaissance d'implémentation complète d'OpenCL. Les constructeurs proposent leurs propres implémentations qui sont en fait des surcouches de leurs environnements. Malheureusement cela nuit à l'objectif final d'OpenCL puisque l'on ne peut pas avoir un système hétérogène dans de telles conditions.

3.2 OpenCL

Nous avons choisi d'adopter OpenCL pour sa simplicité d'utilisation car le langage utilisé est proche du C standard 99. Il apparait néanmoins que dans certains cas, les implémentations en OpenCL soient moins performantes que leurs pendants en CUDA. OpenCL est un standard supporté par de nombreux constructeurs et qui se développe très rapidement.

3.2.1 Introduction

La Figure 3.1 représente une plateforme OpenCL. C'est une représentation générique qui permet de décrire toutes les unités de calculs pouvant être utilisées dans la norme.

On peut constater que la plateforme est constituée de deux parties. L'hôte (*host*) et les composants de calculs (*compute devices*). L'hôte est le plus souvent le CPU. Il se charge :

— d'initialiser la plate forme.
— de gérer la mémoire du composant.
— de régenter les différentes applications sur les composants et leurs entrées/sorties.

Les composants de calculs eux ne font qu'exécuter les instructions (fournies par l'hôte) sur les données (fournies par l'hôte).

Dans la Figure 3.2 nous avons représenté le composant de calculs. Il est composé d'unités de calculs (*compute unit* ou CU), qui elles mêmes sont composées d'unités de traitement (*processing element* ou PE). Les unités de traitement sont en fait des cœurs de calculs dégénérés qui ne possèdent aucun élément de contrôle de l'application. Un programme s'exécute au niveau des unités de calculs qui contrôlent l'exécution. Toutes les unités de traitement d'une même unité de calculs exécutent la même instruction au même instant mais sur différentes données. On retrouve ici le modèle SIMD présenté dans le tableau 3.1.

La structure d'un programme OpenCL se décompose donc en deux grandes parties. Le noyau (*kernel*) qui est exécuté sur le GPU et le programme hôte qui est exécuté sur le CPU. L'application hôte peut être écrite dans différents langages comme le C, le Java, le Python, etc. Le noyau, quant à lui, est écrit dans un langage très proche du C standard 99.

3.2.2 Exemple : L'addition de deux vecteurs

Afin d'appréhender l'écriture d'un noyau OpenCL, nous nous proposons d'additionner deux vecteurs de taille n. L'algorithme est très simple et est donné dans l'Algorithme 2.

algorithme 2 Algorithme d'addition de deux vecteurs, X et Y, de taille n.

Algorithme ADDVECT(X, Y)
  Z un vecteur de taille n
  $i \leftarrow 0$
  pour $i < n$ **faire**
    $Z[i] \leftarrow X[i] + Y[i]$
    $i \leftarrow i + 1$
  fin pour
retourner Z
fin Algorithme

Une implémentation classique en C de cet algorithme est donnée dans la Figure 3.3. Le noyau OpenCL correspondant est donné dans la Figure 3.4. On note dans ce noyau la fonction `get_global_id` qui est spécifique à l'OpenCL et qui permet de connaitre l'identité de l'instance qui est exécutée. L'hôte va demander aux composants de calculs d'exécuter ce noyau sur n PE. Chaque instance va additionner la composante des deux vecteurs correspondante à son identité. C'est à dire que l'instance 0 va calculer $X[0] + Y[0]$ alors que l'instance i va calculer $X[i] + Y[i]$. Lorsque toutes les instances seront terminées on aura obtenu le résultat souhaité dans Z.

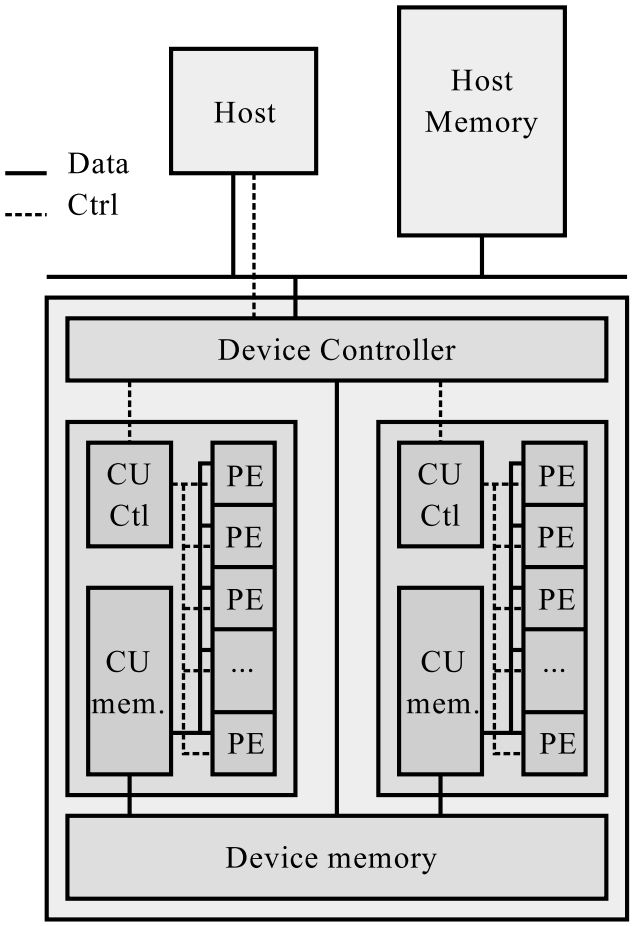


FIGURE 3.1 – Représentation d'une plateforme OpenCL.

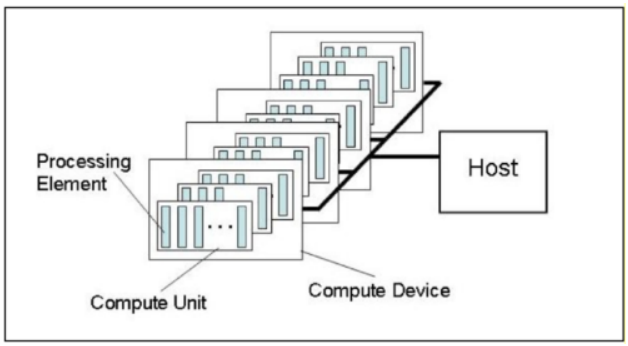


FIGURE 3.2 – Représentation d'un composant de calculs dans la plateforme OpenCL.

Cette exemple nous permet déjà d'implémenter en OpenCL l'attaque par observations CPA puisque les calculs des coefficients de Pearson dépendent essentiellement de sommes.

```
1  void AddVect_cpu (const float* X, const float* Y, float* Z,
       const int n)
2  {
3      for (int i = 0; i < n; i++)
4          Z[i] = X[i] + Y[i];
5  }
```

FIGURE 3.3 – Implémentation en C de l'addition de deux vecteurs.

```
1  __kernel void AddVect_gpu (__global const float* X, __global
       const float* Y, __global float* Z)
2  {
3      const int idx = get_global_id;
4  //La fonction get_global_id renvoie l'ID du thread courant.
5      Z[idx] = X[idx] + Y[idx];
6  }
```

FIGURE 3.4 – Implémentation OpenCL de l'addition de deux vecteurs.

3.3 CPA

Dans le Paragraphe 2.3.3.1, nous avons présenté la CPA. Cette attaque par observations est l'une des plus populaires et se prête très bien à une implémentation OpenCL. L'Équation 2.1 est en fait tronquée puisque l'on ne considère pas l'instant t pour lequel on calcule la corrélation. Dans l'Équation 3.1,nous avons développé les moyennes, ajouté l'aspect temporel et mis en couleurs deux types de sommes. Celles qui dépendent de l'instant t, en orange et les autres, en bleu.

$$\rho(t)_k = \frac{n.\sum_{i=1}^{n} l_i(t).m(x_i,k) - \sum_{i=1}^{n} l_i(t).\sum_{i=1}^{n} m(x_i,k)}{\sqrt{n.\sum_{i=1}^{n} l_i(t)^2 - \left(\sum_{i=1}^{n} l_i(t)\right)^2}.\sqrt{n.\sum_{i=1}^{n} m(x_i,k))^2 - \left(\sum_{i=1}^{n} m(x_i,k)\right)^2}}. \quad (3.1)$$

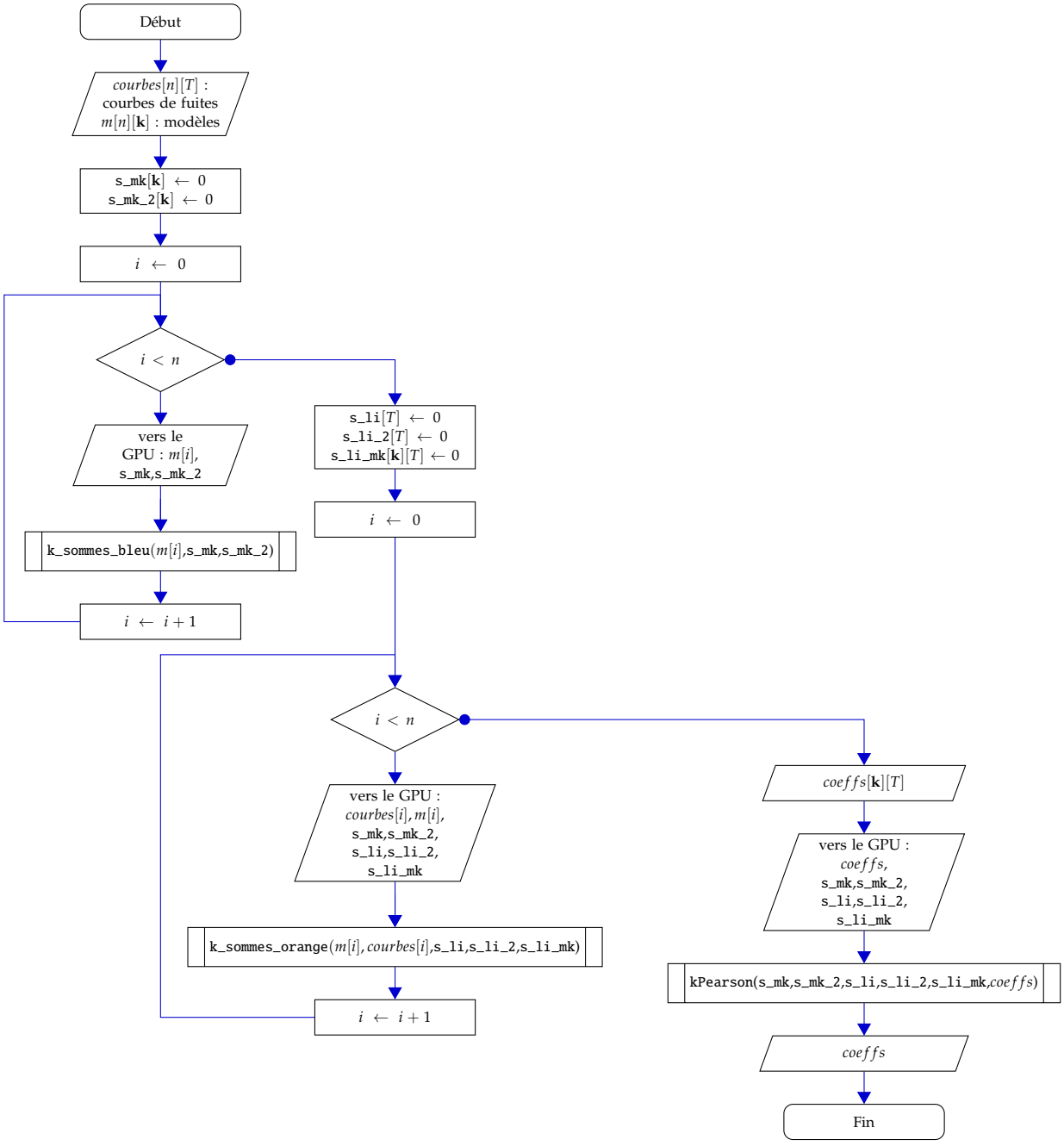
Dans [BLR11], Bartkewitz et Lemke-Rust ont proposé une implémentation de la CPA. La principale différence entre leur travail et le nôtre réside dans l'environnement utilisé, puisqu'ils utilisent CUDA. Ils proposent en outre une manière de calculer les sommes tout en évitant les débordements lorsque la valeur de la somme devient trop importante. Nous avons choisi ici de ne pas tenir compte de ces problèmes ni des problèmes de mémoire pouvant survenir lorsque les courbes traitées sont très grandes, afin de nous focaliser sur le parallélisme. Tout comme eux, nous proposons de diviser le calcul du coefficient de Pearson en trois noyaux.

Dans un premier temps, les sommes qui n'impliquent pas le paramètre t (en bleu) sont calculées. Le second noyau permet d'obtenir les sommes qui dépendent du paramètre t (en orange). Enfin l'ultime noyau permet de regrouper toutes les sommes calculées afin d'obtenir le coefficient de Pearson pour chaque instant t et chaque clé k considérés.

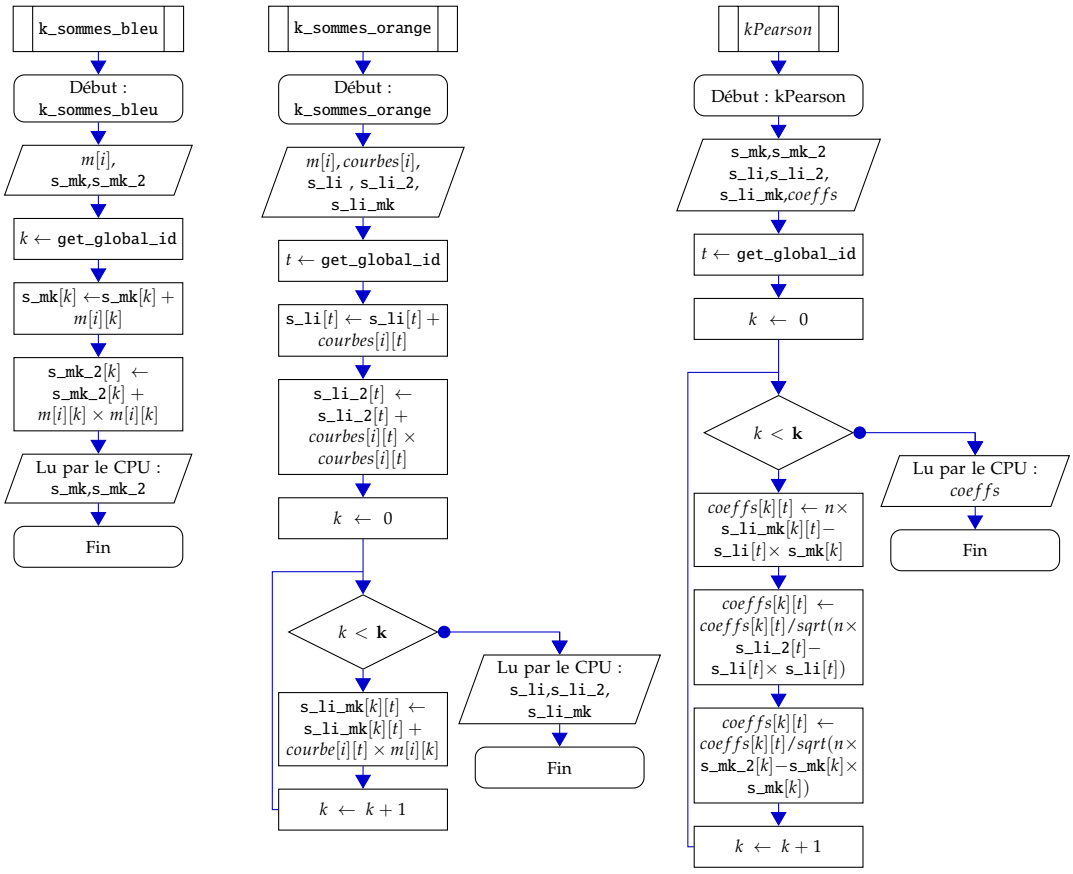
L'Algorithme 3 décrit l'implémentation de l'hôte et l'Algorithme 4 les noyaux utilisés par le composant de calculs. Ces deux algorithmes nous permettent ensemble de mener une attaque CPA en utilisant le GPU.
Le noyau `k_sommes_bleu` permet d'obtenir les sommes en bleu, pour cela il a besoin des modèles m et de deux tableaux pour stocker les résultats, `s_mk` pour $\sum_{i=1}^{n} m(x_i, k)$ et `s_mk_2` pour $\sum_{i=1}^{n} m(x_i, k)^2$.
Le noyau `k_sommes_orange` permet quant à lui d'obtenir les sommes en orange, pour cela il est nécessaire de lui fournir les modèles et les courbes. Il stocke les résultats dans `s_li` pour $\sum_{i=1}^{n} l_i(t)$, `s_li_2` pour $\sum_{i=1}^{n} l_i(t)^2$, `s_li_mk` pour $\sum_{i=1}^{n} l_i(t) \times m(x_i, k)$.
Grâce à ces deux premiers noyaux nous avons tous les éléments nécessaires pour calculer le coefficient de Pearson pour chaque clé et pour chaque instant.
Le noyau `kPearson` permet de recombiner les sommes obtenues par les noyaux précédents pour obtenir le coefficient de Pearson.

Nous avons testé notre implémentation sur un ordinateur équipé d'un processeur XEON X3430 possédant 4 cœurs cadencés à 2.4Ghz et d'une carte graphique NVidia ENGTS450 bénéficiant de 192 cœurs CUDA cadencés à 0.78GHz. Il est intéressant de noter que ces deux composants avaient un prix de lancement équivalent ($\approx$ 200\$). Nous avons utilisé les traces fournies par DPAContest V1 [Par08] et V2 [Par10]. Le temps d'exécution de notre implémentation OpenCL est environ douze fois plus rapide qu'une implémentation séquentielle. Il est important de noter que si l'on considère uniquement des traces de petites tailles (moins de 1 920 points dans notre cas), ce gain sera moins élevé à cause du coût de l'initialisation de l'environnement fait par l'hôte.

algorithme 3 Calcul des coefficients de Pearson sur GPU : **hôte**.



algorithme 4 Calcul des coefficients de Pearson sur GPU : **composant de calculs**.



3.4 Spearman

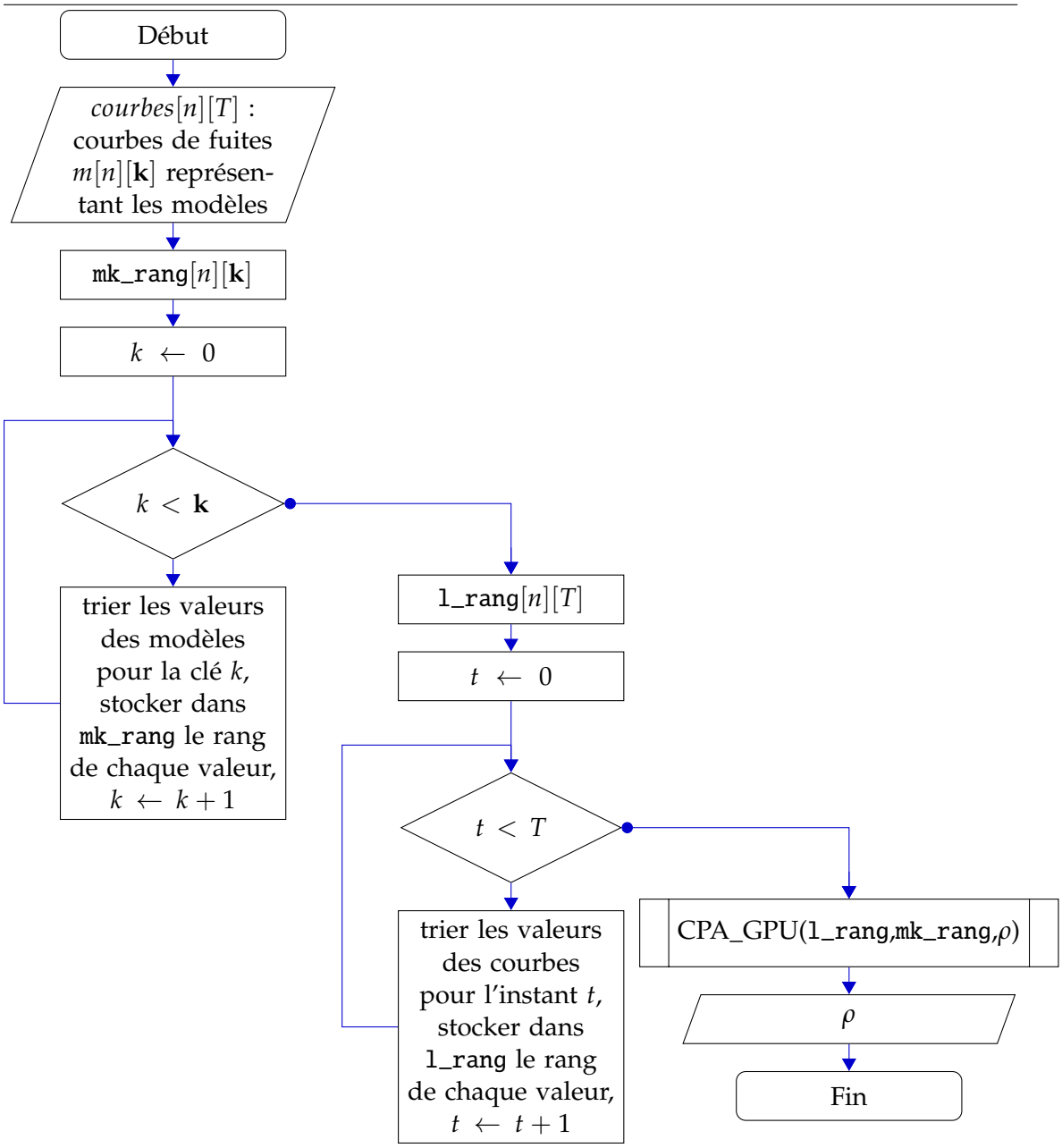
Dans le Paragraphe 2.3.3.2, nous avons présenté la corrélation des rangs de Spearman et l'attaque par observations qui en résulte. La corrélation de Spearman est obtenue en calculant le coefficient de Pearson sur les rangs des données et non sur leurs valeurs. Dans ce qui précède, nous avons déjà proposé une implémentation OpenCL pour le calcul du coefficient de Pearson. Nous allons donc proposer dans la suite différentes possibilités pour implémenter le tri des données nécessaire pour calculer la corrélation de Spearman. Dans un premier temps, nous avons implémenté l'algorithme de *tri rapide* [Hoa62] en OpenCL. Cette implémentation a été très difficile, de plus l'algorithme de *tri rapide* effectue beaucoup de comparaisons et n'est donc pas bien adapté au modèle SIMD. En utilisant cette implémentation et l'implémentation de la CPA précédente nous avons pu obtenir une implémentation cinq fois plus rapide que notre implémentation séquentielle. Ces résultats sont bien loin de ceux obtenus pour la CPA. Cette différence s'explique principalement par le fait que l'algorithme de *tri rapide* n'est pas adapté à une implémentation OpenCL.

3.4.1 Optimisation

Afin d'obtenir de meilleurs résultats, nous avons décomposé le calcul de la corrélation des rangs de Spearman en deux parties : le tri et le calcul du coefficient de Pearson. Nous avons utilisé l'hôte pour trier les données et le coefficient de Pearson est obtenu grâce au composant de calculs. L'Algorithme 5 décrit l'implémentation que nous avons mise en œuvre.

En distribuant le travail à effectuer entre le CPU et le GPU, les performances de notre implémentation ont été grandement améliorées. Nous avons ainsi pu obtenir une implémentation huit fois plus rapide que l'implémentation séquentielle. Afin de pouvoir encore améliorer les performances de notre implémentation, nous avons cherché à étudier les tris qui ne requièrent pas de comparaison.

algorithme 5 Calcul des coefficients de Spearman sur le GPU



3.4.2 Utilisation de tri sans comparaison

Les données que nous cherchons à trier sont des entiers. Afin de trier des entiers, on peut utiliser le tri par comptage. Ce tri extrêmement facile à mettre en œuvre est tout à fait compatible avec le modèle SIMD. Cet algorithme consiste à compter le nombre d'apparitions de chaque valeur et à le stocker. Ensuite, il suffit de relire le tableau dans lequel on a stocké ces nombres pour obtenir le tableau trié. Nous avons décrit ce tri dans l'Algorithme 6. La Figure 3.5 nous montre un exemple de tri comptage pas à pas sur le tableau $\{0, 1, 2, 4, 1, 1, 2\}$.

Cet algorithme est tout à fait compatible avec le modèle SIMD. Son implémentation sur *Graphics Processing Unit* (GPU) est donc efficace. Notre but n'est pas simplement de trier le tableau mais d'obtenir le rang des valeurs qu'il contient. Pour cela, nous avons modifié l'Algorithme 6. La première partie de cet algorithme nous permet d'obtenir un tableau contenant le nombre de fois qu'apparait chaque valeur dans le tableau à trier. Grâce à ce tableau, on obtient le nombre d'éléments de notre tableau inférieur à une valeur c'est à dire le rang de chaque valeur. Lorsque l'on est en présence d'ex æquo, on attribuera le rang moyen. Dans l'Algorithme 7, nous avons décrit notre méthode pour calculer les rangs.

Dans la Figure 3.6 nous avons repris l'exemple précédent que nous avons poursuivi avec notre algorithme afin d'obtenir le rang de chaque valeur. En implémentant cet algorithme en OpenCL, nous avons pu obtenir une implémentation onze fois plus rapide que la version séquentielle. En revanche, nous n'avons pas pu obtenir les mêmes performances que pour la CPA en raison du plus grand nombre de données à transférer vers le GPU.

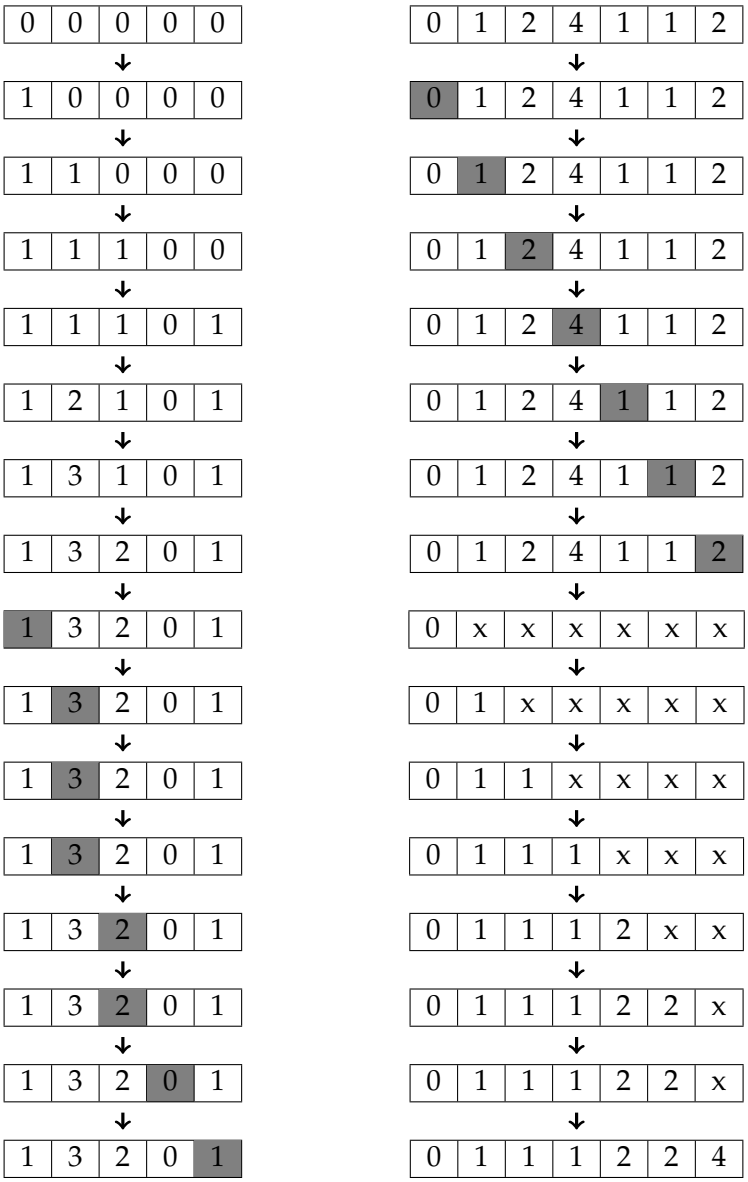
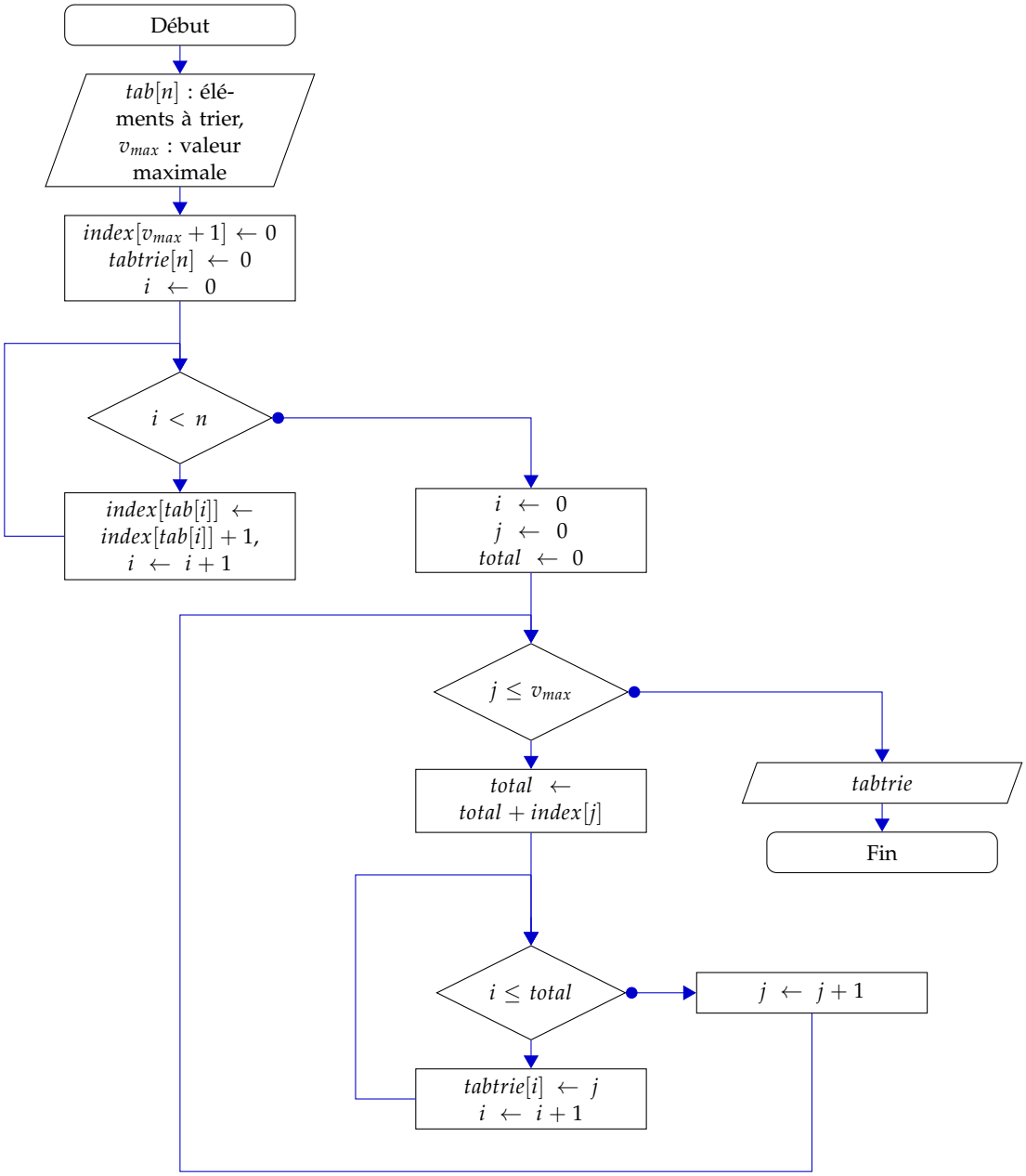


FIGURE 3.5 – Tri comptage de $\{0, 1, 2, 4, 1, 1, 2\}$ pas à pas.

algorithme 6 Algorithme de tri comptage.



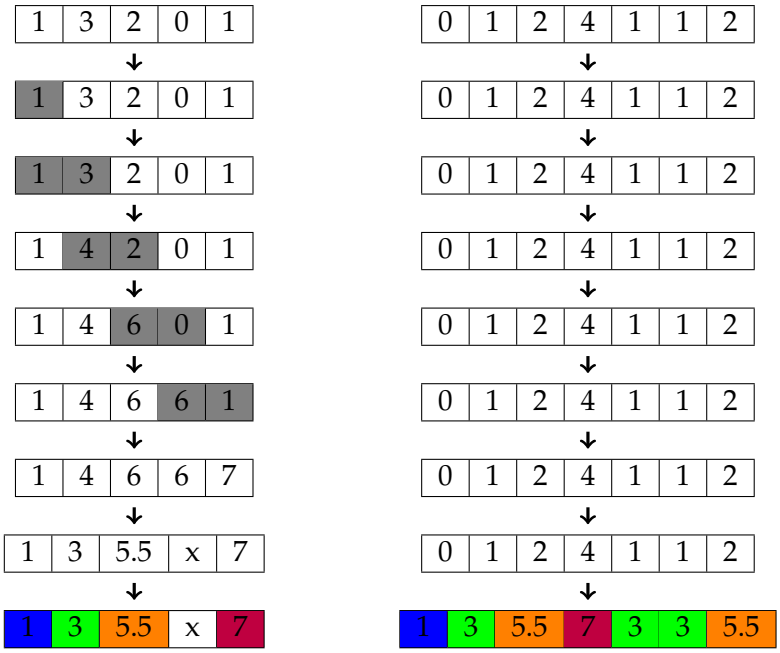
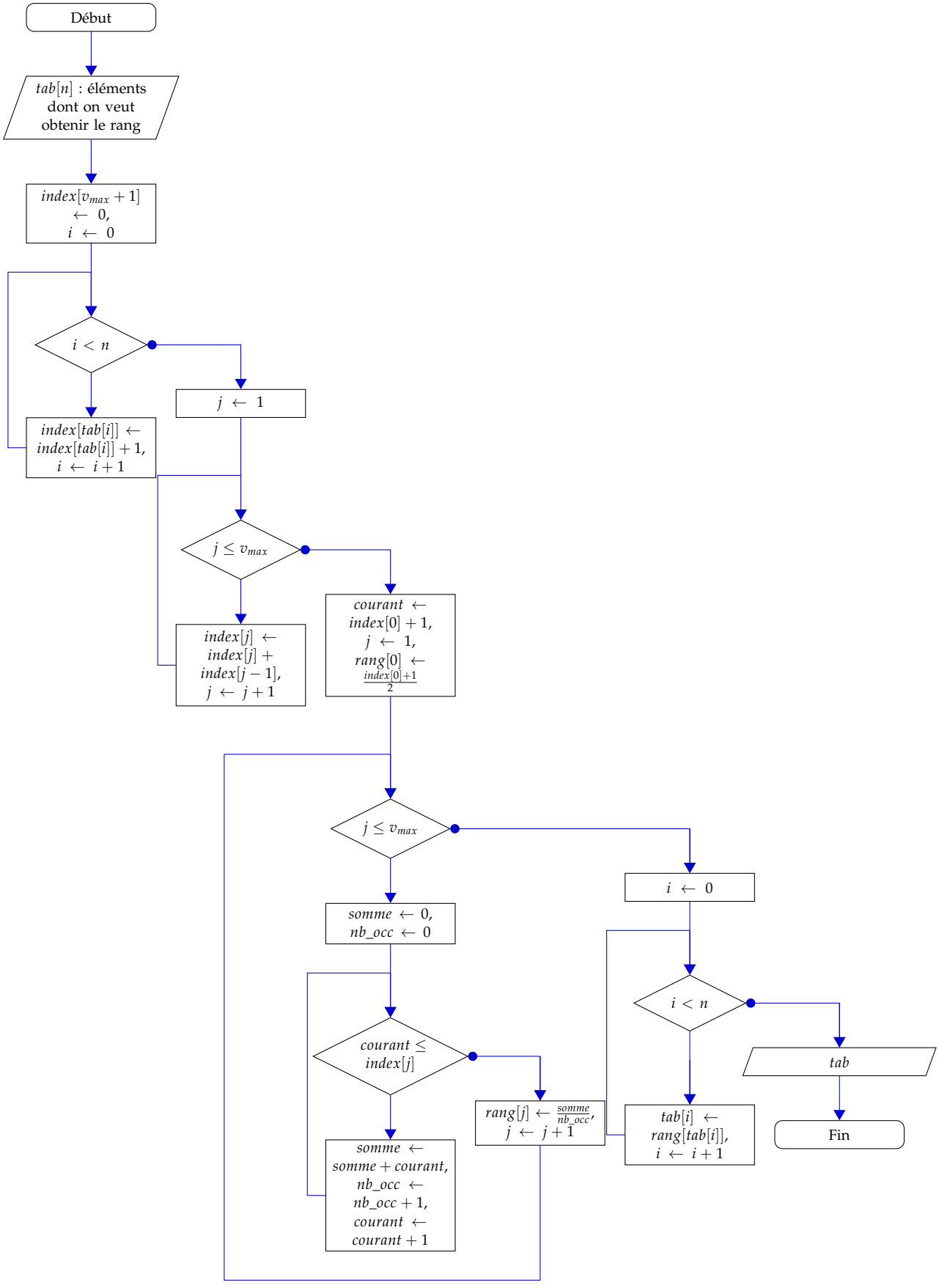


FIGURE 3.6 – Calcul des rangs pour l'ensemble $\{0, 1, 2, 4, 1, 1, 2\}$ pas à pas.

algorithme 7 Algorithme de calcul du rang.



3.5 Résultats pour d'autres attaques par observations

Nous avons implémenté le *tau de Kendall*, le coefficient de Gamma ainsi que la statistique de Kolmogorov-Smirnov. Les performances obtenues diffèrent en fonction des tests utilisés, certains étant mieux adaptés que d'autres au modèle SIMD comme nous avons pu le constater précédemment. Notre implémentation du *tau de Kendall* est neuf fois plus rapide que notre implémentation séquentielle. Ceci est dû au fait que l'on doit parcourir les données un grand nombre de fois. Le coefficient de Gamma étant peu adapté au modèle SIMD, nous avons pu obtenir qu'une implémentation six fois plus rapide que notre version séquentielle. Alors que pour la statistique de Kolmogorov-Smirnov nous avons pu utiliser le tri sans comparaison et ainsi nous avons obtenu une implémentation neuf fois plus rapide que notre implémentation séquentielle. Les algorithmes que nous avons implémentés sont disponibles dans l'Annexe A. Dans le Tableau 3.2 nous présentons les temps de calcul nécessaires pour chacune des implémentations proposées pour les 1000 premières courbes de DPAContest V1.

Attaque	Séquentielle	OpenCL
CPA	20	1,6
Spearman	190	17,3
Kendall	270	30
Gamma	30	5
Kolmogorov-Smirnov	130	14,5

Tableau 3.2 – Temps de calcul pour les différentes implémentations des attaques par observations en secondes pour les 1 000 premières traces de DPAContest V1 pour 1 600 instants.

3.6 Conclusion

Dans ce chapitre nous avons proposé des implémentations sur GPGPU des attaques par observations aux performances remarquables. En effet, notre implémentation de la CPA est douze fois plus rapide que notre implémentation séquentielle de référence. L'utilisation de GPGPU semble être une manière idéale de mettre en œuvre les attaques par observations. Nous n'avons parlé ici que des problèmes liés au modèle SIMD or, il existe une deuxième contrainte pour les GPGPU, les échanges mémoires entre l'hôte et le GPU. Dans le cas de la CPA, le problème n'existe que si l'on considère des courbes d'une taille supérieur à $100Mo$ dans le cas de notre carte graphique car notre algorithme effectue les calculs courbe par courbe. En revanche, pour les autres implémentations, les algorithmes effectuent les calculs point par point et chargent l'ensemble des courbes à chaque fois ; ce problème y est plus présent. Ce problème de mémoire est commun à de nombreuses implémentations utilisant les GPU. AMD/ATI travaillent depuis quelques temps à l'unification de la mémoire entre le CPU et le GPU où *Heterogeneous System Architecture* (HSA). Cette unification permettrait de résoudre ce problème de mémoire. Ce projet appelé *Heterogeneous Uniform Memory Access* (hUMA)[AMD] devrait donner le jour à une nouvelle génération de composants AMD/ATI nommée Kaveri qui pourrait être commercialisée avant la fin de cette année 2013. Il semblerait que les consoles de jeux de dernière génération comme la Playstation4 dispose déjà d'une architecture HSA[Nut13]. L'utilisation de ces nouveaux composants pourrait permettre d'obtenir des performances encore meilleures lorsque la gestion de la mémoire pose problème.

Dans le prochain chapitre nous allons présenter une nouvelle méthode pour étudier les relations entre deux variables aléatoires et l'attaque par observations qui en résulte. Nous proposerons certaines optimisations liées aux canaux cachés ainsi qu'une implémentation en OpenCL.

Chapitre 4

Attaque basée sur le coefficient maximal d'information

En 2011, Reshef *et al.* ont introduit une nouvelle mesure d'indépendance entre deux variables : le *Maximal Information Coefficent* (MIC) [RRF+11]. Cette mesure de dépendance est basée sur l'information mutuelle mais ne dépend plus de la valeur des paramètres tels que la taille de fenêtre, h, pour les estimateurs de densités de probabilité. Ainsi, le MIC peut être utilisé avec moins de connaissances préalables sur les données que l'information mutuelle. C'est donc un outil facile d'utilisation pour un attaquant. Le MIC est basé sur l'*information mutuelle* et vise à se prémunir des problèmes liés à l'estimation de densité de probabilité. L'idée des auteurs est d'étudier les variations de l'information mutuelle lorsque l'on change la taille de la largeur des histogrammes utilisés pour estimer les densités de probabilités. Le MIC est la valeur maximale de l'information mutuelle entre deux variables pour différentes tailles d'histogrammes. L'un des grands atouts du MIC comparé à l'information mutuelle, outre sa simplicité d'utilisation, est sa capacité importante à résister au bruit.

Nous allons présenter comment calculer le coefficient maximal d'information, avant d'expliquer comment l'utiliser dans le cadre d'une attaque par canaux cachés. Enfin, nous proposerons une implémentation du coefficient maximal d'information en OpenCL. Ce chapitre fera l'objet d'une publication dans le Journal de la Société Française de Statistique.

L'idée du MIC est de calculer l'information mutuelle grâce à différents histogrammes en s'autorisant à considérer des histogrammes ayant des classes de tailles différentes. Dans la Figure 4.1 les auteurs de [RRF+11] cherchent à maximiser la valeur de *l'information mutuelle* pour un nombre de classes fixé. On constate que la façon dont sont choisis les histogrammes pour estimer l'*information mutuelle* peut jouer un grand rôle.

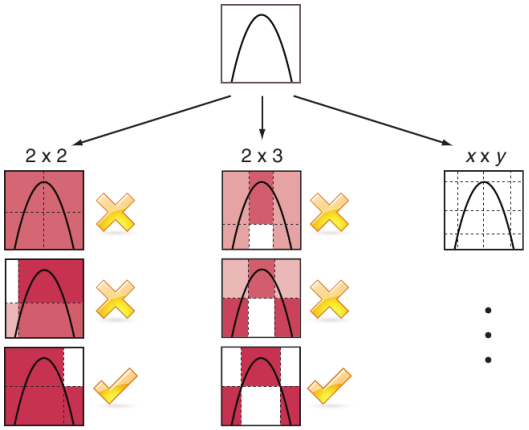


FIGURE 4.1 – Figure extraite de l'article [RRF+11].

4.1 Calcul du *MIC*

Le but du MIC est de détecter s'il existe une relation entre deux variables aléatoires $\mathcal{U}$ et $\mathcal{V}$. Pour cela on dispose de l'observation d'un échantillon de taille n de variable parente $\mathcal{U}$ (resp. $\mathcal{V}$), $\{u_1, \cdots u_n\}$(resp. $\{v_1, \cdots v_n\}$) de U (resp. V). On notera D l'ensemble des paires $(u_i, v_i), \forall i \in \{0, \cdots, n\}$ ordonnées par ordre croissant pour U et V. Les auteurs de [RRF+11] appellent *p-par-q* grille la partition des couples $(u_i, v_i), \forall i \in \{0, \cdots, n\}$ avec un histogramme à p classes pour la première composante et q classes pour la seconde. On notera par $\tilde{D}^G$ la distribution empirique du couple (U, V) associée à la grille G (comme dans le Tableau 4.1). On notera $\mathcal{G}_{(p,q)}$ l'ensemble de toutes les grilles de taille *p-par-q*. Si l'on considère

$$U = (4, 0, 1, 3, 5, 2),$$

$$V = (-0.3, 0.3, 0.01, -0.5, 0.4, 0.1),$$

$$D = ((0, 0.3), (1, 0.01), (2, 0.1), (3, -0.5), (4, -0.3), (5, 0.4)),$$

et

$$G = ([0, 2[, [2, 4[, [4, 6[) \times ([-1, 0[, [0, 1[),$$

une grille de taille 3-*par*-2.

	[-1,0[	[0,1[
[0,2[	0	2
[2,4[	1	1
[4,6[	1	1

Tableau 4.1 – Distribution de D sur G pour l'exemple proposé.

La distribution empirique de (U, V) associée à cette grille est donnée dans le Tableau 4.1. On notera $\hat{I}(\tilde{D}^G)$ l'information mutuelle obtenue grâce aux distributions empiriques associées à la grille G. Pour p et q fixés, le maximum d'information mutuelle sur l'ensemble des *p-par-q* grilles est défini par :

$$I^*(D, p, q) = \max_{G \in \mathcal{G}_{(p,q)}} (\hat{I}(\tilde{D}^G)).$$

Il n'est pas possible de comparer directement deux maxima d'information mutuelle, $I^*(D, p, q)$ et $I^*(D, p', q')$, lorsque la taille des grilles n'est pas la même. En effet, sur une grille de taille *p-par-q* la valeur maximale pouvant être atteinte est $log_2(min(p, q))$. Pour pallier ce problème, les auteurs proposent de normaliser ces maxima avant de les comparer. Une normalisation naturelle des maxima d'information mutuelle est donc donnée par :

$$M(D)_{p,q} = \frac{I^*(D, p, q)}{log_2(min(p, q))}.$$

Il est maintenant possible de comparer deux maxima d'information mutuelle pour des grilles de tailles différentes. On peut donc calculer le maximum des $I^*(D, p, q)$ pour tous les p et q possibles.

Les valeurs p et q sont bornées par le nombre d'observations, n, que l'on possède. Le nombre de grilles possibles est donc borné par n^n. Ce nombre devient très vite trop important pour que l'on puisse calculer l'information mutuelle de l'ensemble de ces grilles. Après une étude empirique, les auteurs ont proposé de borner le produit $p \times q$ par $n^{0.6}$ et finalement défini le coefficient maximal d'information par :

$$MIC(D) = \max_{\forall p,q | pq \leq n^{0.6}} (M(D)_{p,q}).$$

La valeur de ce coefficient est comprise entre 0 et 1. Un MIC de 0 indique une très probable indépendance entre les deux variables considérées. A contrario, un MIC de 1 indique une relation forte entre les deux variables.

Lorsque l'on compare le MIC à l'*information mutuelle*, on remarque tout d'abord une complexité de calcul bien plus importante pour le MIC que pour l'*information mutuelle*. En effet pour obtenir le MIC, on doit calculer de multiples fois l'*information mutuelle* à partir de différents histogrammes. Néanmoins, le MIC peut être calculé sans connaissances préalables sur les variables considérées. De plus, le MIC reste plus stable en présence de bruit que l'*information mutuelle*. Dans le cadre des attaques par observations, le bruit est un problème important, le MIC offre donc une alternative très intéressante à l'*information mutuelle*.

4.2 Maximal Information Coefficient Analysis

Dans le contexte des attaques par observations, on cherche à maximiser le MIC. La clé secrète est donc estimée grâce à :

$$\hat{k^\star} = \underset{k \in \mathcal{K}}{argmax}(MIC(D_k)),$$

où $D_k = \{(l_i, m(x_i, k)) \in \{1, \cdots, n\}$ ordonnés$\}$. Le calcul du MIC est très coûteux et il doit être calculé pour chaque clé possible. Il est donc important de ne pas implémenter le MIC directement à partir des équations précédentes mais d'adapter le MIC au cas très particulier des attaques par canaux cachés.

Dans un premier temps, il est astucieux de noter que les variables que l'on considère ne sont pas de même nature. $L(Z)$ est de nature continue alors que $m(X,k)$ est de nature discrète. On connait les valeurs possibles que peut prendre la variable $m(X,k)$. On peut donc borner q par le nombre de valeurs que peut prendre la variable représentant notre modèle. Ce nombre est souvent suffisamment petit pour réduire grandement la complexité du calcul du MIC. Pour l'AES 9 valeurs sont possibles et 5 pour le DES lorsque l'on considère un modèle basé sur le poids de Hamming.

4.3 Implémentation en OpenCL

Le calcul du MIC est décrit dans l'Algorithme 8.

algorithme 8 Calcul du maximal d'information mutuelle.

```
1:  Algorithme CALCUL DU MIC (D un ensemble de n données ordonnées)
2:      MIC ← 0
3:      pour p, q tel que p · q ≤ n^0.6 faire
4:          I ← MaxMI(D, p, q)
5:          si MIC < I alors
6:              MIC ← I
7:          fin si
8:      fin pour
9:  retourner MIC
10: fin Algorithme
```

La fonction *MaxMI* renvoie la valeur maximale pouvant être atteinte par l'information mutuelle pour les données D sur l'ensemble des grilles de taille *p-par-q*. On peut remplacer la fonction *MaxMI* par une fonction qui va simplement calculer le maximum d'information mutuelle sur l'ensemble des grilles de taille *p-par-q* une par une. Une telle fonction n'est pas réaliste si l'on veut pouvoir utiliser le MIC car il faudrait calculer l'information mutuelle sur un nombre trop important de grilles. Dans [RRF+11] Reshef et *al.* proposent un algorithme heuristique pour remplacer la fonction *MaxMI* lorsque p et q sont fixées. La méthode qu'ils proposent repose sur deux faits. D'une part, l'information mutuelle est bornée inférieurement par l'entropie de la variable qui possède le moins d'information. Dans le cadre des attaques par observations c'est la variable discrète $m(X,k)$. D'autre part, l'entropie marginale des deux variables est maximisée par une équipartition, c'est à dire une partition où tous les histogrammes possèdent la même largeur. Une approche heuristique naturelle sera de considérer que l'un des deux axes est équipartitionné et de chercher à maximiser l'information mutuelle en modifiant la partition de l'autre axe. La méthode de Reshef et *al.* optimise la recherche de la partition permettant de maximiser l'information mutuelle lorsque l'un des axes est équipartitionné.

Pour notre implémentation, nous commençons par fixer l'équipartition de l'histogramme en p classes de la variable $L(Z)$. La variable $m(X,k)$ étant discrète et possédant peu de valeurs possibles, nous faisons une recherche exhaustive pour trouver la partition de l'histogramme en q classes de $m(X,k)$ qui maximisera l'information mutuelle. L'algorithme 9 décrit la procédure que nous avons implémentée en OpenCL. Le noyau OpenCL associé calcule le maximum d'information mutuelle pour une grille de taille *p-par-q*.

algorithme 9 Calcul du maximal d'information mutuelle.

Algorithme CALCUL DU MIC ($D = L(Z), m(X,k)$ un ensemble de n données ordonnées)
  $MIC \leftarrow 0$
  pour p, q tel que $p \cdot q \leq n^{0.6}$ **faire**
    pour $G \in \mathcal{G}_{(p,q)}$ tel que $L(Z)$ soit équipartitionnée **faire**
      $I \leftarrow I_G(L(z); m(X,k))$
      si $MIC < I$ **alors**
        $MIC \leftarrow I$
      fin si
    fin pour
  fin pour
retourner MIC
fin Algorithme

Le nombre de grilles à tester peut être borné, de manière abrupte, par le produit du nombre de partitions possibles pour la variable $m(X,k)$ et de la taille maximale de grille, $n^{0.6}$. Si l'on considère, par exemple, que l'on possède $n = 1\,000\,000$ signaux de consommation de courant d'un AES, on devra calculer l'information mutuelle sur moins de 40 000 000 de grilles. [1] Ce calcul est tout à fait réalisable dans un temps raisonnable.

Nous avons comparé notre implémentation avec l'implémentation en JAVA proposée par Reshef et *al.*, mais aussi avec une implémentation en C proposée par Albanese *et al.* dans [AFV+12] et nommée minepy. L'implémentation en JAVA de Reshef *et al.* n'est pas très optimisée mais permet d'avoir un bon point de départ pour valider notre implémentation. Par contre, elle ne permet pas d'avoir plus de 1 000 réalisations par variable ce qui est très restrictif. Nous avons fait une comparaison entre ces deux implémentations et notre implémentation en OpenCL en effectuant une attaque par observations *Maximal Information Coefficent Analysis* (MICA) sur les données de la version 2 de DPA-Contest [Par10] en utilisant 1 000 et 20 000 traces de 3 253 points d'un AES. Le Tableau 4.2 présente les résultats de cette expérimentation. On constate que notre implémentation est plus efficace que l'implémentation proposée par Reshef *et al.* Lorsque le nombre de traces considéré n'est pas très important, l'implémentation minepy est plus performante que notre implémentation en OpenCL, ceci est dû au coût d'utilisation d'une implémentation en OpenCL (initialisation du contexte, etc...). L'utilisation de l'OpenCL nous permet d'atteindre d'excellentes performances par rapport aux autres implémentations proposées. Néanmoins il est important de noter que nous utilisons des propriétés spécifiques aux canaux cachés pour accélérer le calcul du MIC.

Nombre de traces	JAVA	minepy	OpenCL	CPA séquentielle
1000	2.6	0.03	0.3	0.0125
20000	**X**	2	0.5	0.06

Tableau 4.2 – Temps de calcul pour les différentes implémentations pour un instant t et une clé k en secondes.

1. $\sum_{q=0}^{7} C_8^q \cdot \frac{n^{0.6}}{q} = 37\,711\,093$

4.4 Résultats expérimentaux

Nous avons choisi de présenter ici les résultats que nous avons obtenus en utilisant le MIC en prenant comme fonction g les quatre bits issus de la première boite-S après le OU exclusif entre les registres R et L de la dernière ronde du DES proposé dans la première version de DPAContest et comme fonction de fuite, φ le poids de Hamming (comme nous l'avions fait dans la Section 2.4). Nous comparons ces résultats à ceux obtenus en utilisant la CPA. La Figure 4.2 représente la valeur de la CPA (en haut) et de la MICA (en bas) pour 1 000 traces pour chaque clé. Le nombre de traces étant très important, les deux attaques ont réussi. On peut observer que les pics de relations sont présents aux mêmes instants pour les deux distingueurs. On peut donc en déduire que la relation entre le modèle que l'on a choisi (la distance de Hamming) et la consommation est bien linéaire. Il est aussi intéressant de remarquer que les résultats de la MICA ne présentent pas de *pics harmoniques*

La Figure 4.3 représente la valeur de chaque distingueur pour 200 traces pour chaque clé. Dans cette expérimentation, on a grandement réduit le nombre de traces utilisées. On constate que les deux attaques sont toujours effectives. Pour la bonne clé de la CPA, on peut toujours observer 5 pics alors que l'on observe uniquement un pic pour la MICA. Pour la MICA, on peut remarquer que malgré tout, la courbe représentant la valeur du MIC entre le bon modèle et les traces possède toujours 5 motifs alors que celle représentant la valeur du MIC pour une mauvaise hypothèse de clé ne représente que du bruit comme le montre la Figure 4.4 .

La CPA requiert moins de réalisations que la MICA puisqu'elle reste effective même lorsque l'on a très peu de traces. Mais lorsque l'on possède suffisamment d'information, la MICA est plus efficace que la CPA dans le sens où elle ne possède pas de *pics harmoniques*.

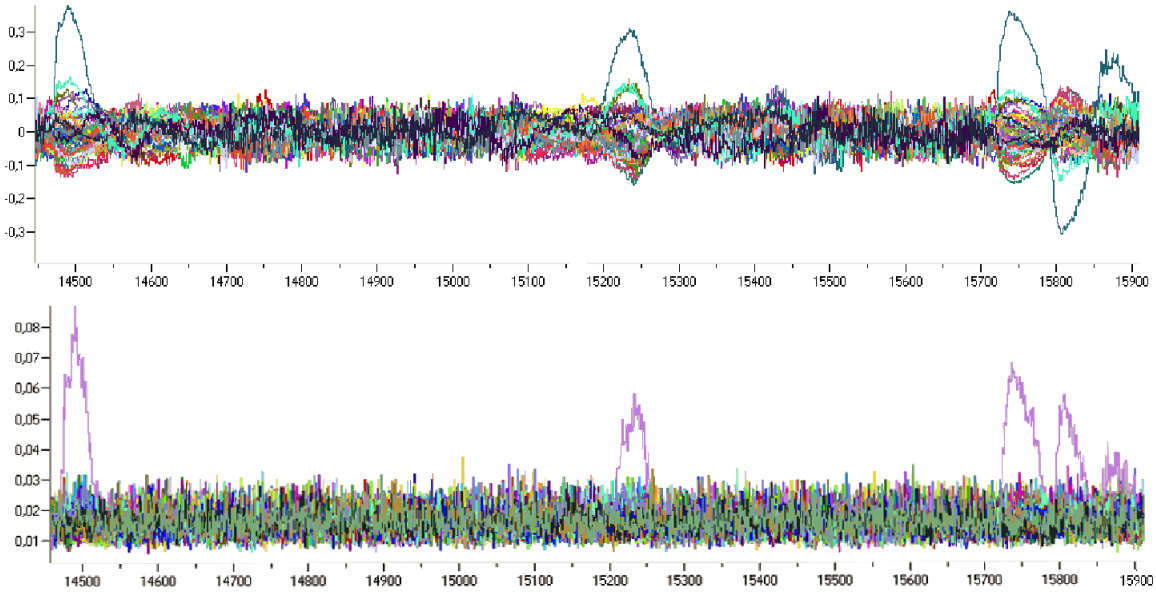


FIGURE 4.2 – En haut le résultat de la *CPA*, en bas, le résultat de la *MICA* pour 1 000 traces de la première version du DPAContest.

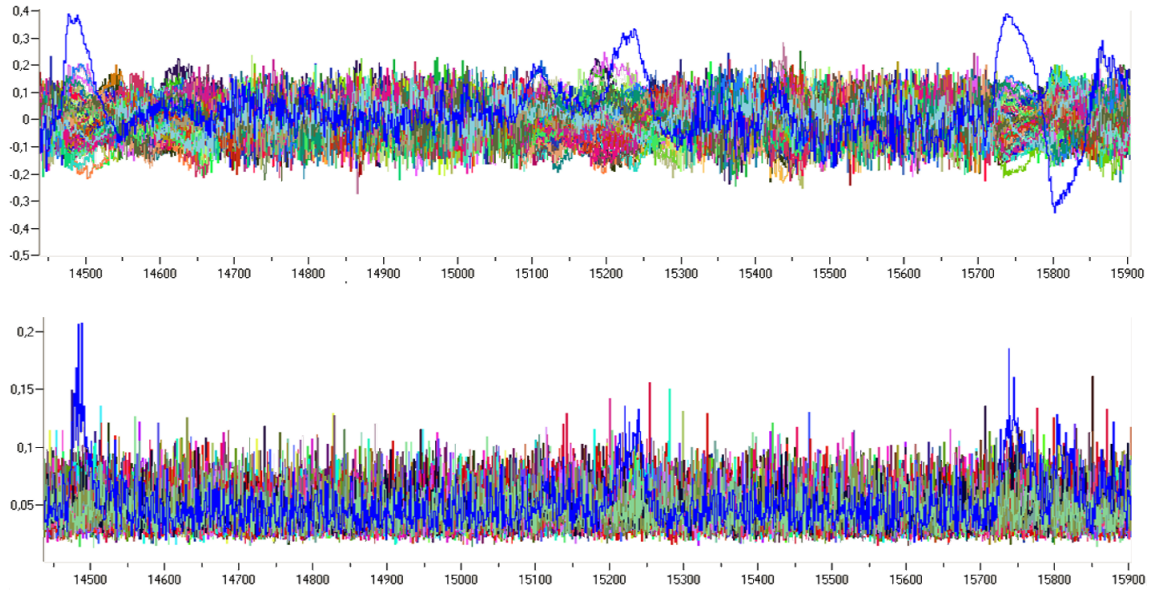


FIGURE 4.3 – En haut le résultat de la *CPA*, en bas, le résultat de la *MICA* pour 200 traces de la première version du DPAContest.

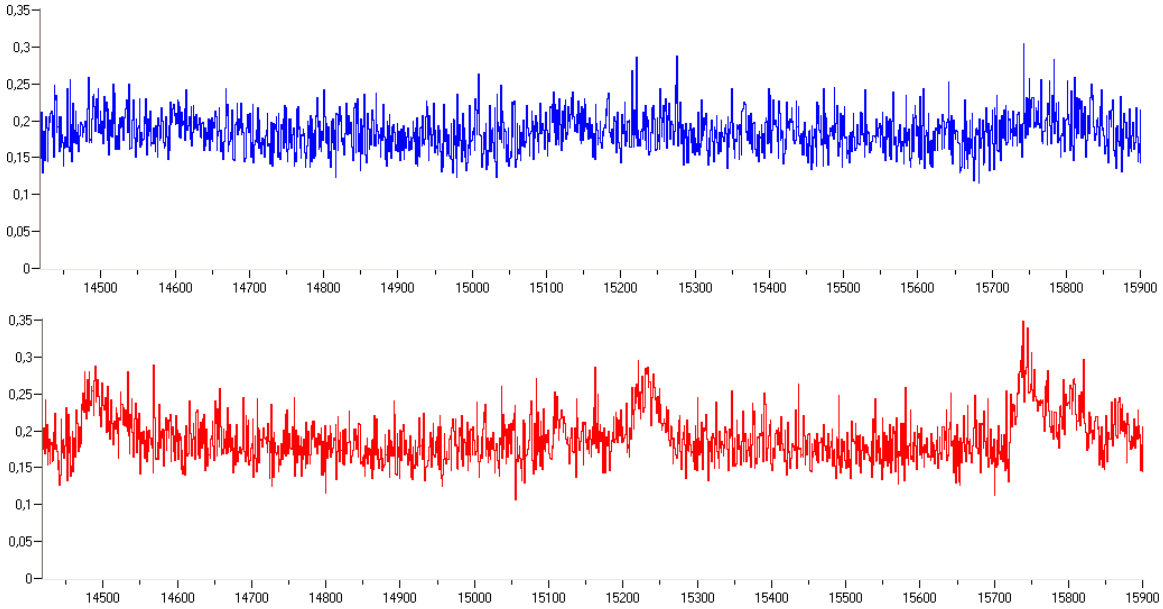


FIGURE 4.4 – Résultat de la *CPA* pour 150 traces de la première version du DPAContest pour la bonne clé (en rouge) et pour une mauvaise clé (en bleu).

4.5 Influence de la taille des grilles

Dans cette section, nous avons voulu étudier l'influence de la taille des grilles sur les résultats de la MICA. En effet, dans [RRF+11], Reshef *et al.* proposent de majorer la taille des grilles que l'on considère par $n^{0.6}$. Si l'on est capable de réduire encore cette taille tout en gardant les mêmes résultats, le temps de calcul du MIC sera réduit. Nous avons choisi d'utiliser 1 000 traces disponibles pour mener les MICA. Nous avons considéré d'une part le modèle correspondant à la clé secrète, c'est à dire un modèle que l'on sait avoir une dépendance avec les données, ainsi qu'un modèle correspondant à une mauvaise clé. Rappelons que lorsque la taille des grilles augmente, la valeur du MIC ne peut diminuer car l'on considère un maximum. Ainsi sur nos figures, les courbes avec l'amplitude la plus faible correspondent aux grilles les plus petites alors que celles avec la plus grande amplitude correspondent aux grilles de plus grande taille. Nous avons dans un premier temps fait varier la taille maximale de partition pour la variable $m(X,k)$ en prenant pour valeur maximale : $5, 6, 7$. Les résultats de cette expérimentation sont présentés sur la Figure 4.5. Comme attendu, la valeur du MIC ne change que très peu car la variable $m(X,k)$ n'a que 5 valeurs possibles.

Dans un second temps, nous avons fait varier la taille maximale de partition pour la variable $L(Z)$ en prenant pour valeur maximale : $5, 10, 15, 20$.

Les résultats présentés dans la Figure 4.6 ici sont plus surprenants que dans l'expérimentation précédente. En effet, le fait d'utiliser des partitions plus petites ne semble pas beaucoup affecter la valeur du MIC calculée. Ce fait est sûrement dû à la variable $m(X,k)$ qui est discrète et qui n'a que peu de valeurs possibles.

Nous avons donc choisi de comparer la MICA en utilisant l'implémentation en JAVA proposée par Reshef *et al.* et la nôtre dans laquelle nous nous sommes limités à des grilles de tailles maximales 5-*par*-5. On peut observer sur la Figure 4.7 que les résultats obtenus sont presque identiques. Dès lors, nous avons testé cette nouvelle implémentation sur d'autres données. Les résultats restent excellents même lorsque l'on calcule le MIC sur de petites grilles. Grâce à cette observation, nous avons pu réduire de manière drastique le temps de calculs nécessaire pour une attaque par observations MICA. Nous avons pu tester cette amélioration uniquement dans le cas où le coefficient de Pearson donne un résultat dans le cadre des attaques par observations. En revanche, nous n'avons pas pu valider notre proposition sur des données où le coefficient de Pearson serait inefficace.

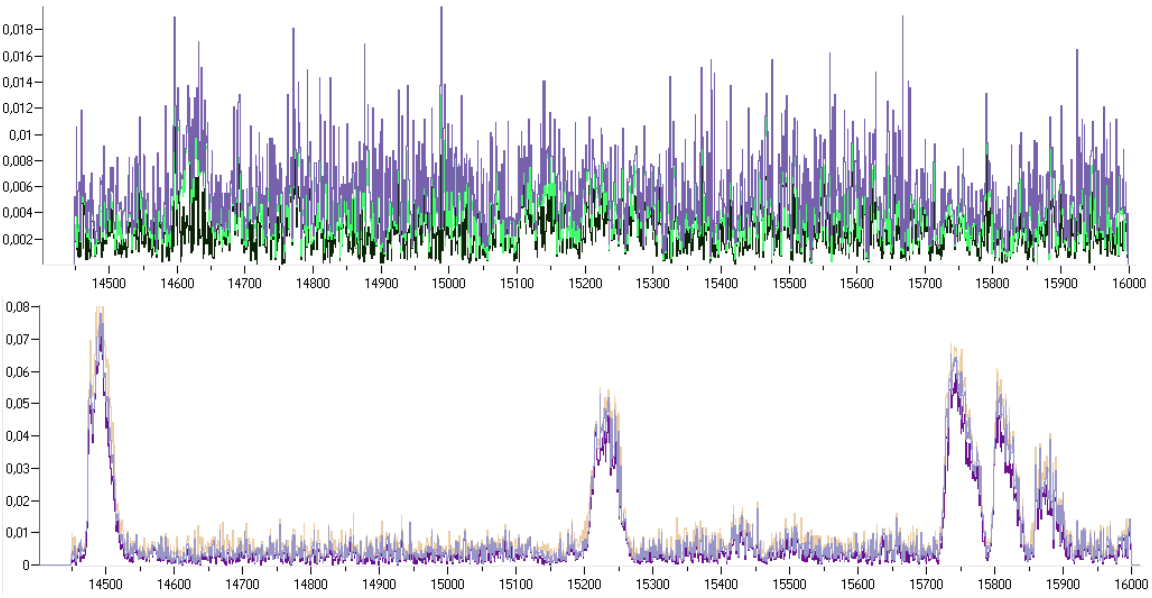


FIGURE 4.5 – Le résultat de la *MICA* pour 1 000 traces de la première version du DPA-Contest pour une fausse clé 6 (en haut) et la vraie clé 60 (en bas) lorsque l'on fait varier la taille maximale pour les partitions de la variable $m(X, k)$.

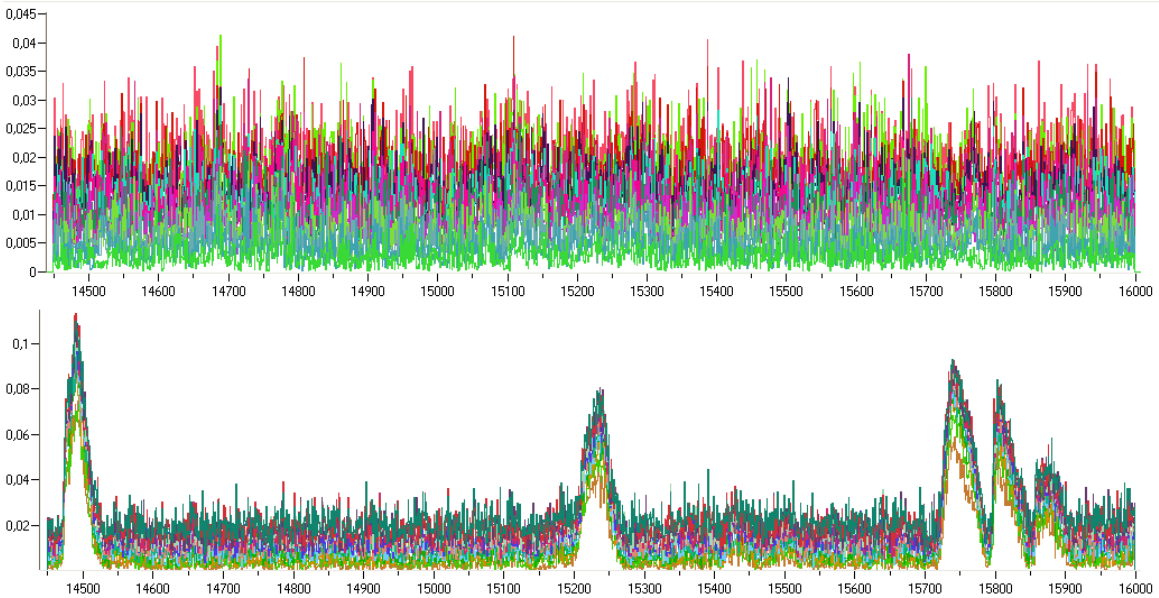


FIGURE 4.6 – Le résultat de la *MICA* pour 1 000 traces de la première version du DPA-Contest pour une fausse clé 6 (en haut) et la vraie clé 60 (en bas) lorsque l'on fait varier la taille maximale pour les partitions de la variable $L(Z)$.

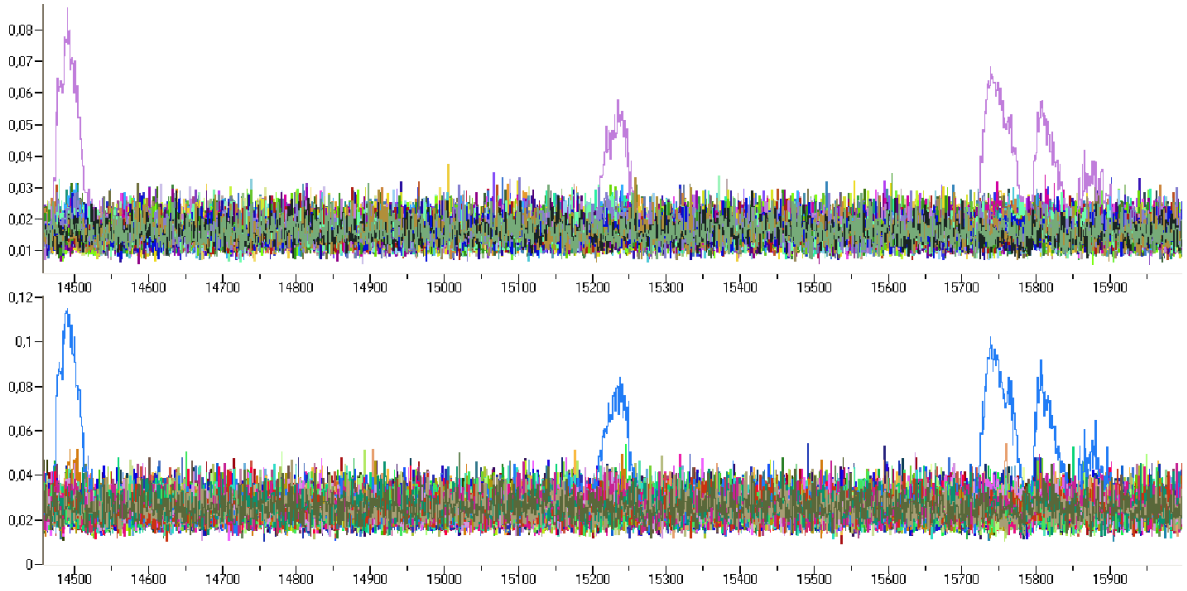


FIGURE 4.7 – Le résultat de la *MICA* pour 1 000 traces de la première version du DPAContest pour l'implémentation en JAVA de Reshef *et al.*(en haut) et pour notre implémentation en limitant la taille maximale des grilles à des grilles de 5-*par*5 (en bas).

4.6 Autres modèles de fuite

Le principal intérêt du MIC est de pouvoir détecter des relations entre les variables $L(Z)$ et $m(X, k^\star)$ qui ne soient par linéaires ou d'ordre. Il est donc intéressant d'étudier l'attaque par observations MICA lorsque la relation entre la fuite du composant et le modèle correspondant à la clé secrète n'est pas linéaire. Nous avons donc étudié les valeurs du coefficient de Pearson, de l'information mutuelle et du coefficient maximal d'information lorsque l'attaquant se trompe et utilise un modèle en poids de Hamming alors que la relation entre la fuite et le modèle est donnée par une autre fonction g. Pour simuler les données, nous choisissons 50 entiers x entre 0 et 15 et nous obtenons la fuite simulée en calculant $g(x)$ pour les différents modèles de fuite suivants :

1. $g(x) =$ poids de Hamming(x).
2. $g(x) =$ poids de Hamming$(x) +$ la valeur du premier bit de x.
3. $g(x) = P($poids de Hamming$(x))$ où P est une permutation fixe.
4. $g(x) =$ un nombre aléatoire entre 0 et 8.

La première fonction est très classique et c'est celle que nous avons déjà utilisée précédemment. La seconde représente le cas où un bit fuit plus que les autres [LCC08]. Pour la troisième, nous avons considéré le cas où les données sont stockées de manière masquée. Enfin la dernière fonction représente un modèle aléatoire que l'on peut observer lorsque l'on considère, par exemple, une mauvaise clé.

Nous avons simulé 50 traces et repété 10 000 fois la simulation. Le Tableau 4.3 présente le résultat moyen des deux distingueurs sur les données simulées. Rappelons que les relations entre $L(Z)$ et $m(X, k)$, pour $k \neq k^\star$, sont comparables à ce que l'on observerait en prenant des valeurs au hasard pour le modèle c'est à dire à la quatrième fonction. Comme on pouvait s'y attendre, le coefficient de Pearson donne les meilleurs résultats lorsque la relation est linéaire. On constate également que le MIC donne de bons résultats dans tous les cas y compris lorsque le modèle de fuite est une permutation. Le MIC donne les meilleurs résultats lorsque tous les bits de la variable observée n'ont pas tous la même contribution à la fuite. On constate que lorsque le modèle de fuite utilisé pour l'attaque est imparfait, le MIC semble être un bon choix de distingueur.

Modèle de fuite	ρ	MIC
1	0.69	0.61
2	0.58	0.84
3	0.02	0.45
4	0	0

Tableau 4.3 – Moyenne sur 10 000 simulations de 50 traces de la différence entre la valeur du coefficient (Pearson et MIC) pour chaque modèle considéré et celle pour le modèle 4 (mauvaise clé).

4.7 Relations entre la consommation de courant et l'émanation électromagnétique

Dans les attaques par canaux cachés, on utilise principalement deux fuites : la consommation de courant et l'émanation électromagnétique. Ces deux types de fuites permettant chacune de mener une attaque par observations, il nous a semblé intéressant

d'étudier, grâce au MIC, les relations pouvant exister entre elles. En effet, si pour une attaque par observations, on ne considère que les instants durant lesquels on détecte une relation forte entre les deux types de fuites, on réduit la complexité de l'attaque.

Nous avons donc calculé le MIC entre la consommation de courant et l'émanation électromagnétique pour les deux premières rondes d'un DES câblé. Les Figures 4.8 à 4.11 représentent les résultats que nous avons obtenus. Hélas, on constate que les valeurs maximales du MIC correspondent au pics de consommation de courant ce qui ne fait que confirmer un fait déjà connu : l'information utile pour les attaques par observations se trouve a proximité des pics d'horloge.

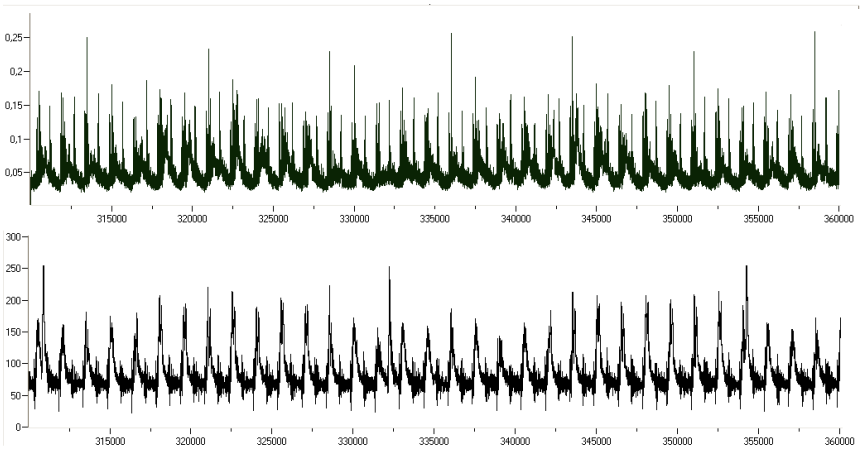


FIGURE 4.8 – Le résultat du MIC entre la consommation de courant et l'émanation électromagnétique pour 1 000 traces (en haut) et une courbe de consommation de courant (en bas) pour les deux premières rondes du DES.

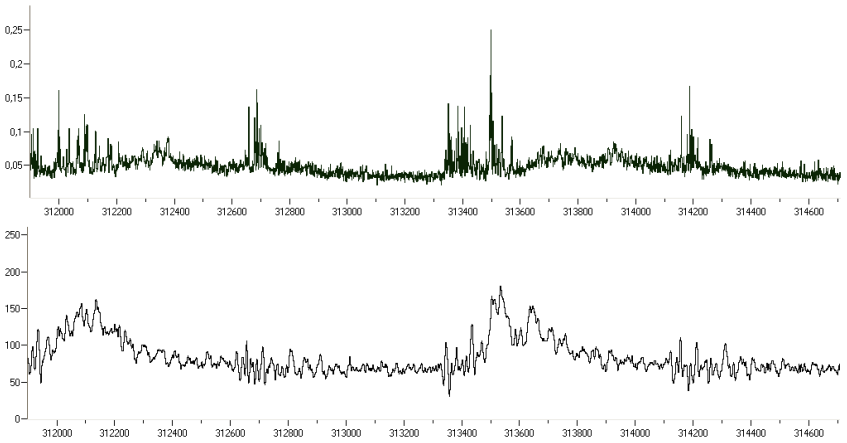


FIGURE 4.9 – Le résultat du MIC entre la consommation de courant et l'émanation électromagnétique pour 1 000 traces (en haut) et une courbe de consommation de courant (en bas) pour les deux premiers pics de consommation de courant du DES.

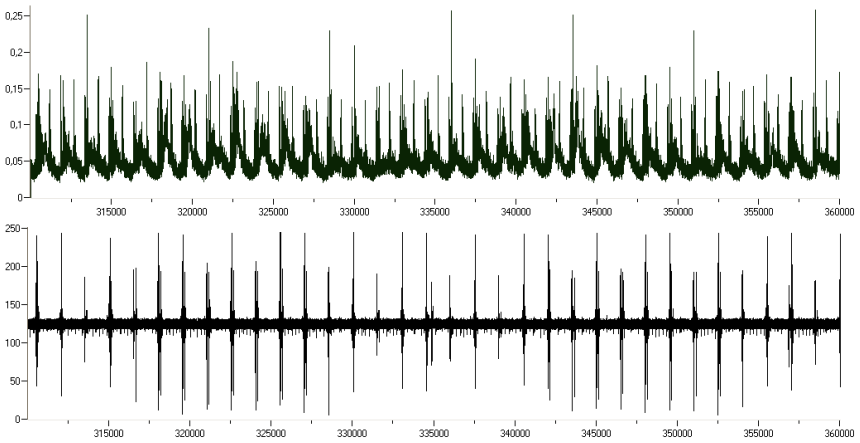


FIGURE 4.10 – Le résultat du MIC entre la consommation de courant et l'émanation électromagnétique pour 1 000 traces (en haut) et une courbe de l'émanation électromagnétique (en bas) pour les deux premières rondes du DES.

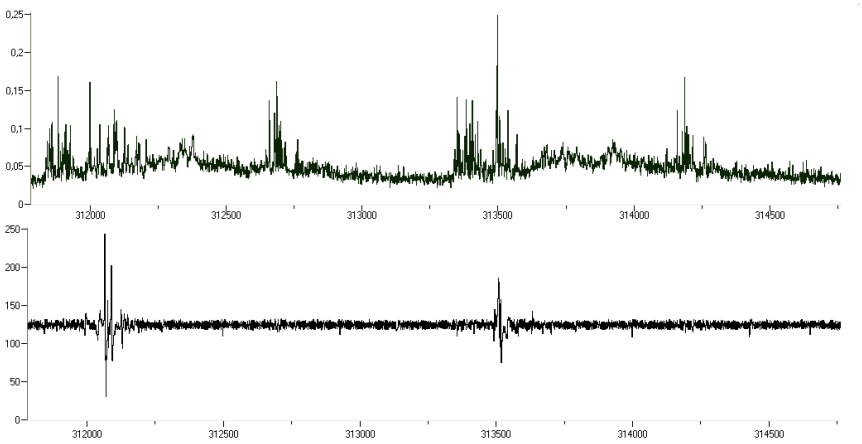


FIGURE 4.11 – Le résultat du MIC entre la consommation de courant et l'émanation électromagnétique pour 1 000 traces (en haut) et une courbe de l'émanation électromagnétique (en bas) pour les deux premiers pics de consommation de courant du DES.

4.8 Conclusion

Dans ce chapitre nous avons présenté un nouveau distingueur basé sur le coefficient maximal d'information qui permet de découvrir des relations générales entre les variables $L(Z)$ et $m(X, k^\star)$. Comme la *Mutual Information Analysis* (MIA), la MICA peut être efficace même lorsque la relation entre la fuite et le modèle correspondant à la clé secrète n'est pas linéaire. Malheureusement, cette généricité arrive avec un coût très important en calcul. Nous avons toutefois fait plusieurs propositions afin que le MIC puisse être utilisé de manière réaliste dans le cadre des attaques par canaux cachés. Notre implémentation tire parti de la particularité de ces attaques. Nous avons ainsi obtenu une implémentation en OpenCL très efficace comparée à l'implémentation minepy. Le principal avantage du MIC est d'être défini de manière unique, alors que *l'information mutuelle* dépend des choix faits pour estimer l'entropie conditionnelle et l'entropie marginale. De plus, le MIC réagit très bien en présence de bruit comme l'ont montré Reshef *et al.* dans [RRF+11]. Nos expérimentations soulignent la capacité de ce distingueur à retrouver la clé secrète utilisée par un composant en utilisant peu de traces lorsque le modèle de fuite est bien défini. Elles ont aussi montré que lorsque le modèle n'est pas bien défini, le MIC permet d'avoir une bonne estimation de la clé secrète utilisée par le composant. De plus, notre distingueur donne d'excellents résultats en simulation lorsqu'un bit fuit plus que les autres. Lorsque la fuite n'est pas bien modélisée, notre distingueur semble être un choix intéressant même si sa complexité de calcul est très importante. En revanche, si le modèle de fuite est bien défini, la CPA reste le meilleur choix. Le MIC permettant de détecter un très grand nombre de relations, nous l'avons utilisé pour découvrir des relations entre la consommation de courant d'un composant et son émanation électromagnétique. Malheureusement, cette expérimentation ne nous a rien appris que nous ne savions déjà : l'information se trouve au niveau des pics d'horloge.

Dans le chapitre suivant, nous allons vous présenter une nouvelle attaque par observations qui présente une faible complexité de calcul. Mais surtout qui ne nécessite ni la connaissance du texte clair ni celle du texte chiffré pour être menée.

Chapitre 5

Attaque par signatures

L'IDÉE porteuse de cette thèse est d'utiliser plusieurs instants d'un algorithme afin de s'affranchir de la connaissance du message clair et du message chiffré qui sont généralement nécessaires à une attaque par observations. Dès 2007, des attaques ne nécessitant pas cette connaissance ont vu le jour [HP07, Jaf07]. Ces attaques combinent la cryptanalyse algébrique et les attaques par canaux cachés. La première de ces attaques montre qu'il est important de masquer l'ensemble de l'algorithme et non seulement une partie comme il était proposé par Akkar *et al.* dans [ABG04, AG03]. La seconde attaque vise un AES en mode compteur lorsque la valeur initiale du compteur n'est pas connue. Ces deux attaques reposent d'une part sur la capacité de l'attaquant à déterminer le poids de Hamming de certaines variables manipulées par le composant et d'autre part sur une bonne connaissance de l'algorithme attaqué afin de définir un système d'équations qui utilise les valeurs des poids de Hamming obtenues. Ces deux attaques sont les premières à avoir vu le jour et ont été généralisées par Renauld et Standaert dans [RS11] sous le nom d'attaques algébriques par canaux cachés. Ces attaques nous ont semblé intéressantes. Cependant, leur mise en œuvre est complexe et les équations peuvent être difficiles à résoudre. Le principal défaut des attaques algébriques par canaux cachés est leurs comportements face au bruit. En effet, lorsqu'on est en présence d'un bruit important, il est très fréquent que le poids de Hamming obtenu soit erroné. Dans ce cas, la résolution des équations issues de ces informations erronées peut être très difficile, voir impossible, et la clé retrouvée ne sera probablement pas celle recherchée. Actuellement de nombreux travaux sont effectués afin de pallier ce problème comme dans [OKPW10, ORSW12, MBZ+12, OW12]. Ces attaques peuvent être appliquées à une partie de l'algorithme ou à son ensemble. Elles diffèrent des attaques par observations classiques car elles n'utilisent pas les émanations du composant pour retrouver directement la clé secrète utilisée. Dans la suite, nous allons proposer une nouvelle attaque, l'attaque par signatures, qui tire partie de la connaissance du poids de Hamming de certaines variables afin de retrouver une clé secrète. Contrairement aux attaques algébriques par canaux cachés, l'attaque par signatures repose sur des propriétés statistiques plutôt que sur des propriétés algébriques, ce qui la rend bien plus tolérante au bruit. L'attaque par signatures permet de retrouver des sous-clés secrètes sans avoir à utiliser des équations complexes.

Dans ce chapitre, nous commencerons par présenter une première attaque basée sur la connaissance du poids de Hamming de certaines variables. Cette hypothèse peut sembler très restrictive, elle est néanmoins commune aux attaques algébriques par canaux cachés. Nous validerons cette attaque grâce à des simulations. Nous expliquerons ensuite pourquoi cette attaque est difficile à mettre en pratique et nous montrerons comment améliorer la méthode pour palier ces problèmes. Ceci nous permettra de pouvoir expérimenter notre attaque dans des conditions réelles en présentant une attaque plus générale qui peut être facilement mise en œuvre.

5.1 Première attaque par signatures

L'idée de l'attaque par signatures est d'utiliser la forme particulière des distributions conjointes de l'entrée et de la sortie d'une fonction utilisant une clé secrète.

5.1.1 Étude des distributions conjointes

Les attaques présentées dans les chapitres précédents estiment la clé secrète comme étant la clé qui va maximiser la mesure de dépendance entre $L(Z)$ et $m(X,k)$. Le modèle de la fuite dépend de la fonction φ conjecturée par l'attaquant qui représente la façon dont fuit le composant en fonction des variables manipulées mais aussi de la fonction intermédiaire attaquée, g. Nous allons utiliser ces deux fonctions pour échafauder les attaques par signatures.

Pour une première attaque par signatures, nous intéressons aux variables aléatoires $\varphi(X)$ et $\{\varphi(g(X,k))\}_{k\in\mathcal{K}}$. Nous étudierons plus particulièrement la distribution conjointe de ces deux variables.

Pour obtenir la distribution correspondant à la clé k, on calcule $\varphi(x)$ et $\varphi(g(x,k))$ pour tous les $x \in \mathcal{X}$, ce qui nous permet d'obtenir la loi de $\varphi(X)$ et celle de $\varphi(g(X,k))$.

Par exemple, si l'on considère la fonction :

$$\begin{array}{rccl} g: & \mathbb{Z}/2\mathbb{Z}\times\mathbb{Z}/2\mathbb{Z} & \longrightarrow & \mathbb{Z}/2\mathbb{Z} \\ & (x,k) & \longmapsto & z = x\oplus k, \end{array}$$

et si φ est définie par :

$$\begin{array}{rccl} \varphi: & \mathbb{Z}/2\mathbb{Z} & \longrightarrow & \mathbb{Z}/2\mathbb{Z} \\ & x & \longmapsto & x. \end{array}$$

Lorsque $k = 0$, on aura $\varphi(x) = \varphi(g(x,k))$. Par ailleurs, lorsque $k = 1$, $\varphi(x) = 1 - \varphi(g(x,k)) \bmod 2$.
Le Tableau 5.1 représente les distributions pour les clés 0 et 1. Ces deux distributions sont très clairement différentes.

$k = 0$

$\varphi(x)$ / $\varphi(z)$	0	1
0	1/2	0
1	0	1/2

$k = 1$

$\varphi(x)$ / $\varphi(z)$	0	1
0	0	1/2
1	1/2	0

Tableau 5.1 – Distributions conjointes de $(x, z = x \oplus k)$.

Cet exemple nous montre que les distributions considérées peuvent dépendre de la clé utilisée dans la fonction. Il est ici possible de distinguer si la clé utilisée est $k = 0$ ou $k = 1$ uniquement en observant ces distributions. Si les distributions sont différentes pour chaque clé, il est possible d'estimer quelle est la clé utilisée. Un grand nombre de fonctions cryptographiques possèdent cette propriété, comme nous le verrons plus loin.

Le plus souvent, la fonction φ peut être remplacée par le poids de Hamming de la variable manipulée. C'est ce que nous ferons par la suite. On peut également imaginer que la fonction φ soit remplacée par une autre fonction si l'attaquant possède des informations sur la fuite considérée.

Avec la seule connaissance de la fonction qu'il considère, un attaquant peut facilement construire ces distributions. Si de plus, il a accès aux poids de Hamming des entrées et des sorties de la fonction au cours de son exécution sur un composant, il peut construire

une distribution empirique. Il ne lui reste plus qu'à comparer cette distribution issue d'une clé secrète et celles qu'il a obtenues de manière théorique. La distribution théorique qui sera la plus proche de celle du composant sera issue de la même clé et l'attaquant aura ainsi estimé la valeur de la clé secrète. Il est donc possible de mener une attaque avec uniquement la connaissance du poids de Hamming des entrées et des sorties de la fonction attaquée et les distributions théoriques.

5.1.2 Formalisation

Nous allons maintenant formaliser cette attaque. Les éléments de $\mathcal{X}$ sont composés de $log_2(\mathbf{x})$ bits et ceux de $\mathcal{Z}$ de $log_2(\mathbf{z})$ bits. Notre attaque se décompose en trois phases distinctes.

Dans une première phase, nous calculons les distributions des paires $(\varphi(x), \varphi(g(x,k)))$, $\forall x \in \mathcal{X}$ et pour toutes les clés $k \in \mathcal{K}$ possibles. Dans la suite, ces distributions seront appelées les *signatures théoriques* et seront notées $S(g,k)$. Avec la notation précédente, on aura :

$$S(g,k) = \{p_{i,j} | i \in \{0,\ldots,log_2(\mathbf{x})\}, j \in \{0,\ldots,log_2(\mathbf{z})\}\}$$

avec $p_{i,j}$ la probabilité que $\varphi(x) = i$ et $\varphi(g(k,x)) = j$. Cette partie de l'attaque ne dépend pas du composant que l'on attaque mais uniquement de la fonction que l'on souhaite attaquer. Si les *signatures théoriques* ne dépendent pas de la clé k utilisée, l'attaque ne pourra pas être menée. Il faudra alors changer de fonction g. Il est possible aussi que les *signatures théoriques* ne soient pas toutes différentes et on ne pourra pas alors estimer de manière unique quelle sera la clé secrète utilisée par le composant. Si au moins deux des *signatures théoriques* sont différentes, on pourra obtenir de l'information sur la clé secrète. Ces distributions peuvent être réutilisées pour mener des attaques sur d'autres composants.

La seconde phase de l'attaque est liée au composant attaqué. On obtient plusieurs couples de la fuite des entrées et des sorties de la fonction g implantée sur le composant cible. La fonction g est alors toujours utilisée avec la même clé secrète $k^\star$. À partir des paires que l'on a obtenues, on calcule la fréquence du couple $(\varphi(x), \varphi(z))$. Ces fréquences nous donnent une distribution que l'on appellera *signature du composant* que l'on notera S_d et qui dépend de la clé $k^\star$ utilisée par le composant attaqué.

$$S_d = \{f_{i,j} | i \in \{0,\ldots,log_2(\mathbf{x})\}, j \in \{0,\ldots,log_2(\mathbf{z})\}\}$$

où $f_{i,j}$ est la fréquence du couple $(\varphi(x), \varphi(z)) = (i,j)$. Si les entrées considérées sont distribuées de manière uniforme, la *signature du composant* sera proche de la *signature théorique* de la clé secrète $k^\star$.

La troisième phase de l'attaque est la comparaison entre les *signatures théoriques* et la *signature du composant*. Il nous faut donc expliquer ce que sont deux distributions proches et pour cela, introduire la distance du χ^2. La distance du χ^2, bien connue des statisticiens, sert à comparer deux distributions. Plus cette distance est petite, plus les distributions sont considérées comme proches. La distance du χ^2 est définie par :

$$\chi^2(S(g,k), S_d) = \sum_{i=0}^{log_2(\mathbf{x})} \sum_{j=0}^{log_2(\mathbf{z})} \delta(p_{i,j}, f_{i,j})$$

La distance δ entre $p_{i,j}$ et $f_{i,j}$ est définie par :

$$\delta(p_{i,j}, f_{i,j}) = \begin{cases} \frac{(p_{i,j}-f_{i,j})^2}{p_{i,j}} & , p_{i,j} \neq 0 \\ 0 & , p_{i,j} = f_{i,j} \\ \infty & , p_{i,j} = 0 \neq f_{i,j} \end{cases}$$

Une bonne estimation de la clé secrète utilisée par le composant minimisera la distance entre $S(g,k)$ et S_d. On définit ainsi :

$$\hat{k^\star} = \underset{k}{\operatorname{argmin}}(\chi^2(S(g,k), S_d))$$

La mise en œuvre de l'attaque par signatures est décrite dans l'Algorithme 10.

algorithme 10 Attaque par signatures

Algorithme ATTAQUE PAR SIGNATURES (n paires (x_i, z_i))
$g : \mathcal{K} \times \mathcal{X} \to \mathcal{Z}$
  pour $k \in \mathcal{K}$ **faire**
    $S(g,k) \leftarrow 0$ ▷ $S(g,k) \in \{0 \dots log_2(\mathbf{x})\} \times \{0 \dots log_2(\mathbf{z})\}$
    pour $x \in \mathcal{X}$ **faire** ▷ Calcul des *signatures théoriques*
      $S(g,k)(\varphi(X), \varphi(g(k,X))) \leftarrow S(g,k)(\varphi(X), \varphi(g(k,X))) + \frac{1}{\mathbf{x}}$
    fin pour
  fin pour
  $S_d \leftarrow 0$ ▷ $S_d \in \{0 \dots log_2(\mathbf{x})\}$ x $\{0 \dots log_2(\mathbf{z})\}$
  pour $i \leftarrow 0, n-1$ **faire** ▷ Calcul de la *signature du composant*
    $S_d(x_i, z_i) \leftarrow S_d(x_i, z_i) + \frac{1}{n}$
  fin pour
  $key \leftarrow 0$
  pour $k \in \mathcal{K}$ **faire** ▷ Confrontation de la *signature du composant* aux *signatures théoriques*
    si $\chi^2(S(g,k), S_d) < \chi^2(S(g,key), S_d)$ **alors**
      $key \leftarrow k$
    fin si
  fin pour
retourner key
fin Algorithme

La première étape de l'algorithme requiert une multiplication et $\mathbf{k} \cdot \mathbf{x}$ additions pour le calcul des *signatures théoriques*. Pour la *signature du composant*, on a besoin d'effectuer une multiplication et n additions. Enfin, on doit effectuer $2 \cdot \mathbf{k} \cdot log_2(\mathbf{x}) \cdot log_2(\mathbf{z})$ multiplications et $\mathbf{k} \cdot log_2(\mathbf{x}) \cdot log_2(\mathbf{z})$ additions pour calculer $\mathbf{k}$ distances du χ^2. La complexité total de l'algorithme est donc de $2 \cdot (\mathbf{k} \cdot log_2(\mathbf{x}) \cdot log_2(\mathbf{z}) + 1)$ multiplications et $\mathbf{k} \cdot (\mathbf{x} + log_2(\mathbf{x})) + log_2(\mathbf{z})$ additions.

Cet algorithme nous montre que lorsque l'on connaît la fuite de certaines variables, on est capable de retrouver la clé secrète utilisée par le composant. Plus le nombre d'échantillons est élevé plus la *signature du composant* est proche de la *signature théorique* de la clé secrète. La *signature théorique* correspondant à la clé secrète et la *signature du composant* n'ont pas besoin d'être identiques. Il suffit que ces deux signatures soient proches, au sens de la distance du χ^2.

Avant de mettre en œuvre une attaque par signatures, il est important d'étudier les *signatures théoriques*.

5.1.3 Étude préalable des *signatures théoriques*

Une telle étude nous permet de connaître, *a priori*, la faisabilité de l'attaque au niveau algorithmique. Afin d'étudier les *signatures théoriques*, nous proposons trois indicateurs qui dépendent de la fonction g que l'on attaque. Ils seront donc les mêmes si l'on attaque une même fonction sur deux composants différents. Dans un premier temps, on calcule toutes les distances entre deux *signatures théoriques*.

$$\chi^2(S(g,k), S(g,k')),\ \forall k \neq k' \in \{0,\ldots,\mathbf{k}-1\}$$

Si la distance entre deux *signatures théoriques* est nulle, on ne pourra pas faire la différence entre les deux clés correspondantes.

Le premier indicateur i_1 est défini comme étant le nombre de cas où l'on ne pourra pas distinguer deux *signatures théoriques*. Si cet indicateur n'est pas nul, on ne pourra pas retrouver la clé secrète utilisée par le composant avec certitude ; mais on sera capable de donner un ensemble de clés réduit contenant cette clé secrète. L'indicateur i_2 est le nombre de paires (k,k') tel que $\chi^2(S(g,k), S(g,k')) = \infty$. Cet indicateur est intéressant pour la distance du χ^2 car on sait que cette distance peut être infinie. Plus cet indicateur sera important, plus les *signatures théoriques* seront séparées, dans le sens où un plus grand nombre de paires de *signatures théoriques* auront des supports disjoints. Ce second indicateur ne peut pas excéder $\mathbf{k}\cdot\mathbf{k}-1$, qui est le nombre de paires possibles. Le dernier indicateur représente la distance non nulle minimale entre deux *signatures théoriques* ${\chi^2}_{min}$.

$${\chi^2}_{min} = \min_{k\neq k'\in\{0,\ldots,\mathbf{k}-1\}} \{\chi^2(S(g,k), S(g,k')) > 0\} \tag{5.1}$$

Si cet indicateur a une valeur suffisamment grande, on sera en mesure de distinguer les différentes signatures.

Dans le Tableau 5.2, on présente nos trois indicateurs de notre attaque basée sur les distances entre *signatures théoriques* pour trois fonctions g différentes [1] en considérant que φ représente le poids de Hamming :

- la fonction de substitution de l'AES (SubBytes)
- l'addition modulo 256 : $g(x,k) \equiv k + x \bmod 256$
- l'addition suivie d'une mise au carré modulo 256 : $g(x,k) \equiv (k+x)^2 \bmod 256$

	$i_1 : \chi^2 = 0$	$i_2 : \chi^2 = \infty$	${\chi^2}_{min}$
$g(x,k) = \text{SubBytes}(k \oplus x)$	0	65274	0.154516
$g(x,k) \equiv (k+x) \bmod 256$	210	62666	0.008722
$g(x,k) \equiv (k+x)^2 \bmod 256$	256	64976	0.028375

Tableau 5.2 – Indicateur de l'efficacité de l'attaque par signatures pour les trois fonctions considérées.

En étudiant d'un peu plus près le Tableau 5.2, on constate que la fonction SubBytes de l'AES semble être une excellente candidate pour l'attaque par signatures. En plus du fait que l'indicateur i_1 soit nul pour cette fonction, la plupart des signatures sont séparées par la distance maximale. Pour la seconde fonction, l'addition modulo 256, beaucoup de clés ne peuvent pas être distinguées, néanmoins les ensembles de clés possibles qui seront proposés ne contiendront que deux clés. La troisième fonction semble être la moins adaptée à une attaque par signatures et comme pour l'addition modulo 256 les clés retrouvées iront par paires.

1. Le choix des deux dernières fonctions a été initialement motivé par le souhait d'attaquer l'algorithme de chiffrement par flot Rabbit [BVP+03] qui utilise ce type d'opérations.

5.1.4 Simulations

Dans cette section nous proposons d'expérimenter notre attaque à travers des simulations. Nous nous sommes placés dans le cadre peu réaliste où l'on considère que l'on connait exactement le poids de Hamming de x_i et de $g(x_i, k)$ pour $i \in \{1, \cdots, n\}$. Chaque simulation peut être décomposée de la manière suivante :

- Génération de n entrées aléatoires x_i et des sorties correspondantes pour $g(x_i, k^\star)$ pour la fonction g choisie avec une clé aléatoire fixe.
- Calcul des distances du χ^2 entre la *signature du composant* et les *signatures théoriques* grâce à l'Algorithme 10.
- Comptage du nombre de clés éliminées grâce à l'attaque par signatures, c'est-à-dire les clés dont les *signatures théoriques* sont plus éloignées que la distance entre la *signature du composant* et la *signature théorique* correspondant à la clé secrète.

Dans les expérimentations suivantes (Figure 5.1 à 5.3), nous représentons la moyenne du nombre de clés que l'on a pu éliminer sur 100 000 expérimentations pour différents nombres d'échantillons. Comme le laissait supposer notre étude préalable, l'attaque par signatures est très efficace si l'on considère la fonction SubBytes de l'AES et l'addition modulo 256 puisqu'avec seulement 50 traces, on a discriminé plus 90% des clés possibles. Pour la troisième fonction considérée, on constate qu'il faudra plus de traces pour réussir l'attaque. Pour l'addition modulo 256 ainsi que l'addition suivie d'une mise au carré modulo 256 on constate qu'il restera toujours deux clés entre lesquelles on ne pourra pas choisir.

Nous avons aussi effectué des simulations en utilisant les boites-S du DES. Nous avons utilisé ces boites-S comme une unique fonction avec 512 clés possibles. L'entrée de cette fonction est composée de trois paramètres : la donnée, la clé secrète et la boite-S à utiliser. La Figure 5.4 montre qu'avec moins de 100 échantillons, nous avons pu retrouver la clé secrète, mais également la boite-S utilisée.

L'attaque par signatures semble donc être efficace pour retrouver la clé secrète utilisée par une grande variété de fonctions. Cependant cette attaque est très dépendante de notre capacité à bien estimer les poids de Hamming des entrées et des sorties de la fonction que l'on attaque. En effet, comme on peut le constater dans l'expérimentation suivante, cette attaque est très sensible aux erreurs pouvant être introduite sur $\varphi(x_i)$ et $\varphi(g(x_i, k^\star))$.

5.1.5 Rencontre entre la première attaque par signatures et le monde réel

Les données obtenues en utilisant les canaux cachés sont toujours bruitées. Il est donc très peu probable de connaître le poids de Hamming exact de la variable manipulée. Dans cette section, nous allons introduire des erreurs dans la valeur des poids de Hamming utilisés dans l'attaque par signatures afin d'étudier le comportement de notre attaque face aux erreurs. La distance du χ^2 peut atteindre une valeur infinie dès qu'une des valeurs de la distribution observée comporte une valeur nulle alors que la valeur correspondante pour la distribution théorique est non nulle ($p_{i,j} = 0$ et $f_{i,j} \neq 0$). Or ce cas peut être fréquemment rencontré lorsque que l'on attaque un composant.

Il existe différentes manières d'estimer le poids de Hamming d'une variable manipulée à partir des canaux cachés. Nous pouvons par exemple utiliser une attaque par profilage [CRR02] comme il est proposé dans [RS11] ou des *clusters* comme dans [LMV+]. Mais même en utilisant ces techniques, il arrive très fréquemment d'introduire des erreurs dans les poids de Hamming que l'on va considérer et ces erreurs sont difficiles à distinguer. Dans [RS11], Renauld et Standaert obtiennent le poids de Hamming d'une variable avec une probabilité de 0.993. Cette probabilité est très élevée mais nous avons

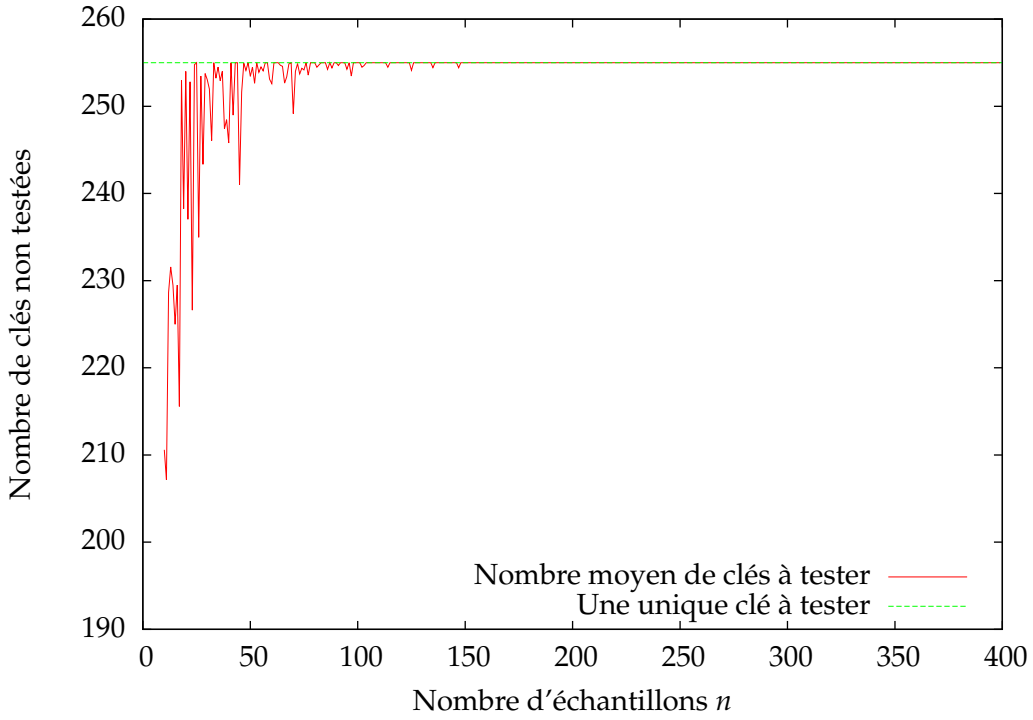


FIGURE 5.1 – Moyenne des résultats obtenus pour 100 000 attaques par signatures sur le *SubBytes de l'AES* pour différents nombres d'échantillons.

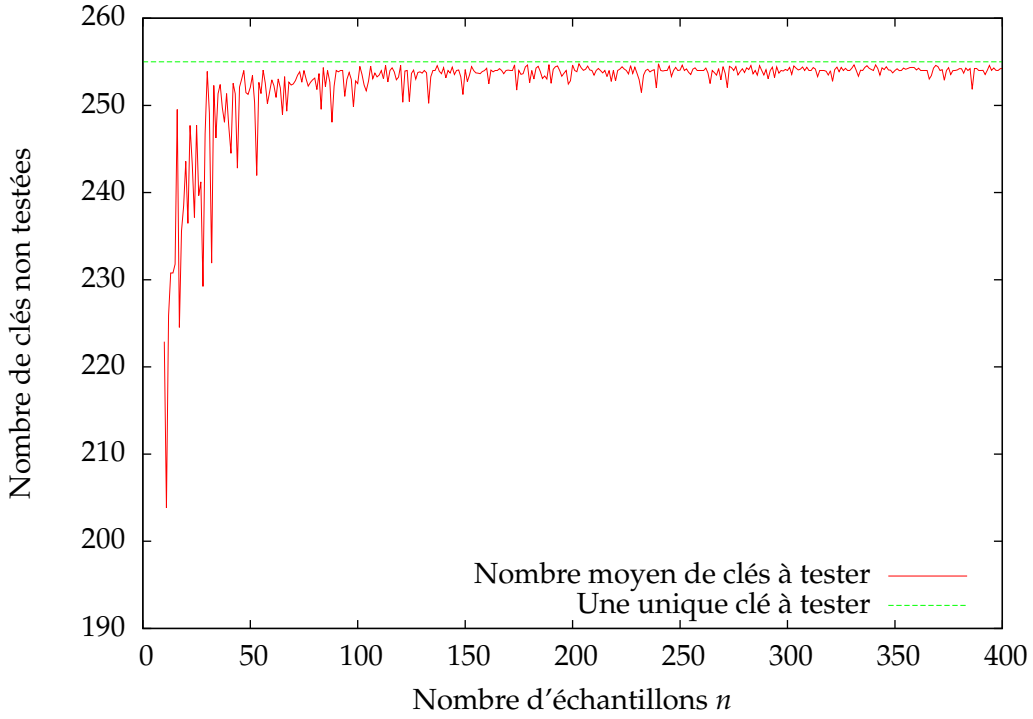


FIGURE 5.2 – Moyenne des résultats obtenus pour 100 000 attaques par signatures sur *l'addition modulo 256* pour différents nombres d'échantillons.

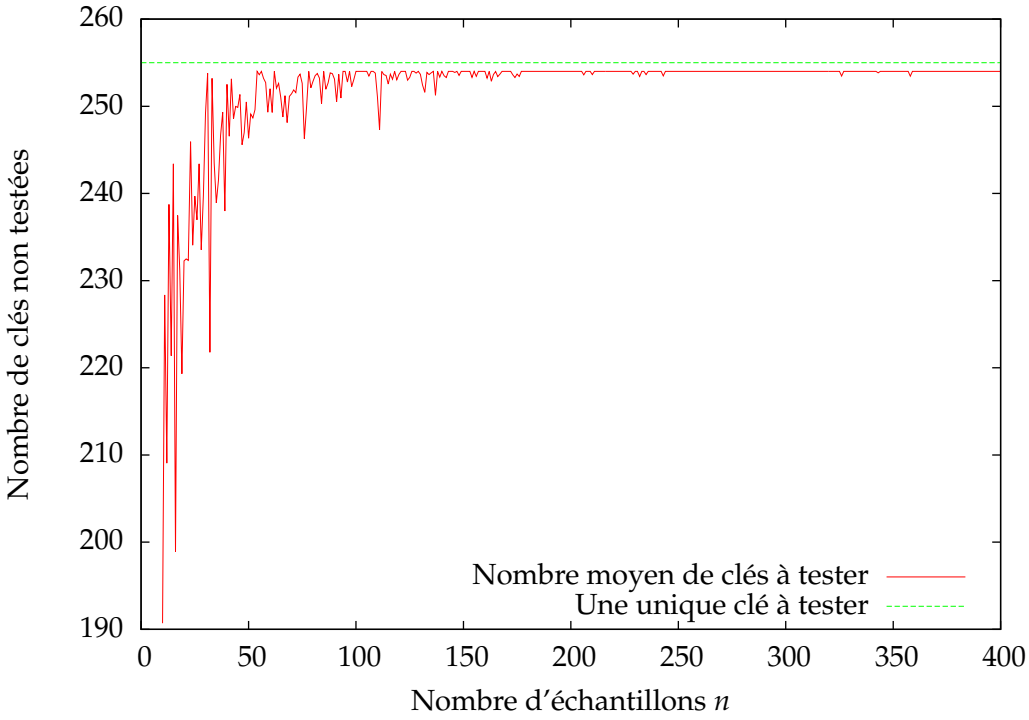


FIGURE 5.3 – Moyenne des résultats obtenus pour 100 000 attaques par signatures sur *l'addition suivie d'une mise au carré modulo 256* pour différents nombres d'échantillons.

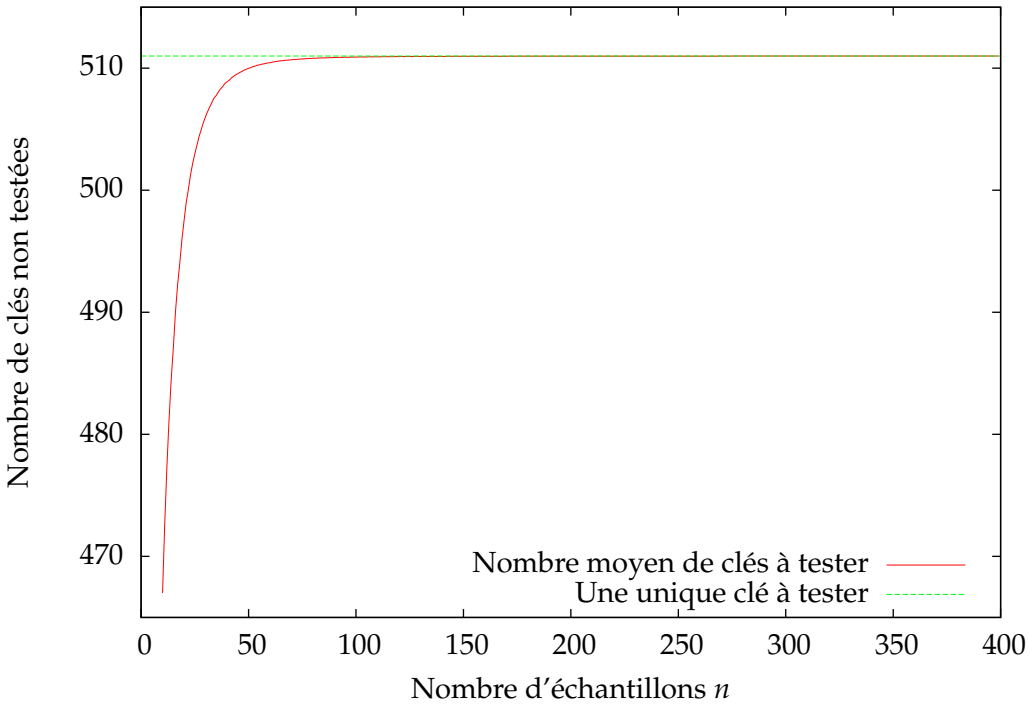


FIGURE 5.4 – Moyenne des résultats obtenus pour 100 000 attaques par signatures sur *les boites-S du DES* pour différents nombres d'échantillons.

besoin de retrouver un nombre important de poids de Hamming (les simulations précédentes suggèrent que l'on aura besoin d'estimer 150 couples de poids de Hamming). Or la probabilité d'obtenir 300 poids de Hamming sans erreur est de 0.993^{300} soit environ 0.122, ce qui est faible. Ainsi, il semble important d'étudier le comportement de notre attaque face à la présence d'erreurs dans les poids de Hamming que l'on utilise.

Nous proposons ici des expérimentations plus proches de la réalité que celles proposées précédemment. Pour cela, nous ajoutons un certain pourcentage d'erreurs aux poids de Hamming utilisés dans nos expérimentations. Nous proposons de considérer deux types d'erreurs. Tout d'abord une erreur aléatoire, c'est-à-dire que la valeur du poids de Hamming sera aléatoire. Le second type d'erreurs que nous allons considérer est une erreur pour laquelle on ajoutera ou on retranchera 1 du poids de Hamming. Une erreur de ce type sera appelée *petite erreur*.

Une simulation est maintenant décrite par :

- Génération de n entrées aléatoires, x_i, et des sorties correspondantes, $g(x_i, k^\star)$, pour la fonction SubBytes de l'AES choisie avec une clé aléatoire fixe..
- Calcul du poids de Hamming de ces entrées et sorties ($\varphi(x_i)$ et $\varphi(g(x_i, k^\star))$), ajout d'erreurs ($\varphi(x_i) + \epsilon_i^x$ et $\varphi(g(x_i, k^\star)) + \epsilon_i^z$) et enfin calcul de la *signature du composant*.
- Calcul des distances du χ^2 entre la *signature du composant* et les *signatures théoriques* grâce à l'Algorithme 10.

Si l'on souhaite par exemple ajouter 1% d'erreur à 100 valeurs, on tire aléatoirement un entier entre 0 et 99 et on modifie la valeur du poids de Hamming correspondant. Dans le cas d'une *petite* erreur, on tire aléatoirement un entier entre 0 et 1. Si cet entier est 0, on retranche 1 à la valeur considérée. S'il vaut 1, on ajoutera 1 au poids de Hamming.

Nous dirons qu'une attaque par signatures n'a pas échoué lorsque la distance entre la *signature du composant* et la *signature théorique* correspondant à la bonne clé n'est pas infinie. En effet, si cette distance est infinie on est certain que l'attaque échouera. Dans les Figures 5.5 et 5.6 nous faisons varier le nombre d'échantillons utilisés pour l'attaque par signatures et nous comptons le nombre de fois où l'attaque par signature n'a pas échouée. Les fréquences de ces deux figures sont obtenues en simulant 1 000 000 attaques par signatures, pour différentes probabilités d'erreurs : $0,01; 0,1; 0,5$. Les fréquences ainsi obtenues sont représentées dans la Figure 5.5 pour les *petites erreurs* et dans la Figure 5.6 pour les erreurs aléatoires.

Dans les Figures 5.5 et 5.6 on constate que la présence d'erreurs dans les poids de Hamming devient très rapidement problématique. Le fait que les attaques par signatures échouent lorsque des erreurs sont introduites est intimement lié à la distance utilisée lors de la comparaison de la *signature du composant* et les *signatures théoriques*. Deux stratégies ont été envisagées afin de résoudre le problème des erreurs de lecture. D'une part, adapter la distance du χ^2 afin de lui permettre d'absorber les *petites erreurs*, et d'autre part, essayer d'autres distances moins restrictives en cas d'erreurs de lecture.

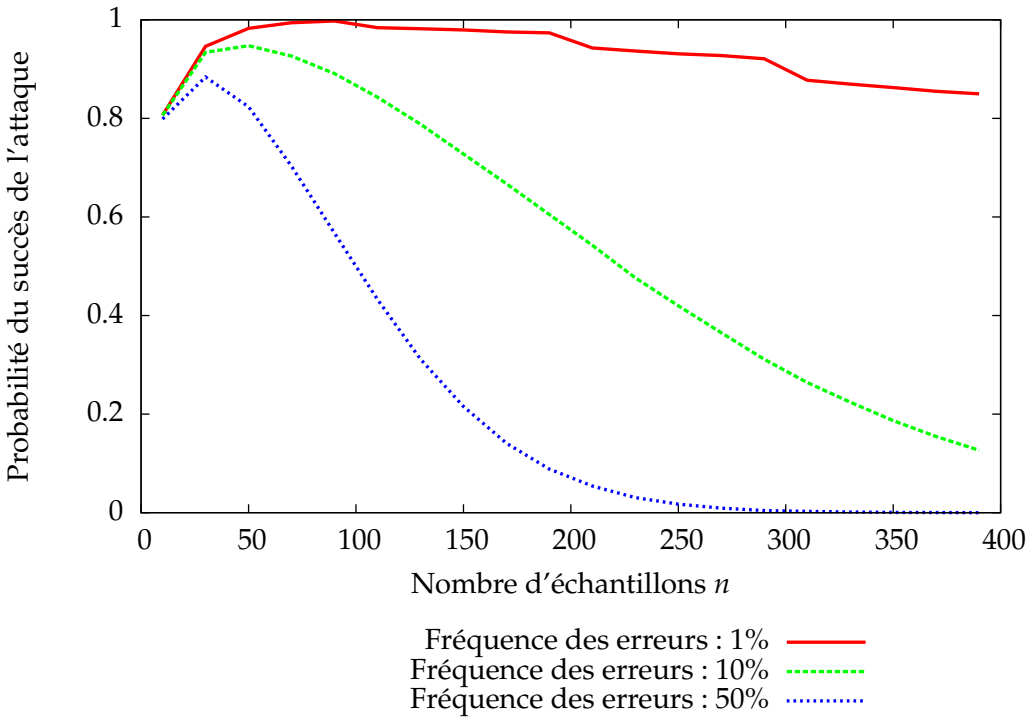


FIGURE 5.5 – Probabilité du possible succès des attaques par signatures sur la fonction SubBytes de l'AES lorsqu'on introduit de *petites erreurs* pour différents nombres d'échantillons.

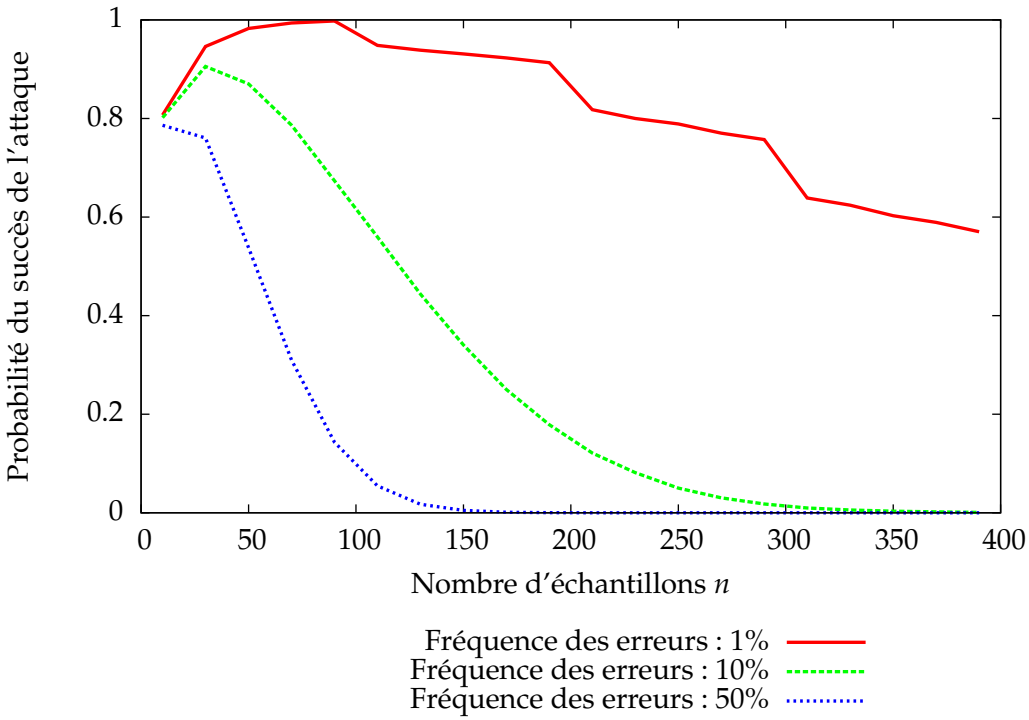


FIGURE 5.6 – Probabilité du possible succès des attaques par signatures sur la fonction SubBytes de l'AES lorsqu'on introduit des erreurs aléatoires pour différents nombres d'échantillons.

5.2 Attaque par signatures

5.2.1 Distance du χ^2 à fenêtre glissante

Nous proposons de recombiner les distributions $S(g,k)$ et S_d de la manière suivante :

$$S(g,k) = \{p_{0,0}, p_{0,1}, \ldots, p_{log_2(\mathbf{x}),log_2(\mathbf{z})}\}$$

devient

$$w_s(S(g,k)) = \{w_s(p_{0,0}), w_s(p_{0,1}), \ldots, w_s(p_{log_2(\mathbf{x})-2s,log_2(\mathbf{z})-2s})\}$$

et

$$S_d = \{f_{0,0}, f_{0,1}, \ldots, f_{log_2(\mathbf{x}),log_2(\mathbf{z})}\}$$

devient

$$w_s(S_d) = \{w_s(f_{0,0}), w_s(f_{0,1}), \ldots, w_s(f_{log_2(\mathbf{x})-2s,log_2(\mathbf{z})-2s})\}$$

.

Les nouvelles distributions $w_s(S(g,k))$ et $w_s(S_d)$ sont calculées à partir des distributions originales $S(g,k)$ et S_d à l'aide d'une fenêtre de taille s comme schématisé dans la Figure 5.7 :

$$w_s(p_{i,j}) = \sum_{x=i-s}^{i+s} \sum_{y=j-s}^{j+s} p_{x,y} \text{ et } w_s(f_{i,j}) = \sum_{x=i-s}^{i+s} \sum_{y=j-s}^{j+s} f_{x,y}.$$

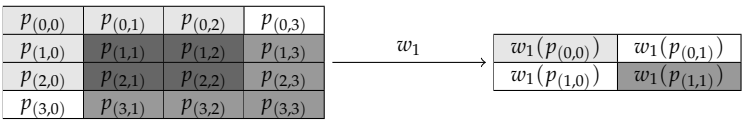


FIGURE 5.7 – Recombinaison d'une distribution en vue de calculer une distance du χ^2 à fenêtre glissante pour fenêtre de taille 1.

Et finalement nous calculons $\chi^2(w_s(S(g,k)), w_s(S_d))$. Nous noterons dans la suite cette distance $\chi^2_s(S(g,k), S_d)$, où s est la taille de la fenêtre. Elle nous permet d'absorber au moins des *petites erreurs* de plus ou moins la taille de la fenêtre. Comme cette distance est faite pour absorber les erreurs, elle réduit aussi les différences entre les distributions que l'on souhaite comparer. Le choix de la taille de la fenêtre est donc très important et une attention particulière doit y être apportée.

Afin de valider l'intérêt de l'utilisation de ces nouvelles distances, nous les avons étudier au travers des indicateurs proposés dans la Section 5.1.3. Le Tableau 5.3 présente les indicateurs d'efficacité de l'attaque par signatures pour la fonction SubBytes de l'AES lorsque l'on utilise la distance χ^2_s pour différents s. Ce tableau nous montre que si l'on souhaite attaquer la fonction SubBytes de l'AES, il ne faut pas prendre une taille de fenêtre plus grande que 2. D'une manière générale, les simulations ont montré qu'une taille de fenêtre de 1 est suffisante pour obtenir d'excellents résultats.

	$i_1 : \chi^2_s = 0$	$i_2 : \chi^2_s = \infty$	$\chi^2_{s\,min}$
$s = 0$	0	65274	0.154516
$s = 1$	0	0	0.017322
$s = 2$	0	0	0.001147
$s = 3$	4424	0	0.000015

Tableau 5.3 – Indicateurs de l'efficacité de l'attaque par signatures pour la fonction SubBytes de l'AES lorsque l'on utilise la distance χ^2_s, pour différentes valeurs de s.

5.2.2 Autres distances

La distance du χ^2 à fenêtre glissante permet de se prémunir du cas où un poids de Hamming erroné serait utilisé. Malheureusement, cette distance permet uniquement de faire face aux *petites erreurs*. Il existe un grand nombre d'autres distances qui pourrait convenir à une attaque par signatures. Dans l'article [Cha07], l'auteur présente une étude très complète de distances pouvant être utilisées pour comparer deux densités de probabilité, $P = (P_1, ..., P_n)$ et $Q = (Q_1, ..., Q_n)$, lorsqu'elles sont représentées en utilisant des histogrammes (comme présenté dans la Section 2.3.4.1). Pour une attaque par signatures, les signatures peuvent être vues comme des densités de probabilités représentées de cette manière. L'auteur de [Cha07] commence par présenter les très classiques : distance euclidienne, distance de Manhattan, distance de Minkowski et distance de Chebychev (Tableau 5.4).

Euclidienne	$d_{L_2}(P,Q) = \sqrt{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert^2}$
Manhattan	$d_{L_1}(P,Q) = \sum_{i=1}^{n} \lvert P_i - Q_i \rvert$
Minkowski	$d_{L_p}(P,Q) = \sqrt[p]{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert^p}$
Chebychev	$d_{L_\infty}(P,Q) = \max_i \lvert P_i - Q_i \rvert$

Tableau 5.4 – Distances classiques.

Le cas des attaques par signatures est particulier car on cherche la *signature théorique* qui minimisera la distance avec la *signature du composant*. Ainsi, des distances différentes pourront donner les mêmes résultats lorsqu'elles seront utilisées dans une attaque par signatures. On dira que deux distances, d_1, d_2 sont *équivalentes* si elles donnent les mêmes résultats lors d'une attaque par signatures, c'est à dire s'il y a conservation de l'ordre lorsque l'on change de distance. S'il existe une fonction T croissante telle que $\forall P, Q,\ d_1(P,Q) = T(d_2(P,Q))$ alors d_1 et d_2 sont *équivalentes*.

Certaines distances citées dans l'article [Cha07] sont *équivalentes* à la distance de Manhattan.

La distance de Sørensen, par exemple, est définie par :

$$\frac{\sum_{i=1}^{n} |P_i - Q_i|}{\sum_{i=1}^{n} (P_i + Q_i)} = \frac{\sum_{i=1}^{n} |P_i - Q_i|}{\sum_{i=1}^{n} (P_i) + \sum_{i=1}^{n} (Q_i)} = \frac{\sum_{i=1}^{n} |P_i - Q_i|}{2} = \frac{1}{2} d_{L_1}(P,Q)$$

En effet, $\sum_{i=1}^{n} P_i = \sum_{i=1}^{n} Q_i = 1$, car P et Q sont des densités de probabilités. La fonction permettant de transformer la distance de Sørensen en distance de Manhattan est la fonction $T(x) = 2x$. Ces deux distances sont donc *équivalentes*. De la même manière, la distance de Manhattan est *équivalente* aux distances de Sørensen, de Gower, de Lorentzian, de l'intersection, de Czekanowski. Nous présenterons dans la suite uniquement des distances non *équivalentes* et nous citerons à titre indicatif des distances *équivalentes* à des distances déjà présentées.

Le second groupe de distances proposé par l'auteur sont les distances issues de la distance de Manhattan. Le Tableau 5.5 présente ces distances.

Soergel	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{n} max(P_i, Q_i)}$
Kulczynski	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{n} min(P_i, Q_i)}$
Canberra	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{n} P_i + Q_i}$

Tableau 5.5 – Distances issues de la distance de Manhattan.

La distance de Soergel est *équivalente* à la distance de Tanimoto. De même, la distance de Canberra est *équivalente* à la distance du χ^2 carré [2] et à la distance probabiliste symétrique du χ^2.

Le troisième groupe proposé dans l'article [Cha07] est la famille des intersections que nous avons présentée dans le Tableau 5.6

Wave Hedges	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert}{max(P_i, Q_i)}$
Motyka	$\frac{\sum_{i=1}^{n} max(P_i, Q_i)}{\sum_{i=1}^{n} P_i + Q_i}$
Ruzicka	$\frac{\sum_{i=1}^{n} min(P_i, Q_i)}{\sum_{i=1}^{n} max(P_i, Q_i)}$

Tableau 5.6 – Distances des intersections.

Le quatrième groupe est lié au produit scalaire (Tableau 5.7).

Produit scalaire	$1 - \sum_{i=1}^{n} P_i \cdot Q_i$
Moyenne harmonique	$1 - 2\sum_{i=1}^{n} \frac{P_i \cdot Q_i}{P_i + Q_i}$
Cosinus	$1 - \frac{\sum_{i=1}^{n} P_i \cdot Q_i}{\sqrt{\sum_{i=1}^{n} {P_i}^2} \sqrt{\sum_{i=1}^{n} {Q_i}^2}}$
Jaccard	$\frac{\sum_{i=1}^{n} (P_i - Q_i)^2}{\sum_{i=1}^{n} {P_i}^2 \sum_{i=1}^{n} {Q_i}^2 \sum_{i=1}^{n} P_i \cdot Q_i}$
Dice	$\frac{\sum_{i=1}^{n} (P_i - Q_i)^2}{\sum_{i=1}^{n} {P_i}^2 \sum_{i=1}^{n} {Q_i}^2}$

Tableau 5.7 – Distances basées sur le produit scalaire.

Ce groupe est très intéressant pour mener une attaque par signatures. En effet si l'on compare par exemple, la distance du produit scalaire et la distance de Manhattan, on remarque que les erreurs sont atténuées par la distance du produit scalaire alors qu'elles sont exacerbées par la distance de Manhattan. Notons $\epsilon = \{\epsilon_i | i \in \{1, \cdots, n\}\}$ le vecteur représentant les erreurs et ϵ_i la valeur de l'erreur pour la i-ème coordonnée. Le vecteur ϵ doit satisfaire :

$$\sum_{i=1}^{n} P_i + \epsilon_i = 1$$

et

$$0 \leq p_i + \epsilon_i \leq 1 \forall i \in \{1, \cdots, n\}.$$

2. Cette distance est différente de la distance présentée dans la Section 5.1.2 et est définie dans [Cha07]

Dans une attaque par signatures, il est important que la distance entre une signature et la même signature à laquelle on ajoute quelques erreurs soit la plus faible possible. Étudions la distance entre les densités de probabilité P et $P + \epsilon$ pour la distance de Manhattan et pour la distance du produit scalaire. Pour la distance de Manhattan (d_{L_1}) on a :

$$\sum_{i=1}^{n} |P_i - (P_i + \epsilon_i)| = \sum_{i=1}^{n} |\epsilon_i| = d_{L_1}(P, P) + \sum_{i=1}^{n} |\epsilon_i|.$$

D'autre part pour la distance du produit scalaire (d_{ps}), on a :

$$1 - \sum_{i=1}^{n} P_i \cdot (P_i + \epsilon_i) = 1 - \sum_{i=1}^{n} P_i \cdot P_i - \sum_{i=1}^{n} P_i \cdot \epsilon_i = d_{ps}(P, P) - \sum_{i=1}^{n} P_i \cdot \epsilon_i.$$

Comme $P_i < 1, \forall i \in \{1, \cdots, n\}$ et $|\epsilon_i| \geq 0, \forall i \in \{1, \cdots, n\}$ on en déduit :

$$|\epsilon_i| \geq P_i \cdot |\epsilon_i| \geq -P_i \cdot \epsilon_i, \forall i \in \{1, \cdots, n\}.$$

On en conclut que la distance du produit scalaire est mieux adaptée à une attaque par signatures que la distance de Manhattan car l'impact des erreurs est plus important pour la distance de Manhattan que pour la distance du produit scalaire.

Le groupe suivant est représenté par la distance de fidélité et la distance des cordes carrées dans le Tableau 5.8. La distance de fidélité est *équivalente* aux distances de Bhattacharyya, de Hellinger et de Matusita.

Fidélité	$1 - \sum_{i=1}^{n} \sqrt{P_i \cdot Q_i}$
Corde carrée	$\sum_{i=1}^{n} \left(\sqrt{P_i} - \sqrt{Q_i} \right)^2$

Tableau 5.8 – Distance de fidélité et distance des cordes carrées.

Le sixième groupe est basé sur la distance du χ^2 que nous avons utilisée à l'origine pour notre attaque. Nous présentons ce groupe dans le Tableau 5.9. La distance de divergence est *équivalente* à la distance de Clark.

χ^2	$\sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{P_i}$
χ^2 Pearson	$\sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{Q_i}$
Divergence	$2 \sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{(P_i + Q_i)^2}$
χ^2 additive symétrique	$\sum_{i=1}^{n} \frac{(P_i - Q_i)^2 \cdot (P_i + Q_i)}{P_i \cdot Q_i}$

Tableau 5.9 – Distances issues de la distance du χ^2.

Une autre façon de mesurer la distance entre deux signatures est basée sur l'entropie de Shannon. Le Tableau 5.10 présente les distances recensées dans [Cha07] et issues de l'entropie de Shannon. La distance de Jensen-Shannon est *équivalente* à la distance de Topsøe.

Kullback-Leibler	$\sum_{i=1}^{n} P_i \cdot ln(\frac{P_i}{Q_i})$
Jeffreys	$\sum_{i=1}^{n} (P_i - Q_i) \cdot ln(\frac{P_i}{Q_i})$
K divergence	$\sum_{i=1}^{n} P_i \cdot ln(\frac{2 \cdot P_i}{P_i + Q_i})$
Topsøe	$\sum_{i=1}^{n} P_i \cdot ln(\frac{2 \cdot P_i}{P_i + Q_i}) + Q_i \cdot ln(\frac{2 \cdot Q_i}{P_i + Q_i})$
Différence de Jensen	$\sum_{i=1}^{n} \frac{P_i \cdot ln(P_i) + Q_i \cdot ln(Q_i)}{2} - \frac{P_i + Q_i}{2} \cdot ln(\frac{P_i + Q_i}{2})$

Tableau 5.10 – Distances basées sur l'entropie de Shannon.

Le pénultième groupe présenté dans [Cha07] présente les distances basées sur des combinaisons d'autres distances présentées précédemment (Tableau 5.11).

Taneja	$\sum_{i=1}^{n} \frac{P_i + Q_i}{2} \cdot ln(\frac{P_i + Q_i}{2\sqrt{P_i \cdot Q_i}})$
Kumar-Johnson	$\sum_{i=1}^{n} \frac{(P_i^2 - Q_i^2)^2}{2(P_i \cdot Q_i)^{\frac{3}{2}}}$
Moyenne(L_1, L_∞)	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert + \max_i \lvert P_i - Q_i \rvert}{2}$

Tableau 5.11 – Distances basées sur des combinaisons de distances.

Enfin le dernier groupe proposé est composé de variations sur des distances présentées précédemment. Nous avons présenté ces distances dans le Tableau 5.12.

Vicis-Wave Hedges	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert}{min(P_i, Q_i)}$
Vicis-Symmetric χ^2	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert^2}{min(P_i, Q_i)^2}$
Vicis-Symmetric χ^2	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert^2}{min(P_i, Q_i)}$
Vicis-Symmetric χ^2	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert^2}{max(P_i, Q_i)}$
Max Symmetric χ^2	$max(\sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{P_i}, \sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{Q_i})$
Min Symmetric χ^2	$min(\sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{P_i}, \sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{Q_i})$

Tableau 5.12 – Distances basées sur des combinaisons de distances.

5.2.3 Comparaison des distances pour les attaques par signatures

Nous avons exploré les différentes distances (non *équivalentes*) afin de trouver la distance qui serait la plus adaptée à une attaque par signatures contre une implémentation de l'AES dans le cas de *petites* erreurs mais aussi dans le cas d'erreurs aléatoires. Pour cela, nous avons simulé 10 000 attaques par signatures en imposant un taux d'erreurs de 50% et compté pour chaque distance combien de *signatures théoriques* sont plus proches de la *signature du composant* simulée que de celles correspondant à la clé secrète. Ce nombre représente le nombre de clés restant à tester et est illustré dans les différents graphiques qui

suivent. L'annexe B contient des graphiques plus détaillés. Dans un premier temps, nous avons représenté les résultats pour les petites erreurs dans la Figure 5.8. Il est intéressant de remarquer qu'un grand nombre de distances considérées donne des résultats proches voire équivalents. Néanmoins pour les *petites erreurs*, trois distances ne donnent aucun résultat (χ_2 , Taneja et Kumar-Johnson) car elles peuvent prendre des valeurs infinies lors de la division par zéro. Sept distances donnent des résultats bien moins bons que les autres : Chebychev, Canberra, Wave Hedges, divergence, Jensen différence, Vicis-Wave Hedges et Vicis-Symmetric χ^2.

On observe que la distance la plus efficace pour mener à bien une attaque par signatures lorsque les poids de Hamming considérés sont biaisés par de *petites* erreurs, est la distance du χ^2 à fenêtre glissante de taille 1, χ_1^2. Ceci est dû à la fenêtre glissante mise en œuvre qui permet d'absorber les erreurs introduites. Mais, il est plus intéressant de ne pas avoir à remanier les signatures avant d'appliquer les distances. En effet en faisant cela, on risque de perdre une partie de l'information contenue dans les distributions. C'est pour cela que l'on n'a pas cherché à appliquer cette technique de fenêtre sur d'autres distances car seule la distance du χ^2 peut prendre la valeur infinie. On remarque sur la figure précédente que certaines distances sont bien meilleures que d'autres. La distance du produit scalaire, de Cosine et de Pearson χ^2 offrent de très bons résultats. La distance χ_2^2 ne produit pas de résultats concluants car la fenêtre considérée est trop grande pour avoir une bonne séparation des distributions.

Dans la Figure 5.9 nous avons représenté les résultats des expériences menées dans les même conditions que précédemment (10 000 simulations et 50% d'erreurs) pour des erreurs aléatoires.

On constate que, comme dans le cas des *petites* erreurs, les distances ne réagissent pas toutes de la même manière en présence d'erreurs aléatoires. Celles qui ne sont vraiment pas adaptées aux attaques par signatures lorsque l'on considère des erreurs aléatoires, sont bien plus nombreuses que dans le cas des *petites* erreurs. Une distance a de bien meilleurs résultats que toutes les autres : la distance de Pearson χ^2. D'autres distances, bien qu'ayant de moins bons, ont des résultats plutôt satisfaisants comme la distance de Kullback-Leibler, la famille *Vicis-Symmetric*, la distance du produit scalaire ou la distance des moyennes harmoniques.

Si l'on n'a aucune connaissance du type d'erreurs auquel on est confronté, cinq distances donneront de bons résultats :

— La distance χ_1^2.
— La distance du χ^2 de Pearson.
— La distance du produit scalaire.
— La distance de Kullback-Leibler.
— La distance des moyennes harmoniques.

Nous avons présenté ici les résultats obtenus en considérant que la fonction attaquée est la fonction SubBytes de l'AES. On obtient des résultats similaires en considérant les boites-S du DES ou encore l'addition modulo 256 ou l'addition suivie d'une mise au carré modulo 256. Nous avons ici simulé le comportement de notre attaque en présence d'un bruit et nous avons proposé d'utiliser d'autres distances que la distance du χ^2 afin d'améliorer sa résistance au bruit. Afin de pouvoir faire des expérimentations sur un composant réel, il nous faut encore proposer une manière d'estimer les poids de Hamming que l'on va utiliser pour l'attaque par signatures.

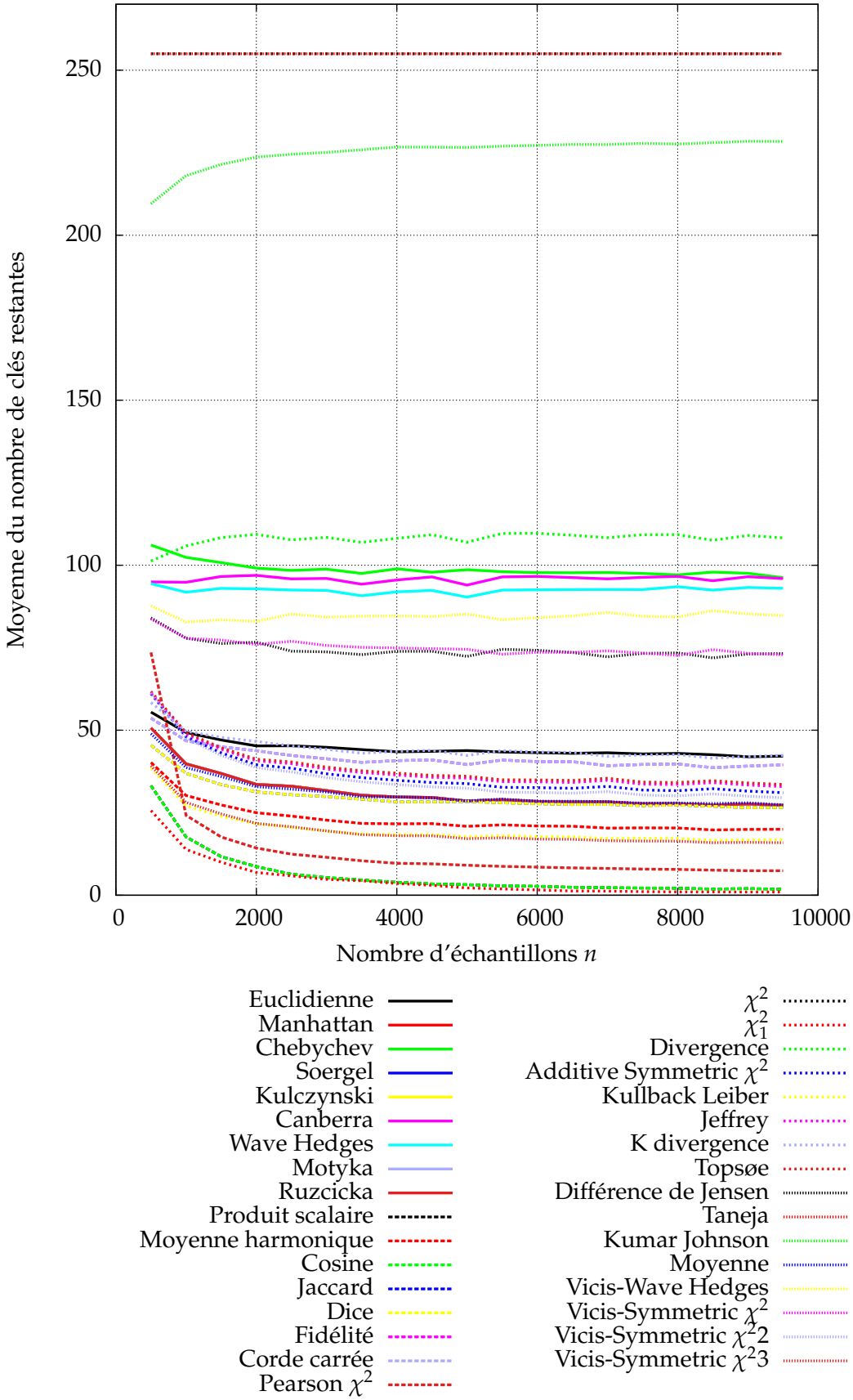


FIGURE 5.8 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour l'ensemble des différentes distances et 50% de *petites* erreurs.

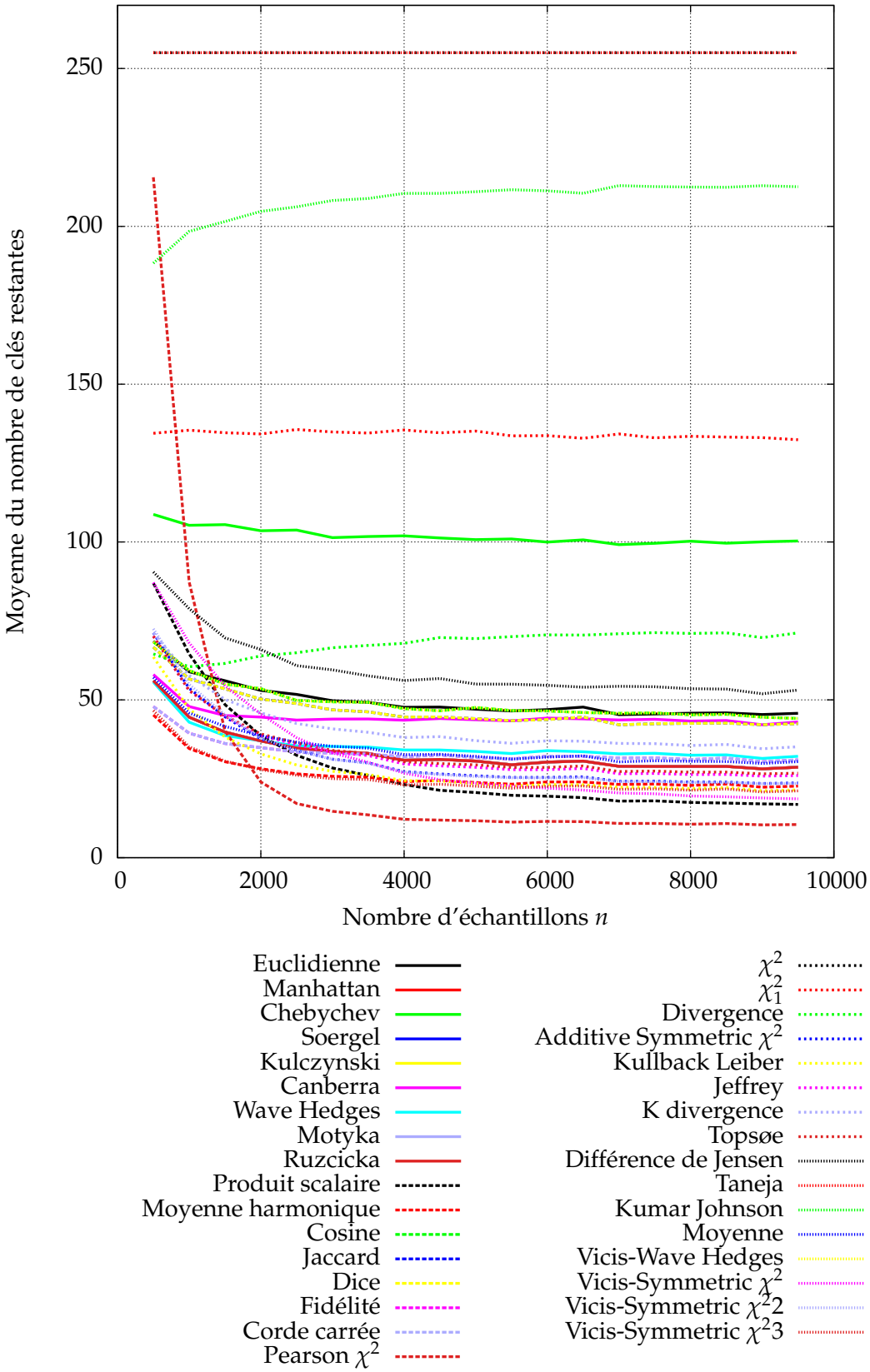


FIGURE 5.9 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour l'ensemble des distances différentes et 50% d'erreurs aléatoires.

5.3 Estimation du poids de Hamming

Il existe dans la littérature plusieurs propositions visant à retrouver le poids Hamming de certaines variables manipulées par un composant grâce aux informations contenues dans les signaux compromettants. Malheureusement ces propositions tendent à calculer une estimation la plus précise possible et pour cela requiert beaucoup d'informations sur le composant que l'on souhaite attaquer. Les méthodes proposées dans [RS11, LMV$^+$] reposent sur les attaques par profilage. Ces méthodes permettent d'avoir d'excellentes approximations du poids de Hamming des variables manipulées. Notre attaque étant résistante aux erreurs pouvant être faites dans l'estimation du poids de Hamming, nous avons voulu proposer un nouvel estimateur moins restrictif. Cet estimateur est très grossier comme nous le verrons dans nos expérimentations et ne peut certainement pas être utilisé pour mener à bien une attaque algébrique par canaux cachés par exemple. Néanmoins cet estimateur est suffisamment précis pour que l'attaque par signatures puisse être mise en œuvre.

L'amplitude d'une fuite, à un instant donné, dépend généralement du poids de Hamming de la variable manipulée à cet instant. Cette relation est souvent une relation d'ordre. C'est à dire que plus le poids de la variable manipulé est important plus l'amplitude de la fuite le sera (si la relation est croissante). Notre estimateur tire partie de cette particularité comme nous allons le voir.

Soit $l_i(t)$ un ensemble de n mesures correspondant au même instant t de n traces. On commence par trier cet ensemble par ordre croissant. On peut supposer sans perte de généralité que les valeurs des données que l'on considère sont uniformes. On connait ainsi la distribution du poids de Hamming qui sera très particulière. Si l'on prend l'exemple de l'ensemble $\mathcal{X}$ le poids de Hamming maximal est $log_2(\mathbf{x})$. Pour l'ensemble des n éléments de $l_i(t)$, on aura $\frac{n \cdot C^p_{log_2(\mathbf{x})}}{\mathbf{x}}$ éléments qui auront un poids de Hamming de p. On attribuera le poids de Hamming maximal aux plus grandes valeurs de notre ensemble $l_i(t)$. A contrario, les valeurs les plus petites se verront appliquer un poids de Hamming de 0.

Considérons par exemple que notre ensemble $l_i(t)$ est composé de 100 éléments et que la variable manipulée à l'instant t est une variable représentée par 2 bits. On associera le poids de Hamming de la manière suivante :

— 2 pour les $\frac{100 \cdot C^2_2}{2^2} = 25$ éléments ayant les plus grandes valeurs.

— 1 pour les $\frac{100 \cdot C^1_2}{2^2} = 50$ éléments suivants.

— 0 pour les $\frac{100 \cdot C^0_2}{2^2} = 25$ éléments ayant les plus petites valeurs.

On peut aussi classer l'ensemble $l_i(t)$ par ordre décroissant et inverser l'attribution des valeurs du poids de Hamming si la fuite est inversement proportionnelle au poids de Hamming. La qualité de cet estimateur dépend de la qualité des traces que l'on a obtenues, mais aussi du choix de l'instant t.

Nous allons maintenant présenter des expérimentations réalisées sur un composant implémentant un AES logiciel.

5.4 Expérimentations

Pour valider notre attaque, nous l'avons expérimentée sur une implémentation d'un AES logiciel sur un composant ATMega2561. La représentation interne des données est faite sur huit bits. Tout d'abord, nous avons validé notre méthode d'estimation du poids de Hamming. Pour cela, nous avons commencé par choisir les instants des signaux à considérer. Dans [APcXSQ06, GBTP08, GLRP06] les auteurs proposent différentes ma-

nières pour déterminer ces instants que l'on appellera *points d'intérêt*. Nous avons choisi d'utiliser les instants ayant la plus grande variance. Ce choix est délicat et nous détaillerons dans la suite. Ce procédé est simple mais suffisant dans le cadre de notre expérimentation.

Dans un premier temps, nous avons acquis 1 000 signaux d'émanations électromagnétiques de notre composant. Sur ces signaux, on peut distinguer chaque étape de la première ronde de l'AES comme l'on peut le voir sur la Figure 5.10. Puis nous avons calculé la variance de ces signaux en chaque instant qui est représentée dans la Figure 5.11. Ces deux figures sont synchronisées. Nous avons choisi de nous intéresser aux neuf instants des signaux qui possèdent une variance supérieure à dix fois la variance moyenne. Si ces *points d'intérêt* sont situés en dehors des phases de l'algorithme qui nous intéresse, nous ne les considérerons pas. Par exemple, l'un de ces points d'intérêt est situé en tout début de trace, avant même la phase AddRoundKey et SubBytes, c'est-à-dire avant que les variables considérées pour l'attaque soient normalement manipulées. Quatre de ces instants sont compris dans la phase de SubBytes et peuvent donc correspondre aux entrées de la fonction SubBytes (SB) de l'AES que l'on va attaquer. Les quatre derniers instants appartiennent à la région correspondant à la fonction MixColumn (MC) et peuvent donc être associés à la sortie de la fonction que l'on souhaite attaquer.

Afin de valider notre estimateur, nous avons comparé les valeurs obtenues grâce à notre méthode et les valeurs réelles pour chacun des 16 octets manipulés par l'algorithme. Le Tableau 5.13 (respectivement Tableau 5.14) présente le pourcentage d'estimations exactes du poids de Hamming pour les quatre *points d'intérêt* de la région de la fonction SubBytes (respectivement de la région du MixColumn).

PoI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PoI SB 1	24%	21%	28%	18%	78%	22%	24%	29%	23%	24%	29%	24%	23%	21%	24%	25%
PoI SB 2	21%	19%	20%	27%	24%	21%	25%	24%	26%	21%	29%	22%	24%	24%	28%	23%
PoI SB 3	25%	24%	21%	27%	22%	29%	81%	26%	31%	21%	24%	28%	27%	29%	26%	24%
PoI SB 4	22%	23%	24%	68%	21%	27%	23%	25%	23%	31%	25%	21%	23%	25%	21%	22%

Tableau 5.13 – Pourcentage de bonne estimation du poids de Hamming pour les entrées de la fonction que l'on souhaite attaquer pour l'ensemble des 16 octets de l'état.

PoI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PoI MC : 1	18%	21%	25%	27%	22%	24%	27%	29%	23%	22%	28%	20%	23%	21%	28%	29%
PoI MC : 2	25%	24%	21%	27%	22%	29%	73%	26%	31%	21%	24%	28%	27%	29%	26%	24%
PoI MC : 3	23%	25%	28%	75%	25%	25%	24%	28%	21%	24%	21%	24%	22%	24%	29%	23%
PoI MC : 4	24%	21%	28%	18%	64%	22%	24%	29%	23%	24%	29%	24%	23%	21%	24%	25%

Tableau 5.14 – Pourcentage de bonne estimation du poids de Hamming pour les sorties de la fonction que l'on souhaite attaquer pour l'ensemble des 16 octets de l'état.

Cette implémentation utilise les données octet par octet. Notre estimateur associera beaucoup de bonnes valeurs du poids de Hamming pour, au plus, un octet de l'état (cf. Section 1.4). Par exemple, le premier *point d'intérêt* de la zone du SubBytes peut être associé au quatrième octet de l'état avant la fonction AddRoundKey. Pour les autres octets le nombre de poids de Hamming correct sera proche de celui obtenu aléatoirement en utilisant uniquement la distribution du poids de Hamming. Notre estimateur donne une bonne approximation du poids de Hamming pour six octets différents.

Nous allons maintenant expérimenter le comportement de l'attaque par signatures sur notre implémentation. Nous avons choisi d'utiliser la distance du produit scalaire qui, d'après notre étude de la Section 5.2.2, fait partie des trois meilleures distances dans le cas des *petites erreurs* mais donne aussi de très bons résultats dans le cas des erreurs

aléatoires. Les *points d'intérêt* que l'on a trouvés précédemment correspondent aux octets 3, 4 et 6 de l'état avant la fonction AddRoundKey et avant la fonction MixColumn. Nous avons quatre *points d'intérêt* à considérer dans chacune des zones qui nous intéressent, donc $4 \times 4 = 16$ couples (x, z) à tester. Le Tableau 5.15 représente les résultats de l'attaque par signatures pour chacun des couples (x, z) lorsqu'on utilise la distance du produit scalaire. Nous indiquons la valeur de la clé qui correspond à la distribution théorique la plus proche de la distribution du composant, ainsi que la valeur de la distance associée. La dernière colonne précise si l'octet retrouvé est réellement un octet de la clé secrète. Nous avons classé ces résultats en fonction de la distance entre la *signature du composant* et la *signature théorique* la plus proche. Si l'on considère quatre *points d'intérêts* dans chaque zone, une attaque par signatures ne nécessite que quelques secondes pour être effectuée.

Nous savons que l'attaque a réussi car les trois premières lignes correspondent bien aux octets 3, 4 et 6 de l'état interne de l'AES et les octets retrouvés sont bien les octets 3, 4 et 6 de la clé. Même si l'on ne retrouve que quelques octets, on diminue la sécurité proposée par l'algorithme.

Cependant un attaquant n'a pas la possibilité de valider ainsi l'attaque : il ne connait pas la correspondance entre les points d'intérêt et les octets de l'état et il ne sait pas décider directement quels octets de clés sont corrects. Il doit encore deviner la position des octets de clé qu'il obtient mais aussi la valeur des octets manquants sachant que certains octets peuvent être erronés. Il sera donc avantageux d'utiliser plus de *points d'intérêt* afin d'augmenter ses chances d'obtenir des octets de clé corrects.

Le Tableau 5.16 représente les seize meilleurs résultats de l'attaque par signatures lorsque l'on utilise la distance du produit scalaire et si l'on considère 50 *points d'intérêt* par zone. Comme l'on considère plus de *points d'intérêt*, l'attaque requiert environ deux minutes. Dans notre cas, on retrouve alors dix octets de la clé secrète. Il reste donc encore $10! \cdot 2^{6 \cdot 8} \approx 2^{70}$ de clés à tester. Ce nombre peut sembler important, mais il l'est bien moins que les 2^{128} clés à tester à l'origine. De plus, il est possible d'appliquer notre attaque à différentes rondes et de combiner les résultats pour réduire encore le nombre de clés à tester.

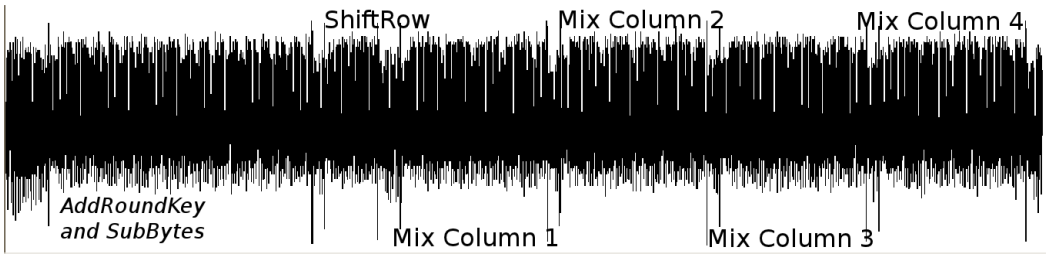


FIGURE 5.10 – Émanation électromagnétique d'un ATMega2561 durant l'exécution de la première ronde d'un AES128 logiciel.

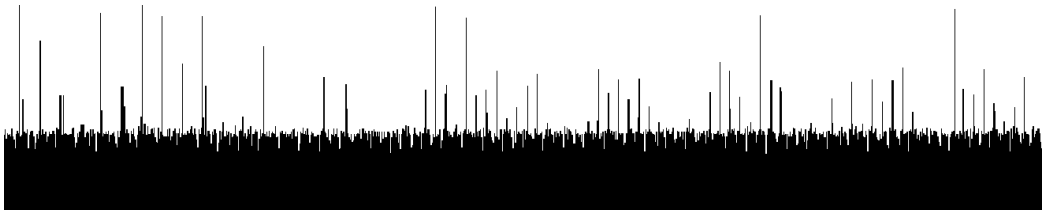


FIGURE 5.11 – Variance obtenue pour 1 000 signaux d'émanations électromagnétiques pour un ATMega2561 durant l'exécution de la première ronde d'un AES128 logiciel.

PoI dans la zone SB	*PoI* dans la zone MC	Octet de clé	Distance	Vrai ?
2	1	54	0.0036051	✓
3	2	31	0.0036711	✓
0	3	61	0.0037023	✓
3	3	224	0.0037423	X
0	0	200	0.0037556	X
2	3	234	0.0037823	X
2	0	39	0.0037883	X
1	3	154	0.0037976	X
3	0	55	0.0037986	X
0	2	216	0.0038011	X
1	1	159	0.0038514	X
2	2	206	0.0038786	X
1	2	21	0.0038983	X
3	1	218	0.0039113	X
0	1	257	0.0039115	X
1	0	197	0.0039612	X

Tableau 5.15 – Résultats de l'attaque par signatures pour les 8 instants avec la plus grande variance.

PoI dans la zone SB	*PoI* dans la zone MC	Octet de clé	Distance	Vrai ?
8	23	31	0.035180	✓
42	18	23	0.035773	✓
33	45	54	0.035813	✓
1	8	228	0.035867	✓
16	49	191	0.035941	✓
12	38	138	0.035977	X
11	21	61	0.035996	✓
48	20	224	0.036023	X
5	19	61	0.036023	✓
28	12	207	0.036051	✓
13	33	25	0.036094	X
21	42	39	0.036121	X
9	34	197	0.036137	X
25	47	198	0.036137	✓
38	15	109	0.036187	X
17	31	145	0.036203	✓

Tableau 5.16 – Résultats de l'attaque par signatures pour les 50 instants avec la plus grande variance pour chaque région.

5.5 Conclusion

Nous avons présenté une nouvelle attaque par observations basée sur la distribution conjointe des entrées et des sorties d'une fonction cryptographique.

Les deux principaux avantages de cette attaque sont sa grande résistance au bruit et par conséquent aux erreurs introduites dans la valeur du poids de Hamming des variables que l'on considère et le fait qu'elle ne requiert pas la connaissance de l'entrée ou de la sortie de l'algorithme. Elle représente donc une alternative intéressante aux attaques algébriques par canaux cachés, lorsque l'estimateur du poids de Hamming utilisé n'est pas très précis. L'estimateur que nous avons proposé n'offre pas la possibilité d'attaquer une implémentation câblée car alors la fuite est trop concentrée. On peut toutefois imaginer attaquer de telles implémentations en utilisant d'autres estimateurs. Notre estimateur diffère de ceux proposés dans [RS11, LMV$^+$] car il ne nécessite pas de phase de profilage et offre donc un cadre plus général à notre attaque. Si nous n'avons ici considéré que le cas où la fonction de fuite, φ, est le poids de Hamming, il est tout à fait imaginable d'utiliser d'autres fonctions comme la distance de Hamming par exemple. L'implémentation que nous avons considérée n'a pas de contre-mesure. Une bonne contre-mesure à notre attaque serait de rendre aléatoire l'ordre d'exécution des différentes opérations constituant une ronde de l'AES. En revanche, notre attaque permet de mettre en défaut la contre-mesure classique qui vise à masquer les données utilisées en attaquant la fonction AddRoundKey à la place de la fonction SubBytes. De plus, comme notre attaque ne nécessite pas la connaissance de l'entrée ou de la sortie de l'algorithme, elle peut être appliquée à n'importe quelle ronde de l'AES. Ainsi une contre-mesure qui n'est mise en œuvre que dans les premières et dernières rondes comme proposée dans [ABG04, AG03] pourrait être mise en défaut par notre attaque. L'attaque par signatures peut être améliorée en utilisant un estimateur plus efficace, mais nous avons voulu mettre en valeur sa simplicité de réalisation et sa résistance au bruit. Les implémentations logicielles d'algorithmes cryptographiques sont particulièrement bien adaptées à une attaque par signatures car les données sont manipulées par morceaux. Cette attaque peut éventuellement aussi s'appliquer au contexte de la rétro ingénierie, on considère alors que les *signatures théoriques* sont obtenues non plus en changeant une clé mais en changeant d'opération.

Chapitre 6

Conclusion et perspectives

Ce chapitre clos ce manuscrit. Nous allons commencer par rappeler les objectifs qui ont motivé cette thèse. Nous ferons ensuite une synthèse des résultats que nous avons obtenus, avant de suggérer quelques perspectives.

L'un des buts de cette thèse était de proposer de nouvelles attaques par observations ne nécessitant ni la connaissance du texte clair ni celle du texte chiffré. De telles attaques présentent un intérêt certain car elles permettent de mettre en défaut certaines contremesures répandues dans le monde de la carte à puce. En outre, les attaques par observations classiques n'utilisent qu'une petite partie des signaux obtenus afin d'en retirer de l'information. En effet, le fait de devoir connaitre l'entrée ou la sortie de l'algorithme implique le plus souvent une exploitation partielle des signaux.

Durant notre état de l'art, nous avons mis en évidence l'aspect hautement parallélisable des attaques par observations. Nous avons donc proposé des implémentations en OpenCL des attaques par observations classiques. Ces implémentations ont pour but de prouver la faisabilité et l'intérêt de l'utilisation de GPGPU dans le cadre des attaques par canaux cachés. Elles peuvent sans nul doute être améliorées. De plus, les nouvelles améliorations matérielles proposées par les constructeurs de cartes graphiques semblent très prometteuses dans le cadre de l'utilisation généraliste des GPU.

En décembre 2011, une nouvelle méthode appelée MIC pour étudier les relations entre deux variables a été présentée. Nous l'avons utilisée dans le cadre des attaques par observations. Les résultats que nous avons obtenus sont excellents, néanmoins la complexité des calculs à effectuer est très importante. Nous avons proposé une implémentation OpenCL de cette attaque. Nous avons aussi remarqué que dans le cadre des attaques par observations, nous pouvions grandement réduire le nombre de calculs à effectuer en diminuant la taille maximale des grilles considérées. Malgré ces améliorations, la MICA reste une attaque lente mais plus facile à mettre en œuvre que la MIA et plus résistante au bruit. De plus, contrairement à la MICA, lorsque la MIA échoue, il est difficile de savoir s'il faut remettre en cause les paramètres ou le modèle de fuite. Cependant, il serait intéressant d'observer les résultats de la MICA lorsque la relation entre les signaux et les modèles n'est plus linéaire. On peut aussi imaginer utiliser le MIC à certains instants très précis du signal afin de détecter une relation particulière non linéaire. L'étude de cette relation permettra de modifier en conséquence le modèle de fuite avant d'effectuer une CPA sur l'ensemble du signal.

L'état de l'art nous a aussi permis de découvrir les attaques algébriques par canaux cachés. Ces attaques à la frontière entre la cryptanalyse algébrique et les attaques par observations, ont attiré notre attention de part les méthodes mathématiques souvent très complexes mises en œuvre mais aussi par l'utilisation complète qu'elles font du signal. Nous avons proposé l'attaque par signatures qui utilise la distribution conjointe du poids de Hamming de l'entrée et du poids de Hamming de la sortie d'une fonction cryptographique. Lorsque la connaissance du poids de Hamming est parfaite cette attaque donne d'excellents résultats bien que l'on ait besoin de bien plus de signaux que pour les attaques algébriques par canaux cachés. La résistance aux erreurs d'estimation est très importante puisqu'avec suffisamment de signaux, nous pouvons retrouver la clé secrète

même lorsque 50% des valeurs du poids de Hamming sont erronées. Pour cela, il est important de bien choisir les distances que nous allons utiliser pour comparer deux distributions. Nous avons fait une étude exhaustive des distances permettant de comparer deux distributions et nous avons ainsi identifié cinq distances plus pertinentes que les autres. La forte résistance au bruit de notre attaque permet d'utiliser des méthodes d'estimation du poids de Hamming bien moins contraignantes que les attaques de type *Template* proposées dans le cadre des attaques algébriques par canaux cachés. Nous avons limité notre étude à la connaissance partielle du poids de Hamming des données manipulés. Il serait très intéressant de considérer d'autres cas comme celui de la distance de Hamming. Une perspective prometteuse serait de combiner les résultats de cette attaque sur différentes rondes en utilisant la connaissance de l'algorithme de différenciation de la clé afin d'obtenir une attaque plus performante utilisant l'ensemble du signal. Il serait intéressant d'essayer d'utiliser cette attaque sur des algorithmes implémentés de manière câblée même si les calculs nécessaires pour obtenir les signatures théoriques peuvent devenir plus complexes que pour effectuer une attaque par force brute.

Pour la MICA comme pour l'attaque par signatures, nous avons soulevé l'importance de l'aptitude à détecter des *points d'intérêt* dans le signal. Cette capacité, plus que les implémentations parallèles et les subtilités d'optimisation, peut amener vers une bien plus grande performance des attaques par observations.

Annexe A

Algorithmes pour les distingueurs classiques en OpenCL

CEtte annexe est dédiée à la présentation des algorithmes des différents distingueurs classiques dans les attaques par canaux cachés. Ces algorithmes sont plus précis et complets que les diagrammes présentés dans le chapitre 3.

algorithme 11 Calcul des coefficients de Spearman sur le GPU.

Algorithme SPEARMAN_GPU ($courbes[n][T]$: courbes de fuites, $m[n][\mathbf{k}]$: modèles)
▷ Code hôte
pour $k = 0\,; k <$ **k faire**
Trier les valeurs des modèles pour la clé k
Attribuer à chaque valeur son rang et les stocker dans le tableau `mk_rang` de taille **k**.
fin pour
pour $t = 0\,; t < T$ **faire**
Trier les valeurs des courbes pour l'instant t
Attribuer à chaque valeur son rang et les stocker dans le tableau `l_rang` de taille T.
fin pour
Exécuter CPA_GPU(l_rang,mk_rang)
fin Algorithme

algorithme 12 Algorithme de tri comptage.

Algorithme TRI_COMPTAGE ($tab[n]$: tableau à trier, v_{max} : valeur maximale des éléments de tab)
Initialiser un tableau, $index$ de taille $v_{max} + 1$ à zéro.
Initialiser un tableau, $tabtrie$ de taille n à zéro.
pour $i = 0 ; i < n$ **faire**
$index[tab[i]] + +$
▷ On va ainsi obtenir le nombre d'apparitions de chaque valeur.
fin pour
$i = 0$
$total = 0$
pour $j = 0 ; j <= v_{max}$ **faire**
$total+ = index[j]$
pour $i <= total$ **faire**
$tabtrie[i] = j$
$i + +$
fin pour
fin pour
fin Algorithme

algorithme 13 Algorithme de calcul du rang.

Algorithme ALGO_RANG ($tab[n]$: tableau à trier, v_{max} : valeur maximale des éléments de tab)
Initialiser un tableau, $index$ de taille $v_{max} + 1$ à zéro.
pour $i = 0 ; i < n$ **faire**
$index[tab[i]] + +$ ▷ On va ainsi obtenir le nombre de fois qu'apparait chaque valeur.
fin pour
pour $j = 1 ; j \leq v_{max}$ **faire**
$index[j]+ = index[j-1]$ ▷ On obtient ainsi le nombre d'éléments plus petits ou égaux à j
fin pour
$courant = index[0] + 1$
$rang[0] = \frac{index[0]+1}{2}$
pour $x = 0 ; x \leq v\_max$ **faire**
$somme = 0$
$nb\_occ = 0$
pour $courant \leq index[x]$ **faire**
$somme+ = courant$
$nb\_occ + +$
$courant + +$
fin pour
$rang[x] = \frac{somme}{nb\_occ}$ ▷ $rang[x]$ contient le rang moyen de la valeur x
$x + +$
fin pour
pour $i = 0 ; i < n$ **faire**
$tab[i] = rang[tab[i]]$
fin pour
fin Algorithme

algorithme 14 Calcul des coefficients de Pearson sur le GPU.

Algorithme CPA_GPU ($courbes[n][T]$: courbes de fuites, $m[n][\mathbf{k}]$: modèles) ▷ Code hôte
  Créer les tableaux `s_mk` de taille **k**, `s_mk_2` de taille **k**
  pour $i = 0 ; i < n$ **faire**
    Copier les modèles correspondant à la i-ème courbe vers le GPU($m[i]$)
    Exécuter K_SOMMES_BLEU($m[i]$,s_mk,s_mk_2)
  fin pour
  Créer les tableaux `s_li` de taille T, `s_li_2` de taille T, `s_li_mk` de taille $T \times \mathbf{k}$
  pour $i = 0 ; i < n$ **faire**
    Copier les modèles correspondant à la i-ème courbe vers le GPU($m[i]$)
    Copier la i-ème courbe vers le GPU($courbes[i]$)
    Exécuter K_SOMMES_ORANGE($m[i]$,$courbes[i]$,s_li,s_li_2,s_li_mk)
  fin pour
  Créer le tableau $coeffs$ de taille $T \times \mathbf{k}$
  Exécuter KPEARSON(s_mk,s_mk_2,s_li,s_li_2,s_li_mk,$coeffs$)
  Copier les **k** courbes représentant le coefficient de Pearson le GPU
fin Algorithme
Algorithme K_SOMMES_BLEU ($m[i]$,s_mk ǀ ,s_mk_2) ▷ Code GPU
  $k \leftarrow$ *id du thread*
  `s_mk` += $m[i][k]$;
  `s_mk_2` +=$m[i][k] \times m[i][k]$;
fin Algorithme
Algorithme K_SOMMES_ORANGE ($m[i]$,$courbes[i]$,s_li,s_li_2,s_li_mk) ▷ Code GPU
  $t \leftarrow$ *id du thread*
  `s_li` += $courbes[i][t]$;
  `s_li_2` += $courbes[i][t] \times courbes[i][t]$;
  pour $k = 0 ; k <$ **k faire**
    `s_li_mk`$[k \times T + t] += courbes[i][t] \times m[i][k]$;
  fin pour
fin Algorithme
Algorithme KPEARSON (s_mk,s_mk_2,s_li,s_li_2,s_li_mk,$coeffs$) ▷ Code GPU
  $t \leftarrow$ *id du thread*
  pour $k = 0 ; k <$ **k faire**
    $tmp = (n \times$ `s_li_mk`$[k \times T + t]) - ($`s_li`$[t] \times$ `s_mk`$[k]) ;$
    $tmp/ = sqrt((n \times$ `s_li_2`$[t]) - ($`s_li`$[t] \times$ `s_li`$[t])) ;$
    $tmp/ = sqrt((n \times$ `s_mk_2`$[k]) - ($`s_mk`$[k] \times$ `s_mk`$[k])) ;$
    $coeffs[k \times T + t] += tmp ;$
  fin pour
fin Algorithme

Annexe B

Résultats de l'attaque par signatures pour les différentes distances de la classification de Cha en fonction de leur efficacité

DANS cette annexe nous résultats détaillés de l'attaque par signatures dans les mêmes conditions que dans la Section 5.2.2

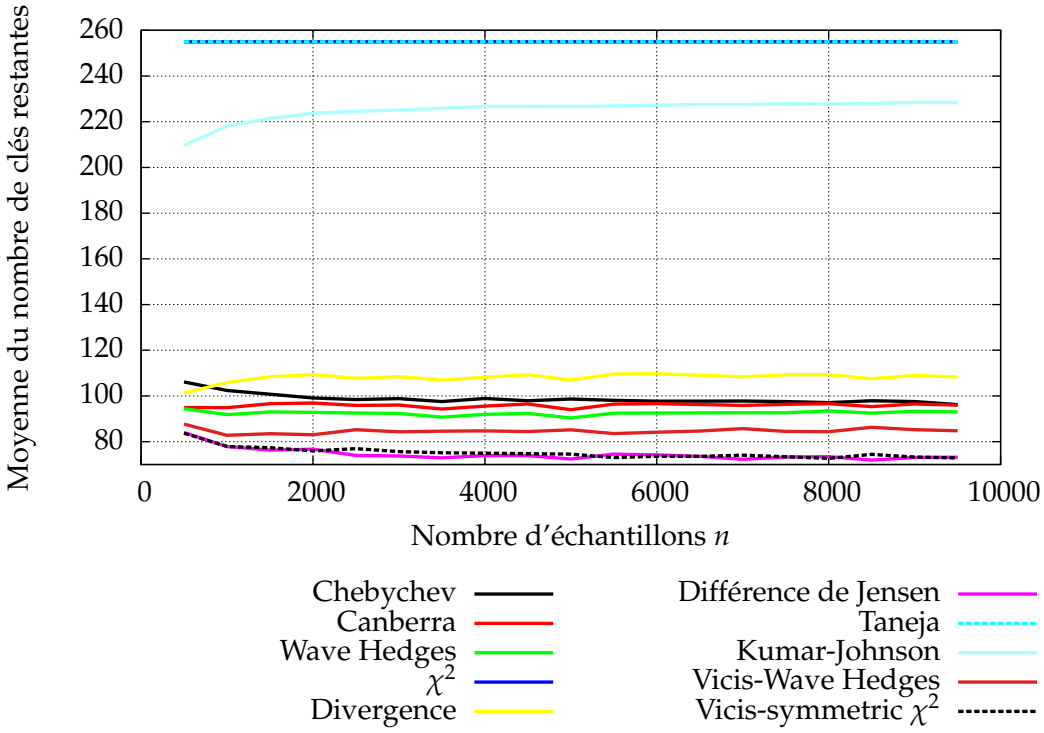


FIGURE B.1 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances inadaptées à l'attaque par signatures et 50% de *petites* erreurs.

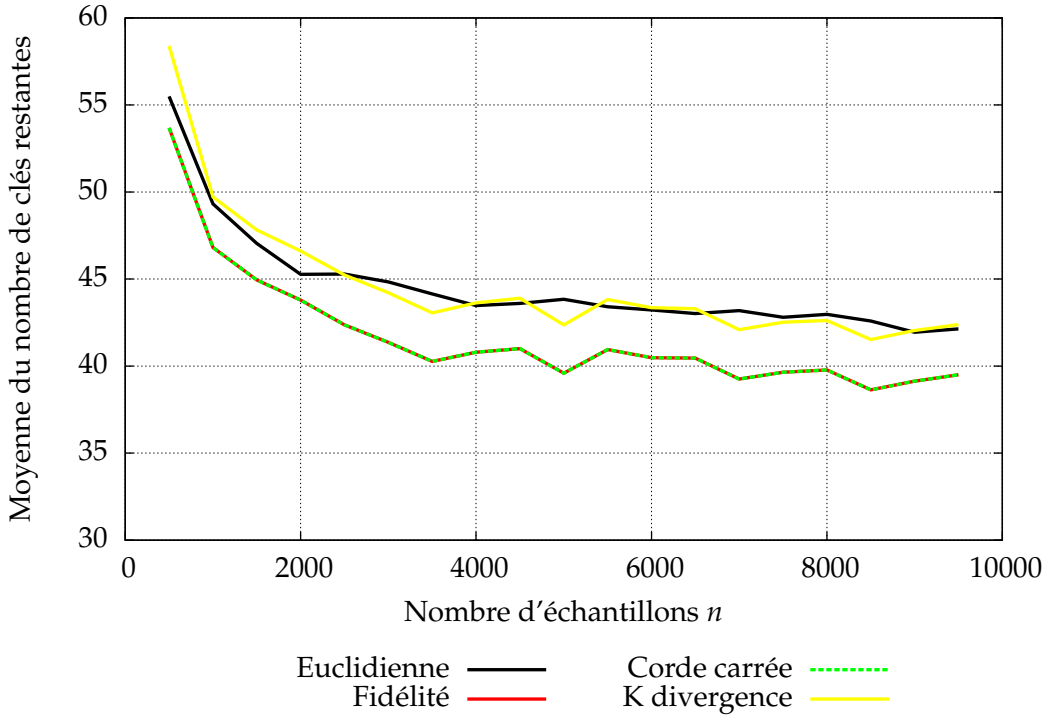


FIGURE B.2 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances moyennement adaptées à l'attaque par signatures et 50% de *petites* erreurs.

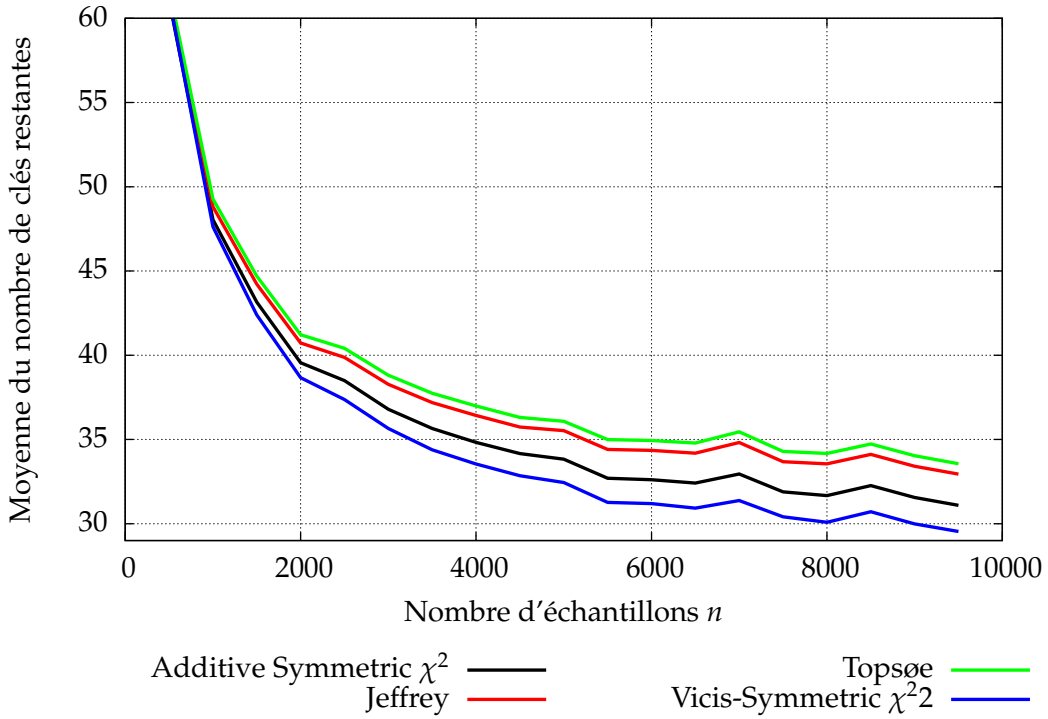


FIGURE B.3 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances plutôt adaptées à l'attaque par signatures et 50% de *petites* erreurs.

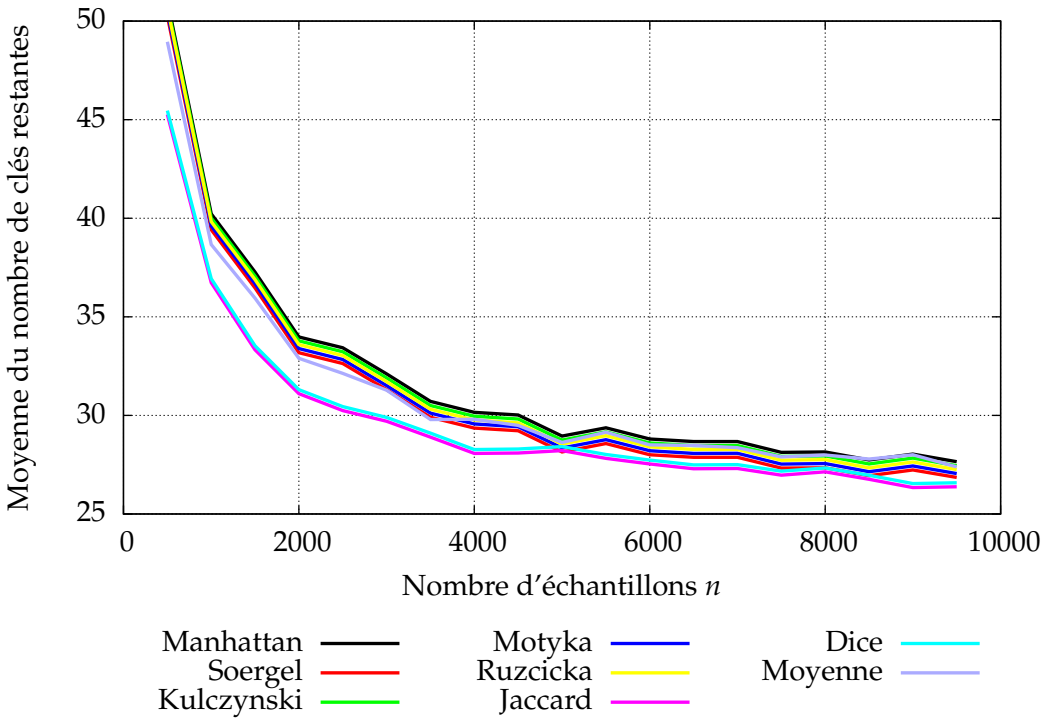


FIGURE B.4 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances adaptées à l'attaque par signatures et 50% de *petites* erreurs.

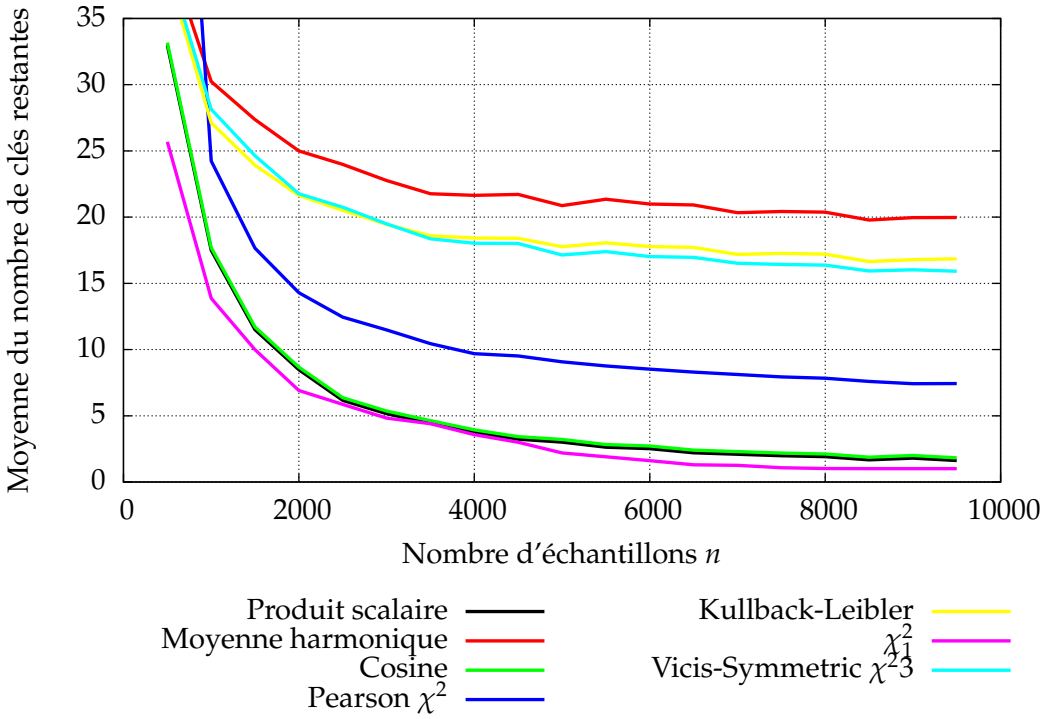


FIGURE B.5 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances parfaitement adaptées à l'attaque par signatures et 50% de *petites* erreurs.

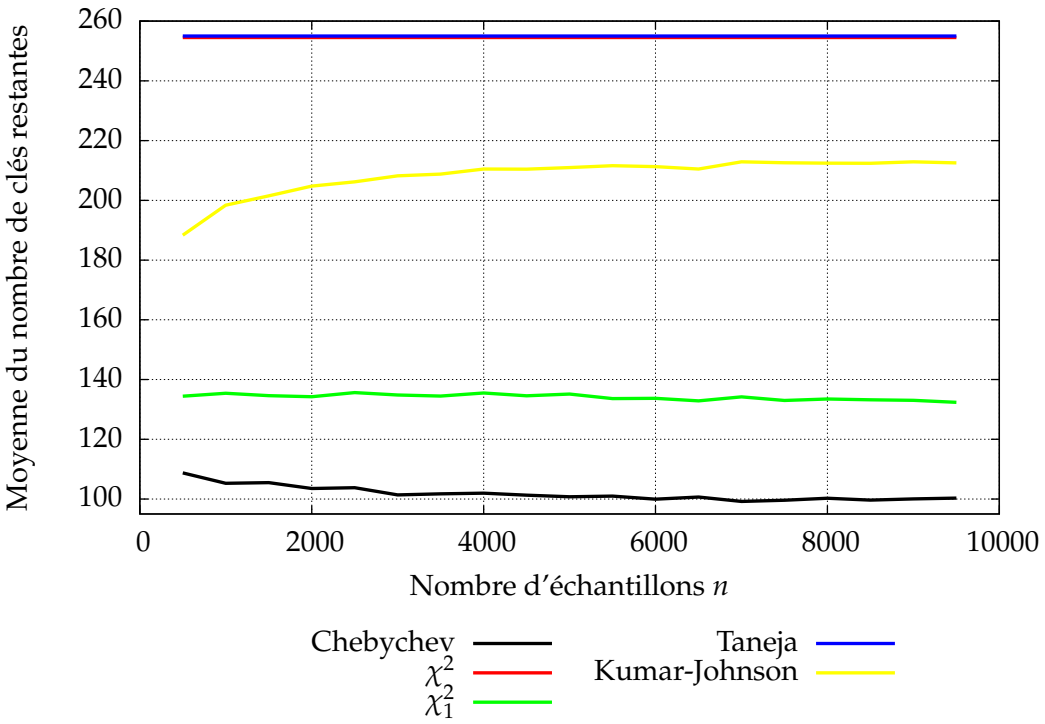


FIGURE B.6 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances inadaptées à l'attaque par signatures et 50% d'erreurs aléatoire.

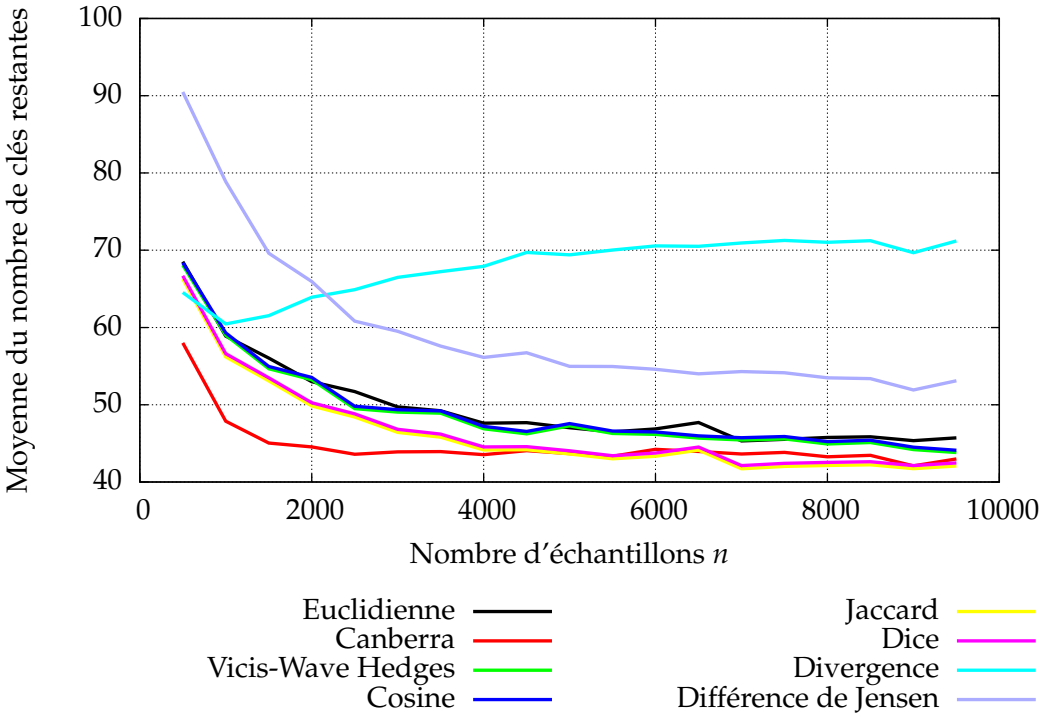


FIGURE B.7 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances moyennement adaptées à l'attaque par signatures et 50% d'erreurs aléatoire.

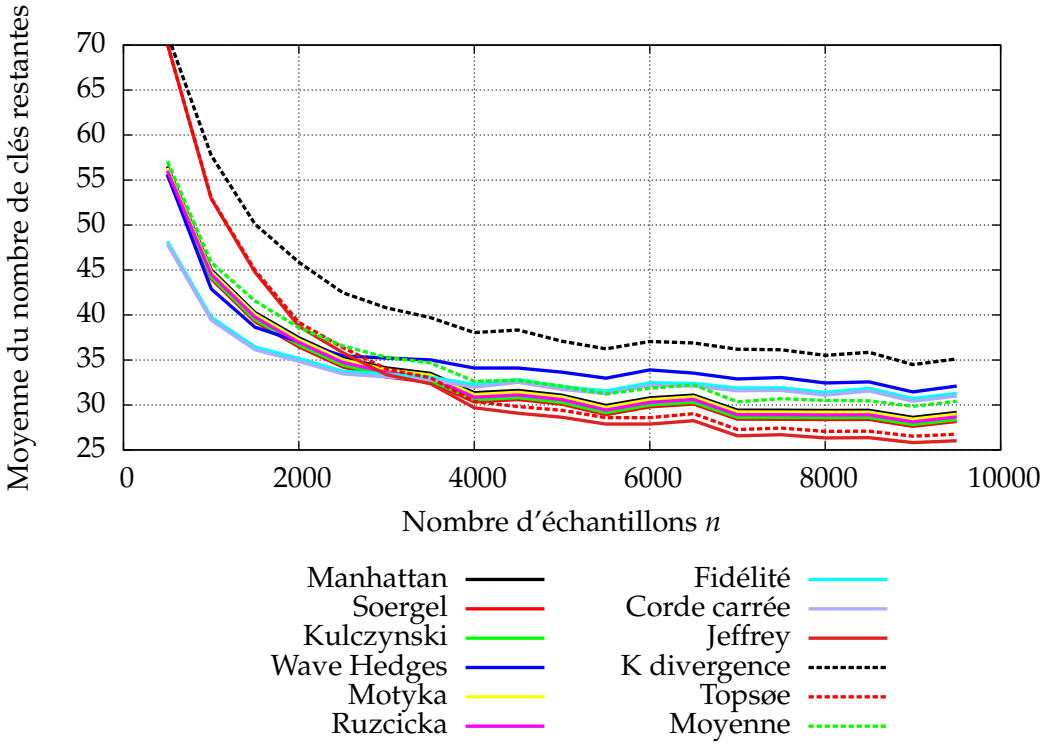


FIGURE B.8 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances adaptées à l'attaque par signatures et 50% d'erreurs aléatoire.

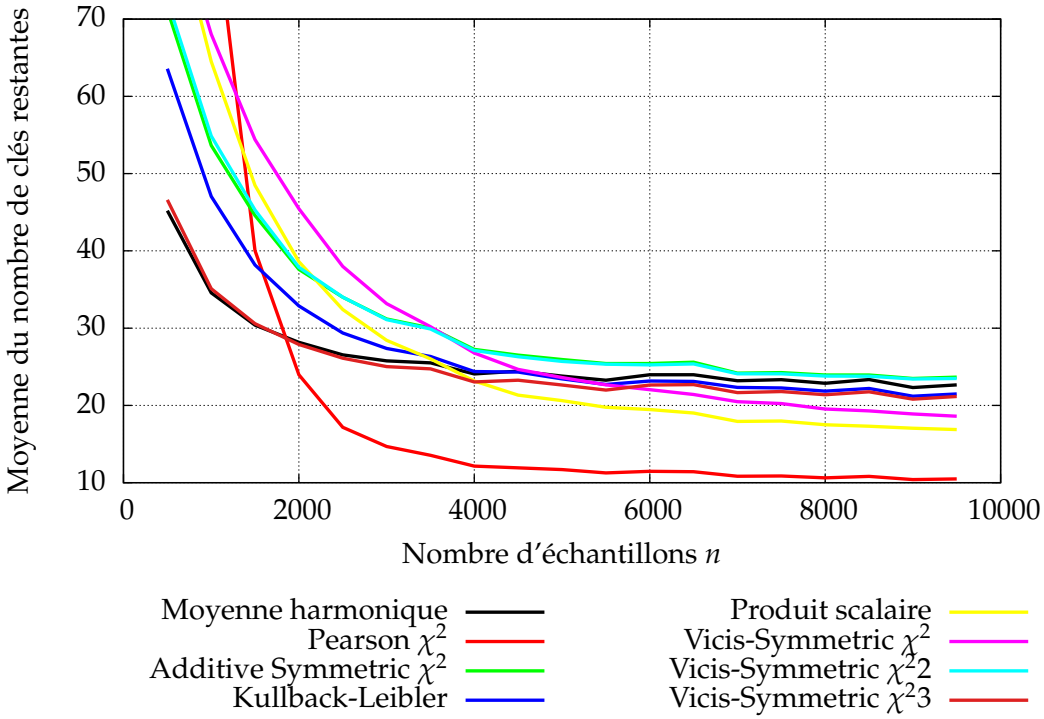


FIGURE B.9 – Moyenne des résultats obtenus pour 10 000 attaques par signatures pour les distances parfaitement adaptées à l'attaque par signatures et 50% d'erreurs aléatoire.

Bibliographie

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-Channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002.

[ABG04] Mehdi-Laurent Akkar, Régis Bevan, and Louis Goubin. Two Power Analysis Attacks against One-Mask Methods. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2004.

[AFV+12] Davide Albanese, Michele Filosi, Roberto Visintainer, Samantha Riccadonna, Giuseppe Jurman, and Cesare Furlanello. cmine, minerva & minepy : a C Engine for the MINE Suite and its R and Python Wrappers. *arXiv preprint arXiv :1208.4271*, 2012.

[AG03] Mehdi-Laurent Akkar and Louis Goubin. A Generic Protection against High-Order Differential Power Analysis. In *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 192–205. Springer, 2003.

[AMD] AMD/ATI. AMD et HSA – Lancement d'une nouvelle ère d'expériences numériques saisissantes. `http://www.amd.com/fr/products/technologies/hsa/Pages/hsa.aspx`.

[APcXSQ06] Cédric Archambeau, Eric Peeters, François Xavier Standaert, and Jean-Jacques Quisquater. Template Attacks in Principal Subspaces. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004 : 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BGLR08] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Comparative Evaluation of Rank Correlation Based DPA on an AES Prototype Chip. In *ISC*, pages 341–354, 2008.

[BGP+11] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, Francois-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual Information Analysis : a Comprehensive Study. *J. Cryptology*, 24 :269–291, 2011.

[BLR11] Timo Bartkewitz and Kerstin Lemke-Rust. A High-Performance Implementation of Differential Power Analysis on Graphics Cards. In *Smart Card Research and Advanced Applications*, pages 252–265. Springer, 2011.

[BVP+03] Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius. Rabbit : A New High-Performance Stream Cipher. In

Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers, volume 2887 of *Lecture Notes in Computer Science*, pages 307–329. Springer, 2003.

[Cha07] Sung-Hyuk Cha. Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4) :300–307, 2007.

[CJ01] Christophe Clavier and Marc Joye. Universal Exponentiation Algorithm. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 300–308. Springer, 2001.

[Cop94] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM journal of research and development*, 38(3) :243–250, 1994.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DH76] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. Information Theory, IEEE Transactions on 22(6) : 644-654, 1976.

[DRTV07] Jean-Guillaume Dumas, Jean-Louis Roch, Éric Tannier, and Sébastien Varrette. *Théorie des Codes-Compression, Cryptage, Correction : Compression, Cryptage, Correction*. Dunod, 2007.

[Dub98] Gilles Dubertret. *Initiation à la Cryptographie*. Vuibert supérieur, 1998.

[Fly72] Michael J Flynn. Some Computer Organizations and their Effectiveness. *Computers, IEEE Transactions on*, 100(9) :948–960, 1972.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*, volume 141. Wiley New York, 2003.

[Gam84] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[GLRP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. Stochastic Methods. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis : Concrete Results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[Gro10] Khronos Group. The OpenCL Specification, Version 1.1, September 2010.

[Hoa62] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1) :10–16, 1962.

[HP07] Helena Handschuh and Bart Preneel. Blind Differential Cryptanalysis for Enhanced Power Attacks. In *Selected Areas in Cryptography*, pages 163–173. Springer, 2007.

[Hud] Hans Christoph Hudde. GPU assisted Mutual Information Analysis Attacks on AES.

[Jaf07] Joshua Jaffe. A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.

[Ken38] Maurice Kendall. A New Measure of Rank Correlation. *Biometrika*, 30 (1/2) :81–93, 1938.

[Ker83a] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des sciences militaires*, IX janvier :5–38, 1883.

[Ker83b] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des sciences militaires*, IX février :161–191, 1883.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Introduction to Differential Power Analysis and Related Attacks. 1999.

[KKLM09] Marcelo Kaihara, Thorsten Kleinjung, Arjen Lenstra, and Peter Montgomery. On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography : version 2.1. *cryptology eprint archive*, report 2009/389,, 2009.

[Knu81] Donald Knuth. *The Art of Computer Programming*, volume 2, chapter Seminumerical Algorithms. Addison-Wesley, Reading, Massachusetts, 1981.

[LCC+06] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. A Proposition for Correlation Power Analysis Enhancement. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop*, volume 4249 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2006.

[LCC08] Thanh-Ha Le, Cécile Canovas, and Jessy Clédiere. An Overview of Side Channel Analysis Attacks. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 33–43. ACM, 2008.

[LMV+] Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cedric Meuter, Gianluca Bontempi, and Olivier Markowitch. Semi-Supervised Template Attack.

[Luc98] Stefan Lucks. Attacking Triple Encryption. In *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 1998.

[Man05] Itsik Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2005.

[MBZ+12] Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, Michael Walter, and Johannes Buchmann. Improved Algebraic Side-Channel Attack on AES. In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pages 146–151. IEEE, 2012.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits : A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6632 of *Lecture Notes in Computer Science*, page 69. Springer, 2011.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[Nut13] Christian Nutt. Inside the PlayStation 4 With Mark Cerny. `http://www.gamasutra.com/view/feature/191007/inside_the_playstation_4_with_mark_.php`, april 2013. [Online ; accessed 29-Juny-2013].

[OKPW10] Yossef Oren, Mario Kirschbaum, Thomas Popp, and Avishai Wool. Algebraic Side-Channel Analysis in the Presence of Errors. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop*, volume 6225 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2010.

[ORSW12] Yossef Oren, Mathieu Renauld, Fran&ccedil ;ois-Xavier Standaert, and Avishai Wool. Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model. In *CHES*, pages 140–154. Springer, 2012.

[oS77] United States National Bureau of STANDARDS. The data encryption standard. FIPS pub 46. Technical report, National Bureau of Standards, Washington, DC, USA, 1977.

[oS01] United States National Bureau of STANDARDS. Advanced encryption standard. FIPS pub 197. Technical report, National Bureau of Standards, Washington, DC, USA, 2001.

[OW12] Yossef Oren and Avishai Wool. Tolerant Algebraic Side-Channel Analysis of AES. Technical report, Cryptology ePrint Archive, Report 2012/092, 2012.

[Par08] Télécom ParisTech. DPA Contest v1, 2008.

[Par10] Télécom ParisTech. DPA Contest v2, 2010.

[Par13] Télécom ParisTech. DPA Contest v3, 2013.

[PR10] Emmanuel Prouff and Matthieu Rivain. Theoretical and Practical Aspects of Mutual Information-Based Side Channel Analysis. *International Journal of Applied Cryptography*, 2(2) :121–138, 2010.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema) : Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001.

[RRF+11] David Reshef, Yakir Reshef, Hilary Finucane, Sharon Grossman, Gilean Mc Vean, Peter Turnbaugh, Eric Lander, Michael Mitzenmacher, and Pardis Sabeti. Detecting Novel Associations in Large Data Sets. *science*, 334(6062) :1518–1524, 2011.

[RS11] Mathieu Renauld and François-Xavier Standaert. Algebraic Side-Channel Attacks. In *Information Security and Cryptology*, pages 393–410. Springer, 2011.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Ddigital Signatures and Public-key Cryptosystems. Communications of the ACM, 21 : 120–126,, 1978.

[Sin99] Simon Singh. *Histoire des codes secrets. De l'Égypte des pharaons à l'ordinateur quantique*. Jean-Claude Lattès, 1999.

[VCcXS09] Nicolas Veyrat-Charvillon and François Xavier Standaert. Mutual Information Analysis : How, When and Why ? In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2009.

[Ven10] Alexandre Venelli. Efficient Entropy Estimation for Mutual Information Analysis Using B-splines. In *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, pages 17–30. Springer, 2010.

[vOW96] Paul C. van Oorschot and Michael J. Wiener. Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 1996.

[WOM11] Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. An Exploration of the Kolmogorov-Smirnov Test as a Competitor to Mutual Information Analysis. In Springer, editor, *Smart Card Research and Advanced Applications*, pages 234–251, 2011.

Résumé :

Cette thèse porte sur les attaques par observations. Ces attaques étudient les variations d'émanation d'un composant pour retrouver une clé secrète. Ces émanations peuvent être multiples, par exemple, la consommation de courant électrique, le rayonnement électromagnétique, etc. Généralement, ces attaques font appel à des méthodes statistiques pour examiner la relation entre les émanations du composant et des modèles de consommation imaginés par l'attaquant. Trois axes sont développés dans cette thèse.

Dans un premier temps, nous avons implémenté différentes attaques par observations sur *GPGPU* en utilisant l'API OpenCL. Ces implémentations sont plus performantes que les implémentations classiques, ce qui permet à un attaquant de pouvoir traiter plus de données.

Dans un second temps, nous avons proposé l'utilisation dans le cadre des attaques par observations d'une nouvelle mesure de dépendance proposée par Reshef *et al.* appelée MIC. L'avantage du MIC, par rapport à l'information mutuelle, est que son calcul ne dépend pas du choix d'un noyau ou d'une taille de fenêtre. Son utilisation dans une attaque par observations est donc aisée, même si la complexité des calculs à effectuer est souvent très importante.

Enfin, nous avons introduit une nouvelle attaque, basée sur la distribution conjointe de l'entrée et de la sortie d'une sous-fonction cryptographique. Si cette distribution dépend de la valeur de la clé impliquée par la fonction, on est capable de retrouver la clé secrète utilisée par le composant. Cette nouvelle attaque a la particularité de ne nécessiter ni la connaissance du texte clair, ni la connaissance du texte chiffré, ce qui lui permet d'être efficace même en présence de certaines contre-mesures.

Abstract :

The main subject of this manuscript is the Side Channel Attacks. These attacks investigate the variation of device emanations to retrieve a secret key. These emanations can be the power consumption, the electromagnetic radiation, etc. Most of the time, those attacks use statistical methods to examine the relationship between the emanations and some leakage models supposed by the attacker. Three main axis are developed here.

First, we have implemented many side channel attacks on *GPGPU* using the API OpenCL. These implementations are more effective than the classical ones, so an attacker can exploit more data.

Then, in order to provide a new side channel attack, we have suggested the use of a new dependency measurement proposed by Reshef *et al.*, the MIC. The MIC is more advantageous than the mutual information, because its computation does not depend of a kernel choice nor a windows size. So, its use in side channel analysis is simple, even if the time complexity is large.

Finally, we have introduced a new attack based on the join distribution of the input and the output of a cryptographic sub-function. If the distribution depends on the key used in the function, we can retrieve the secret key. This attack can be efficient even in presence of some countermeasures because it does not required the knowledge of both plain text or cipher text.