Contributions à l'analyse de canaux auxiliaires sans connaissance des clairs et chiffrés, et à la recherche de S-boxes compactes

Léo Reynaud

Université de Limoges

Contributions à l'analyse de canaux auxiliaires sans connaissance des clairs et chiffrés, et à la recherche de S-boxes compactes

Auteur :

Léo Reynaud

Thèse présentée pour l'obtention du grade de docteur en Informatique de l'Université de Limoges

Jury :

Directeur :	Christophe Clavier (Université de Limoges)
Rapporteurs :	Louis Goubin (Université de Versailles Saint-Quentin en Yvelines)
	Sylvain Guilley (Telecom ParisTech)
Examinateurs :	Yanis Linge (STMicroelectronics)
	Antoine Wurcker (Serma Technologies)
	Marine Minier (Loria)

Université de Limoges

xlim

Remerciements

Une thèse n'est pas une étape qu'on effectue seul, que ce soit sur le plan professionnel ou personnel. Ces remerciements sont une façon de dire merci à tous ceux qui y ont participé et sans qui cette thèse serait bien différente, ou n'aurait tout simplement pas vu le jour. Il n'est évidemment pas possible de remercier tout le monde, mais j'espère que ceux que j'oublie se sentiront tout de même inclus dans ces lignes.

Je tiens tout d'abord à remercier mon jury de thèse : merci à mes rapporteurs Louis Goubin et Sylvain Guilley pour avoir pris le temps de lire ma thèse avec attention et pour leurs retours bienveillants. Merci également à mes examinateurs avec qui j'ai eu la chance de travailler au cours de cette thèse. Yanis Linge pour son enthousiasme et son aide lorsque nous avons commencé à travailler sur ses distributions jointes. Marine Minier pour sa patience et ses cours de cryptographie très pédagogiques. Antoine Wurcker que je connais depuis longtemps puisque je participais à ses cours lorsque lui-même était à ma place. Sa gentillesse et sa passion sont pour beaucoup dans mon choix de faire une thèse. Je tiens aussi à remercier ceux qui avaient accepté mais n'ont pas pu être examinateurs, Benoît Feix et Julien Francq.

Lors de cette thèse j'ai eu la chance de participer à deux projets et je tiens à remercier les équipes de chacun, travailler à leur côté a été une expérience enrichissante. Merci à Hugues Thiebeauld pour m'avoir inclus dans le projet SCATTER, et merci à Aurélien Vasselle et Guillaume Bethouart pour nos échanges et qui ont toujours œuvré à nous faciliter la tache. Merci à toute l'équipe de PACLIDO, projet qui m'a permis de sortir de ma zone de confort et de m'ouvrir sur de nouvelles choses. Marine Minier qui a toujours répondu à mes questions, même les plus naïves, sur la cryptographie, et sa capacité à trouver des solutions à tous les problèmes. Julien Francq pour sa bienveillance et sa rigueur au travail, mais aussi sa compagnie en dehors. Paul Huynh avec qui c'est un plaisir de travailler et de sortir, et Alexandre Adomnicai pour nos coups de téléphones sur le threshold. Kevin Le Gouguec pour son expertise et patience vis-à-vis de mon codage, Virginie Lallemand pour ses conseils avisés sur les S-boxes, Cécile Abdo et Alexis Duque pour l'organisation du projet.

Je n'ai pas eu l'occasion de travailler avec eux, mais un grand merci à Arnaud Tisserand et Benoît Gérard de m'avoir invité à présenter mes travaux à Rennes, et qui m'ont aussi rassuré lors de la rédaction de cette thèse. Merci beaucoup à Vincent Verneuil pour sa bienveillance, sa compagnie

et sa confiance, notamment lors de ma première conférence, ce qui m'a été d'une très grande aide. Merci aussi à Markus Hinkelmann qui m'a permis de surmonter la peur de présenter en anglais.

Lors des conférences auxquelles j'ai participé, j'ai eu la chance de rencontrer d'autres doctorants/-postdocs, ce qui a rendu ces événements plaisants et mémorables. Merci à Maxime, Aymeric, Felix, Ko, Matthias, Thierno, Jean-Christophe et d'autres. Merci aussi à ceux que j'ai côtoyés au bureau pendant ces années, je pense notamment à François, Théo, Arthur, Xavier et Maxime.

Ces remerciements sont aussi l'occasion de remercier des personnes qui ne font pas du tout partie du cadre professionnel, mais qui sont tout aussi importantes. Merci donc à mes amis, et pardon pour ceux dont les noms n'apparaîtront pas par manque de place. Merci à Marine, Tom, Gwen avec qui j'ai passé énormément de temps pendant mes études, et qui ont participé à faire de la fac une de mes périodes préférées. Merci à Maxime, Alexis, Audrey et Maxime pour les voyages et les vacances qui ont toujours été fantastiques. Merci à Kevin, Romain, Pierre et Margot pour les week-ends entre amis. Merci à Kevin, Julia, Boy, Alex, Angélique et Ben pour les soirées, les repas, les jeux et le basket. Merci à Golo, Polo et Loïc pour être là toutes les semaines quoi qu'il arrive, et qui ont toujours quelque chose à fêter. Merci à Pierre, Célia et Charles que je connais depuis longtemps et dont je suis toujours proche malgré les années et la distance. Merci à ma seconde famille, Jerem, Antoine (blond), et Rémi (grand blond) avec qui je passe toujours des moments fantastiques et avec qui j'ai vécu mille histoires.

Je tiens aussi à remercier mes parents sans qui je ne serais jamais allé aussi loin, dans les études mais bien évidemment aussi dans la vie. Depuis petit, ils m'ont toujours soutenu et beaucoup appris, et ils continuent d'ailleurs toujours à le faire.

Enfin, je tiens à remercier tout particulièrement deux personnes. Merci à Christophe Clavier, directeur de cette thèse, mais pas seulement. Sa pédagogie, sa bienveillance et bien d'autres choses font de lui un mentor avec qui j'ai pris beaucoup de plaisir à travailler. C'est grâce à lui si j'ai eu la chance de commencer cette thèse, et c'est aussi grâce à lui que j'ai pu la terminer. D'une écoute et d'une patience à toute épreuve (sauf pour s'attaquer à des problèmes qu'il s'empresse de résoudre), il a aussi toujours été disponible pour n'importe quel sujet, en témoignent les près de 2000 mails que nous avons échangés lors de ces trois ans.

Merci aussi à Aude avec qui je passe des moments merveilleux depuis maintenant plusieurs années, et je l'espère encore pour beaucoup d'autres. Elle a eu la dure tâche de me supporter et de me soutenir, et aussi de relire et corriger ce manuscrit lorsqu'il était encore plus proche du brouillon.

Table des matières

Table des figures

Liste des tableaux

Avant-propos

Introduction

Cette thèse a été faite au sein du laboratoire XLIM rattaché à l'Université de Limoges. Ce laboratoire comporte trois pôles scientifiques, un pôle électronique, un pôle photonique, et un pôle mathématique, informatique et image. C'est au sein de ce dernier que s'est déroulé ce travail de recherche qui prolonge des travaux effectués en stage de Master Cryptis $2^{\text{ème}}$ année.

De nombreuses études portent sur les attaques applicables aux appareils utilisant la cryptographie afin de développer des contre-mesures efficaces. Les attaques par canaux cachés sont une vraie menace et suscitent beaucoup de recherches depuis le milieu des années 90. Elles ont pour particularité de nécessiter le clair ou le chiffré afin de déduire de l'information sur la clé secrète. Récemment, un nouveau type d'attaques par canaux cachés propose de se passer de ces deux éléments grâce à l'utilisation de distributions jointes. Aux vus des avantages et nouvelles perspectives que procure cette attaque, nous avons décidé d'étudier son fonctionnement. Ce document propose des améliorations, et son extension à divers cas de figures d'implémentations protégées par masquage booléen.

La cryptologie légère est particulièrement d'actualité puisqu'elle est utilisée dans des milieux très contraints tels que l'internet des objets, domaine très dynamique aujourd'hui. Ayant participé à la conception d'un algorithme de chiffrement léger lors du projet PACLIDO, et plus particulièrement à l'optimisation de la compacité de la S-box, nous avons remarqué que trouver des S-boxes 8 bits compactes est un sujet ayant une portée pratique. Dans cette optique, nous proposons une nouvelle approche sur l'obtention de circuits compacts de S-boxes 8 bits.

Structure du document

Le premier chapitre présente les notions de cryptologie, cryptographie et cryptanalyse, ainsi que les éléments nécessaires à la compréhension des attaques par canaux cachés. Des exemples de telles attaques sont donnés afin de rendre les explications plus concrètes. Le second chapitre

porte sur les S-boxes. Il présente particulièrement certaines notions de qualités cryptographiques, qualités nécessaires à la résistance face à la cryptanalyse classique qui est, elle aussi, présentée dans ce chapitre. Le troisième chapitre présente les attaques par distributions jointes ainsi que les améliorations que nous y avons apportées, notamment l'étude des cas de figures masqués. Enfin, le quatrième chapitre décrit, d'une part, notre contribution au projet PACLIDO, à savoir la recherche de la S-box compacte de Lilliput-TBC, et, d'autre part, notre recherche de circuits compacts 8 bits.

Publications

[MRLM16] Abdelhak Mesbah, Léo Reynaud, Jean-Louis Lanet, and Mohamed Mezghiche. The Hell Forgery, Polymorphic Codes Shoot Again. In *15th Smart Card Research and Advanced Application Conference*, 2016.

[CR17] Christophe Clavier and Léo Reynaud. Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 24–44. Springer, 2017.

[CRW18] Christophe Clavier, Léo Reynaud, and Antoine Wurcker. Quadrivariate Improved Blind Side-Channel Analysis on Boolean Masked AES. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 153–167. Springer, 2018.

[CR19] Christophe Clavier and Léo Reynaud. Systematic and Random Searches for Compact 4-Bit and 8-Bit Cryptographic S-Boxes. Cryptology ePrint Archive, Report 2019/1379, 2019.

Soumission à l'appel NIST pour standardisation de cryptographie légère

[ABC$^+$] Alexandre Adomnicai, Thierry P. Berger, Christophe Clavier, Julien Francq, Paul Huynh, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud, and Gaël Thomas. Lilliput-AE : a New Lightweight Tweakable Block Cipher for Authenticated Encryption with Associated Data. Submission to the NIST Lightweight Cryptography Standardization Process.

Glossaire

AES	Advanced Encryption Standard Algorithme de chiffrement symétrique établi comme standard par le NIST en 2001 pour remplacer le DES
ANF	Algebraic Normal Form Représentation particulière d'une S-box grâce aux équations de chaque fonction composante
Arduino Uno	Microcontrôleur programmable
ARK	Fonction AddRound Key de l'AES
attaque	Ensemble de techniques utilisées afin de déduire de l'information sur la clé secrète
clé	Élément secret utilisé par les algorithmes de chiffrement
contre-mesure	Changements apportés à un algorithme ou son implémentation afin de résister à des attaques
CPA	Correlation Power Analysis Attaque exploitant la corrélation entre les consommations de courant et les poids de Hamming d'une valeur intermédiaire afin de gagner de l'information sur la clé
DDT	Difference Distribution Table Table des différentielles de sortie en fonction des différentielles d'entrée d'une S-box
DES	Data Encryption Standard Algorithme de chiffrement symétrique établi comme standard par le NIST en 1975 jusqu'à 2001 où il a été remplacé par l'AES

DFA	Differential Fault Analysis Attaque exploitant les résultats de chiffrement fautés afin de gagner de l'information sur la clé
DPA	Differential Power Analysis Attaque exploitant la différentielle de consommation de courant afin de gagner de l'information sur la clé
FPGA	Field-Programmable Gate Array Circuits intégrés reprogrammables
glitch	Comportement inadéquat de portes logiques qui changent plusieurs fois d'état
HD	Hamming Distance Poids de Hamming du XOR de deux valeurs
HW	Hamming Weight Poids de Hamming, correspond au nombre de 1 dans la représentation binaire
LAT	Linear Approximation Table Biais d'approximation des sorties en fonction des entrées d'une S-box
Lilliput-TBC	Lilliput Tweakable Bloc Cipher Algorithme de chiffrement par bloc paramétrable
LSB	Least Significant Bit Bit de poids faible
MC	Fonction MixColumns de l'AES
MIA	Mutual Information Analysis Attaque exploitant l'information mutuelle entre la consommation de courant et les poids de Hamming afin de gagner de l'information sur la clé
MISTY	Mitsubishi Improved Security Technology Instance particulière d'un réseau de Feistel
modèle	Distribution jointe théorique

MSB	Most Significant Bit Bit de poids fort
MV	Maximum de Vraisemblance
NIST	National Institute of Standards and Technology Institut américain chargé d'établir les standards cryptographiques notamment
PACA	Passive and Active Combined Attacks Attaque utilisant à la fois une analyse passive et une analyse active afin de passer outre des contre-mesures empêchant l'application de l'une ou l'autre indépendamment
PACLIDO	Protocoles et Algorithmes Cryptographiques Légers pour l'Internet Des Objets Projet ayant pour but d'étudier la sécurisation de l'internet des objets
RSA	Algorithme de chiffrement asymétrique tirant son nom de ses inventeurs Rivest, Shamir et Adleman
SB	Fonction SubBytes de l'AES
S-box	Table de substitution, élément non linéaire de certains algorithmes de chiffrement
SPA	Simple Power Analysis Attaque exploitant la lecture visuelle d'une ou plusieurs traces de consommation de courant afin de gagner de l'information sur la clé
SPN	Substitution Permutation Network Algorithme basé sur l'alternance de fonctions de substitution et de permutation
SR	Fonction ShiftRows de l'AES
thresholdable	Est dit d'un circuit si il est possible de le masquer par TI sous les conditions exprimées
thresholder	Action de masquer par TI un circuit
TI	Threshold Implementation Contre-mesure de type masquage principalement matérielle basée sur le calcul multi-parties

Chapitre 1

Introduction aux attaques physiques

Nous présentons ici le contexte de cette thèse. Tout d'abord nous présentons les rudiments de la cryptologie, ce qui passe par un peu d'histoire de la cryptographie et de la cryptanalyse. Ensuite, nous nous intéressons aux notions d'attaques par analyse de canaux cachés. Ainsi, des attaques connues telles que la SPA sur le key schedule, la DPA, la CPA et la MIA seront exposées. Des contre-mesures capables d'empêcher ces attaques seront ensuite présentées, et une attention toute particulière sera apportée à celle de type masquage.

1.1 Introduction à la cryptologie

La cryptologie signifie la science du secret et englobe à la fois la cryptographie et la cryptanalyse. La cryptographie est l'art de rendre des informations incompréhensibles de tous, procédé appelé chiffrement, hormis des individus destinataires, qui procéderont au déchiffrement. Cela est souvent possible grâce au partage d'un procédé/méthode de chiffrement, secret ou non, ou encore d'une valeur/mot secret permettant le déchiffrement simple. Secret désigne connu ou possédé uniquement par l'émetteur et le(s) destinataire(s).
D'un autre côté, des individus peuvent souhaiter avoir accès à ces informations sans en être les destinataires, souvent pour en tirer un avantage. L'art de déchiffrer ces informations sans en posséder le droit légitime et les connaissances nécessaires est appelé cryptanalyse.
Plusieurs raisons expliquent ce comportement, et nous pouvons citer comme plus illustre exemple la communication militaire. Dès lors que deux camps s'opposent, la connaissance des faits et gestes et des ordres de l'ennemi donne un avantage décisif puisqu'il permet de s'adapter à la situation avant même que les faits n'arrivent. C'est pourquoi un camp cherche toujours à communiquer ses ordres de manière à ce que uniquement ses troupes les comprennent, et c'est aussi la raison pour laquelle le camp adverse tente de les comprendre également.

La cryptographie repose sur trois grands principes :

— La confidentialité : Faire en sorte que seuls les destinataires puissent déchiffrer l'information

— L'authenticité : Permettre d'identifier avec certitude l'émetteur du message

— L'intégrité : Assurer que l'information ne puisse pas être modifiée après l'envoi sans que le destinataire ne s'en rende compte

L'information, ou message, est appelée le texte clair (ou le clair). Le principe de la cryptographie est de transformer ce texte clair en un texte chiffré (ou le chiffré) grâce à un algorithme de chiffrement et une clé, le secret. Le texte chiffré peut être transmis non secrètement à son destinataire puisqu'il est, par définition, incompréhensible à tout autre individu. Le destinataire retrouve le message clair grâce à un algorithme de déchiffrement et une clé (potentiellement la même que pour le chiffrement). Dans la suite de ce chapitre, nous noterons le message M, le chiffré C, l'algorithme de chiffrement E, de déchiffrement E^{-1}, et la clé K.

La figure 1.1 représente la transmission d'un message chiffré comme présenté précédemment. Alice possède un clair qu'elle chiffre grâce à la clé, puis l'envoie à Bob qui peut déchiffrer grâce à la clé. Entre les deux, Eve écoute le message chiffré et cherche à retrouver le clair.

Il existe principalement deux moyens de communiquer secrètement. Le premier, qui est aussi le plus ancien, consiste à garder le procédé de chiffrement secret. Faire reposer la sécurité de la communication sur le secret du moyen de chiffrement pose beaucoup de problèmes. D'une part, tous les individus possédant ce système peuvent déchiffrer toutes les communications, ce qui

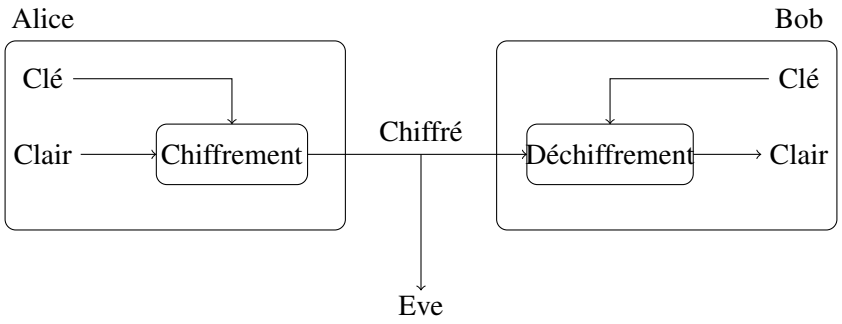


FIGURE 1.1 – Alice transmet un message à Bob de manière sécurisée

implique de posséder un moyen différent pour chaque groupe de communication. D'autre part, un seul individu dévoilant publiquement le système de chiffrement le rend tout de suite obsolète, impliquant la création d'un nouveau procédé. Cela peut provenir d'un individu malveillant faisant partie du groupe possédant la connaissance ou l'outil de chiffrement, ou encore de l'étude approfondie de l'outil s'il a été subtilisé par exemple. Le second moyen est d'avoir un procédé de chiffrement paramétré par une valeur secrète, appelée communément clé, partagée par les individus voulant communiquer. Cette méthode résout les problèmes soulevés précédemment puisqu'il est facile de générer une clé, il est donc possible d'en créer autant que de groupes de communications, et la divulgation d'une clé au public n'implique le changement que de cette valeur, et non pas de tout le procédé. À la fin du XIX^{e} siècle, Kerckhoffs introduit un principe qui porte son nom, et qui stipule que la sécurité doit reposer uniquement sur le secret de la clé.

La cryptanalyse peut consister soit à retrouver le clair à partir du chiffré sans en être le destinataire, c'est-à-dire sans posséder toutes les informations généralement nécessaires à son déchiffrement, ou bien à retrouver l'algorithme/procédé si celui-ci est secret, ou encore à retrouver la clé. Pour donner un exemple simple, un chiffrement consistant en une substitution des lettres d'un message peut être compromis par l'étude statistique de la distribution des lettres du chiffré et de la langue du clair. Étant donné la distribution des lettres de la langue et de celles du chiffré, un classement ordonné de chacune donne la correspondance lettre clair/lettre chiffré. Dans la cryptanalyse moderne, l'objectif de l'adversaire consiste principalement à retrouver la clé utilisée. Même si le principe de Kerckhoffs est largement appliqué de nos jours, certaines instances gardent leurs procédés secrets en plus d'utiliser une clé. L'art de retrouver ce procédé est appelé la rétro-conception.

Si l'ensemble des clés possibles est petit, alors une manière simple de la retrouver est de posséder un couple clair/chiffré, puis de chiffrer le clair avec toutes les clés, jusqu'à l'obtention du bon chiffré. Dans le cas contraire, la cryptanalyse cherche à réduire ce champ jusqu'à un nombre raisonnable à tester. La cryptanalyse que nous dirons "classique" repose principalement sur l'étude conjointe du comportement des clairs et des chiffrés afin d'en déduire de l'information sur la clé grâce à la connaissance des liens mathématiques impliqués dans l'algorithme de chiffrement.

Intrinsèquement, cette cryptanalyse tire partie des faiblesses cryptographiques de l'algorithme. Un autre type de cryptanalyse, dite "par canaux cachés" ou "par analyse de canaux auxiliaires", étudie des phénomènes physiques produits par les implémentations de ces mêmes algorithmes. L'individu tentant de retrouver la clé est souvent appelé l'attaquant ou l'adversaire. Les méthodes et actions employées par un attaquant afin de retrouver la clé sont appelées des attaques.

La cryptologie est un domaine très intéressant de nos jours puisque la cryptographie est employée partout, que ce soit dans nos poches avec les cartes bancaires et nos téléphones, dans nos bureaux pour la sécurité d'accès, ou pour l'industrie. Il est très important d'étudier les attaques afin de s'en prémunir et ainsi de sécuriser du mieux possible toutes ces communications.

1.2 La cryptographie

À l'heure du numérique, nous considérons que les informations sont sous format binaire. Pour cette raison nous parlerons des clairs, des chiffrés et des clés sous forme binaire, et le plus souvent sous forme d'octets.

De nos jours, deux grandes familles d'algorithmes de chiffrement sont employées, la cryptographie symétrique et la cryptographie asymétrique, chacune ayant des propriétés bien particulières.

1.2.1 Le chiffrement symétrique

Le chiffrement symétrique est le plus intuitif et le plus ancien. Il suppose le partage d'un secret, la clé K, entre les deux parties communicantes puisqu'elle est indispensable au chiffrement et au déchiffrement (d'où son nom de "symétrique"). Généralement, cette clé est relativement petite, et le chiffrement rapide.

Parmi les algorithmes symétriques, il existe deux grandes familles : les algorithmes dits à flots, et les algorithmes dits de chiffrement par blocs.

Le chiffrement à flot consiste généralement à générer un flot de bits grâce à un générateur pseudo aléatoire paramétré par la clé, et de masquer, c'est-à-dire xorer, ce flot avec le clair pour obtenir le chiffré. Puisque la séquence de bits générés, appelée "suite chiffrante" est reproductible grâce à la connaissance de la clé, le destinataire est capable de générer exactement la même, puis de xorer le chiffré avec cette même séquence, redonnant ainsi le message.

Le chiffrement par blocs consiste à découper le message en blocs de taille fixe, blocs souvent eux-mêmes découpés en un certain nombre d'octets, puis de chiffrer chacun de ces blocs. Le chiffrement consiste généralement en plusieurs tours, chacun constitué d'un petit nombre de fonctions simples appliquées à l'état S. L'état initial consiste en le message, et celui-ci est transformé par l'application itérative des fonctions jusqu'à donner l'état final, le chiffré. La clé est généralement utilisée à chaque tour dans l'une des fonctions, xorée à l'état par exemple.

Ce sont les chiffrements par blocs qui seront traités dans ce document. Nous nous intéressons particulièrement à deux formes de chiffrement par bloc : ceux de type substitution-permutation tels que l'AES, et les schémas de Feistel tels que le DES.

La cryptographie moderne est souvent associée à Shannon grâce à sa publication [Sha49]. Il y introduit notamment la notion de *diffusion* et de *confusion* comme propriétés nécessaires afin de rendre les algorithmes robustes à la cryptanalyse classique. Nous avons brièvement exposé que des études statistiques peuvent donner de l'information à un attaquant par des biais qui apparaîtraient dans les chiffrés. La diffusion a pour but d'effacer de tels biais. Une notion liée à la diffusion est l'effet d'avalanche. Ce nom provient du fait qu'un chiffrement ayant une bonne diffusion doit associer à un message très légèrement modifié, un chiffré complètement différent. Poussé à son maximum, la modification d'un seul bit du message modifie tous les bits de sortie avec une probabilité de $\frac{1}{2}$. La diffusion est souvent réalisée par permutation des bits de l'état. La confusion quant à elle doit rendre la relation entre la clé et le chiffré compliquée. Pour cela, il est par exemple possible de faire en sorte que chaque bit du chiffré dépende de chaque bit de la clé, et que les équations qui lient ces deux soient complexes. La confusion est souvent réalisée par l'utilisation de tables de substitutions.

1.2.2 Le chiffrement asymétrique

La contrainte principale liée à la cryptographie symétrique est le partage de la clé secrète par les deux parties avant l'établissement de l'échange sécurisé. C'est donc une étape paradoxale puisque établir cet échange sécurisé nécessite le partage sécurisé d'une information. La clé étant petite, cela a quand même pour avantage d'être plus facile à transmettre secrètement.
Le chiffrement asymétrique, dit à clé publique, est apparu en 1976 et solutionne ce problème. Il a été introduit dans [DH76]. La cryptographie asymétrique propose qu'un individu ne possède pas seulement une clé secrète, mais aussi une clé publique. La clé publique permet de chiffrer, et la clé privée de déchiffrer. Cet individu peut alors afficher sa clé publique, et quiconque souhaitant communiquer vers lui peut alors l'utiliser afin de chiffrer des messages. La clé privée étant indispensable au déchiffrement, seul le possesseur est capable de déchiffrer. Ce système ne permet la communication que dans un seul sens. Pour assurer un dialogue, il est donc nécessaire que les deux parties s'échangent leurs clés publiques, ce qui peut se faire de manière non sécurisée, et gardent secrètes leurs clés privées respectives.
Généralement, la cryptographie asymétrique se base sur des fonctions à sens unique, et la sécurité repose sur l'incapacité à résoudre des problèmes difficiles tels que la factorisation d'entiers par exemple. Généralement les clés sont plus longues que celles de cryptographie symétrique, et le chiffrement plus long. La cryptographie asymétrique est donc parfois utilisée pour apporter une solution au premier problème du symétrique, à savoir le partage sécurisé de la clé avant l'échange. Il suffit pour cela de transmettre de manière asymétrique un message contenant la clé qui sera utilisée ensuite dans un chiffrement symétrique plus rapide.

Nous exposons rapidement ici un algorithme asymétrique car il sera étudié dans des exemples d'attaques de la littérature, ainsi que deux algorithmes symétriques de chiffrement par blocs qui seront les plus traités dans ce document.

1.2.3 Le chiffrement asymétrique RSA

Le RSA, du nom de ses trois inventeurs Rivest, Shamir et Adleman, est un chiffrement asymétrique. Il emploie donc deux clés, une publique destinée à chiffrer, et une privée destinée à déchiffrer. Les étapes de la création de ces clés sont :

— Tirer deux grands nombres premiers p et q

— Calculer $n = pq$

— Calculer $\phi(n) = (p-1)\cdot(q-1)$ avec ϕ l'indicatrice d'Euler

— Choisir e premier avec $\phi(n)$ et généralement petit

— Calculer $d = e^{-1} \bmod \phi(n)$

La clé publique consiste en n et e. Le chiffrement d'un message M consiste à calculer $C = M^e \bmod n$. La clé privée consiste en d et n. Le déchiffrement consiste en le calcul $M = C^d \bmod n$. La sécurité repose sur le fait que, sans la connaissance de p et de q, il est difficile de calculer $\phi(n)$, et que factoriser n est difficile.

1.2.4 Le chiffrement symétrique standard AES

L'AES [FIP01] est un chiffrement par bloc de type substitution-permutation (SPN) publié en 2001 et a été choisi comme remplaçant du DES comme standard de chiffrement par le NIST. Il existe trois variantes de cet algorithme, chacune ayant pour particularité sa taille de clé (128, 192 et 256 bits) et son nombre de tours (10, 12 et 14). Nous présentons ici la version la plus simple à clé de 128 bits, décrite en figure 1.3. La taille du bloc, et donc du message, de l'état et du chiffré est, elle, toujours de 128 bits, soit 16 octets. Pour des raisons de clarté, l'état est souvent représenté comme une matrice 4×4 d'octets. L'indice de position de chacun des 16 octets dans cette matrice est croissant de haut en bas puis de gauche à droite. Ainsi la colonne la plus à gauche contient de haut en bas les octets 0, 1, 2 et 3, puis la seconde colonne 4, 5, 6 et 7, etc. Une telle matrice est représentée en figure 1.2.

Les quatre fonctions qui constituent chacun des 10 tours sont les suivantes :

— `AddRoundKey` notée ARK

— `SubBytes` notée SB

— `ShiftRows` notée SR

— `MixColumns` notée MC

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

FIGURE 1.2 – Indice de positions de la matrice 4×4 de l'AES

Le `AddRoundKey` effectue un xor entre l'état S et la clé de tour K_i, tous deux de 128 bits. Chacune des clés de tour est dérivée de la clé principale comme nous l'expliquerons plus tard. Ainsi $\texttt{AddRoundKey}(S) = S \oplus K_i$. C'est la seule fonction impliquant la clé dans le calcul de l'état.
Le `SubBytes` effectue une substitution de chacun des 16 octets de S en suivant une table appelée `Sbox` et notée SB. Les S-boxes sont une partie très importante de la robustesse des algorithmes en assurant la confusion, et le chapitre 2 est principalement consacré à ces dernières. L'AES utilise une permutation des éléments de $GF(2^8)$ basée sur l'inversion et qui présente de fortes propriétés cryptographiques.
Le `ShiftRows` est une opération de permutation des octets de l'état. Cette fonction permet en partie la diffusion. Chaque ligne $i = \{0,...,3\}$ de la matrice d'état subit une rotation de i octets vers la gauche.
Enfin, le `MixColumns` participe également à la diffusion en calculant chaque nouvelle colonne de l'état grâce à une multiplication matricielle.

Chacune des 11 clés de tours K_i, $i \in \{0,...,10\}$, de l'AES 128 est calculée de manière déterministe par le `KeySchedule` grâce à la clé K, dite clé maître, avec comme particularité $K_0 = K$. Nous représentons la clé étendue de 176 octets sous forme de 11 matrices 4×4, chacune correspondant à une clé de tour K_i, et dont la répartition des octets est faite de la même manière que celle des octets de message dans l'état. Une clé de tour K_i est constituée de 16 octets $\{k_{(i,0)},...,k_{(i,15)}\}$. Les octets de la clé étendue sont calculés grâce aux règles suivantes :

$$\begin{aligned} k_{(i+1,j)} &= k_{(i,j)} \oplus k_{(i+1,j-4)} && j \notin \{0,1,2,3\} \\ k_{(i+1,j)} &= k_{(i,j)} \oplus SB\left(k_{(i,12+(j+1) \bmod 4)}\right) \oplus Rcon_{i,j} && j \in \{0,1,2,3\} \end{aligned}$$

La première équation stipule que les 3 dernières colonnes d'une clé de tour K_{i+1} résultent du xor entre leur colonne précédente et la colonne de même indice de la clé K_i. La seconde ligne stipule que la première colonne d'une de tour résulte du xor entre la colonne précédente ayant subi une rotation et SB, une constante *Rcon* et la colonne de même indice de la clé K_i. Le schéma de la

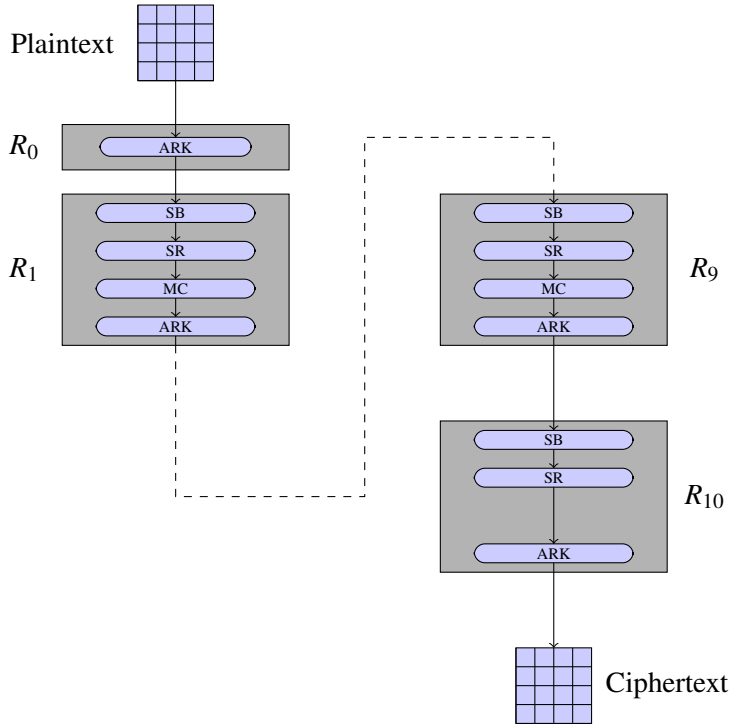


FIGURE 1.3 – AES 128 bits

figure 1.7 en page 22 et issu de [VBC05] donne les relations entre octets de clé étendue.

1.2.5 Schéma/réseau de Feistel

Le DES, prédécesseur de l'AES, est un algorithme de chiffrement par blocs basé sur un schéma/-réseau de Feistel. Plutôt que de présenter ce dernier, nous proposons plutôt d'expliquer le schéma en général, le DES n'étant qu'une instance particulière du schéma.

Un schéma de Feistel classique consiste en plusieurs tours manipulant deux branches qui se croisent. La figure 1.4 présente un tel schéma.
Le message M est divisé en deux parties, la partie gauche L_0 et la partie droite R_0. Cette dernière subit la fonction de tour paramétrée par la clé de tour, puis est xorée à la première qui, elle, ne subit aucune modification. Les branches sont ensuite échangées, et un nouveau tour commence. Le chiffré est la concaténation des deux dernières branches sans croisement. Les équations qui caractérisent un tel schéma sont :

$$\begin{aligned} M &= (L_0||R_0) \\ L_{i+1} &= R_i \\ R_{i+1} &= f(R_i) \oplus L_i \\ C &= (R_n||L_n) \end{aligned}$$

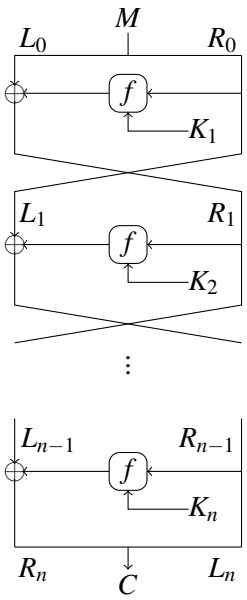


FIGURE 1.4 – Schéma de Feistel

Où f est la fonction de tour et peut consister elle-même en plusieurs fonctions. Généralement, f consiste en au moins trois fonctions, la première xore la clé, la seconde applique une S-box et la troisième diffuse.

Nous pouvons citer deux particularités principales inhérentes à un tel schéma. Premièrement, seule la moitié de l'état est modifiée à chaque tour et la clé ne fait que la moitié de la taille du bloc. Deuxièmement, le chiffrement et le déchiffrement sont strictement identiques, pourvu que les mêmes clés de tour soient utilisées en sens inverse.

1.3 La cryptanalyse

1.3.1 La cryptanalyse classique

La cryptanalyse classique, qui consiste à retrouver la clé grâce à l'étude de couples (clairs, chiffrés) et de la connaissance de l'algorithme employé en tirant parti des liens mathématiques induits par ce même algorithme, sera étudiée plus en détail dans le chapitre 2. Nous y présenterons notamment les attaques algébriques, différentielles et linéaires.

1.3.2 La cryptanalyse par canaux cachés/auxiliaires

La cryptanalyse par canaux cachés/auxiliaires est plus récente, et tire parti du fonctionnement de l'appareil sur lequel est implémenté l'algorithme de chiffrement. Cela peut passer par l'exploitation des manifestations physiques produites par l'appareil lors de l'exécution d'un chiffrement,

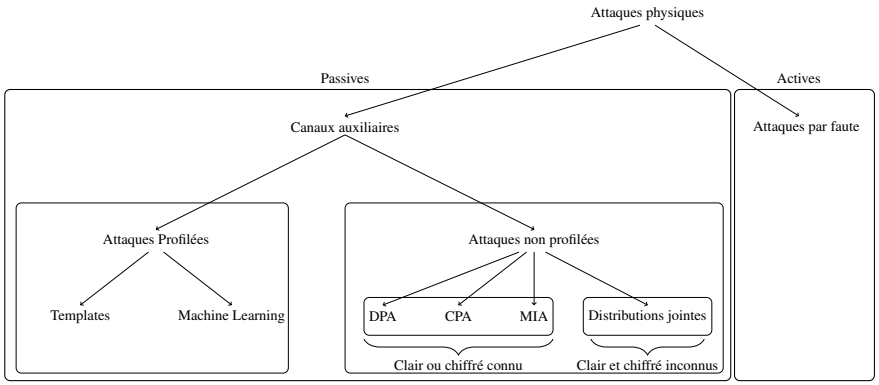


FIGURE 1.5 – Typologie des attaques physiques

jusqu'à l'altération de l'exécution de l'algorithme afin de déterminer la clé. Les attaques inhérentes à cette branche sont appelées les attaques par canaux cachés ou par canaux auxiliaires ou encore attaques physiques. Contrairement à la cryptanalyse classique qui exploite les propriétés mathématiques liées à l'algorithme lui-même, la cryptanalyse par canaux auxiliaires exploite les faiblesses liées à la réalisation physique du chiffrement. Introduites par Kocher au milieu des années 90, les attaques physiques sont souvent divisées en deux classes :

— Les attaques par observations ou passives. Exemple : DPA, CPA, MIA

— Les attaques actives. Exemple : Attaques par injection de fautes

La première catégorie exploite les données physiques résultantes des calculs sans les perturber, alors que la seconde agit sur l'exécution.
Notons que le terme "Side Channel", traduit par attaque par canaux cachés/auxiliaires, semble parfois faire référence aux attaques physiques en général, et parfois aux attaques physiques par observations en particulier. Dans ce document, nous nous intéressons particulièrement aux attaques par observations, et nous ferons références à ces dernières en utilisant le terme attaques par canaux cachés ou auxiliaires. Lorsque la manifestation physique observée dans ce type d'attaque est liée à une donnée manipulée par l'algorithme, nous parlerons de fuite. La figure 1.5 représente quelques attaques physiques.

La force de l'attaquant est en grande partie liée à ces classes d'attaques. Un attaquant uniquement capable de faire des observations sera dit faible car peu de matériel et d'efforts sont nécessaires. Au contraire, un attaquant capable de modifier l'appareil en profondeur se doit d'être équipé spécialement et met en œuvre beaucoup plus de ressources.
Les mesures physiques les plus exploitées à ce jour concernent le courant utilisé par l'appareil, à savoir la consommation de courant, et les émanations électromagnétiques. La mesure de ces grandeurs physiques est généralement réalisée sur une certaine période de temps (toute ou partie de l'exécution cryptographique), ce qui résulte en un ensemble de mesures en fonction du temps,

ensemble que nous appelons trace de courant. Puisque la consommation et les émanations électromagnétiques sont liées, de nombreuses attaques s'appliquant à un type de traces s'appliquent également à l'autre. Dans la suite de ce document, nous ne distinguerons pas les deux et parlerons systématiquement de traces de courant. Puisque les attaques par canaux cachés exploitent les fuites, nous parlerons de fuites de courant.

Implémentation logicielle et matérielle

L'implémentation d'un algorithme de chiffrement sera souvent orientée logiciel ou matériel. Le premier cas de figure suppose qu'un algorithme codé est exécuté par un microprocesseur. C'est par exemple le cas des microcontrôleurs. L'exécution est généralement caractérisée par sa nature séquentielle puisqu'un cycle d'horloge, qui sert à synchroniser l'exécution de l'algorithme, correspond à l'exécution d'une seule instruction.
Le second cas de figure suppose que l'algorithme est exécuté grâce à un ensemble de portes logiques. Le circuit est dédié ou reprogrammé spécifiquement pour cet algorithme. C'est par exemple le cas des FPGA. L'exécution est cette fois-ci généralement caractérisée par sa nature parallèle puisqu'en un seul cycle d'horloge, le signal traverse plusieurs portes logiques, ce qui correspond à plusieurs instructions.

Fuite univariée et multivariée

Selon que l'implémentation de l'algorithme de chiffrement soit orientée logiciel ou matériel, deux types de fuites différentes peuvent être observés [MM17]. Le premier, généralement associé aux implémentations logicielles, est la fuite multivariée. Cela traduit qu'une fuite, c'est-à-dire un instant temporel, n'est liée qu'à une seule instruction/donnée/variable à la fois. Dans un tel cas de figure, si trois variables fuient, alors la trace de courant comporte trois fuites à trois instants temporels distincts, chacune liée à une seule variable. C'est principalement ce cas qui sera abordé dans ce document. La fuite univariée correspond généralement aux implémentations matérielles. Le circuit exécutant plusieurs instructions au cours d'un même cycle d'horloge, la fuite correspond à la somme des consommations de plusieurs variables. Dans ce cas de figure, plusieurs instructions/variables successives peuvent ne produire qu'une seule consommation, combinaison des consommations individuelles de chacune.

Attaque temporelle

L'une des premières attaques par canaux cachés est une attaque par observations qui exploite le temps d'exécution des calculs (Timing Attack). Introduite dans [Koc96], elle consiste à récupérer successivement des sous-parties d'une clé RSA. Pour illustrer facilement l'exécution en temps non constant, nous présentons un exemple simple avant l'explication de l'attaque : les chiffrement

et déchiffrement consistent en une exponentiation modulaire. Celle-ci peut être réalisée grâce des algorithmes d'exponentiation rapide tels que le *right-to-left* ou *left-to-right square and multiply*. Ces techniques utilisent la représentation binaire de l'exposant et appliquent successivement des élévations au carré et des multiplications modulaires. L'algorithme 1 présente l'exponentiation left-to-right qui tient son nom de ce qu'il considère les bits de l'exposant de la gauche vers la droite, du MSB au LSB.

Algorithme 1 : Exponentiation rapide left-to-right

Données : m, n, $e = (e_{k-1}, ..., e_0)_2$

Résultat : $m^e \bmod n$

```
1 r = 1
2 pour b de k-1 à 0 faire
3 |   r = r^2 mod n
4 |   si e_b == 1 alors
5 |   |   r = r · m mod n
6 |   fin
7 fin
8 retourner r
```

Remarquons que cet algorithme ne s'exécute pas en temps constant, c'est-à-dire que le nombre d'instructions exécutées, et donc le temps d'exécution, dépend des bits de l'exposant/clé. Cela est dû au branchement conditionnel qui décide de faire ou non la multiplication modulaire. Pour illustrer ce comportement, si les durées d'une multiplication et d'une mise au carré sont identiques, alors l'exponentiation avec un exposant constitué uniquement de 1 sera deux fois plus longue que celle avec un exposant de même taille essentiellement composé de 0. La connaissance de la clé et de la durée de chaque instruction permet de connaître de manière déterministe, au bruit près, la durée du calcul.

Dans [Koc96], les auteurs exploitent un autre aspect de l'algorithme rendant le temps d'exécution non constant : la réduction modulaire de Montgomery peut nécessiter ou pas la soustraction de n. Dans le premier cas, la multiplication est plus longue. La connaissance de l'architecture permet de connaître à l'avance les couples de valeurs dont la multiplication modulaire nécessite une telle soustraction. Dans ces conditions, à clé constante, le temps d'exécution est différent d'un clair à l'autre. Connaissant les b premiers bits de la clé, l'algorithme utilisé, une approximation du temps de chaque instruction (élévation au carré et les deux cas de multiplication), et les couples de valeurs dont la multiplication est longue, il est possible de calculer le temps attendu t_b pour le calcul de ces b bits étant donné un message. Supposons qu'un attaquant mesure un grand nombre N de temps d'exécutions $(T_i)_{1 \leq i \leq N}$ pour différents messages avec une même clé. L'hypothèse de b bits de clé permet de calculer le temps partiel $(t_{(b,i)})_{1 \leq i \leq N}$ attendu pour chacun des messages. La différence entre le temps relevé et le temps partiel calculé pour un message i est

notée $\delta_i = T_i - t_{(b,i)}$. Si l'hypothèse est correcte, alors la variance des δ_i est la variance des bits restants (hors hypothèse) uniquement puisque $t_{(i,b)}$ est correctement évalué. Si l'hypothèse est incorrecte, la variance n'étant plus uniquement engendrée par les autres bits, sa valeur attendue est donc supérieure. L'hypothèse engendrant la variance la plus faible est sélectionnée comme étant la bonne. L'attaquant peut ensuite faire l'hypothèse de plus de bits et procéder de la même manière jusqu'à retrouver la clé entière.

Analyse simple de courant

La SPA, attaque par analyse simple de courant (Simple Power Analysis), exploite le relevé de la consommation de courant de l'appareil effectuant le chiffrement pour en déduire entre autres l'état interne des données, le déroulement de l'algorithme, ou encore de l'information vis-à-vis de la la clé, ce qui en fait une attaque par observations. Un exemple simple de SPA permettant de déduire un à un les bits d'une clé RSA est décrit par Kocher dans [KJJ+98]. Il suppose un RSA utilisant une exponentiation rapide telle que décrite dans l'algorithme 1. Rappelons le caractère non constant de ce dernier selon que le bit considéré soit 1 ou 0. Dans le premier cas une mise au carré et une multiplication sont effectuées, et dans le second, la multiplication n'est pas faite. Si, de par la conception de l'appareil, la consommation est différente selon qu'il exécute une multiplication ou une mise au carré, et si cette différence est identifiable uniquement grâce à l'observation de la trace de courant, alors un attaquant peut en déduire la clé. Étant donné une trace de courant d'un RSA utilisant la clé d, si l'attaquant est capable d'identifier les différentes instructions effectuées (carrés et multiplications), alors il lui suffit d'en déduire successivement les bits de l'exposant grâce à la démarche suivante : si un carré est suivi d'une multiplication, le bit courant est 1. Si deux carrés se suivent, le bit courant est 0.

Analyse différentielle de courant

La DPA, analyse différentielle de courant (Differential Power Analysis), présentée dans [KJJ+98, KJJ99], symbolise les fondations des attaques statistiques par canaux auxiliaires. La DPA est une attaque par observations puisqu'elle exploite des traces de consommation de courant. C'est une analyse statistique puisque, contrairement à la SPA, qui ne nécessite qu'une seule trace, celle-ci en exploite plusieurs à la fois afin d'en faire ressortir des comportements qui ne sont pas visibles à l'œil nu, grâce à des calculs statistiques. Pour fonctionner, la consommation de courant doit être liée au poids de Hamming de la donnée manipulée à l'instant de la fuite. Le poids de Hamming (Hamming Weight, abrégé en HW) d'une donnée x sera noté noté $\mathrm{HW}(x)$ et correspond au nombre de 1 dans la représentation binaire de x. L'attaque de base est présentée sur le DES, mais celui-ci n'étant pas présenté dans ce document, nous proposons d'expliquer son principe sur l'AES puisque l'attaque y est strictement identique. Soient N traces de P points $(T_i = (T_i[1], ..., T_i[P]))_{1 \leq i \leq N}$ issues des chiffrements de N messages aléatoires connus

$(M_i = (M_i[0],...,M_i[15]))_{1\leq i\leq N}$. En aparté, remarquons que cette attaque peut aussi utiliser les chiffrés à la place des messages. Nous supposons que ces traces sont synchronisées, c'est-à-dire qu'à un instant t donné, l'instruction exécutée par l'appareil est la même pour toutes les traces, et ce pour tout t. L'attaque consiste à faire une hypothèse sur une portion de la clé, typiquement un octet dans le cas de l'AES, puis de calculer une donnée intermédiaire, dite donnée ciblée, puis enfin de confronter le comportement de la consommation de l'appareil au comportement attendu vis-à-vis de la variable calculée. La donnée ciblée doit naturellement être fonction de la portion de clé dont l'attaquant fait l'hypothèse, sans quoi le comportement attendu serait indépendant de la valeur de celle-ci. L'AES est un algorithme de chiffrement opérant sur des octets. Une cible communément utilisée est la sortie de S-box de premier tour puisque chacune des sorties n'utilise qu'un seul octet de la clé, ce qui permet de faire 2^8 hypothèses pour chacun des 16 octets. Diviser ainsi la clé en sous-parties puis en déduire la valeur de chacune indépendamment est appelé *diviser et régner* (divide and conquer). Voici les étapes de la DPA :

- Faire l'hypothèse g d'un octet o de la clé
- Pour chacun des messages, calculer la sortie de la S-box o : $x = \mathrm{SB}(M_i[o] \oplus g)$
- Partitionner les N traces en 2 sous-ensembles S_0 et S_1 selon les critères suivants :
  - S_0 : Le bit b de x vaut 0
  - S_1 : Le bit b de x vaut 1

  Où b est un indice de bit arbitraire, $0 \leq b \leq 7$. Pour la suite, les traces T_i appartenant au sous-ensemble S_0 se voient renommées en $(T_j^0)_{1\leq j\leq \#S_0}$ et celles appartenant à l'ensemble S_1 se voient renommées en $(T_j^1)_{1\leq j\leq \#S_1}$.
- Calculer la trace de la différence des traces moyennes des deux ensembles :

$$\Delta_g[j] = \frac{\sum_{i=0}^{\#S_1-1} T_i^1[j]}{\#S_1} - \frac{\sum_{i=0}^{\#S_0-1} T_i^0[j]}{\#S_0}$$

Pour simplifier les explications, commençons par le cas où chaque trace ne contient qu'un seul point, celui correspondant à la manipulation de la sortie de la S-box ciblée. Si l'hypothèse correspond à la clé, les deux ensembles sont effectivement triés selon les critères imposés puisque les valeurs calculées sont effectivement celles manipulées par l'appareil. Dans ce cas, la moyenne des poids de Hamming des valeurs ciblées de l'ensemble S_0 est 3.5 puisqu'un bit est systématiquement égal à 0. La même moyenne pour S_1 est 4.5 puisqu'un bit est systématiquement égal à 1. Si la consommation est fonction du poids de Hamming, puisque la différence des moyennes des deux ensembles est non nulle, la différence des consommations l'est aussi. Lorsque l'hypothèse n'est pas la bonne valeur, les ensembles ne sont pas correctement construits puisque les valeurs permettant d'appliquer les critères de sélection ne sont pas celles réellement manipulées. Grâce au grand nombre de traces, et à la particularité de confusion de la S-box, l'attribution d'une trace à un ensemble ou à l'autre s'apparente à une attribution aléatoire. Remarquons que l'attribution

n'est aléatoire qu'à la valeur du message $M_i[0]$ près puisque toutes les traces ayant la même valeur d'octet de message sont attribuées au même ensemble. De ce fait, les moyennes des fuites liées aux consommations des valeurs ciblées de chacun des ensembles sont à peu près équivalentes, et la différence des moyennes des consommations est faible.
L'hypothèse à laquelle correspond la différence des moyennes la plus élevée est sélectionnée comme étant la bonne valeur de clé.
Lorsque la trace comporte plusieurs points, deux types de points peuvent exister. Ceux qui fuient vis-à-vis de la valeur ciblée ou d'une fonction de cette valeur, et ceux indépendants de cette dernière. Le comportement des premiers est décrit dans le paragraphe précédent et ce sont eux qui permettent à l'attaque de fonctionner. Les seconds étant indépendants de la valeur ciblée, leur comportement peut s'apparenter à un comportement aléatoire, et donc une différence des moyennes proche de 0. La trace de la différence des traces moyennes contient soit uniquement des points proches de 0, soit des points proches de 0 et un ou plusieurs points de valeur plus élevée, ce qui se caractérise par un pic. L'hypothèse dont est issue la plus haute valeur de pic est sélectionnée comme étant la bonne valeur de clé. Remarquons que ces traces nous renseignent à la fois sur la valeur de la clé, mais aussi sur l'instant de manipulation de la donnée ciblée.
L'attaque se résume à un partitionnement des traces pour chaque hypothèse d'une partie de la clé, puis à l'utilisation d'un outil communément appelé distingueur, ici la différence des moyennes, qui confronte l'adéquation du comportement de chaque ensemble avec le comportement attendu. Ce principe est récurrent des attaques par canaux cachés.

Analyse par corrélation de courant

La CPA, analyse par corrélation de courant (Correlation Power Analysis), présentée dans [BCO04], est une autre attaque statistique par observations très similaire à la DPA. Son efficacité est néanmoins bien supérieure, c'est-à-dire que l'attaque requiert moins de traces pour en déduire la clé. Pour fonctionner, les traces doivent être synchronisées, les clairs (ou les chiffrés) doivent êtres accessibles, et l'appareil fuite selon une fonction affine du poids de Hamming de la donnée manipulée. Le but de cette attaque est de calculer la corrélation linéaire entre la série des consommations en un point et la série des poids de Hamming de la variable ciblée calculés selon chaque hypothèse de portion de clé. Si le modèle de consommation est affine en le poids de Hamming, la corrélation est maximale lorsque les poids de Hamming calculés sont effectivement ceux de la valeur ciblée.
Soient les mêmes N messages et traces que définis lors de la DPA. De la même manière que cette dernière, la CPA cible une valeur intermédiaire fonction d'un octet de clé, usuellement la sortie d'une S-box au premier tour. Soit $M_i[o]$ l'octet de message correspondant issu de M_i. Pour chaque hypothèse d'octet de clé g, les sorties de la S-box sont $V_g = (v_{(g,i)} = \mathrm{SB}(M_i[o] \oplus g))_{1 \leq i \leq N}$ et leurs poids de Hamming $H_g = (h_{(g,i)} = HW(v_{g,i}))_{1 \leq i \leq N}$. Pour chaque instant p des traces de consommation, il est possible de calculer le coefficient de corrélation de Pearson entre la série

des consommations à cet instant et la série des poids de Hamming calculés pour cette hypothèse. L'expression de ce coefficient est :

$$\rho_{g,p} = \frac{\mathrm{Cov}(T_p, H_g)}{\sqrt{\mathrm{Var}(T_p) \cdot \mathrm{Var}(H_g)}} = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(T_i[p] \cdot h_{(g,i)}\right) - \frac{1}{N}\sum_{i=0}^{N} T_i[p] \cdot \frac{1}{N}\sum_{i=1}^{N} h_{(g,i)}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N} T_i[p]^2 - \left(\frac{1}{N}\sum_{i=1}^{N} T_i[p]\right)^2} \cdot \sqrt{\frac{1}{N}\sum_{i=1}^{N} h_{(g,i)}^2 - \left(\frac{1}{N}\sum_{i=1}^{N} h_{(g,i)}\right)^2}}$$

où T_p désigne la série des consommations à l'instant p.
Le raisonnement est identique à celui de la DPA : les instants indépendants de la variable ciblée présentent une faible corrélation, et les instants liés à la variable ciblée donnent une forte corrélation dans le cas de la bonne hypothèse de clé, et faible dans les autres cas. La trace de CPA présente un pic à/aux l'instant(s) de manipulation de cette variable pour la bonne hypothèse de clé. La CPA a pour avantage d'exploiter tous les bits de la valeur ciblée, contrairement à un seul en DPA, ce qui la rend plus efficace.

Analyse par information mutuelle

La MIA, analyse par information mutuelle (Mutual Information Analysis), présentée dans [GBTP08] est une autre attaque statistique qui utilise les mêmes hypothèses que la CPA, à cela près que l'attaque fonctionne sur les appareils ayant un modèle de consommation fonction du poids de Hamming, qu'il soit affine ou non. Cette attaque utilise l'information mutuelle comme distingueur. L'entropie de Shannon $\mathrm{H}(X)$ d'une variable aléatoire discrète X définie sur l'espace $\mathcal{X}$ ayant pour distribution de probabilité $P_X = \{\Pr(X = x) \mid x \in \mathcal{X}\}$ est définie comme :

$$\mathrm{H}(X) = \sum_{x \in \mathcal{X}} \Pr(X = x) \cdot \log_2\left(\frac{1}{\Pr(X = x)}\right)$$

Elle mesure la quantité moyenne d'information apportée par l'observation de cette variable aléatoire. Plus intuitivement, elle désigne l'incertitude d'un tirage. Par exemple, le tirage à partir d'une distribution uniforme est très incertain puisque tout est équiprobable, ce qui résulte en une grande entropie. Au contraire, un tirage d'une distribution contenant de fortes et de faibles probabilités donnera une faible entropie.
L'entropie conjointe $\mathrm{H}(X,Y)$ est définie grâce aux probabilités conjointes :

$$\mathrm{H}(X,Y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \Pr(X = x, Y = y) \cdot \log_2\left(\frac{1}{\Pr(X = x, Y = y)}\right)$$

L'entropie conditionnelle $H(X \mid Y)$ s'exprime :

$$\mathrm{H}(X \mid Y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \Pr(X = x, Y = y) \cdot \log_2\left(\frac{1}{\Pr(X = x \mid Y = y)}\right)$$

L'information mutuelle $\mathrm{I}(X,Y)$ entre X et Y est définie comme :

$$\begin{aligned}\mathrm{I}(X,Y) &= \sum_{x\in\mathcal{X},y\in\mathcal{Y}} \Pr(X=x,Y=y)\log_2\left(\frac{\Pr(X=x,Y=y)}{\Pr(X=x)\Pr(Y=y)}\right)\\ &= \sum_{x\in\mathcal{X}} \Pr(X=x)\sum_{y\in\mathcal{Y}} \Pr(Y=y\mid X=x)\log_2\left(\frac{\Pr(Y=y\mid X=x)}{\Pr(Y=y)}\right)\end{aligned}$$

Et peut s'exprimer comme fonction de l'entropie :

$$\mathrm{I}(X,Y) = \mathrm{H}(X)+\mathrm{H}(Y)-\mathrm{H}(X,Y) = \mathrm{H}(X)-\mathrm{H}(X\mid Y)$$

L'information mutuelle tient compte de la dépendance entre les deux variables. Ainsi, une forte information mutuelle indique que les variables sont fortement dépendantes, et une faible valeur qu'elles sont plutôt indépendantes.

La MIA utilise ce principe comme distingueur. Lorsque le modèle de consommation est fonction du poids de Hamming, l'information mutuelle entre la série des poids de Hamming de la donnée ciblée et la consommation doit être élevée.

Nous reprenons les notations de la CPA. Rappelons que nous désignons $T[p]$ l'ensemble des points p de toutes les traces, que nous notons T_p pour plus de clarté. L'information mutuelle d'une hypothèse d'octet de clé g pour un point particulier p des traces se calcule comme :

$$\mathrm{I}(H_g,T_p) = \sum_{h=0}^{8}\Pr(H_g=h)\sum_{t\in\mathcal{T}_{g,p,h}}\Pr(T_p=t\mid H_g=h)\cdot\log_2\left(\frac{\Pr(T_p=t\mid H_g=h)}{\Pr(T_p=t)}\right)$$

Ici $\mathcal{T}_{g,p,h}$ correspond à toutes les valeurs de consommation atteintes au point p en considérant uniquement les traces i vérifiant $H_{(g,i)}=h$. De la même manière que pour les attaques précédentes, la plus forte valeur correspond à la valeur de k.

Attaques profilées

Les attaques profilées, dont les plus connues sont certainement les attaques par templates introduites dans [CRR02], sont différentes des attaques précédentes puisqu'elles consistent à dresser le profil de consommation d'un type d'appareil pour chacune des valeurs d'une partie de la clé, puis lors de l'acquisition d'une trace de courant d'un appareil attaqué, de déduire de quel profil, et donc de quelle valeur de clé, cette trace est issue. Pour cela, l'attaquant doit avoir accès à un appareil au comportement très proche de celui/ceux attaqué(s) appelé clone, et avoir le contrôle, ou sinon au moins la connaissance, de la clé pour chiffrer des messages. Elles restent cependant non-invasives puisque l'appareil attaqué n'est pas perturbé.

Grâce au contrôle de la clé, l'attaquant peut fixer une valeur d'un octet de clé, et chiffrer des messages aléatoires. Il obtient ainsi un jeu de traces T^k pour chacune des valeurs k d'un octet de clé. La première partie du profil consiste à calculer la trace moyenne, et la seconde partie consiste à calculer la matrice de covariance du bruit supposé gaussien. Étant donné une trace d'un appareil

dont la clé est inconnue, l'attaquant calcule la probabilité d'une telle trace sous l'hypothèse de chacun des profils. La valeur ayant la probabilité la plus élevée est sélectionnée comme étant la valeur de k. Il est aussi possible d'utiliser plusieurs traces en calculant le produit des probabilités de chacune des traces sous chaque hypothèse de profil.
Remarquons que les très récentes attaques de machine learning rentrent dans cette catégorie d'attaques profilées.

Analyses de fautes

Les attaques par fautes sont des attaques actives et exploitent des perturbations du déroulement de l'algorithme en changeant les données, ou en perturbant le flot d'instructions par exemple. L'une des premières attaques par fautes, la DFA (Differential Fault Analysis) pour analyse différentielle de fautes, a été présentée dans [BS97] et s'applique initialement sur le DES. Elle suppose la capacité de chiffrer deux fois le même message, et d'induire une faute, et notamment le changement d'une valeur d'un octet, au cours du deuxième chiffrement. Le principe est de tirer parti des propriétés des S-boxes. De par la structure particulière des schémas de Feistel, l'induction d'une faute dans les données d'entrée de la fonction du dernier tour permet plusieurs choses : tout d'abord de lire la faute induite δ_{in}, de connaître les entrées de la dernière fonction de tour, et de calculer δ_{out}, la différentielle en sortie de fonction de dernier tour. Or, ce couple de différentielles n'est possible que pour certaines valeurs d'entrées de S-box. Il est donc possible d'éliminer des valeurs de clés qui ne permettent pas à ces différentielles d'exister. Le processus est répété avec d'autres couples de chiffrements non fautés/fautés jusqu'à l'obtention d'une seule ou peu de valeurs possibles. Ces propriétés de différentielles de la S-box seront abordées en détail dans le chapitre 2.

1.3.3 Les attaques combinées

Évoquons aussi la possibilité de réaliser plusieurs types d'attaques simultanément afin d'en combiner les effets. Il est ainsi parfois possible de rendre certaines attaques, a priori impossibles, efficaces. Une attaque PACA (Passive Active Combined Attacks) sur le RSA est introduite dans [AVFM07]. Le principe est d'attaquer une implémentation protégée contre l'analyse simple de courant et les attaques par injection de fautes. La protection contre la SPA est réalisée grâce à un algorithme régulier qui ne permet pas de faire la différence entre les différentes instructions à la simple lecture des traces de courant, et la protection contre l'injection de fautes est réalisée par une vérification après calcul qui décide de retourner ou non le résultat du calcul. Cette attaque cible une instruction d'affectation d'un registre, ce qui a pour effet d'y affecter la valeur 0. La multiplication par une telle valeur étant visible, les motifs de consommations sont différents selon l'instruction exécutée. Ainsi, même si le résultat du calcul fauté n'est pas récupéré par l'attaquant, ce dernier peut tout de même analyser la trace de courant altérée par la faute.

1.4 Les attaques visant le key schedule

Ici nous exposons en détail des attaques permettant de récupérer entièrement la clé d'un AES 128 bits grâce à certaines informations sur la clé étendue. Cette partie est importante puisque nous combinons ces attaques avec des attaques du chapitre 3.

Une attaque visant le key schedule de l'AES est introduite par Mangard dans [Man03] qui fait la remarque que les implémentations masquées protègent essentiellement le déroulement de l'algorithme et pas le key schedule, et qu'il est donc plus facile d'attaquer ce dernier. Cette attaque suppose un attaquant capable de déterminer des poids de Hamming de variables manipulées pendant le key schedule. Dans le cadre d'une attaque par canaux cachés sur la consommation, cela correspond à trouver les points d'intérêt des variables ciblées et à déterminer la valeur de poids de Hamming de chacune grâce à la consommation. L'attaque originale suppose un attaquant ayant suffisamment connaissance de l'algorithme pour trouver les points d'intérêt, et que le modèle de consommation est suffisamment simple pour déduire le poids de Hamming par analyse simple du courant (SPA). Notons que l'attaque ne dépend pas du moyen de calcul des poids de Hamming et s'applique quelle que soit la technique employée.

Nous renvoyons vers la partie sur l'AES pour la définition du key schedule. Rappelons seulement que des dépendances existent entre des octets de clé étendue, et que la connaissance d'une des 11 clés de tour permet de retrouver de manière déterministe les 10 autres. Un attaquant naïf peut par exemple déterminer les poids de Hamming des 176 octets de clé étendue, que nous appellerons l'empreinte mesurée, calculer la clé étendue pour une hypothèse des 16 octets d'une clé de tour répondant aux contraintes induites par les poids de Hamming, puis vérifier que les poids de Hamming de la clé étendue calculée sous cette hypothèse, dite empreinte calculée, soient cohérents avec l'empreinte mesurée. Ce processus peut être répété jusqu'à trouver la bonne hypothèse. D'après ce papier, dans le cas où l'empreinte calculée correspond à l'empreinte mesurée, il est plus que probable que la clé soit la bonne puisque la probabilité que deux clés partagent les mêmes valeurs de poids de Hamming sur leurs 176 octets est très faible, et posent la question de l'existence de telles clés. Plus tard dans [CMW14], les auteurs montreront que ces clés existent mais que leur nombre est faible. Notons tout de même que cette attaque est impossible à mettre en œuvre puisque cela nécessite de faire l'hypothèse de 16 octets avec pour seule contrainte les poids de Hamming de chacun.

1.4.1 Attaque originale

Mangard propose de découper une clé de tour en quatre parties chevauchantes pouvant être attaquées indépendamment. Ces quatre parties sont constituées de cinq octets et ont été choisies afin de rendre un grand nombre d'octets de clé étendue et de valeurs intermédiaires de sorties de S-box calculables. Cinq octets est une limite raisonnable pour en faire toutes les hypothèses. La figure 1.6 récapitule ceci. Elle représente les 5 premières clés de tour $\{K_0, ..., K_4\}$, chacune étant constituée de 16 octets $\{k_{(i,0)}, ..., k_{(i,15)}\}$, et des valeurs intermédiaires de sorties de S-boxes calculées au cours du key schedule. La clé de tour 3 est découpée en 4 parties, les octets $\{k_{(3,0)}, k_{(3,4)}, k_{(3,8)}, k_{(3,12)}, k_{(3,13)}\}$ constituant la partie 0 par exemple. Les octets colorés représentent les octets des clés de tour 0, 1, 2 et 4 et les sorties de S-box calculables pour une hypothèse de la partie 0.

L'attaque consiste, pour chacune des 4 parties, à faire toutes les hypothèses des 5 octets correspondants, puis de vérifier que les valeurs des octets calculés sont bien les mêmes que ceux déterminés par SPA. Si ce n'est pas le cas, l'hypothèse est écartée. Lorsque la liste des candidats possibles a été établie pour chacune des quatre parties, la clé complète est une combinaison de quatre candidats, chacun provenant d'une liste. Dans un premier temps, le chevauchement des parties aide à réduire la liste des combinaisons possibles. Ensuite, pour chaque clé candidate, l'attaquant est capable de calculer tous les octets de la clé étendue, et peut éliminer celles dont l'empreinte calculée ne correspond pas à l'empreinte mesurée. À l'issue de cette dernière étape, il ne reste qu'un faible nombre de clés candidates. Grâce à un couple clair/chiffré, il est possible de réduire ce nombre à l'unique vraie clé.

Le grand intérêt de cette attaque est de montrer qu'il est calculatoirement possible de déterminer la clé en ayant uniquement des données sur des poids de Hamming de la clé étendue (et optionnellement certaines sorties de S-boxes).

1.4.2 Optimisation

Plus tard cette attaque sera améliorée par VanLaven et al. dans [VBC05] où les auteurs proposent une méthode permettant d'augmenter l'efficacité de l'attaque en faisant l'hypothèse des octets de la clé étendue qui maximise les dépendances d'autres octets. Pour trouver ces octets, ils choisissent de représenter de manière astucieuse le key schedule comme étant un hypergraphe des relations entre chacun des octets de la clé étendue. Cet hypergraphe est visible sur la figure 1.7. Chacune des 16 lignes correspond à un indice d'octet et est constitué de 11 valeurs, chacune représentant une clé de tour. La clé maître est représentée par la première valeur de chaque ligne. Notons tout d'abord que les indices ne sont pas représentés dans l'ordre croissant, mais plutôt lus de gauche à droite lorsque l'on représente une clé de tour sous forme de matrice de quatre lignes et autant de colonnes, cela afin de rendre les dépendances plus lisibles. Notons ensuite

FIGURE 1.6 – Les 4 parties et octets de clé de tour et valeurs de `Sbox` calculées grâce à la partie 0

que l'hypergraphe est cyclique, donc que la 17^{me} ligne représentée n'est autre que la première. Les dépendances sont représentées par les triangles de couleur, les gris clairs représentant une relation grâce à de simples XORs, et les triangles foncés représentant l'implication de la S-box dans le calcul. Pour calculer les clés de tour depuis la clé maître, le key schedule utilise l'octet inférieur gauche ainsi que l'octet supérieur d'un triangle pour calculer l'octet inférieur droit. Par exemple, le triangle foncé tout à gauche de la dernière ligne de la figure 1.7 correspond au calcul de l'octet 0 de clé de tour 1 grâce à l'octet 0 de la clé de tour 1 et à l'octet 13 de la clé de tour 0. Les relations entre octets étant connues et inversibles, il est possible de calculer n'importe quelle valeur d'un triangle connaissant les deux autres.

Afin de maximiser le nombre d'octets calculables sous l'hypothèse d'un certain nombre d'octets, les auteurs proposent d'utiliser la propriété suivante : considérons une portion de l'hypergraphe présenté en figure 1.7 formant un triangle équilatéral constitué de plus de 2 valeurs de coté. La connaissance des valeurs d'un coté permet de calculer l'ensemble des valeurs de ce triangle. Ainsi la connaissance de i valeurs bien choisies équivaut à la connaissance de $\frac{i\cdot(i+1)}{2}$ valeurs. Un exemple avec $i = 4$ est représenté sur la figure 1.8. La connaissance des 4 octets noirs permet le calcul des 3 gris foncés, qui, eux, permettent le calcul des 2 plus clairs qui permettent eux-mêmes le calcul de l'octet blanc. Les auteurs ont donc cherché une séquence de 16 octets (16 suffisants à

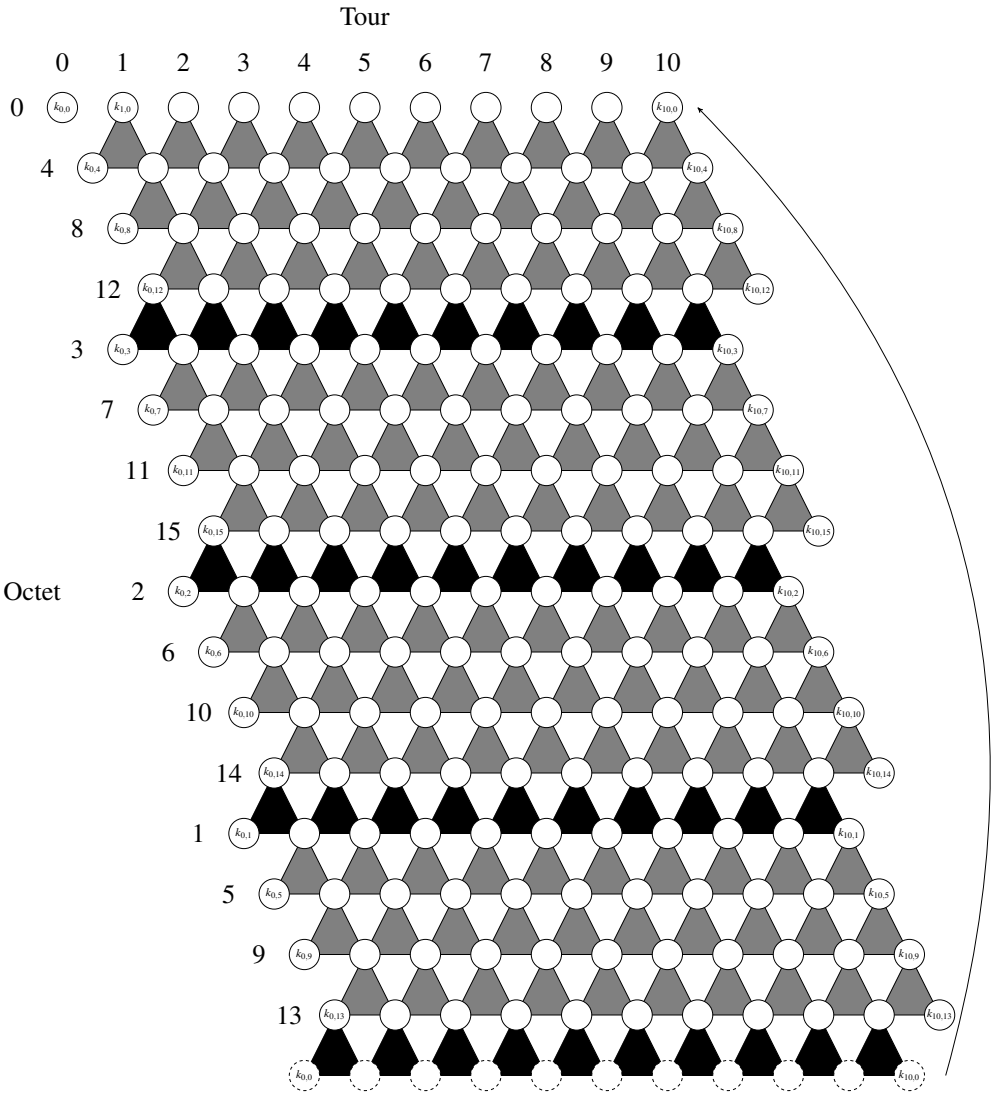


FIGURE 1.7 – Hypergraphe représentant les dépendances du key schedule

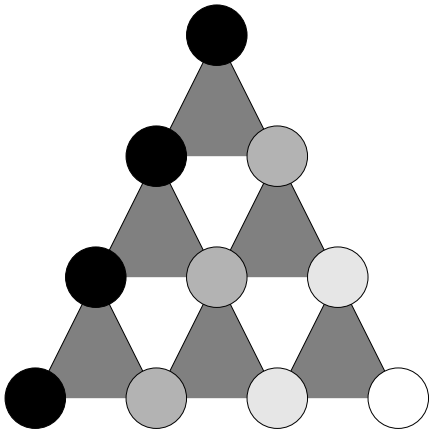


FIGURE 1.8 – Sous-partie d'hypergraphe du key schedule

calculer les 176 octets de la clé étendue) qui, lorsque l'ont fait l'hypothèse progressive de chacun des ses octets, permet de maximiser le nombre d'octets de clé étendue calculables à chaque instant, permettant ainsi d'écarter plus rapidement de mauvaises hypothèses. Ils proposent aussi d'utiliser les fuites provenant des calculs employant la S-box, et donc de maximiser le nombre de valeurs supérieures d'un triangle foncé. Ayant une telle séquence, ils utilisent un algorithme de type guess-compute-backtrack avec pour condition de backtrack la non correspondance entre les poids de Hamming des valeurs calculées et celles mesurées.

Une autre amélioration est aussi apportée vis-à-vis des erreurs possibles faites lors des mesures des poids de Hamming des octets de clé étendue. Jusqu'alors, les attaques présentées ne permettent aucune erreur de mesure, une incohérence menant à l'élimination de la clé candidate. Dans ce papier, les auteurs permettent de telles erreurs en modifiant la condition de backtrack. Pour cela, ils proposent une fonction qui utilise d'une part les carrés des différences entre l'empreinte calculée et l'empreinte mesurée, et d'autre part les carrés des différences entre les octets de l'empreinte mesurée qui ne peuvent pas être calculés sous l'hypothèse courante et leur entier le plus proche. Un seuil permet d'établir si la condition est remplie ou non, et permet de ne garder que les candidats dont l'empreinte calculée est proche de celle mesurée.

1.5 Les contre-mesures

La sécurité est une course perpétuelle entre deux camps, les concepteurs/implémenteurs de sécurité, et les attaquants. Afin de résister aux attaques précédentes, les concepteurs utilisent des contre-mesures, modifications ou ajouts apportés aux algorithmes/appareils ayant pour but de rendre des attaques inefficaces, ou au moins de rendre la tâche plus compliquée aux attaquants. Une contre-mesure est généralement efficace contre un type particulier d'attaque, et il est donc

parfois nécessaire d'en combiner plusieurs si cela est possible. Cependant, elles induisent très souvent un coût logiciel et/ou matériel généralement non négligeable, ce qui peut empêcher de toutes les appliquer. Un compromis doit souvent être choisi entre performance/coût et sécurité.
Un type de contre-mesure peut être implémenté par des moyens logiciels, des moyens matériels, ou par les deux. Les contre-mesures logicielles modifient directement l'algorithme et sont implémentées dans le code. Les contre-mesures matérielles sont indépendantes de ce dernier mais peuvent modifier son exécution. Elles ont toutes principalement pour but de :

— Réduire/bruiter la fuite d'information

— Désynchroniser les traces

— Randomiser les données manipulées

Pour illustrer nos propos, nous présentons quelques exemples de contre-mesures.

Réduire ou bruiter la fuite

L'une des premières contre-mesures logicielles, et qui est maintenant appliquée systématiquement à la très vaste majorité des algorithmes, est l'exécution en temps constant. En assurant que les instructions utilisées soient identiques quelle que soit la clé et que seules les données manipulées changent, l'algorithme se retrouve naturellement protégé des attaques temporelles puisque la durée d'exécution est identique d'une clé à l'autre.
Une autre manière d'empêcher les fuites, mais de matérielle cette fois, peut être de créer artificiellement de la consommation. De cette manière, la fuite est moins corrélée aux données manipulées.
Une contre-mesure rendant les attaques par fautes inefficaces consiste à effectuer le chiffrement deux fois, ce qui permet de détecter une faute dans l'une ou l'autre des exécutions si les résultats ne sont pas identiques, et de ne pas livrer le chiffré le cas échéant. L'attaquant n'a donc pas accès au chiffré fauté souvent essentiel dans une attaque par fautes.
Dans le but de prévenir des attaques par observations, il est par exemple possible d'induire des consommations non liées au calcul, ce qui a pour effet de bruiter les consommations et de les rendre moins facilement exploitables.

1.5.1 La désynchronisation

La synchronisation est une hypothèse courante nécessaire aux attaques par canaux cachés. Un moyen de les complexifier est de désynchroniser les exécutions. Une première méthode, le "shuffling", consiste à changer l'ordre d'exécution de certaines instructions de manière aléatoire. Cela est particulièrement possible lors de boucles dont chaque itération est indépendante. Par exemple, le `SubBytes` de l'AES s'applique de manière indépendante aux 16 octets de l'état, ce qui permet de l'appliquer dans n'importe quel ordre. Une deuxième méthode consiste à inclure des délais aléatoires au cours de l'exécution, souvent générés grâce à l'exécution d'instructions inutiles. Ces deux méthodes ont pour résultat de faire en sorte qu'un instant temporel ne soit

plus lié à la manipulation de la même donnée de manière systématique, ce qui bruite les calculs statistiques.

1.5.2 La randomisation

La randomisation est parmi les contre-mesures les plus utilisées de nos jours et consiste à rendre les données réellement manipulées différentes de celles attendues, notamment par l'attaquant, le masquage étant sa réalisation la plus commune. Dans sa forme la plus simple, le masquage consiste à ajouter une valeur aléatoire à chaque exécution aux données d'entrées, d'adapter l'algorithme en ce sens, puis de la retirer aux données de sorties. Les attaques statistiques qui nécessitent le calcul de données intermédiaires sont rendues inefficaces puisque les valeurs manipulées ne sont pas celles calculées, même dans le cas de la bonne hypothèse de clé, et les fuites n'y sont donc pas associées. Le masquage étant important pour la compréhension d'attaques présentées dans le chapitre 3, nous en donnons ici plus de détails.

1.5.3 Le masquage logiciel

Le masquage [GP99, CJRR99, CG00, Mes00a, PR13] permet la randomisation des données. Il en existe plusieurs variantes qui diffèrent principalement par le fait qu'elles soient orientées logicielles ou matérielles. Le masquage matériel, et plus particulièrement celui connu sous le nom d'implémentation threshold sera exposé en détail dans le chapitre 2. Nous présentons ici un masquage logiciel : le masquage booléen. Celui-ci consiste à partager une donnée x en $d+1$ *shares* $(x_i)_{1\leq i\leq d+1}$ grâce au tirage de d valeurs aléatoires $(r_i)_{1\leq i\leq d}$, ce qui consiste en un masquage d'ordre d, ou une implémentation d'ordre $d+1$. Pour cela, un moyen consiste à choisir :

$$\begin{aligned} x_i &= r_i && 1 \leq i \leq d \\ x_{d+1} &= x \oplus \bigoplus_{i=1}^{d} r_i \end{aligned}$$

La sécurité apportée par cette méthode tient en l'incapacité d'un attaquant à reconstruire la donnée non masquée même si celui-ci a connaissance de d shares. Plus particulièrement, l'uniformité du tirage des masques assure que chaque valeur de x est équiprobable quelle que soit la distribution des valeurs de d shares parmi les $d+1$.

Certaines instructions permettent d'être appliquées directement à la donnée masquée, qui par convention fait référence à x_{d+1}, puis d'être démasquées par la suite. Le XOR en est un exemple trivial. Toute série de XOR appliquée à la donnée non masquée x peut être appliquée de manière équivalente à la donnée masquée x_{d+1}, cette dernière nécessitant seulement d'être démasquée

pour donner le même résultat :

$$x \oplus c_1 \oplus ... \oplus c_s = (x_{d+1} \oplus c_1 \oplus ... \oplus c_s) \oplus \bigoplus_{i=0}^{d} r_i$$

Pour d'autres instructions, cela est impossible. C'est par exemple le cas de la fonction de la S-box de l'AES.

Nous proposons ici une explication simple d'un masquage d'ordre 1 de l'AES, c'est-à-dire que toute donnée interne x soit partagée en une donnée masquée x' et un masque r tels que $x' = x \oplus r$. Le but de ce masquage est de résister aux attaques classiques telles que la CPA qui nécessitent le calcul d'une variable intermédiaire selon une hypothèse de clé. Si cette valeur intermédiaire manipulée par l'algorithme est différente, sous-entendue masquée, de celle calculée, alors l'attaque ne fonctionne plus. Plus particulièrement, pour que l'attaque soit réellement inefficace, le masque r doit être inconnu de l'attaquant, et les distributions de x' ainsi que de r doivent être uniformes. C'est le cas si r est tiré aléatoirement de manière uniforme. Les fonctions `AddRoundKey`, `ShiftRows` et `MixColumns` de l'AES sont des fonctions linéaires, ce qui permet de ne les appliquer qu'à une seule share. Soit l'état S :

$$\begin{aligned} \mathrm{ARK}(S \oplus r) &= \mathrm{ARK}(S) \oplus r \\ \mathrm{SR}(S \oplus r) &= \mathrm{SR}(S) \oplus r \\ \mathrm{MC}(S \oplus r) &= \mathrm{MC}(S) \oplus r \end{aligned}$$

Un AES uniquement constitué de ces fonctions protégé par un masquage d'ordre 1 consisterait à tirer un masque aléatoire r secret, à calculer les 16 octets de message masqué $M' = (m'_i)_{1 \leq i \leq 16}$ correspondants à $m'_i = m_i \oplus r$, à appliquer normalement l'AES à M' résultant en le chiffré $C' = (c'_i)_{1 \leq i \leq 16} = AES(M')$, puis à retourner $C = (c_i = c'_i \oplus r)_{1 \leq i \leq 16}$. Notons qu'une attention particulière doit être apportée au niveau du `MixColumns`. Celui-ci consiste en les XOR de plusieurs résultats intermédiaires de multiplications polynomiales qui, effectués dans un certain ordre, peuvent compromettre la sécurité du masquage. Soient les quatre octets d'état $\{s_1, s_2, s_3, s_4\}$ d'une colonne manipulée par le `MixColumns`. Le calcul d'un octet de la colonne résultat s'exprime de la manière suivante :

$$\bigoplus_{i=1}^{4} a_i \cdot s_i$$

où $a_i \in \{1,2,3\}$ des coefficients. Dans le cadre d'une implémentation masquée par r, chaque terme $a_i \cdot s_i$ devient $a_i \cdot s'_i = a_i \cdot s_i \oplus a_i \cdot r$. Par exemple, le XOR intermédiaire des deux termes dont les coefficients sont 1 démasque puisque $s'_i \oplus s'_j = s_i \oplus r \oplus s_j \oplus r = s_i \oplus s_j$. De la même manière, le calcul intermédiaire du XOR des coefficients 1, 2 et 3 revient à démasquer. Il est donc important de xorer successivement et dans l'ordre les octets dont les coefficients sont 1, 2, 1 et 3, ce qui

revient à être masqué par $3 \cdot r$, $2 \cdot r$ et enfin r.

La S-box n'étant pas linéaire, il est nécessaire de procéder autrement puisque $\mathrm{SB}(S \oplus r) \neq \mathrm{SB}(S) \oplus r$. Dans le cadre d'une implémentation logicielle, la S-box correspond à une lecture en table. La S-box masquée est calculée lors du tirage du masque et sera utilisée en lieu et place de l'originale. Cette S-box masquée SB' est construite telle que :

$$\mathrm{SB}'(x) = \mathrm{SB}(x \oplus r) \oplus r$$

De cette manière, $\mathrm{SB}'(x') = \mathrm{SB}(x \oplus r \oplus r) \oplus r = \mathrm{SB}(x) \oplus r$.

Cette proposition n'est évidemment pas l'unique moyen de protéger un AES, mais présente la version la plus simple. Dans la suite de ce document, nous rencontrerons des variantes où le masque n'est pas le même aux différents points de l'algorithme, différent d'un indice de l'état à l'autre, ou encore différent d'un tour à l'autre. Les mécaniques restent cependant globalement les mêmes. Ces schémas particuliers seront exposés lors de l'explication des attaques correspondantes.

1.6 Les attaques visant le key schedule à l'ordre 2

1.6.1 Cas masqué

Une attaque visant le key schedule dans le cadre d'implémentations masquées est présentée dans [CMW14]. Les auteurs étudient deux schémas de masquages booléens de la clé étendue. Le premier consiste à tirer 11 masques, un masque par clé de tour, et à protéger les 16 octets d'une telle clé grâce à ce masque. Le second schéma consiste cette fois à tirer 16 masques, un par indice dans l'état qui sera protégé par le même masque tout au long du chiffrement.
Afin d'attaquer de tels schémas, les auteurs remarquent qu'il est possible d'établir des contraintes dès lors que l'on considère deux valeurs protégées par le même masque. Soient $x' = x \oplus r$ et $y' = y \oplus r$, les propriétés suivantes sont vérifiées :

$$\begin{aligned} &|\mathrm{HW}(x') - \mathrm{HW}(y')| \leq \mathrm{HD}(x,y) \leq \min(8, \mathrm{HW}(x') + \mathrm{HW}(y') \\ &\mathrm{HD}(x,y) \equiv \mathrm{HW}(x') + \mathrm{HW}(y') \pmod 2 \end{aligned}$$

où $\mathrm{HD}(x,y) = \mathrm{HW}(x \oplus y)$ est la distance de Hamming entre x et y. Des contraintes s'appliquent donc aux valeurs non masquées x et y par l'observation des poids de Hamming du couple des valeurs masquées par le même masque r.
L'attaque proposée dans ce papier comporte deux phases. La première consiste à récupérer de telles contraintes entre octets de clé étendue grâce à des mesures sur plusieurs exécutions des

valeurs masquées, et la seconde à appliquer une méthode de guess-compute-backtrack utilisant les contraintes précédemment obtenues.

11 masques

La première phase permet d'obtenir des distances de Hamming de couples d'octets d'une même clé de tour. Une fois les distances des couples de deux clés successives $K_i = \left(k_{(i,j)}\right)_{1\leq j\leq 16}$ et $K_{(i+1)} = \left(k_{(i,j)}\right)_{1\leq j\leq 16}$ déterminées, ils réalisent une exploration sous l'hypothèse du couple de masques. Cette exploration consiste à faire successivement l'hypothèse de chacun des octets non masqués de K_i, et à chaque fois de calculer les octets résultants de la clé suivante K_{i+1}. À chaque étape, les contraintes liées aux poids de Hamming mesurés des valeurs masquées ainsi que celles liées aux distances de Hamming au sein de chacune des clés de tour sont vérifiées, une incohérence menant à un backtrack. Les auteurs proposent de commencer par faire l'hypothèse des octets $k_{(i,12)}$ et $k_{(i,3)}$, ce qui rend possible le calcul de $k_{i+1,3}$, puis de faire l'hypothèse et de calculer les octets possibles de bas en haut puis de gauche à droite dans la représentation matricielle quatre par quatre.

16 masques

La première phase permet d'obtenir des distances de Hamming de couples d'octets de même indice. Cette fois, les auteurs utilisent un couple d'indices plutôt qu'un couple de clés de tour. Étant donné un indice $a \in \{0,...,3\}$, ils définissent le second comme étant $b = 12 + ((a+1) \bmod 4)$. Ces deux indices d'octets sont impliqués dans le calcul de l'octet de position a au tour suivant. De la même manière que précédemment, pour chaque couple de masques associé aux indices a et b, une exploration est effectuée. Une fois l'hypothèse faite de la valeur $k_{(0,a)}$, l'exploration consiste à faire l'hypothèse de la valeur $k_{(i,b)}$ puis de calculer $k_{(i+1,a)}$ pour tous les $i \in \{0,...,9\}$ en testant à chaque instant toutes les contraintes qui s'exercent sur les distances de Hamming des couples de même indice. Ensuite, ils proposent de faire l'hypothèse de $k_{(10,b)}$, ce qui permet le calcul des octets $k_{(i,c)}$ pour tout $i \in \{10,...,1\}$ et $c = b-4$, avec pour unique nouvelle hypothèse le masque de l'indice c. Le processus recommence avec l'hypothèse de l'octet $k_{0,c}$ qui permet de calculer les octets d'un nouvel indice. Les auteurs remarquent aussi que l'hypothèse du masque de l'indice a peut être évitée.

1.6.2 Conclusion

Nous avons vu qu'il existe des moyens de déterminer la clé grâce à des contraintes sur les poids et/ou distances de Hamming entre des octets de la clé étendue, et ce même dans des cas masqués. Nous utilisons de telles attaques lorsque nous nous posons la question de l'exploitation de l'information apportée par certaines attaques dans le chapitre 3.

1.7 Attaques d'ordre supérieur

Suite à la présentation des contre-mesures, et notamment celle qui consiste à masquer les implémentations, nous présentons ici rapidement le principe d'attaques d'ordre supérieur qui fonctionnent malgré de telles protections.
Une implémentation protégée à l'ordre 1 résiste aux attaques présentées plus haut qui sont dites, elles aussi, d'ordre 1 puisqu'elles n'exploitent la fuite qu'à un seul point temporel dans le cadre multivarié. La raison pour laquelle ces attaques ne fonctionnent plus provient du fait que l'attaquant n'est plus en mesure de prédire une donnée intermédiaire puisque le masque est tiré de manière uniforme, et que l'information d'un seul instant, d'une seule share est insuffisante. Cependant, ce raisonnement ne tient que si le masque est inconnu. Les attaques d'ordres plus élevés utilisent plusieurs instants temporels. Si une fuite du masque existe effectivement dans la trace, en plus de celle vis-à-vis de la donnée masquée, l'étude jointe de ces fuites peut éventuellement permettre d'appliquer des contraintes sur la donnée non masquée, et donc de monter une attaque. Présentée dans [Mes00b], la DPA second ordre combine les instants de manipulation de la donnée masquée et du masque en utilisant la valeur absolue de la différence des deux consommations. Il s'avère que cette combinaison de fuites est corrélée à la donnée non masquée. Une attaque DPA classique où le calcul ne porte plus sur la consommation de l'instant de fuite, mais plutôt sur la valeur absolue de la différence des consommations aux instants de fuite de la donnée masquée et du masque, permet de déterminer la clé. Cette attaque qui combine deux instants est dite d'ordre deux. Il existe plusieurs moyens de combiner ces instants comme l'explique [PRB09]. Il faut toutefois noter que la corrélation issue de la combinaison de plusieurs instants est naturellement moindre que celle d'un seul instant non protégé. De plus, le bruit joue un rôle d'autant plus important que le nombre de points combinés augmente comme décrit dans [CJRR99]. Ces attaques sont donc moins efficaces et ne rendent pas les contre-mesures complètement obsolètes puisqu'elles augmentent les difficultés opposées à l'attaquant. Dans le cas où ces instants ne sont pas connus, il est possible de mener l'attaque sur toutes les combinaisons de points possibles, ce qui augmente d'autant plus la difficulté de l'attaque.

Dans le cas univarié, la combinaison est alors souvent faite intrinsèquement puisque la consommation en un instant fuit vis-à-vis de plusieurs instructions, et si la manipulation du masque et de la donnée masquée sont proches, alors leur fuite est combinée. La consommation est donc la somme, entre autres, des consommations liées aux manipulations de la donnée masquée et du masque. Un moyen de tirer de l'information de telles fuites est l'exploitation des différents moments statistiques. La DPA utilise le moment statistique d'ordre 1, la moyenne. Soient les distributions des valeurs de nombreuses consommations (bruitées) associées à deux valeurs de poids de Hamming différentes (dans le cadre d'une consommation fonction du poids de Hamming). Comme l'explique par exemple [MM17], les moyennes de ces deux distributions sont différentes. En présence d'une implémentation masquée à l'ordre 1, les distributions de la somme

des consommations de la valeur masquée et du masque présentent, cette fois ci, la même moyenne. Cependant, elles ne présentent pas la même variance, moment statistique d'ordre 2.

Les attaques d'ordre supérieur permettent donc de déterminer la clé grâce à la combinaison (ou à l'utilisation de moments statistiques supérieurs) des fuites de chacune des shares de la donnée ciblée. Ces attaques ne fonctionnent que si ces shares fuient effectivement, et sont très sensibles au bruit. Ce principe étant général, il est applicable aux attaques que nous avons présentées telles que la DPA, CPA et MIA.

Chapitre 2

Étude des S-boxes

Le chapitre précédent présente volontairement des lacunes au niveau de la cryptanalyse dite classique. Dans cette partie nous nous y intéressons plus en détail puisque le dernier chapitre porte sur la construction de S-box, composant essentiel à certains algorithmes de chiffrement et qui permet justement d'être résistant à la cryptanalyse. Nous parlerons notamment de trois types de cryptanalyses : l'algébrique, la différentielle et la linéaire. Chacune exploitant une caractéristique bien précise de la S-box, nous ne manquons pas de les introduire. Fait intéressant, il est possible de classifier les S-box de manière à ce que ces caractéristiques soient les mêmes pour chaque élément d'une même classe. Nous présentons ici cette classification qui sera utile pour la recherche de S-box 4 bits du chapitre 4. Nous présentons aussi un masquage particulièrement orienté matériel, l'implémentation threshold ou TI, permettant notamment de se protéger en plus des attaques rendues possibles par les glitchs.

2.1 Introduction

La cryptanalyse classique est très différente de celle par canaux cachés. Elle ne prend pas en compte l'implémentation, mais s'attaque à la fonction mathématique elle-même. De par la complexité de ce dernier, des équations comprenant les éléments du clair, du chiffré et de la clé sont, elles aussi, complexes et du même coup difficiles à résoudre. La cryptanalyse classique consiste généralement à établir des liens plus simples, des biais par exemple, entre les clairs et les chiffrés afin de réduire la complexité de ces équations. Dans certains cas, des algorithmes réduits à un certain nombre de tours sont attaqués. Les S-boxes jouent un rôle important contre ces analyses. Ici nous détaillons trois de leurs caractéristiques essentielles, que nous appellerons critères, ainsi que les types de cryptanalyses exploitant des critères de faible qualité. Ces critères sont le degré algébrique exploité par la cryptanalyse algébrique, l'uniformité différentielle exploitée par la cryptanalyse différentielle et enfin la linéarité exploitée par la cryptanalyse linéaire.

Nous commençons donc par présenter ce qu'est réellement une S-box. Nous renvoyons notamment vers les travaux [Can16, Blo11] pour le lecteur intéressé pour plus de précisions sur les sections suivantes.

2.2 Définition de la S-box

Une S-box SB est une fonction de n vers m bits (de $\mathbb{F}_2^n$ vers $\mathbb{F}_2^m$). Lorsque $m = n$, nous parlerons de S-box n bits. Cette fonction peut être représentée de manière simple grâce à une table ordonnée ayant 2^n valeurs correspondant chacune à $\mathrm{SB}(x)$ avec $x = \{0, 1, ..., 2^n - 1\}$. Les implémentations logicielles stockent très souvent les S-box sous forme de table.

Une autre manière de représenter une S-box est sous sa forme ANF (Algebraic Normal Form). Pour cela il convient tout d'abord de la décomposer en un vecteur de m fonctions booléennes à n variables $\mathrm{SB}(x) = \{\mathrm{SB}_1(x), ..., \mathrm{SB}_m(x)\}$ avec $x = \{x_1, ..., x_n\}$ un vecteur binaire dont x_1 est le bit de poids faible, et chaque SB_i une fonction de $\mathbb{F}_2^n$ vers $\mathbb{F}_2$. Chacune de ces fonctions est appelée composante et décrit le comportement du bit d'indice lui correspondant. Ainsi $\mathrm{SB}_i(x) = \mathrm{SB}(x)[i]$ avec $x \in \{0, ..., 2^n - 1\}$ et $\mathrm{SB}(x)[i]$ le bit $1 \leq i \leq m$ de $\mathrm{SB}(x)$. La forme ANF consiste à exprimer chacune de ses composantes sous forme du xor de plusieurs monômes :

$$\mathrm{SB}_i(x_1, ..., x_n) = \bigoplus_{u \in \{0,...,2^n-1\}} \alpha_u x_1^{u_1} x_2^{u_2} ... x_n^{u_n}$$

avec $u = \{u_1, ..., u_n\}$ un vecteur binaire et $\alpha_u \in \{0, 1\}$. Il est possible de construire l'ANF à partir des tables de chacune des composantes grâce à la transformée de Möbius. Cette dernière permet

de calculer les coefficients α_u pour chacun des SB_i et s'exprime comme ceci :

$$\alpha_u = \bigoplus_{v \preceq u} SB_i(v)$$

avec $v \preceq u$ signifiant $v_i \leq u_i \; \forall i \in \{1,...,n\}$. Plus simplement, pour calculer le coefficient α_u, il suffit de considérer la sous-table issue de SB_i ne contenant que les antécédents v tels que $v \preceq u$, puis de compter le nombre d'images à 1. Si ce nombre est impair, alors $\alpha_u = 1$, sinon $\alpha_u = 0$. Deux exemples, $\alpha_0 = SB_i(0)$, $\alpha_{2^n-1} = \bigoplus_{v \in \{0,...,2^n-1\}} SB_i(v)$.

Voici un exemple complet. Pour la simplicité des exemples, nous considérons une S-box 3 bits SB telle que présentée à la table 2.1. Dans la suite de cette partie, les nombres à trois chiffres sont des représentations binaires. Nous aurions tout aussi bien pu nous inspirer d'un cas moins artificiel avec la S-box 4 bits de [Pha02] par exemple et arriver aux mêmes conclusions.

x	000	001	010	011	100	101	110	111
$SB(x)$	0	3	6	2	4	1	7	5
$SB_1(x)$	0	1	0	0	0	1	1	1
$SB_2(x)$	0	1	1	1	0	0	1	0
$SB_3(x)$	0	0	1	0	1	0	1	1

TABLE 2.1 – S-box 3 bits

Nous essayons d'exprimer $SB_1(x) = \alpha_0 \oplus \alpha_1 x_1 \oplus \alpha_2 x_2 \oplus \alpha_3 x_1 x_2 \oplus ... \oplus \alpha_7 x_1 x_2 x_3$. Rappelons que $x = \{x_1, x_2, x_3\}$ avec x_1 le bit de poids faible. En premier lieu, $\alpha_0 = SB_1(0) = 0$ puisque $\{x | x \preceq 000\} = \{000\}$. Calculons à présent α_5 : $\{x | x \preceq 101\} = \{000, 001, 100, 101\}$. La table qui en résulte est celle présentée en 2.2. On peut y voir que le nombre de 1 est pair, ainsi $\alpha_5 = 0$.

x	000	001	100	101
$SB_1(x)$	0	1	0	1

TABLE 2.2 – Sous S-box pour le calcul de α_5

Au contraire, la table correspondante à α_6 contenant les antécédents $\{x | x \preceq 110\} = \{000, 010, 100, 110\}$ est celle présentée en 2.3. Cette fois le nombre de 1 est impair, impliquant que $\alpha_6 = 1$.

x	000	010	100	110
$SB_1(x)$	0	0	0	1

TABLE 2.3 – Sous S-box pour le calcul de α_6

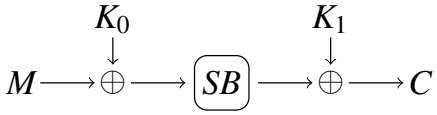


FIGURE 2.1 – Chiffrement 1 tour

Voici les formules complètes des composantes de SB, formant son ANF :

$$\begin{aligned}
\mathrm{SB}_1(x_1,x_2,x_3) &= x_1 \oplus x_1x_2 \oplus x_2x_3 \\
\mathrm{SB}_2(x_1,x_2,x_3) &= x_1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_3 \\
\mathrm{SB}_3(x_1,x_2,x_3) &= x_2 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3
\end{aligned}$$

2.2.1 Degré

L'ANF permet de déduire le degré $deg(\mathrm{SB})$ de la S-box puisque ce dernier est le degré maximum de ses composantes, à savoir le nombre de termes formant le monôme le plus long. Dans l'exemple précédent, le plus long monôme de toutes les composantes réunies est un monôme de deux termes, donc de degré 2, ce qui fait que la S-box est elle-même de degré 2. Notons que le degré maximum d'une permutation n bits est $n-1$.

2.2.2 Cryptanalyse algébrique

La cryptanalyse algébrique cherche à résoudre un système d'équations impliquant la clé. Nous verrons que la complexité des équations est liée au degré de la S-box utilisée. Pour expliquer par un exemple en quoi consiste ce type d'attaques, nous reprenons un exemple similaire à celui présenté dans [Can16]. Considérons un algorithme à un seul tour opérant sur 3 bits et utilisant deux clés de tours d'autant de bits, et représenté en figure 2.1. Il consiste simplement en le xor entre l'état et la première clé de tour, suivi de l'application d'une S-box, et enfin du xor avec la seconde clé de tour. Nous utilisons la S-box présentée en table 2.1.

Le message M est constitué des trois bits $\{m_1,m_2,m_3\}$, le chiffré C de $\{c_1,c_2,c_3\}$ et les deux clés de tours K_0 et K_1 de $\{k_{0,1},k_{0,2},k_{0,3}\}$ et $\{k_{1,1},k_{1,2},k_{1,3}\}$ respectivement, avec l'indice 1 correspondant au bit de poids faible. Voici l'équation résultant de ce chiffrement :

$$C \oplus K_1 = \mathrm{SB}(M \oplus K_0)$$

En utilisant les équations issues de la représentation ANF de la S-box, cela donne :

$$\begin{aligned}
c_1 \oplus k_{1,1} &= (m_1 \oplus k_{0,1}) \oplus (m_1 \oplus k_{0,1})(m_2 \oplus k_{0,2}) \oplus (m_2 \oplus k_{0,2})(m_3 \oplus k_{0,3}) \\
c_2 \oplus k_{1,2} &= (m_1 \oplus k_{0,1}) \oplus (m_2 \oplus k_{0,2}) \oplus (m_1 \oplus k_{0,1})(m_2 \oplus k_{0,2}) \oplus (m_1 \oplus k_{0,1})(m_3 \oplus k_{0,3}) \\
c_3 \oplus k_{1,3} &= (m_2 \oplus k_{0,2}) \oplus (m_1 \oplus k_{0,1})(m_2 \oplus k_{0,2}) \oplus (m_3 \oplus k_{0,3}) \oplus (m_1 \oplus k_{0,1})(m_3 \oplus k_{0,3}) \\
&\quad \oplus (m_2 \oplus k_{0,2})(m_3 \oplus k_{0,3})
\end{aligned}$$

Il est ensuite possible d'isoler les parties uniquement liées au message :

$$c_1 \oplus k_{1,1} = \mathrm{SB}_1(M) \oplus k_{0,1}(1 \oplus m_2) \oplus k_{0,2}(m_1 \oplus m_3) \oplus k_{0,3}m_2 \oplus k_{0,1}k_{0,2} \oplus k_{0,2}k_{0,3}$$
$$c_2 \oplus k_{1,2} = \mathrm{SB}_2(M) \oplus k_{0,1}(1 \oplus m_2 \oplus m_3) \oplus k_{0,2}(1 \oplus m_1) \oplus k_{0,3}m_1 \oplus k_{0,1}k_{0,2} \oplus k_{0,1}k_{0,3}$$
$$c_3 \oplus k_{1,3} = \mathrm{SB}_3(M) \oplus k_{0,2}(1 \oplus m_1 \oplus m_3) \oplus k_{0,3}(1 \oplus m_1 \oplus m_2) \oplus k_{0,1}\kappa_{0,2} \oplus k_{0,1}k_{0,3} \oplus k_{0,2}k_{0,3}$$

Chaque couple (M,C) donne une solution à ces trois équations. La linéarisation d'un tel système amène à considérer chaque monôme de clé comme une inconnue afin de le résoudre plus facilement. Ainsi il n'y a pas 6 inconnues $\{k_{0,1},k_{0,2},k_{0,3},k_{1,1},k_{1,2},k_{1,3}\}$, mais 9 puisque s'ajoutent les inconnues $\{k_{2,1} = k_{0,1}k_{0,2}, k_{2,2} = k_{0,1}k_{0,3}, k_{2,3} = k_{0,2}k_{0,3}\}$. On voit bien que le nombre de nouvelles inconnues est lié au degré de la S-box. En effet, si elle n'avait été que de degré 1, alors il n'y aurait pas eu besoin de considérer des inconnues issues de monômes de degré 2. Si elle avait été de degré 3, il aurait aussi fallu considérer le monôme de degré 3. Le nombre d'inconnues *Nbr* issues des équations d'une S-box n bits dans le cadre d'un chiffrement tel que l'exemple précédent se calcule comme $Nbr = \sum_{i=1}^{n} \binom{n}{i}$. Il est donc très important que la S-box présente le degré le plus haut possible pour se prémunir de ce type d'attaques.

2.3 Cryptanalyse différentielle

2.3.1 Uniformité différentielle

L'uniformité différentielle d'une S-box est liée à la probabilité maximum des différentielles de sortie en fonction des différentielles d'entrées. Tout d'abord qu'est-ce qu'une différentielle d'entrée/sortie ? Une différentielle d'entrée est le xor de deux valeurs données en entrée de S-box, et la différentielle de sortie est le xor des valeurs de sorties correspondantes. Étant donné la S-box de la table 2.1 et les messages 000 et 001, la différentielle d'entrée est $\Delta_{in} = 000 \oplus 001 = 1$, et la différentielle de sortie est $\Delta_{out} = \mathrm{SB}(000) \oplus \mathrm{SB}(001) = 0 \oplus 3 = 3$.

On définit

$$\delta(a,b) = \#\{x \mid \mathrm{SB}(x \oplus a) \oplus \mathrm{SB}(x) = b\}$$

le nombre de couples de valeurs d'entrée réalisant $\Delta_{in} = a$ et $\Delta_{out} = b$. La DDT (Difference Distribution Table) est l'ensemble des $\delta(a,b)$ avec $a \in \mathbb{F}_2^n$ et $b \in \mathbb{F}_2^m$ liés à cette S-box. Si chaque ligne est associée à une différentielle d'entrée, et chaque colonne à une de sortie, la somme des éléments d'une ligne est égale à 2^n. De plus, la première ligne ne contient qu'un seul élément non nul, $\delta(0,0) = 2^n$ puisqu'à entrées égales les sorties sont, elles aussi, égales. L'uniformité différentielle δ_{SB} d'une S-box est le maximum de cette table pour toute différentielle d'entrée non nulle :

$$\delta_{\mathrm{SB}} = \max_{a \neq 0, b}(\delta(a,b))$$

La DDT correspondant à la S-box 2.1 est présentée en table 2.4.

$a\backslash b$	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	0	2	2	2	2	0	0
2	0	2	0	2	2	0	2	0
3	0	2	2	0	0	2	2	0
4	0	2	2	0	2	0	0	2
5	0	2	0	2	0	2	0	2
6	0	0	2	2	0	0	2	2
7	0	0	0	0	2	2	2	2

TABLE 2.4 – DDT S-box 3 bits

Grâce à cette table, nous pouvons en déduire que l'uniformité différentielle est $\delta_{SB} = 2$. Cette valeur est très importante vis-à-vis de la cryptanalyse différentielle.

2.3.2 Principe et exemple de cryptanalyse différentielle

La cryptanalyse différentielle est attribuée à Biham et Shamir dans [BS91] qui l'appliquent d'abord au DES et autres schémas de Feistel. Nous présentons ici le concept afin de montrer en quoi l'uniformité différentielle d'une S-box doit être faible afin de se protéger de ce type d'attaques.

Supposons un schéma de Feistel simple à un tour tel que présenté en figure 2.2. Le message M est séparé en 2 parties égales M_d et M_g les parties droite et gauche respectivement. Dans notre exemple, nous supposons que le message est constitué de 6 bits. Le chiffré C est la concaténation de deux parties que sont $C_g = M_g \oplus X$ et $C_d = M_d$. Nous omettons volontairement la permutation des deux parties de l'état sensée être faite après chaque tour de Feistel pour rendre l'exemple plus compréhensible. La fonction f consiste en le xor avec la clé 3 bits $k = \{k_1, k_2, k_3\}$, puis l'application d'une S-box. Nous considérons toujours que cette S-box est celle de la table 2.1. Étant donné un couple de paires message/chiffré (M, C) et (M^*, C^*), il est possible d'écarter de mauvaises valeurs de clés. En voici le principe : M et M^* étant connus, la différentielle d'entrée Δ_x de la fonction de tour est, elle aussi, connue et correspond à $M_d \oplus M_d^*$. Il est possible de calculer Δ_X comme étant $\Delta_X = M_g \oplus M_g^* \oplus C_g \oplus C_g^*$. D'après ces informations, et comme le xor de la clé ne modifie pas les différentielles, il est possible de restreindre les couples d'entrée possibles de S-box (v, v^*) à ceux respectant les propriété suivante :

$$v \oplus v^* = \Delta_x$$
$$\mathrm{SB}(v) \oplus \mathrm{SB}(v^*) = \Delta_X$$

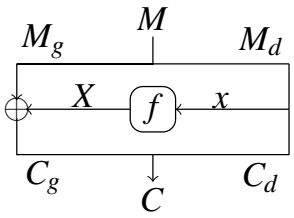


FIGURE 2.2 – Feistel 1 tour

Ce nombre de couples est en fait la valeur de $\delta(\Delta_x, \Delta_X)$ de la DDT de la S-box et nous les notons $(v_i, v_i^*)_{1 \leq i \leq \delta_{(\Delta_x, \Delta_X)}}$. Une fois ces couples identifiés, il est possible de calculer les valeurs possibles de la clé comme étant :

$$\{g_i \mid x \oplus g_i = v_i, x^* \oplus g_i = v_i^*\} \quad \forall i \in \{1, ..., \delta(\Delta_x, \Delta_X)\}$$

L'utilisation d'un autre couple de paires de message/chiffré permet d'avoir d'autres valeurs possibles de la clé. Il est alors possible de faire l'intersection entre les valeurs possibles pour chacun des couples afin d'écarter de nouvelles valeurs de clé jusqu'à obtenir la bonne.

Voici un exemple détaillé. Supposons les valeurs $M = 010\ 101$, $M^* = 011\ 100$, $C = 101\ 101$ et $C^* = 110\ 100$. $\Delta_x = 101 \oplus 100 = 001$ et $\Delta_X = 010 \oplus 011 \oplus 101 \oplus 110 = 010$. Cherchons les couples d'entrée de S-box possibles pour $\Delta_x = 1$ et $\Delta_X = 2$:

$$v = 000 \ \ v^* = 001 \quad \mathrm{SB}(000) = 000 \ \ \mathrm{SB}(001) = 011 \rightarrow \Delta_X = 011 \tag{2.1}$$
$$v = 001 \ \ v^* = 000 \quad \mathrm{SB}(001) = 011 \ \ \mathrm{SB}(000) = 000 \rightarrow \Delta_X = 011 \tag{2.2}$$
$$v = 010 \ \ v^* = 011 \quad \mathrm{SB}(010) = 110 \ \ \mathrm{SB}(011) = 010 \rightarrow \Delta_X = 100 \tag{2.3}$$
$$v = 011 \ \ v^* = 010 \quad \mathrm{SB}(011) = 010 \ \ \mathrm{SB}(010) = 110 \rightarrow \Delta_X = 100 \tag{2.4}$$
$$v = 100 \ \ v^* = 101 \quad \mathrm{SB}(100) = 100 \ \ \mathrm{SB}(101) = 001 \rightarrow \Delta_X = 101 \tag{2.5}$$
$$v = 101 \ \ v^* = 100 \quad \mathrm{SB}(101) = 001 \ \ \mathrm{SB}(100) = 100 \rightarrow \Delta_X = 101 \tag{2.6}$$
$$v = 110 \ \ v^* = 111 \quad \mathrm{SB}(110) = 111 \ \ \mathrm{SB}(111) = 101 \rightarrow \Delta_X = 010 \tag{2.7}$$
$$v = 111 \ \ v^* = 110 \quad \mathrm{SB}(111) = 101 \ \ \mathrm{SB}(110) = 111 \rightarrow \Delta_X = 010 \tag{2.8}$$
$$\tag{2.9}$$

Les deux couples possibles sont donc $(110, 111)$ et $(111, 110)$. Remarquons que chaque couple et son symétrique donnent exactement les mêmes choses, ce qui explique que les valeurs de la DDT sont toujours un multiple de 2. Dans cet exemple, $x = 101$ et $x^* = 100$. Les deux valeurs de

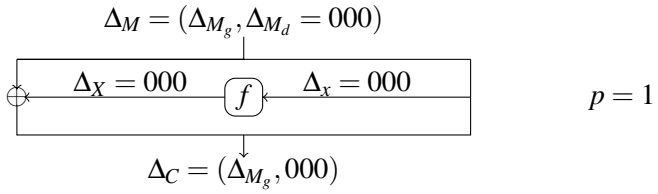


FIGURE 2.3 – Caractéristique d'un Feistel à 1 tour avec probabilité = 1

clé possibles sont donc 010 et 011 car :

$$(x \oplus 010, x^* \oplus 010) = (111, 110)$$
$$(x \oplus 011, x^* \oplus 011) = (110, 111)$$

Il est donc possible de récupérer de l'information sur la clé sur un schéma de Feistel notamment grâce à la connaissance des différentielles d'entrée et de sortie de la dernière fonction de tour. L'exemple précédent est excessivement simple puisque le chiffrement ne comporte qu'un seul tour. En réalité, même si l'on connaît Δ_x grâce au chiffré, il est beaucoup plus difficile de calculer Δ_X puisque la valeur à laquelle X est xorée pour calculer le chiffré n'est pas connue, et si l'algorithme est bien fait, cette valeur est sensée être uniforme à clé fixée lorsque les entrées sont uniformes, et ce pour toutes les clés.

La cryptanalyse différentielle telle que présentée par Biham et Shamir fait en sorte de briser l'uniformité de la distribution de cette valeur grâce aux propriétés différentielles de la S-box, pour ensuite exploiter les différentielles tel que présenté ci-dessus. Pour cela, ils introduisent la notion de caractéristique. Une caractéristique est définie par l'ensemble des différentielles de message, de chiffré, d'entrée et de sortie de tours au cours du chiffrement d'une paire de messages, et à chacune est associée une probabilité. Reprenons l'exemple 2.2. Une caractéristique évidente est celle associée à une différence nulle de la partie droite du message, montrée en figure 2.3. Dans ce cas, il en résulte que le chiffré a la même différentielle que le message, et ce avec certitude.

Une autre caractéristique possible est celle présentée en figure 2.4. Elle illustre le cas présenté dans l'exemple détaillé plus haut. Si l'on reprend les équations 2.9, on peut y voir qu'il existe deux équations parmi les huit qui résultent en une différentielle de sortie de 010 pour une différentielle d'entrée de 001. La probabilité est donc de $\frac{2}{8}$. Si les messages sont choisis de manière à ce que $\Delta_{M_g} = 010$, il y a alors une probabilité de $\frac{1}{4}$ que celle-ci s'annule avec Δ_X, résultant en la caractéristique présentée. La probabilité maximum que peut atteindre une caractéristique est fortement liée à l'uniformité différentielle de la S-box. Par exemple, les caractéristiques sur un tour ayant la plus forte probabilité ont une probabilité de $\frac{\delta_{SB}}{2^n}$.

Dans le cadre d'un schéma de Feistel comportant plus de tours, rappelons que la difficulté pour

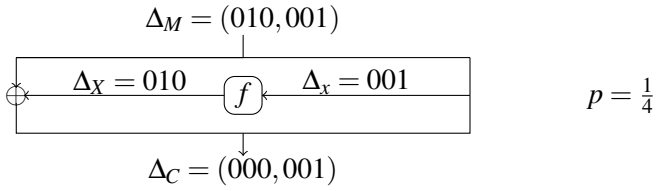


FIGURE 2.4 – Caractéristique d'un Feistel à 1 tour avec probabilité = $\frac{1}{4}$

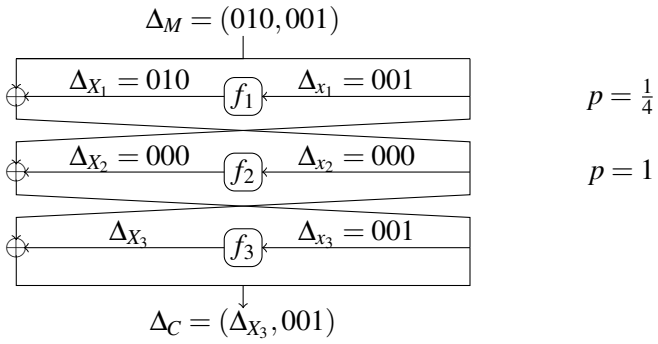


FIGURE 2.5 – Caractéristique Feistel 2 tours avec probabilité = $\frac{1}{4}$

réaliser une attaque différentielle est la capacité de l'attaquant à calculer Δ_X, la différentielle de sortie de la dernière S-box, et que cette valeur est calculable grâce à la connaissance du chiffré, et de la seconde entrée du xor. La figure 2.5 présente une caractéristique sur 2 tours de probabilité $p = \frac{1}{4} * 1$ couplée à un troisième tour. Sur ce schéma, f_i correspond au xor avec la clé de tour K_i et à l'application de la S-box. On en déduit que pour des messages ayant pour différentielle 010 001, avec une probabilité $\frac{1}{4}$ la première différentielle en entrée de dernier xor est $\Delta_{x_2} = 000$. Puisque cette caractéristique n'est pas systématique, et donc que δ_{x_2} n'est pas connu avec certitude, il n'est plus possible d'écarter de manière certaine les clés. Pour améliorer l'efficacité de l'analyse, il convient de différencier les paires de messages qui réalisent cette caractéristique, dites bonnes paires, de celles ne la réalisant pas, dites mauvaises paires. Un critère peut être appliqué pour écarter certaines mauvaises paires. Ce critère est de sélectionner les couples dont la différentielle des chiffrés réalise :

$$\Delta_C = \{(\Delta_{C_g}, \Delta_{x_r}) \mid \delta(\Delta_{x_r}, \Delta_{C_g} \oplus \Delta_{x_{r-1}}) > 0\}$$

avec x_r l'entrée de dernier tour. Ce critère exprime le fait que la différentielle de la partie droite du chiffré corresponde bien à la partie gauche de la sortie de la caractéristique, et que la différentielle de la partie gauche du chiffré soit possible sous condition de différentielle d'entrée et que la partie droite de la caractéristique soit bien celle attendue. Autrement dit, que sous conditions de la caractéristique, la différentielle du chiffré soit possible. Une fois ces couples sélectionnés, il suffit de reproduire l'attaque telle que décrite dans l'exemple détaillé, puis d'attribuer un score

à chaque clé comme étant le nombre de couples de messages qui permettent cette valeur, et de sélectionner K_3 comme étant la clé ayant le plus haut score.

À une caractéristique de probabilité élevée correspond un taux élevé de bonnes paires, et donc il est nécessaire d'effectuer moins de chiffrements afin de mener l'attaque. Puisque cette probabilité est liée à l'uniformité différentielle de la S-box, il est important que cette valeur soit la plus faible possible.

2.4 Cryptanalyse linéaire

2.4.1 Linearité

La linéarité d'une S-box est liée à sa propension à être approximée par une fonction linéaire. Pour la définir, il est nécessaire d'introduire d'autres notions.

Le biais $\mathcal{B}$ d'une fonction booléenne f définie sur $\mathbb{F}_2^n$ est :

$$\mathcal{B}(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)}$$

Une fonction dont le biais est à 0 est une fonction équilibrée, c'est-à-dire qui comporte autant de 0 que de 1. Le biais indique donc à quel point une fonction est éloignée de l'équilibre.

Nous définissons $\varphi_a(x)$ la combinaison linéaire entre a et x comme la fonction linéaire définie sur $\mathbb{F}_2^n$ telle que :

$$\varphi_a(x) = a \cdot x = \bigoplus_{i=1}^{n} a_i \cdot x_i$$

avec $a = \{a_1, ..., a_n\}$ et $x = \{x_1, ..., x_n\}$. Nous dirons aussi que x est masqué par a.
Le coefficient de Walsh de f en a est :

$$\mathcal{B}(f \oplus \varphi_a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}$$

Pour une fonction $\mathrm{SB} = \{\mathrm{SB}_1, ..., \mathrm{SB}_m\}$ définie de $\mathbb{F}_2^n$ vers $\mathbb{F}_2^m$ et $b = \{b_1, ..., b_m\}$, $\varphi_{\mathrm{SB}_b}(x)$ est la combinaison linéaire entre $\mathrm{SB}(x)$ et b et est définie comme la fonction booléenne telle que :

$$\varphi_{\mathrm{SB}_b}(x) = \bigoplus_{i=1}^{m} \mathrm{SB}_i(x) \cdot b_i$$

De la même manière, nous dirons que SB est masquée par b.

Nous définissons donc le biais $\mathcal{B}(a,b)$ en $a \in \mathbb{F}_2^n$ et $b \in \mathbb{F}_2^m$ d'une fonction SB comme :

$$\mathcal{B}(a,b) = \mathcal{B}(\varphi_{\mathrm{SB}_b} \oplus \varphi_a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\varphi_{\mathrm{SB}_b}(x) \oplus \varphi_a(x)}$$

Ce biais porte sur la fonction booléenne $\varphi_{\mathrm{SB}_b} \oplus \varphi_a$. Cette fonction décrit l'égalité entre les entrées masquées par a, et les sorties masquées par b.

La table d'approximation linéaire (LAT) de SB correspond aux biais pour tout couple (a,b). La linéarité $\mathcal{L}(\mathrm{SB})$ de SB est la valeur absolue maximale de cette table pour des masques non nuls :

$$\mathcal{L}(\mathrm{SB}) = \max_{a \neq 0, b \neq 0} |\mathcal{B}(a,b)|$$

Si l'on reprend l'exemple 2.1, voici le détail du calcul de $\mathcal{B}(011,101)$. Tout d'abord $\varphi_{011}(x) = 1 \cdot x_1 \oplus 1 \cdot x_2 \oplus 0 \cdot x_3$, d'où $\varphi_{011} = \{0,1,1,0,0,1,1,0\}$. Ensuite $\varphi_{\mathrm{SB}_{101}}(x) = 1 \cdot \mathrm{SB}_1(x) \oplus 0 \cdot \mathrm{SB}_2(x) \oplus 1 \cdot \mathrm{SB}_3(x)$, d'où $\varphi_{\mathrm{SB}_{101}} = \{0,1,1,0,1,1,0,0\}$. Le biais se calcule comme :

$$\mathcal{B}(011,101) = (-1)^{0\oplus 0} + (-1)^{1\oplus 1} + (-1)^{1\oplus 1} + (-1)^{0\oplus 0} + (-1)^{0\oplus 1} + (-1)^{1\oplus 1} + (-1)^{1\oplus 0} + (-1)^{0\oplus 0}$$
$$\mathcal{B}(011,101) = 4$$

La LAT complète de cette S-box est présentée en table 2.5.

$a \backslash b$	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	4	0	4	-4	0	4	0
2	0	0	4	4	4	-4	0	0
3	0	4	4	0	0	4	-4	0
4	0	4	-4	0	4	0	0	4
5	0	0	4	-4	0	0	4	4
6	0	-4	0	4	0	4	0	4
7	0	0	0	0	4	4	4	-4

TABLE 2.5 – LAT S-box 3 bits

D'après cette table, la linéarité de la S-box est $\mathcal{L}(\mathrm{SB}) = 4$. Par exemple puisque $\mathcal{B}(1,1) = 4$, on en déduit que $x_1 = \mathrm{SB}_1(x)$ avec probabilité $\frac{6}{8}$.

2.4.2 Principe et exemple de cryptanalyse linéaire

Le biais décrit la probabilité d'égalité entre une combinaison linéaire des entrées et une combinaison linéaire des sorties de la S-box. Il est donc possible d'établir des équations ayant une

certaine probabilité d'être réalisées. Le but de la cryptanalyse linéaire est d'impliquer le message, le chiffré et la clé dans ces équations. Des probabilités d'équations ayant une valeur différente de $\frac{1}{2}$ permettent de résoudre ces systèmes, i.e retrouver la clé, grâce à un grand nombre de couples de message/chiffré. Plus la probabilité en est éloignée, et plus l'attaque est efficace. Elle a notamment été appliquée sur le DES par Matsui dans [Mat93].

Pour en expliquer le principe, reprenons l'exemple d'un Feistel 1 tour sur 3 bits décrit à la figure 2.2. Soient le message $M = \{m_1, ..., m_6\}$, le chiffré $C = \{c_1, ..., c_6\}$ et la clé $K = \{k_1, ..., k_3\}$. f consiste en le xor de la clé et l'application de la S-box. Reprenons la S-box décrite en table 2.1. D'après sa LAT présentée en table 2.5, $\mathcal{B}(1,1) = 4$, et donc $p(x_1 = \mathrm{SB}_1(x)) = \frac{6}{8}$. Nous pouvons donc établir l'équation suivante avec probabilité $\frac{6}{8}$:

$$m_1 \oplus k_1 \oplus m_4 = \mathrm{SB}_1(m_1 \oplus k_1) \oplus m_4 \tag{2.10}$$

$$m_1 \oplus k_1 \oplus m_4 = c_4 \tag{2.11}$$

$$k_1 = m_1 \oplus m_4 \oplus c_4 \tag{2.12}$$

Prenons 8 messages et calculons leurs chiffrés pour la clé $k = 011$:

$$M = 100\ 000 \rightarrow C = 110\ 000$$
$$M = 100\ 001 \rightarrow C = 010\ 001$$
$$M = 100\ 010 \rightarrow C = 111\ 010$$
$$M = 100\ 011 \rightarrow C = 100\ 011$$
$$M = 100\ 100 \rightarrow C = 001\ 100$$
$$M = 100\ 101 \rightarrow C = 011\ 101$$
$$M = 100\ 110 \rightarrow C = 101\ 110$$
$$M = 100\ 111 \rightarrow C = 000\ 111$$

D'après l'équation 2.12, on en déduit les équations suivantes :

$$k_1 = 0 \oplus 0 \oplus 0 = 0$$
$$k_1 = 1 \oplus 0 \oplus 0 = 1$$
$$k_1 = 0 \oplus 1 \oplus 0 = 1$$
$$k_1 = 1 \oplus 0 \oplus 0 = 1$$
$$k_1 = 0 \oplus 1 \oplus 0 = 1$$
$$k_1 = 1 \oplus 1 \oplus 0 = 0$$
$$k_1 = 0 \oplus 1 \oplus 0 = 1$$
$$k_1 = 1 \oplus 0 \oplus 0 = 1$$

La proportion d'équations qui indiquent la bonne valeur de k_1 est bien de $\frac{6}{8}$.

Pour porter l'attaque à plus de tours, il convient de procéder un peu de la même manière que pour les attaques différentielles, à savoir combiner plusieurs équations consécutives. La figure 2.6 montre les égalités attendues grâce au biais $\mathcal{B}(1,1)$ pour un Feistel 3 tours sur 6 bits, avec $x_i = \{x_{i,1}, x_{i,2}, x_{i,3}\}$ et $X_i = \{X_{i,1}, X_{i,2}, X_{i,3}\}$, respectivement les entrées et sorties de la fonction f_i. Notons $M = \{m_1, ..., m_6\}$, $C = \{c_1, ..., c_6\}$, et $K_i = \{k_{i,1}, ..., k_{i,3}\}$ pour $i \in \{1,3\}$. De cette figure, nous pouvons en déduire les équations suivantes au premier et dernier tour :

$$x_{1,1} \oplus X_{1,1} = k_{1,1}$$
$$x_{3,1} \oplus X_{3,1} = k_{3,1}$$

$x_{1,1}$ et $x_{3,1}$ sont exprimables en fonction du message et du chiffré. Les équations précédentes donnent :

$$m_1 \oplus X_{1,1} = k_{1,1}$$
$$c_1 \oplus X_{3,1} = k_{3,1}$$

ici $X_{3,1}$ est exprimable en fonction de $X_{1,1}$ puisque $X_{3,1} = X_{1,1} \oplus m_4 \oplus c_4$. Le xor des équations précédentes donne donc :

$$m_1 \oplus X_{1,1} \oplus c_1 \oplus X_{3,1} = k_{1,1} \oplus k_{3,1} \tag{2.13}$$
$$m_1 \oplus c_1 \oplus m_4 \oplus c_4 = k_{1,1} \oplus k_{3,1} \tag{2.14}$$

La probabilité qu'une telle équation soit vraie est calculable de la manière suivante :

$$\frac{1}{2} + 2^{r-1} \prod_{i=1}^{r} \left(p_i - \frac{1}{2} \right)$$

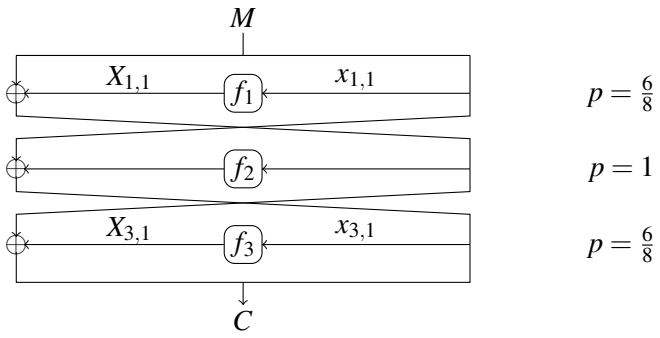


FIGURE 2.6 – Égalités Feistel 3 tours

avec r le nombre de tours et p_i la probabilité associée à une égalité d'un tour. Dans notre cas, la probabilité associée à la figure 2.6 est $p = \frac{1}{2} + 2^{3-1} \cdot (\frac{6}{8} - \frac{1}{2})^2 \cdot (1 - \frac{1}{2}) = 0.625$

Pour une clé étendue $K = \{011, 100, 110\}$, les 64 couples messages chiffrés sont les suivants :

000 000 001 111	000 001 101 111	000 010 000 111	000 011 011 111
000 100 110 111	000 101 100 111	000 110 010 111	000 111 111 111
001 000 001 101	001 001 110 011	001 010 000 101	001 011 101 010
001 100 010 100	001 101 111 011	001 110 100 010	001 111 011 100
010 000 110 100	010 001 001 001	010 010 000 011	010 011 100 100
010 100 111 110	010 101 101 110	010 110 010 011	010 111 011 001
011 000 100 001	011 001 001 010	011 010 000 110	011 011 011 110
011 100 010 010	011 101 110 101	011 110 111 001	011 111 101 101
100 000 110 110	100 001 010 110	100 010 000 000	100 011 101 011
100 100 011 101	100 101 100 000	100 110 111 101	100 111 001 011
101 000 011 010	101 001 101 100	101 010 000 100	101 011 010 000
101 100 111 000	101 101 110 010	101 110 100 110	101 111 001 110
110 000 011 000	110 001 010 101	110 010 000 001	110 011 100 101
110 100 110 001	110 101 111 100	110 110 001 100	110 111 101 000
111 000 100 011	111 001 110 000	111 010 000 010	111 011 010 001
111 100 011 011	111 101 101 001	111 110 001 000	111 111 111 010

L'équation 2.14 appliquée à ces 64 couples donne les proportions suivantes :

$$m_1 \oplus c_1 \oplus m_4 \oplus c_4 = 0 \qquad p = \frac{24}{64}$$
$$m_1 \oplus c_1 \oplus m_4 \oplus c_4 = 1 \qquad p = \frac{40}{64} = 0.625$$

Ainsi, $k_{1,1} \oplus k_{3,1}$ est bien désigné par la bonne proportion de couples.

L'efficacité de l'attaque est fortement liée aux probabilités de chacune des équations, i.e au biais. Pour résister à ce type d'attaque, il est donc important que le biais maximum soit le plus faible possible. C'est pour cela que la linéarité d'une S-box doit être la plus faible possible.

2.5 Classification des S-boxes 4 bits

Les S-box présentent différents niveaux de sécurité vis-à-vis de la cryptanalyse comme nous venons de le voir. La sélection de cette fonction doit minimiser ou maximiser plusieurs critères à la fois. Nous sommes donc en droit de nous demander quelles sont globalement les meilleures. Pour répondre à ce problème, de nombreux travaux de classification des S-boxes ont été menés. Alors que les S-boxes 8 bits proposent un trop large éventail de possibilités pour être complètement étudiées, les 4 bits l'ont été avec succès pour plusieurs types de classifications. Ici nous présentons particulièrement la classification par équivalence affine pour les S-boxes 4 bits. Cette partie sera utile pour le chapitre 4 puisque la recherche de la S-box de Lilliput-TBC se base grandement sur cette classification.

2.5.1 Équivalence linéaire/affine

L'équivalence linéaire entre deux S-boxes SB_1 et SB_2 ne détériore pas les qualités différentielles, linéaires et algébriques. Elle est définie de la manière suivante :

$$\mathrm{SB}_1(x) = B(\mathrm{SB}_2(A(x))) \quad \forall x \in \mathbb{F}_2^n$$

Avec A et B deux tables linéaires. Si un tel couple (A,B) n'existe pas, alors SB_1 et SB_2 ne sont pas linéairement équivalentes.

L'équivalence affine préserve, elle aussi, ces critères et est définie de la manière suivante :

$$\mathrm{SB}_1(x) = B(\mathrm{SB}_2(A(x \oplus a)) \oplus b) \quad \forall x \in \mathbb{F}_2^n$$

avec a et b deux valeurs dans $\mathbb{F}_2^n$.

De Cannière [DC07] a classifié les permutations 4 bits en 302 classes d'équivalence affine. Toute S-box étant une permutation 4 bits fait donc partie de l'une de ces 302 classes. Parmi elles, 1 est de degré 1 dite affine, 6 sont de degré 2 dites quadratiques, et les 295 autres sont de degré 3 dites cubiques. Leander et Poschmann avaient identifié 16 classes optimales telles que $deg(\mathrm{SB}) = 3$, $\delta(\mathrm{SB}) = 4$ et $\mathcal{L}(\mathrm{SB}) = 8$ dans [LP07].

2.6 Implementation threshold

L'implémentation threshold (TI) est une contre-mesure qui s'apparente au masquage, et a pour but principal de supprimer les fuites liées aux glitchs. Nous expliquons ici rapidement ce que sont les glitchs, puis nous nous intéresserons à ce type particulier de masquage d'implémentations matérielles.

2.6.1 Les glitchs

Lorsque l'on considère un circuit constitué de portes logiques, on suppose généralement que chaque porte ne change d'état qu'une seule fois, lorsque ses différentes entrées ont été calculées. Or pour plusieurs raisons, il est possible que cette supposition ne soit pas vérifiée et que certaines portes ne basculent pas directement dans leur état final. Ce comportement est appelé *glitch*. Les raisons de ce comportement peuvent être en partie expliquées par l'arrivée différée des entrées. Cela peut se produire à cause d'une différence de longueurs de fils, d'une différence de délai de calcul d'une porte en amont, d'une différence du nombre de portes résultant en chacune des entrées, ou une combinaison de tout cela. L'impact des glitchs a été présenté notamment dans [MPG05] où les auteurs concluent qu'une porte protégée effectuant un calcul masqué grâce à du masquage classique est en fait vulnérable aux attaques de type DPA car la consommation d'une telle porte est fonction du nombre de glitchs qui est lui-même corrélé à une donnée non masquée. Pour expliquer cette conclusion, nous reprenons l'exemple donné dans [NRS11]. La figure 2.7 présente une porte AND protégée. La porte AND prend en entrée deux bits x et y et calcule $z = xy$. Cette porte protégée prend en entrée les bits masqués $x' = x \oplus r_x$ et $y' = y \oplus r_y$, leurs masques respectifs r_x et r_y, le masque de sortie r_z, et calcule $z' = z \oplus r_z$. Le moyen d'y parvenir est l'équation suivante :

$$z' = (((r_x r_y \oplus r_z) \oplus r_x y') \oplus r_y x') \oplus x' y')$$

L'ordre particulier des XOR permet de toujours se retrouver au moins masqué par r_z. Ils donnent l'exemple particulier où un glitch se produit sur x'. Cette valeur est utilisée dans les deux portes les plus à gauche. Ces portes sont susceptibles de produire elles-mêmes un glitch, selon les valeurs de y' et de r_y. Ces glitchs se propageront à leur tour aux portes XOR suivantes. La table 2.6 donne le nombre de portes impactées par un glitch en fonction des valeurs de r_y et y', et donc de y.

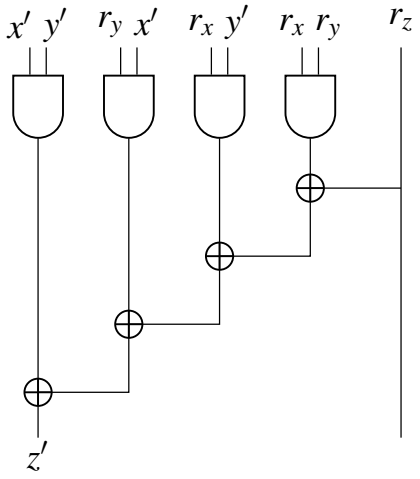


FIGURE 2.7 – Porte AND masquée

y	r_y	y'	AND	XOR
0	0	0	0	0
1	0	1	1	1
1	1	0	1	2
0	1	1	2	2

TABLE 2.6 – Nombre de portes impactées par un glitch sur x'

Lorsque r_y et y' sont nuls, les portes AND correspondantes sont systématiquement à 0, même si x' varie. Aucune porte n'est donc impactée par le glitch. Lorsque y' n'est pas nul, ce AND est touché, et le XOR suivant aussi. Lorsque c'est r_y qui n'est pas nul, le glitch impact les deux XOR utilisant la sortie du AND. Et enfin lorsque les deux valeurs ne sont pas nulles, les deux AND et les deux XOR sont impactés. Si la consommation est liée au nombre de portes subissant le glitch, alors elle est corrélée à la valeur de y puisque son espérance pour $y = 0$ est différente de $y = 1$.

2.6.2 Le masquage threshold

Pour résoudre le problème créé par les glitchs, Nikova et al. proposent l'implémentation threshold dans [NRR06], basée principalement sur le principe de calcul multi-parties (multi-party computation). Ce principe consiste à diviser les variables d'entrée en plusieurs shares, tout comme le masquage logiciel, mais aussi de diviser le calcul lui-même en sous-calculs. La thèse [Bil15] parle de manière étendue de ce sujet. Ici nous expliquons les bases nécessaires à la compréhension de la partie traitant de la S-box Lilliput-TBC.

Soit la fonction $f(x) = a$ définie de $\mathbb{F}_2^n$ vers $\mathbb{F}_2^m$. Pour réaliser son implémentation threshold, il convient tout d'abord de diviser la variable d'entrée x en s_e shares grâce au masquage booléen tel que $x = \bigoplus_{i=1}^{s_e} x_i$. Nous appelons $\mathbf{x} = \{x_1, ..., x_{s_e}\}$ le "sharing" de x. Le calcul est ensuite divisé en s_s sous-calculs $f_j(\mathbf{x})$, ce qui forme le sharing de sortie $\mathbf{a} = \{a_1 = f_1(\mathbf{x}), ..., a_{s_s} = f_{s_s}(\mathbf{x})\}$. La décomposition du calcul en sous-calculs est appelée le sharing de f et est notée $\mathbf{f} = \{f_1, ..., f_{s_s}\}$. En dehors du fait que, de la même manière que le masquage classique, le sharing des variables d'entrée doit être uniforme, l'implémentation threshold doit respecter trois propriétés :

— L'exactitude

— La non-complétude

— L'uniformité

L'exactitude correspond tout simplement au fait que le résultat donné par l'implémentation soit bien le même qu'une implémentation classique. Cela se définit de la manière suivante : Pour $f(x) = a$, alors l'implémentation threshold telle que construite ci-dessus doit satisfaire $\bigoplus_{i=1}^{s_s} f_i(\mathbf{x}) = a$ pour tout sharing $\mathbf{x}$ de x. Une manière d'y arriver est de décliner chaque terme du calcul en tous ceux formés grâce aux shares des variables concernées, puis de les disperser et les xorer dans chacun des sous-calculs. Exemple : $f(x,y) = xy \oplus x$ et $s_e = 2$. Dans ce cas, le terme xy se décline en $\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}$, et x en $\{x_1, x_2\}$.

La non-complétude permet d'assurer qu'un attaquant observant d variables, et plus particulièrement d sous-calculs, soit incapable de retrouver de l'information sur des variables non masquées. Dans le monde du threshold et du matériel, cela s'apparente à observer d fils du circuits. Dans le monde logiciel, cela s'apparente à observer d instants. Ainsi une implémentation threshold protégeant contre un attaquant observant d sous-calculs s'apparente à une implémentation protégée à l'ordre d. La non-complétude d'un sous-calcul est atteinte si au moins d shares de la variable d'entrée sont absentes du calcul. La non-complétude d'un calcul est atteinte si tous les sous-calculs respectent la non-complétude. Dans le cas où la fonction possède plusieurs variables d'entrée, la non-complétude doit être vraie pour chacune de ces variables. Le nombre de shares de sortie est contraint par le degré deg de la fonction, à savoir $s_s \geq deg + 1$.

Enfin, l'uniformité se réfère à l'uniformité du sharing en sortie, et est nécessaire uniquement si ce calcul est suivi par d'autres. En effet, l'uniformité des entrées est nécessaire à la sécurité d'une telle contre-mesure. Ainsi, si deux calculs sont consécutifs et que le premier n'assure pas l'uniformité, alors la sécurité du second est compromise. L'uniformité est réalisée si : $\forall x \in \mathbb{F}_2^n$, $\forall a = f(x) \in \mathbb{F}_2^m$, $\forall \mathbf{a}$ un sharing de a, alors :

$$|\{(\mathbf{x} \mid \mathbf{f}(\mathbf{x}) = \mathbf{a}\}| = \frac{2^{n\cdot(s_e-1)}}{2^{m\cdot(s_s-1)}}$$

Autrement dit, l'uniformité est respectée si tout sharing de a a le même nombre de sharing de

x antécédents. Dans le cas particulier où $n = m$ et $s_e = s_s$, $\mathbf{f}$ forme une bijection des sharings d'entrée vers ceux de sortie.

Voici des exemples de constructions réalisant les deux premières propriétés. Nous nous concentrons sur des cas où $s_e = s_s$ puisque c'est celui que nous utiliserons pour Lilliput-TBC :

Fonction affine Une fonction affine est très simple à implémenter. Il suffit pour cela que l'un des sous-calculs comporte la constante, et que chacun ne traite qu'une share de chacune des entrées. Soient $f(x) = 1 \oplus x$ et $s_e = s_s = 2$. Une implémentation threshold de cette fonction est :

$$f_1 = x_1 \oplus 1$$
$$f_2 = x_2$$

Fonction quadratique Le sharing direct est une méthode permettant de réaliser de manière automatique le sharing d'une fonction. Il consiste à établir $s_e = s_s = deg + 1$, et à faire en sorte que chaque sous-calcul i soit indépendant (n'utilise pas) de la share i de chacune des variables. Soient $f(x,y) = xy \oplus x$, $s_e = s_s = deg + 1 = 3$. L'implémentation threshold direct de cette fonction est :

$$f_1 = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_2$$
$$f_2 = x_3y_3 \oplus x_3y_1 \oplus x_1y_3 \oplus x_3$$
$$f_3 = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_1$$

L'uniformité, quant à elle n'est pas systématique. Il existe plusieurs moyens d'y parvenir. Parmi eux, nous pouvons citer celui qui consiste à augmenter/diminuer le nombre de shares d'entrée et/ou de sortie, celui qui consiste à déplacer certains termes, dit termes de correction, dans d'autres sous-calculs sans briser la non-complétude (c'est par exemple le cas de x_3y_3 dans l'exemple précédent qui pourrait faire partie de f_1), ou encore celui consistant à remasquer des shares de sortie grâce au tirage de masques uniformes.

2.6.3 Threshold de S-box 4 bits

Une étude détaillée portant sur l'implémentation threshold des S-boxes 4 bits est présentée dans [BNN+15] et propose notamment la décomposition des circuits cubiques en plusieurs quadratiques afin de diminuer le nombre de shares nécessaires.
Leur étude passe en revue chacune des 302 classes d'équivalence affine afin de déterminer le nombre de shares minimum permettant le sharing direct, et le sharing avec termes de correction. Ils proposent aussi l'étude des classes cubiques atteignables grâce à la composition de S-boxes

quadratiques. Cette décomposition permet de réduire une classe de degré 3 en plusieurs de degré 2 nécessitant moins de shares et étant potentiellement thresholdable directement. Grâce à leurs résultats, et notamment la liste des classes atteignables par composition, nous avons pu optimiser la compacité de la S-box thresholdée de Lilliput-TBC, optimisation sur laquelle nous reviendrons dans le chapitre 4.

2.7 Construction de S-boxes 8 bits à partir de S-boxes 4 bits

Une problématique liée aux S-boxes est la compacité des circuits. Celle-ci est induite par les faibles ressources, de surface notamment, proposées par les appareils embarqués. La cryptographie dite légère, "lightweight", est donc un sujet de recherche actuel très intéressant. L'une des parties les plus coûteuses étant la S-box, beaucoup de travaux ont vu le jour afin de réduire au maximum ces composants. De plus, des algorithmes sur 4 bits, et donc plus compactes voient le jour, ce qui explique les nombreux papiers détaillant les caractéristiques des S-boxes 4 bits. Pour résoudre ce problème de compacité, deux approches principales sont explorées. La première consiste à optimiser une S-box déjà existante. L'AES a notamment vu la sienne faire le sujet de nombreuses recherches d'optimisations telles que [Can05, BP09]. Dans le cadre des S-boxes 4 bits qui présentent un espace bien plus restreint, certains travaux permettent de trouver avec certitude les circuits les plus compacts [Sto16, JPST17]. Cependant, cela ne permet pas de dire si oui ou non ces circuits sont les plus compacts pour les critères que réalisent les S-boxes optimisées. La seconde approche consiste à construire des circuits compacts et à en trouver avec des critères suffisants. Ullrich et al. [UDCI+11] explorent systématiquement les circuits ayant un certain nombre de portes avant d'en ajouter de nouvelles, trouvant ainsi les circuits optimaux de certaines classes d'affine équivalence pour leur jeu d'instructions/portes. D'un autre côté, Canteaut et al. par exemple étudient la construction de S-boxes 8 bits à partir de 4 bits [CDL15]. Ils utilisent pour cela des S-boxes 4 bits à l'intérieur de schémas de type Feistel ou basés sur MISTY notamment.
D'autres travaux combinent les idées de constructions grâce à des schémas et les implémentations threshold comme dans [BGG+17].

Chapitre 3

Attaques par signatures / distributions jointes

Les attaques par canaux cachés telles que la DPA, CPA, ou encore MIA sont des attaques statistiques qui reposent sur la connaissance du message clair ou du message chiffré comme nous avons pu le voir. Un autre type d'attaques a vu le jour récemment et a pour avantage majeur de ne plus nécessiter ces connaissances : les attaques dites à clair et chiffré inconnus. Une attaque profilée présentée dans [HTM09] construit des profils ayant pour but de déduire les poids de Hamming de variables internes afin d'écarter des hypothèses de clé. Dans ce document, nous nous intéressons particulièrement à un autre type d'attaques à clair et chiffré inconnus qui ont pour avantage de ne pas nécessiter de phase de profilage : les attaques par signatures comme nommées dans la thèse de Linge [Lin13] et originellement présentées dans [LDL13], que nous appellerons aussi par distributions jointes. Le principe de cette attaque repose sur l'observation des distributions jointes de variables internes. Les informations liées à ces variables sont seulement issues des observations des fuites et, lorsqu'elles sont bien choisies, permettent de discriminer la clé. Des travaux basés sur ce principe voient actuellement le jour. Dans ce contexte, il existe deux axes principaux d'amélioration. Le premier concerne l'obtention des distributions internes, ce qui peut par exemple passer par l'obtention des instants de manipulation des variables, les points d'intérêts, ou encore la conversion ou obtention des observations et données exploitables, le poids de Hamming par exemple. Korkikian explore cela dans sa thèse [Kor16] en proposant une technique à base d'injection de fautes. Le second axe est celui qui porte sur la seconde phase de l'attaque, à savoir le choix du distingueur qui permet de déduire la clé grâce aux distributions. Cet axe est par exemple exploré dans la thèse de Le Bouder [LB14] qui utilise l'inférence Bayésienne.

Ce chapitre porte sur des améliorations publiées à CHES 2017 et Cosade 2018. Ces améliorations sont multiples et sont notamment relatives à l'obtention des poids de Hamming des variables, à la sélection des variables utilisées dans les distributions jointes, ou encore à l'application d'une telle

attaque dans le cadre d'implémentations masquées. Il présente aussi des applications réelles de cette attaque. Avant d'exposer cela, nous rappelons tout d'abord le principe de l'attaque.

3.1 Principe des attaques par distributions jointes

Dans ce chapitre nous traitons particulièrement le cas de l'AES. Pour cette raison, la plupart des exemples/explications utilisent l'octet ($\mathrm{GF}(2^8)$) comme ensemble de définition. De plus, k identifie l'octet de clé utilisé lors du `AddRoundKey`. Cet octet est en réalité un octet de clé de tour, mais nous ne ferons que peu la distinction pour ne pas alourdir le texte. Pour la même raison, lorsque les exemples se font sur un octet, nous pourrons faire référence à k sous le nom de clé sans spécifier l'octet. Il en va de même pour d'autres variables.

3.1.1 Distributions jointes

Rappelons avant tout : la distribution discrétisée d'une variable peut s'exprimer comme la probabilité de chacune des valeurs possibles qu'elle peut prendre. La distribution jointe discrétisée de n variables peut s'exprimer comme la probabilité de chacun des n-uplets possibles atteignables. Considérons la fonction φ_k telle que :

$$\begin{aligned} \varphi_k : \quad & \mathrm{GF}(2^8) \mapsto \mathrm{GF}(2^8) \\ & m \mapsto m \oplus k \end{aligned}$$

Il est évident que la distribution de $\varphi_k(m)$ est uniforme lorsque m est uniforme puisqu'elle réalise une bijection. En cela, l'observation de la distribution de cette fonction ne renseigne en rien sur la valeur de k.

Nous avons vu dans la partie traitant des attaques par canaux cachés que la fuite est souvent assimilable à une fonction du poids de Hamming. Dans la suite de ce chapitre, nous faisons la même hypothèse, et c'est pour cela que nous nous intéressons à des distributions de poids de Hamming plutôt qu'à des distributions de valeurs. L'attaque n'est cependant pas intrinsèque à ce modèle de consommation et d'autres modèles sont susceptibles de permettre l'attaque. Nous remarquerons même plus tard dans ce chapitre que le nombre classes distinctes du modèle peut rendre cette attaque plus efficace, le poids de Hamming n'en comportant que neuf.

La distribution des poids de Hamming de la fonction φ_k est définie de la manière suivante :

$$\begin{aligned} \delta_{\varphi_k} &= \{p_h \mid h \in \mathrm{HW}(\varphi_k)\} \\ p_h &= \{\sum_{m \in \mathrm{GF}(2^8)} \Pr(m) \mid \mathrm{HW}(\varphi_k(m)) = h\} \end{aligned}$$

Lorsque m est distribué uniformément, et ce sera le cas pour le reste de ce document, nous

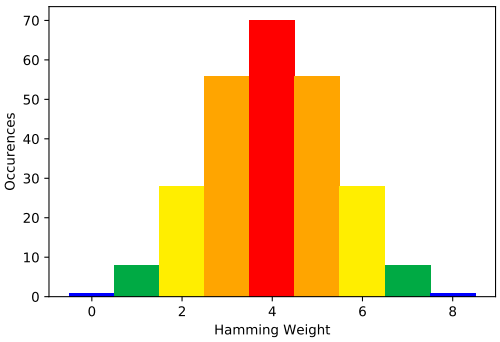


FIGURE 3.1 – Distribution δ_{φ_k} identique pour tout k

pouvons transformer la probabilité d'une valeur de m en effectif :

$$\delta_{\varphi_k} = \{p_h \mid h \in \mathrm{HW}(\varphi_k)\}$$
$$p_h = \frac{1}{2^8}\#\{m \in \mathrm{GF}(2^8) \mid \mathrm{HW}(\varphi_k(m)) = h\}$$

Plus simplement, cette distribution est la probabilité de chacune des valeurs de poids de Hamming de l'ensemble des images de $\varphi_k(m)$ lorsque m prend toutes les valeurs de $\mathrm{GF}(2^8)$. Nous ferons en fait plus souvent référence à cette distribution en omettant le dénominateur 2^8 pour alléger la lecture, ce qui correspondra donc uniquement au nombre d'occurrences de chacun des poids de Hamming des images :

$$\delta_{\varphi_k} = \{p_h \mid h \in \mathrm{HW}(\varphi_k)\}$$
$$p_h = \{\#m \mid m \in \mathrm{GF}(2^8),\ \mathrm{HW}(\varphi_k(m)) = h\}$$

Dans notre exemple, de même que celle de m, cette distribution correspond aux coefficients d'une binomiale :

$$\delta_{\varphi_k}[h] = \binom{8}{h} \qquad 0 \leq h \leq 8$$

La figure 3.1 en est une représentation sous forme d'histogramme où chaque classe/colonne correspond à un poids de Hamming.

Le principe de l'attaque par distributions jointes repose sur l'observation de distributions de non pas une, mais plusieurs (et notamment deux) variables de manière simultanée. Soient deux fonctions φ_k^1 et φ_k^2 paramétrées par k et définies sur $\mathrm{GF}(2^8)$. La distribution jointe de ces deux fonctions est :

$$\delta_{(\varphi_k^1,\varphi_k^2)} = \{p_{(h_1,h_2)} \mid h_1 \in \mathrm{HW}(\varphi_k^1),\ h_2 \in \mathrm{HW}(\varphi_k^2)\} \tag{3.1}$$
$$p_{(h_1,h_2)} = \{\#m \mid m \in \mathrm{GF}(2^8),\ \big(\mathrm{HW}(\varphi_k^1(m)), \mathrm{HW}(\varphi_k^2(m))\big) = (h_1,h_2)\} \tag{3.2}$$

Autrement dit, la distribution jointe est le nombre d'occurrences de chacun des couples de poids de Hamming (h_1, h_2) lorsque m parcourt $\mathrm{GF}(2^8)$. Dans la suite de ce document, φ_k^1 sera très souvent assimilée à la fonction $m \mapsto m$.

Voici un exemple simple sur 1 bit tel que donné dans [Lin13]. Soient $m, x, k \in \mathrm{GF}(2)$, $\varphi_k^1(m) = m$ et $\varphi_k^2(m) = m \oplus k$. Les distributions jointes pour $k = 0$ et $k = 1$ sont les suivantes :

$k = 0$

HW(m) \ HW($m \oplus 0$)	0	1
0	1	
1		1

$k = 1$

HW(m) \ HW($m \oplus 1$)	0	1
0		1
1	1	

FIGURE 3.2 – Distributions jointes des HW de m et $m \oplus k$ pour $k = 0$ et $k = 1$

D'après cette figure, il est clair que les distributions sont différentes entre les deux valeurs de k. Il est donc possible, pour ces fonctions en particulier, de déduire quelle valeur de k a été utilisée en observant la distribution de couples. Dans cet exemple très particulier, puisque les couples atteints sont propres à l'une ou l'autre des distributions, un seul couple suffit.

La figure 3.3 montre une portion d'un tour d'AES appliqué à un seul octet. On peut y identifier m l'entrée de `AddRoundKey` et $x = m \oplus k$ la sortie avec k la clé, puis $y = \mathrm{SB}(x)$ la sortie de `SubBytes`. Dans le cadre d'attaques sur l'AES, il est courant d'attaquer la sortie de la S-box. Ici, les distributions jointes des poids de Hamming de x et y sont identiques pour toute valeur de clé k. En effet, les distributions ne gardent aucune notion d'ordre d'apparition des valeurs. Or $m \mapsto m \oplus k$ est une bijection dans $\mathrm{GF}(2^8)$ et la S-box est indépendante de la clé. Ainsi k ne fait que changer l'ordre d'apparition des valeurs de x et donc des couples (x, y), ce qui ne change en rien la distribution jointe. À l'inverse, la relation entre $m \mapsto m$ et $m \mapsto \mathrm{SB}(m \oplus k)$ est, elle, bien fonction de k. Lorsque l'on calcule les distributions jointes des poids de Hamming des couples (m, y) pour chacune des valeurs de k, ce qui correspond à $\varphi_k^1(m) = m$ et $\varphi_k^2(m) = \mathrm{SB}(m \oplus k)$ dans l'équation 3.2, il apparaît que chacune est unique. Ces distributions permettent donc de discriminer la valeur de k. Pour illustrer leur unicité, la figure 3.5 montre les histogrammes des distributions de deux valeurs de k, et la figure 3.4 montre la distance euclidienne entre chacune. Ces distributions jointes en particulier seront très largement utilisées dans la suite. Nous les noterons δ_k pour $k \in \mathrm{GF}(2^8)$, et comme ces distributions théoriques sont celles auxquelles nous viendrons comparer des distributions empiriques, nous les appellerons parfois les *modèles*. La probabilité d'un couple (h_1, h_2) d'une distribution jointe δ_k sera notée $\delta_k(h_1, h_2)$.

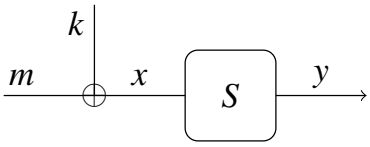


FIGURE 3.3 – `AddRoundKey` et `SubBytes` d'un tour d'AES

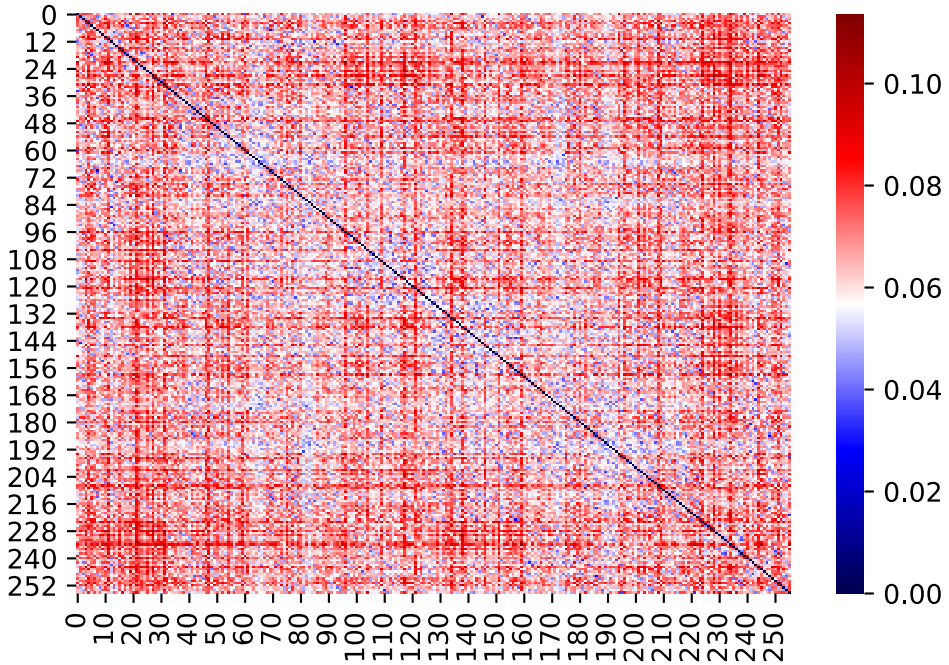
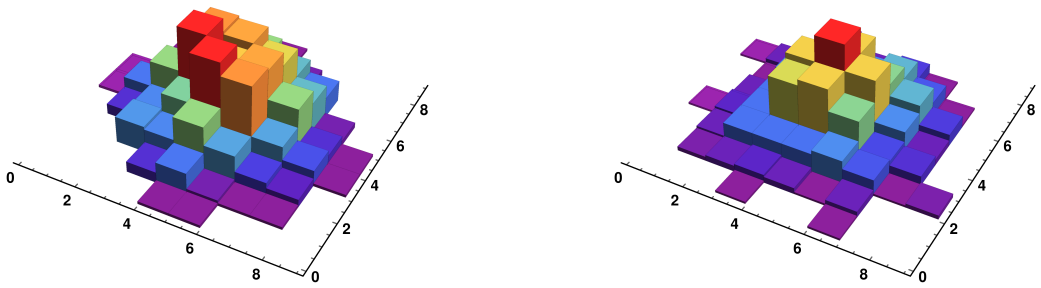


FIGURE 3.4 – Distance euclidienne entre les distributions jointes de m et y



(a) $k = 0x27$ (b) $k = 0x7e$

FIGURE 3.5 – Distributions jointes des HW de m et y

De la même manière, la relation entre $m \mapsto m$ et $\varphi_k(m) = m \oplus k$ est aussi fonction de k. Les distributions jointes des poids de Hamming de m et x permettent aussi de "discriminer" la clé. Cependant, le nombre de distributions différentes cette fois-ci est moindre. Alors que m et y donnent 256 distributions différentes (une par valeur de k), m et x n'en réalisent que 9 différentes. Chacune de ces distributions est en fait relative au poids de Hamming de k. La figure 3.6 présente ces 9 distributions. Dans ce cas, la discrimination de la clé se limite à l'identification de sa classe d'appartenance, à savoir son poids de Hamming. La quantité d'information obtenue est ainsi moindre que dans le cas précédent.

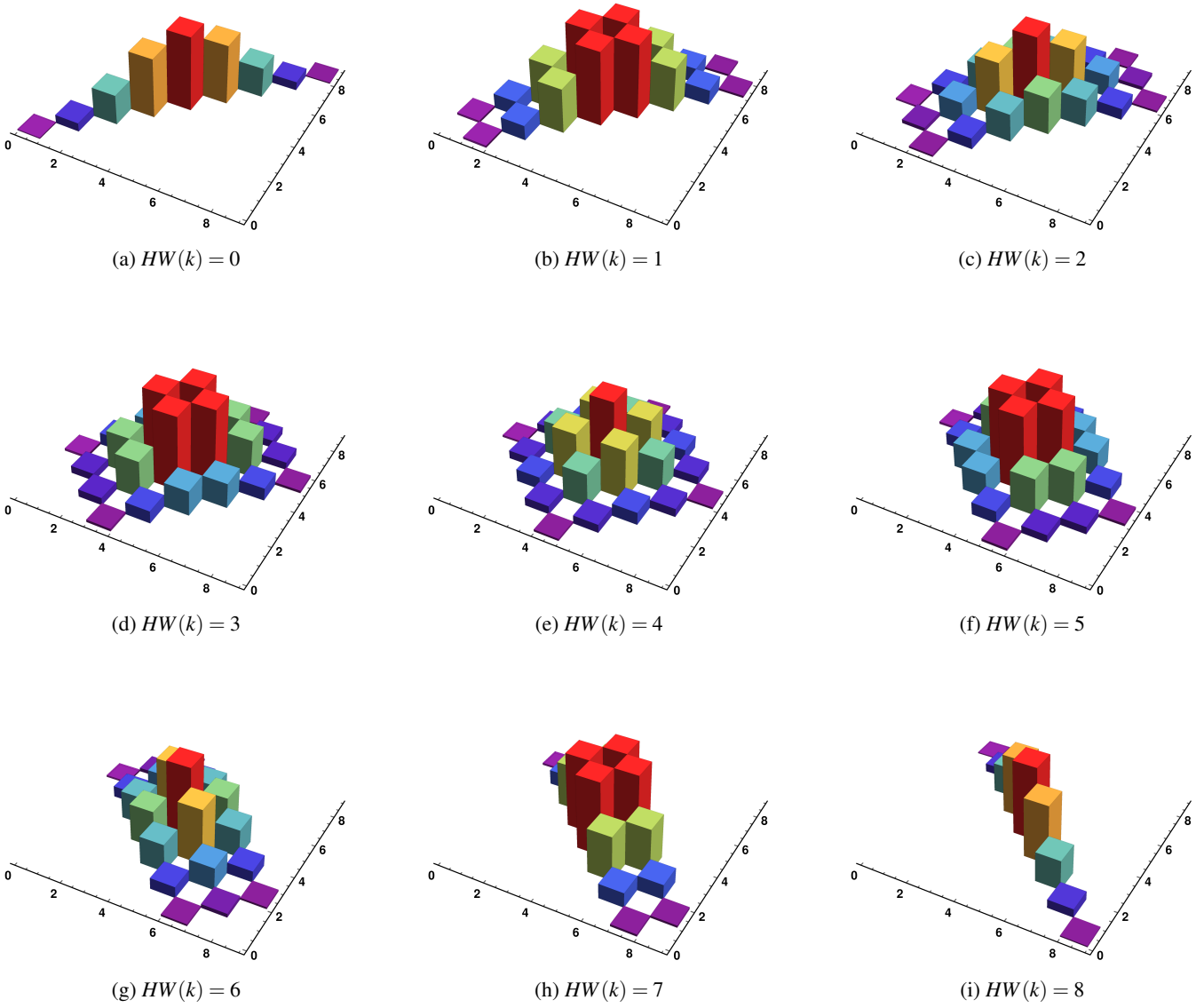
Démonstration : Soit $\delta_k(h_1,h_2)$ la probabilité de la colonne (h_1,h_2) de la distribution δ_k pour m uniforme. Elle s'exprime comme le nombre de valeurs de m qui vérifient à la fois $HW(m) = h_1$ et $HW(m \oplus k) = h_2$. Soient les séries ordonnées $\mathbf{m}$, $\mathbf{x}$ et $\mathbf{C}$ telles que $\mathbf{m} = (m_i = i)_{0 \leq i < 2^8}$, $\mathbf{x} = (x_i = m_i \oplus k)_{0 \leq i < 2^8}$, et $\mathbf{C} = (c_i = (\mathrm{HW}(m_i), \mathrm{HW}(x_i)))_{0 \leq i < 2^8}$ la série des couples de poids de Hamming de $\mathbf{m}$ et $\mathbf{x}$. Soit π une permutation des bits d'un octet. Soit $k' = \pi(k)$ et $\mathbf{m'} = \pi(\mathbf{m})$. π ne modifiant pas le poids de Hamming, $\mathrm{HW}(\mathbf{m}) = \mathrm{HW}(\mathbf{m'})$. De la même manière $\mathrm{HW}(\mathbf{x'}) = \mathrm{HW}(\mathbf{m'} \oplus k') = \mathrm{HW}(\pi(\mathbf{m}) \oplus \pi(k)) = \mathrm{HW}(\pi(\mathbf{m} \oplus k)) = \mathrm{HW}(\mathbf{x})$. Ainsi, $\mathbf{C'}$ étant la série des couples de poids de Hamming de $\mathbf{m'}$ et $\mathbf{x'}$ est identique à $\mathbf{C}$. Les distributions sont donc identiques à permutation de bits près de la clé.

Nous avons jusque là uniquement considéré des poids de Hamming des variables pour être cohérent avec le modèle de fuite en poids de Hamming. Nous avons remarqué précédemment sans explication que le principe de l'attaque n'est pas complètement remis en question si l'on considère un autre modèle, et que l'attaque peut être plus efficace si le nombre de classes distinguables par la fuite est plus élevé. En voici un exemple : dans le cas précédent où l'on considère les distributions jointes des poids de Hamming de m et x, il n'existe que neuf modèles distincts. Si l'on considère un modèle de consommation en valeur, c'est-à-dire que la fuite en un point présente 256 niveaux différents, alors ces distributions deviennent à ce moment toutes différentes. Pour s'en convaincre simplement, il suffit de voir qu'avec ce modèle, les distributions sont basées sur les couples de valeurs $(m, m \oplus k)$. Dans ce cas, l'observation d'un couple ne permet qu'à une seule valeur de k de vérifier ces conditions. Dans cette configuration, il existe 256 distributions jointes différentes.

3.1.2 Attaque originale

Nous présentons ici l'attaque originale par distributions jointes telle qu'exposée dans les travaux de Linge et al. [LDL13].

Nous avons vu dans la section précédente qu'il est possible de discriminer la clé dès lors que les distributions jointes de poids de Hamming de certaines variables internes diffèrent en fonc-



(a) $HW(k) = 0$ (b) $HW(k) = 1$ (c) $HW(k) = 2$

(d) $HW(k) = 3$ (e) $HW(k) = 4$ (f) $HW(k) = 5$

(g) $HW(k) = 6$ (h) $HW(k) = 7$ (i) $HW(k) = 8$

FIGURE 3.6 – Distributions jointes des poids de Hamming de m et x pour chacun des poids de Hamming de k

m	**m'**	$\mathbf{x} = \mathbf{m} \oplus$ 0x1	$\mathbf{x'} = \mathbf{m'} \oplus$ 0x2
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 1	0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 1	0 0 0 0 0 0 1 1	0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1

FIGURE 3.7 – Séries **m** et **m'** et leurs séries associées **x** et **x'** pour π la permutation des deux bits de poids faible

tion de la clé que nous voulons récupérer. Pour ce faire, il suffit de construire une distribution expérimentale tirée à partir d'un k inconnu, puis de déduire de quelle distribution δ_k elle provient. Cependant, avant même de pouvoir faire cette déduction, deux obstacles principaux se présentent. Premièrement, l'attaquant n'a accès qu'à des observations, de consommation de courant ou de rayonnements électromagnétiques par exemple. Il doit donc effectuer la conversion entre observation et donnée, poids de Hamming en l'occurrence, le tout sans avoir connaissance, a priori, du véritable modèle de consommation. Deuxièmement, l'attaquant ne réalise pas deux observations, chacune relative à une variable qu'il cible, mais fait plutôt de nombreuses observations, comme c'est le cas lors de l'acquisition d'une trace de courant par exemple. Il doit donc sélectionner, s'ils existent, les points relatifs aux variables ciblées. De ce fait, il est nécessaire de disposer de traces (re)synchronisées. Dans le cas contraire, l'attaquant serait obligé de déterminer pour chacune des traces quels sont les instants pertinents.

L'attaque se découpe en quatre parties principales :

1. Calculer les distributions jointes théoriques pour les variables ciblées, dites distributions théoriques, ou modèles
2. Déterminer les PoI (Points d'intérêt) liés aux variables ciblées
3. Convertir les fuites/observations en poids de Hamming pour construire une distribution dite expérimentale δ_e
4. Confronter la distribution expérimentale avec les distributions théoriques pour en déduire la clé

L'attaque originale se concentre sur les variables m et y qui sont respectivement les entrée de `AddRoundKey` et sortie de `SubBytes`. La première étape consiste à précalculer les distributions jointes $(\delta_k)_{0 \leq k \leq 2^8}$, exactement comme expliqué dans la partie précédente 3.1.1.

La seconde étape consistant à sélectionner les points d'intérêt est résolue en utilisant des traces de variance. En général, deux types de points/instants sont présents au sein d'une trace de consommation. Le premier type est intrinsèquement lié à l'algorithme et produit généralement

une fuite qui varie peu d'une exécution à une autre. Le second type de points, au contraire, est relatif aux données manipulées par l'algorithme, et dépend donc des données d'entrée. En faisant varier ces entrées, on peut s'attendre à ce que les fuites liées à ce second type de points varient plus fortement que pour le premier. Ainsi, une trace de variance permet généralement de faire ressortir ces derniers, limitant ainsi le nombre de points à considérer. Couplée avec une bonne connaissance de l'algorithme, il est parfois possible d'associer ces instants aux variables ciblées.

La troisième étape consiste à convertir des fuites observées en poids de Hamming. Linge et al. proposent de la traiter de la manière suivante : considérons M observations à l'instant de manipulation de m. Avec m distribué uniformément, le nombre de valeurs de poids de Hamming h est $\frac{M*\binom{8}{h}}{2^8}$. En classant les observations par ordre croissant, il suffit d'attribuer aux $\frac{M*\binom{8}{0}}{2^8}$ plus basses le poids de Hamming 0, puis les $\frac{M*\binom{8}{1}}{2^8}$ suivantes le poids de Hamming 1, etc. À l'issue de cette étape dite de calcul de poids de Hamming par la méthode des couches, et en la reproduisant à l'instant de manipulation de y, il est possible de construire la distribution jointe expérimentale δ_e grâce aux couples de valeurs ainsi calculés.

La dernière étape consiste à déduire de quelle distribution théorique δ_k provient la distribution expérimentale δ_e. Puisqu'elles sont toutes définies sur des couples d'entiers, utiliser des distances se fait de manière naturelle. Linge et al. proposent plusieurs distances telles que l'Inner Product, l'Harmonic Mean, le χ^2 ou celle de Kullback-Leiner, présentée en figure 3.8, selon le cas de figure. La clé est ensuite choisie comme celle minimisant la distance D : $k = \underset{k}{\operatorname{argmin}}\,(D(\delta_k, \delta_e))$.
Une amélioration majeure de cette dernière étape consiste à calculer la probabilité de chaque hypothèse de clé grâce à l'inférence Bayésienne comme exposé dans [LB14]. Nous appellerons cette méthode le maximum de vraisemblance (MV). Plus précisément, celui-ci consiste à mettre à jour la probabilité de chaque valeur de k à chaque couple de fuites observé. Soient $(h_1,h_2)_{1\leq i\leq n}$ n couples de poids de Hamming observés, dits observations dans la suite de cette explication. La probabilité de la valeur k après observation du n-ième couple est notée $\Pr(k \mid \{(h_1,h_2)_i\}_{1\leq i\leq n})$. Le théorème de Bayes permet d'exprimer la probabilité de k après une observation (h_1,h_2) de la manière suivante :

$$\Pr(k \mid (h_1,h_2)) = \frac{\Pr((h_1,h_2) \mid k) \cdot \Pr(k)}{\Pr((h_1,h_2))}$$

Le terme de gauche de cette égalité désigne la probabilité de la clé a posteriori de l'observation. Le terme $\Pr(k)$ désigne, lui, la probabilité a priori. Pour n observations successives, l'application de ce calcul met à jour progressivement la probabilité d'une clé. En omettant le dénominateur qui est identique pour tout k, la probabilité après n observations est la suivante :

$$\Pr(k \mid \{(h_1,h_2)_i\}_{1\leq i\leq n}) = \Pr(k \mid \{(h_1,h_2)_i\}_{1\leq i\leq n-1}) \cdot \Pr((h_1,h_2)_n \mid k)$$

Le terme $\Pr((h_1,h_2)_n \mid k)$ désigne la probabilité d'un couple de poids de Hamming des variables

$$D_{IP} = \sum_{h_1} \sum_{h_2} \delta_k(h_1,h_2) \cdot \delta_e(h_1,h_2)$$

(a) Inner Product

$$D_{HM} = \begin{cases} 1 - 2 \cdot \sum_{h_1} \sum_{h_2} \frac{\delta_k(h_1,h_2) \cdot \delta_e(h_1,h_2)}{\delta_k(h_1,h_2) + \delta_e(h_1,h_2)} & , \delta_k(h_1,h_2) + \delta_e(h_1,h_2) \neq 0 \\ 0 & , \delta_k(h_1,h_2) + \delta_e(h_1,h_2) = 0 \end{cases}$$

(b) Harmonic Mean distance

$$D_{\chi^2} = \begin{cases} \sum_{h_1} \sum_{h_2} \frac{(\delta_k(h1,h2) - \delta_e(h1,h2))^2}{\delta_e(h1,h2)} & , \delta_e(h1,h2) \neq 0 \\ 0 & , \delta_k(h1,h2) = \delta_e(h1,h2) = 0 \\ \infty & , \delta_e(h1,h2) = 0 \neq \delta_k(h1,h2) \end{cases}$$

(c) χ^2 distance

$$D_{KL} = \begin{cases} \sum_{h_1} \sum_{h_2} \delta_{k,h1,h2} \cdot ln(\frac{\delta_{k,h1,h2}}{\delta_{e,h1,h2}}) & , \delta_{e,h1,h2} \neq 0 \\ 0 & , \delta_{e,h1,h2} = 0 \end{cases}$$

(d) Kullback-Leiber distance

FIGURE 3.8 – Distances utilisées dans l'attaque originale

ciblées étant donné la clé k. Si les poids de Hamming sont exacts, il s'agit précisément de $\delta_k(h_1,h_2)$. Lorsque ces poids de Hamming sont issus de calculs, il est possible qu'ils soient inexacts, bruités. Le théorème des probabilités totales permet de calculer précisément la probabilité d'une observation en fonction de la clé en considérant que l'observation peut être issue de n'importe quel couple de poids de Hamming :

$$\Pr((h_1,h_2) \mid k) = \sum_{h_1^*,h_2^*} \Pr((h_1,h_2) \mid (h_1^*,h_2^*)) \cdot \Pr((h_1^*,h_2^*) \mid k)$$

où $h_1 = h_1^* + \omega_1$ et $h_2 = h_2^* + \omega_2$ sont les observations bruitées issues des véritables poids de Hamming manipulés h_1^* et h_2^* et bruitées par ω_1 et ω_2. Ici, $\Pr((h_1^*,h_2^*) \mid k)$ correspond à $\delta_k(h_1^*,h_2^*)$. $\Pr((h_1,h_2) \mid (h_1^*,h_2^*))$, quant à lui, désigne la probabilité de l'observation étant données les vraies valeurs manipulées. Les bruits étant supposés indépendants, cette probabilité s'exprime de la manière suivante :

$$\Pr((h_1,h_2) \mid (h_1^*,h_2^*)) = \Pr(\omega_1 = h_1 - h_1^*) \cdot \Pr(\omega_2 = h_2 - h_2^*)$$

La caractérisation du bruit permet de calculer cette dernière. Un bruit gaussien d'écart-type σ et d'espérance nulle donne le calcul suivant :

$$\Pr((h_1,h_2) \mid (h_1^*,h_2^*)) = \left(\frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_1 - h_1^*}{\sigma_1}\right)^2} \right) \cdot \left(\frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_2 - h_2^*}{\sigma_2}\right)^2} \right) \tag{3.3}$$

Notons que n'importe quelle caractérisation du bruit qui permet de calculer $Pr((h_1,h_2) \mid (h_1^*,h_2^*))$ peut se substituer à ce calcul. Pour le reste de ces travaux, nous approximons le bruit à une gaussienne d'écart-type σ et d'espérance nulle. La difficulté est de déterminer σ.

Grâce aux équations précédentes, nous calculons la probabilité de k a posteriori d'une observation (h_1,h_2) comme étant :

$$\Pr(k \mid (h_1,h_2)) = \frac{\sum_{h_1^*,h_2^*} \left[\left(\frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_1-h_1^*}{\sigma_1}\right)^2} \right) \cdot \left(\frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_2-h_2^*}{\sigma_2}\right)^2} \right) \cdot \delta_k(h_1^*,h_2^*) \right] \cdot \Pr(k)}{\Pr((h_1,h_2))} \tag{3.4}$$

Une remarque importante concernant l'attaque par distributions jointes : nous avons vu qu'elle consiste en quatre étapes. La remarque qui suit s'applique particulièrement aux deux dernières, à savoir la conversion des fuites, et le calcul des probabilités de chacune des clés. Ces étapes étant indépendantes, il est possible de substituer n'importe quelle méthode à une autre résolvant chacune des étapes. Si l'on dispose de N_1 méthodes $(m_{3,i})_{1\leq i\leq N_1}$ pour résoudre l'étape 3, et N_2 méthodes $(m_{4,j})_{1\leq j\leq N_2}$ pour résoudre l'étape 4, n'importe quelle combinaison $(m_{3,i},m_{4,j})$ est susceptible de fonctionner, à des degrés d'efficacité différents.

Cette attaque se distingue donc des attaques par canaux cachés classiques puisqu'elle ne requiert pas l'utilisation du message ou du chiffré pour calculer une valeur intermédiaire. Elle étudie le comportement joint de deux variables internes grâce aux observations de fuites. Un effet direct de ne pas avoir besoin du message/chiffré est qu'elle permet d'attaquer en milieu d'algorithme, ce qui peut permettre dans certains cas d'éviter des contre-mesures présentes uniquement sur les premiers et derniers tours en vue, justement, de contrer les attaques plus classiques. Enfin, cela peut permettre d'attaquer des blocs de chiffrement au milieu d'un mode, blocs qui peuvent parfois n'être pas protégés lorsque l'accès à leurs entrées et sorties ne sont pas disponibles à l'attaquant.

Les difficultés/désavantages concernent les étapes deux et trois, à savoir la sélection des PoI et la conversion des fuites, étapes qui n'étaient pas nécessaires dans la plupart des autres attaques.

Dans les prochaines parties nous présentons des améliorations que nous avons apportées à cette attaque et notamment son adaptation face à des contre-mesures de type masquage.

3.2 Améliorations à l'ordre 1

Ici nous exposons des variantes de certaines étapes à cette attaque, le but principal étant d'en améliorer l'efficacité à l'ordre 1. La première présente une nouvelle manière de convertir les

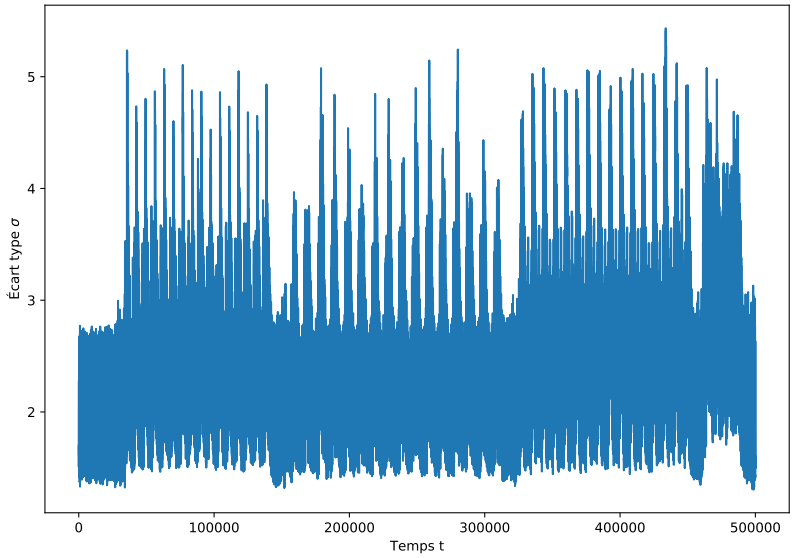


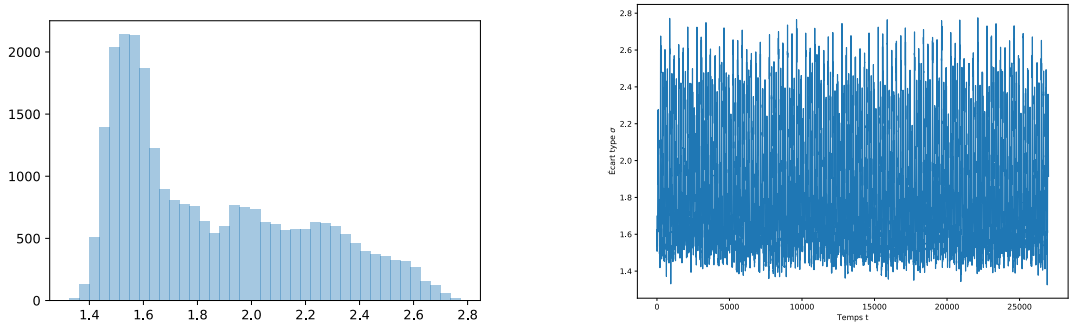
FIGURE 3.9 – Courbe d'écart-type d'un début de tour d'AES

fuites en poids de Hamming. Ensuite, nous étudierons l'utilisation d'autres/plus de variables dans les distributions théoriques et observations.

3.2.1 Calcul des HW par variance

Nous introduisons ici une méthode de conversion des fuites en poids de Hamming. Elle consiste à inverser le modèle de consommation, comme pourrait le faire une régression linéaire. Cependant, pour rester dans le cadre de l'attaque, et contrairement à la régression, cette méthode ne nécessite pas la connaissance des messages et/ou de la clé. Nous considérons un modèle de consommation affine en le poids de Hamming de la donnée manipulée, hypothèse classique utilisée notamment par la CPA par exemple, à savoir $L(x) = \alpha \cdot \mathrm{HW}(x) + \beta$ où x est la donnée manipulée et $L(x)$ la consommation associée. Il est possible de calculer la trace de variance (et écart-type) d'un jeu de traces de consommations, comme présenté en figure 3.9.

Nous avons expliqué qu'il y a généralement deux types de points sur cette courbe : ceux propres à l'algorithme lui-même et dont les consommations sont relativement invariantes, et ceux liés aux données qui provoquent généralement de plus fortes valeurs de variance. À cela s'ajoute le bruit de mesure/exécution qui augmente globalement la variance. Sur la figure 3.9, une partie de ces derniers est facilement identifiable puisque la courbe présente des pics. Sur cette trace qui fuite particulièrement, il est même possible d'identifier des motifs de 16 pics, chaque pic



(a) Distribution

(b) Trace σ

FIGURE 3.10 – Distribution et trace d'écart-type des 27000 premiers points

correspondant au traitement d'un octet de l'état. Dans le détail, le premier motif correspond au chargement des données d'entrée, le second à l'`AddRoundKey`, et le troisième au `SubBytes`. Les points de plus faible variance sont plus compliqués à identifier qu'il n'y paraît sur ce type de trace. En effet, à cause du très grand nombre de valeurs en abscisse, les points affichés sont sélectionnés par l'outil de visualisation. Par exemple, les valeurs des instants [0 ;27000] qui correspondent à une faible activité de l'appareil semblent être distribuées à peu près équitablement autour de deux valeurs principales, 1.3 et 2.7. La figure 3.10 montre la distribution et la trace d'écart-type de ces 27000 premiers points. Il en résulte que l'intuition précédente est fausse. La plupart des valeurs sont en fait distribuées autour de 1.5 et très peu à 2.7 en réalité.

La figure 3.11 correspond à la distribution de la trace d'écart-type présentée dans la figure 3.9. On peut y voir qu'une grande partie des valeurs se situent aux alentours de $\sigma = 1.8$. Cette valeur peut être raisonnablement assimilée au bruit de mesure, ce qui nous sera utile plus tard dans la détermination des valeurs des poids de Hamming des variables.

Considérant la donnée manipulée comme une variable aléatoire X, notre modèle de fuite donne :

$$L(x) = \alpha \cdot \mathrm{HW}(X) + \beta$$

En supposant l'indépendance entre le bruit et la donnée, la variance de la fuite s'exprime de la manière suivante :

$$\mathrm{Var}(L(X)) = \mathrm{Var}(\alpha \cdot \mathrm{HW}(X) + \beta) + \mathrm{Var}(\omega) \tag{3.5}$$

$$= \alpha^2 \cdot \mathrm{Var}(\mathrm{HW}(X)) + \mathrm{Var}(\omega) \tag{3.6}$$

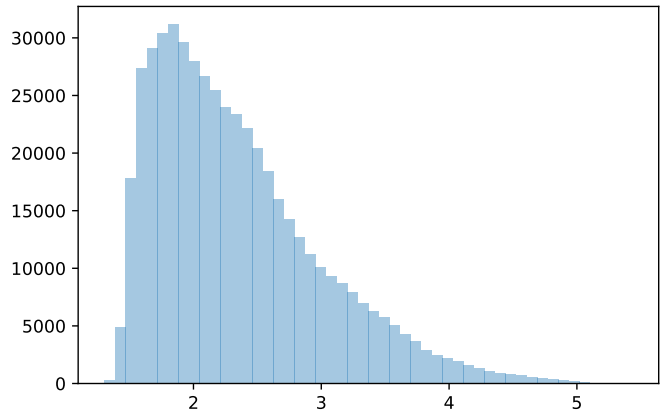


FIGURE 3.11 – Distribution de l'écart-type d'un début de tour d'AES

Si la donnée manipulée est un octet aléatoire issu d'une distribution uniforme, alors la variance de son poids de Hamming est 8 fois la variance d'un seul bit distribué lui-même de manière uniforme. La variance d'un bit étant $\frac{1}{2}\sum_{i=0}^{1}(i-\frac{1}{2})^2 = \frac{1}{4}$, la variance du poids de Hamming d'un octet est donc 2. Grâce à cette valeur, il devient possible de calculer :

$$\alpha = \pm\sqrt{\frac{\mathrm{Var}(L(X)) - \mathrm{Var}(\omega)}{2}} \tag{3.7}$$

La variance de la fuite au point d'intérêt est connue, et la variance du bruit peut être estimée comme expliqué précédemment. Une fois que α est calculé, β s'en déduit par :

$$\begin{aligned}\beta &= \mathrm{E}(L(X)) - \alpha \cdot \mathrm{E}(\mathrm{HW}(X)) \\ &= \mathrm{E}(L(X)) - 4\alpha\end{aligned}$$

où $\mathrm{E}(L(X))$ est approximée par la consommation moyenne en l'instant considéré.

Ayant connaissance des paramètres du modèle de consommation, le poids de Hamming h lié à une consommation l est calculé grâce à la formule suivante :

$$h = \frac{l-\beta}{\alpha}$$

L'avantage que cette technique apporte comparée à la méthode des couches est sa précision, permettant des calculs de réels plutôt que d'entiers. Nous proposons de comparer l'efficacité de ces deux techniques.

3.2.2 Comparaisons en simulation

Dans cette partie nous étudions l'efficacité des deux méthodes de calcul de poids de Hamming dans les cas d'utilisations de plusieurs distances et du maximum de vraisemblance grâce à des

simulations. Pour cela nous avons implémenté les deux méthodes de calcul des poids de Hamming correspondant à l'étape 3, ainsi que trois distingueurs de type distance, et un de type maximum de vraisemblance, correspondant à l'étape 4. Nous avons ensuite construit les attaques utilisant les couches et les distances, puis celles utilisant les couches et le maximum de vraisemblance, et enfin celles utilisant la variance et le maximum de vraisemblance. Nous ne montrons pas les résultats des attaques utilisant la variance et les distances puisque la précision apportée par la technique de la variance ne peut pas être correctement exploitée par les distances. En simulations, l'étape consistant à récupérer les PoIs n'est pas nécessaire, les attaques utilisent deux points qui sont bien liés à la donnée manipulée. Un run correspond à une attaque. À chaque run, un triplet (α, β, k) est tiré au hasard, k étant la valeur à récupérer. Ce triplet reste donc le même au cours d'une attaque, mais changera lors de la suivante. Le début de l'attaque consiste en le tirage d'une série de messages $\mathbf{m}$, puis les valeurs $\mathbf{h} = \mathrm{HW}(\mathrm{SB}(\mathbf{m} \oplus k))$ sont calculées. Ces valeurs sont ensuite bruitées telles que $\mathbf{h}_\sigma = \mathbf{h} + \boldsymbol{\omega}$ où $\boldsymbol{\omega}$ est une série de valeurs tirées d'une loi gaussienne centrée d'écart-type σ. Les mesures sont bruitées avant l'application du modèle de consommation afin que le bruit soit exprimé en bit et non en consommation, ce qui rend les runs comparables. Dans le cas contraire, l'erreur d'observation serait d'autant plus petite que α est grand. Enfin, nous transformons ces poids de Hamming en consommation : $\mathbf{l} = \alpha \cdot \mathbf{h}_\sigma + \beta$.

La figure 3.12 présente les résultats moyennés sur 10000 runs obtenus pour les différentes attaques avec un bruit d'écart-type $\sigma = 1.0$. L'ordonnée représente le rang de la clé et l'abscisse le nombre de traces utilisées. Le rang de la clé correspond à sa position dans le vecteur rangé des scores. Le rang 0 est donc synonyme d'avoir identifié la bonne valeur de k. Le rang diminuant lorsque plus de traces sont utilisées, on en conclut que l'attaque fonctionne. Les trois courbes les plus hautes, donc représentant les attaques les moins efficaces, correspondent aux attaques combinant couches et distances. Ces distances sont le χ^2, l'Inner Product (IP) et la Distance euclidienne (ED), trois distances parmi les plus efficaces d'après les auteurs originaux. Les deux courbes du dessous en rouge utilisent le maximum de vraisemblance avec, pour l'une la méthode des couches, et pour l'autre la méthode de la variance. Nous constatons que le maximum de vraisemblance est globalement meilleur que les distances. Nous attirons l'attention du lecteur sur le fait que cela n'est vrai que si l'approximation du bruit est bonne. En effet, le calcul du maximum de vraisemblance est basé sur le calcul de la probabilité du bruit. Une mauvaise appréciation de ce dernier résulte en une attaque peu efficace. Dans ces simulations, le bruit est approximé de la même manière qu'il est généré. Nous sommes donc dans des conditions optimales. Nous pouvons aussi voir que la méthode de calcul des poids de Hamming par la variance donne de meilleurs résultats que la méthode des couches avec le maximum de vraisemblance. Ces simulations montrent que le maximum de vraisemblance est globalement meilleur que les distances puisque plus efficace à calcul de poids de Hamming identique, et qu'il peut aussi profiter de la plus grande précision d'autres méthodes.

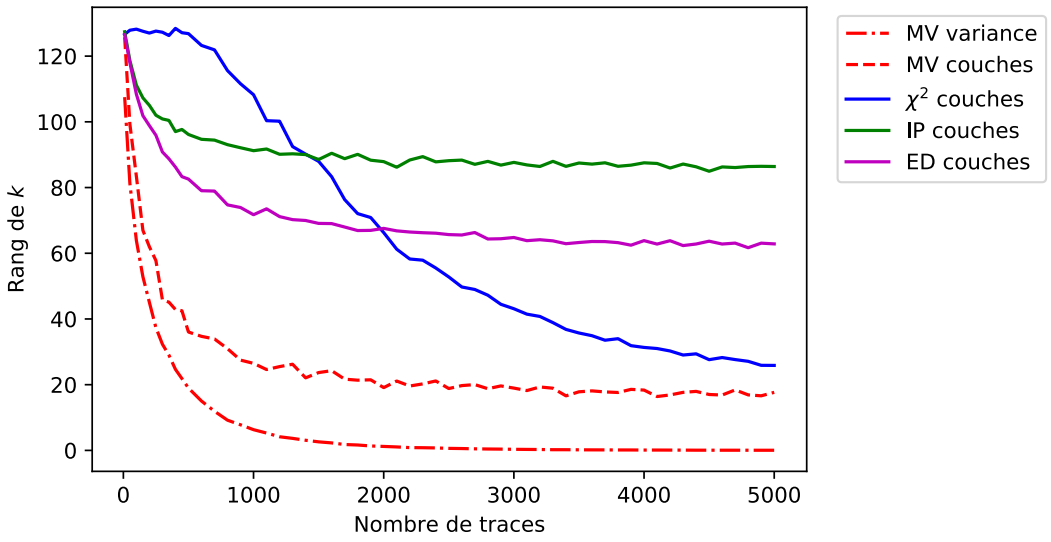


FIGURE 3.12 – Rang de k sur 10000 runs pour différentes attaques en fonction du nombre de traces ($\sigma = 1.0$)

3.2.3 Un autre point d'intérêt : l'entrée de S-box

La partie 3.1.1 montre qu'il est aussi possible de construire les distributions jointes des valeurs d'entrée et de sortie de `AddRoundKey`, respectivement notées m et x. Bien que cela ne permette que de déduire le poids de Hamming de k, nous avons quand même décidé d'étudier l'efficacité d'une telle attaque. La figure 3.13 présente la distance euclidienne entre chacune des distributions jointes de m et x de manière analogue à la figure 3.4. L'échelle permet de constater que les valeurs sont ici beaucoup plus grandes et que les modèles sont donc bien plus distincts. Les modèles n'étant qu'au nombre de neuf, et étant bien plus distincts, l'efficacité de l'attaque est bien plus élevée. En suivant le même principe que dans le paragraphe précédent, nous avons simulé l'attaque utilisant la méthode de calcul des poids de Hamming par variance, et le maximum de vraisemblance comme distingueur, pour différents niveaux de bruit. Les résultats moyennés sur 1000 runs sont visibles sur la figure 3.14. Sans surprise, l'attaque est très efficace et permet de déterminer le Poids de Hamming de k avec certitude en très peu de traces. Nous répondrons à la question de l'exploitation d'une telle information plus tard dans ce document.

3.2.4 Plus de deux variables

Les exemples précédents utilisent les fuites de m et de x ou y. Dans le cas où un appareil fuit sur ces trois variables, il peut être intéressant de tirer profit de toute l'information disponible. Il se peut aussi que plusieurs instants fuitent vis-à-vis de la même variable, et il peut être intéressant

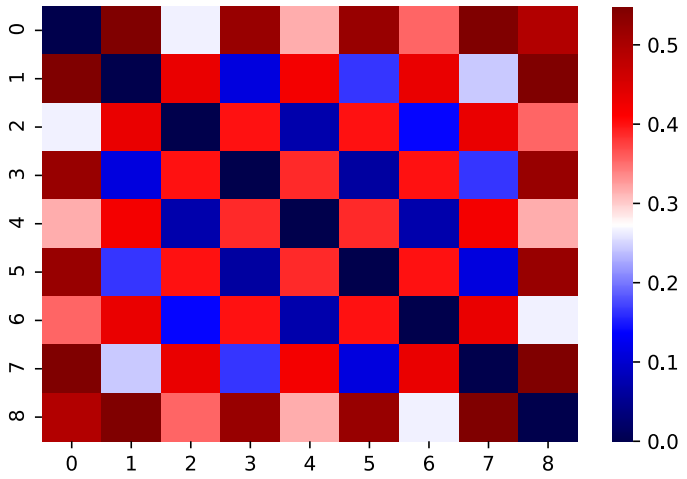


FIGURE 3.13 – Distance euclidienne entre les distributions jointes de m et x

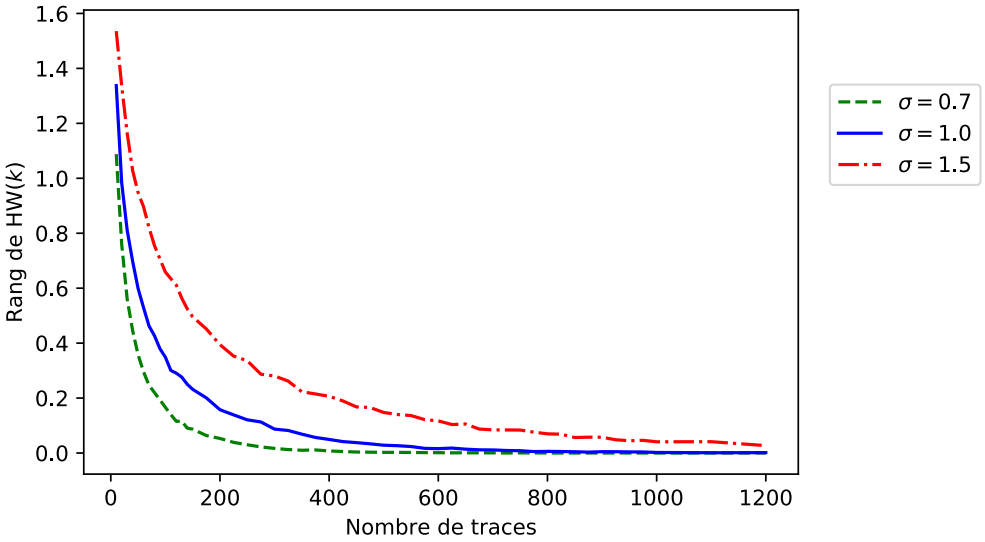


FIGURE 3.14 – Rang de HW(k) pour différents niveaux de bruit

d'en tirer parti.

Observations de m y et y

Ici nous nous intéressons au cas où l'appareil fuit sur une/les variable(s) plusieurs fois dans la trace. Cela est possible notamment lorsque la variable est manipulée pendant plusieurs cycles d'horloges. Comme nous allons le voir, il est possible d'exploiter ces différents points afin d'améliorer l'efficacité de l'attaque. Toujours dans le cas où l'attaquant est capable de récupérer les différents instants liés aux variables visées, et de calculer les poids de Hamming à ces instants, considérer plusieurs fois une même variable diminue l'effet du bruit puisque cela correspond intrinsèquement à moyenner les observations bruitées. Pour incorporer ces nouvelles observations, le calcul du maximum de vraisemblance tel que présenté à l'équation 3.4 doit être légèrement modifié : il suffit d'ajouter les termes qui tiennent compte de la probabilité de chacune des nouvelles observations. Par exemple, si y fuit en deux instants, alors 3.4 devient :

$$\Pr(k \mid (h_1,h_2)) = \frac{\sum_{h_1^*,h_2^*}\left[\left(\frac{1}{\sigma_1\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_1-h_1^*}{\sigma_1}\right)^2}\right)\cdot\left(\frac{1}{\sigma_2\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_2-h_2^*}{\sigma_2}\right)^2}\right)\cdot\left(\frac{1}{\sigma_3\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_3-h_2^*}{\sigma_3}\right)^2}\right)\cdot\delta_k(h_1^*,h_2^*)\right]\cdot\Pr(k)}{\Pr((h_1,h_2))}$$

où h_3 est la seconde observation du poids de Hamming de y, et σ_3 le bruit en cet instant. Remarquons que h_3 est comparé à h_2^* (et non pas à h_3^*), le poids Hamming attendu de y sous l'hypothèse de clé k.

Des simulations nous ont permis de valider cette intuition, l'attaque étant meilleure lorsque deux observations de y sont faites. Ces résultats sont présentés à la figure 3.15.

Observation de m x y

Jusqu'à présent, les distributions jointes que nous avons construites comportent uniquement deux variables, et ne permettent pas d'exploiter pleinement des traces où plus de variables fuient. Pour résoudre ce problème, nous proposons de construire des distributions jointes de toutes les variables qu'il est possible d'identifier. La construction de telles distributions jointes est similaire à celles n'en comportant que deux. Prenons l'exemple des poids de Hamming de trois variables issues de fonctions $(\varphi_k^i)_{1\leq i\leq 3}$ appliquées à m et paramétrées par k. La distribution jointe de ces trois variables est :

$$\delta_{(\varphi_k^1,\varphi_k^2,\varphi_k^3)} = \{p_{(h_1,h_2,h_3)} \mid h_1 \in \mathrm{HW}(\varphi_k^1),\ h_2 \in \mathrm{HW}(\varphi_k^2),\ h_3 \in \mathrm{HW}(\varphi_k^3)\}$$
$$p_{(h_1,h_2,h_3)} = \{\#m \mid m \in \mathrm{GF}(2^8),\ \left(\mathrm{HW}(\varphi_k^1(m)),\mathrm{HW}(\varphi_k^2(m)),\mathrm{HW}(\varphi_k^3(m))\right) = (h_1,h_2,h_3)\}$$

où $\delta_k(h_1,h_2,h_3)$ est la probabilité $p_{(h_1,h_2,h_3)}$ pour la clé k.

Pour illustrer l'utilisation de telles distributions jointes, nous proposons d'étudier le cas où m x et y sont observés conjointement. Après constructions des distributions, nous constatons qu'il en existe 256 différentes, une par hypothèse d'octet de clé. L'effet que cela a, comparé à l'utilisation de deux variables uniquement, est une plus grande distinguabilité : supposons deux couples de valeurs distinctes (m_1,y_1) et (m_2,y_2) dont les observations, c'est-à-dire les couples de poids de Hamming, sont indistinguables, à savoir $(\mathrm{HW}(m_1),\mathrm{HW}(y_1)) = (\mathrm{HW}(m_2),\mathrm{HW}(y_2))$. Grâce à l'observation

de x, il est possible que $(\mathrm{HW}(m_1),\mathrm{HW}(x_1),\mathrm{HW}(y_1)) \neq (\mathrm{HW}(m_2),\mathrm{HW}(x_2),\mathrm{HW}(y_2))$, ce qui rend les observations distinguables. Alors que deux couples étaient indistinguables auparavant, l'ajout d'une troisième variable peut, dans certain cas, permettre de distinguer les deux triplets. Les distributions sont alors plus distinctes les unes des autres, ce qui a généralement pour effet d'améliorer l'efficacité de l'attaque. Le maximum de vraisemblance a encore besoin d'être légèrement modifié pour être appliqué à de telles distributions : cette fois, le théorème des probabilités totales s'applique sur tous les triplets possibles des poids de Hamming de m x et y, ce qui ajoute un terme au calcul de la probabilité d'une observation, comme dans l'exemple où y fuit deux fois. Étant donné une observation (h_1,h_2,h_3), l'équation du maximum de vraisemblance s'écrit maintenant :

$$\Pr(k \mid (h_1,h_2,h_3)) = \frac{\sum_{h_1^*,h_2^*,h_3^*}\left[\left(\frac{1}{\sigma_1\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_1-h_1^*}{\sigma_1}\right)^2}\right)\cdot\left(\frac{1}{\sigma_2\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_2-h_2^*}{\sigma_2}\right)^2}\right)\cdot\left(\frac{1}{\sigma_3\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{h_3-h_3^*}{\sigma_3}\right)^2}\right)\cdot\delta_k(h_1^*,h_2^*,h_3^*)\right]\cdot\Pr(k)}{\Pr((h_1,h_2,h_3))} \tag{3.8}$$

Nous avons mené des simulations pour juger de l'efficacité de l'ajout de l'observation de la variable x. Les résultats sont visibles à la figure 3.15. Nous pouvons voir sans surprise que cette attaque est plus efficace que l'attaque n'observant que m et y, mais elle est aussi meilleure que lorsque y est observé à deux instants.

L'efficacité peut être grandement améliorée par l'ajout de variables. Cependant, cela implique un effort à l'attaquant pour identifier les PoI correspondants. L'efficacité de l'attaque est grandement fonction de la capacité de l'attaquant à identifier les PoIs puisque se tromper a des répercutions dramatiques. Il est préférable d'utiliser moins de variables dont les instants fuient vraiment plutôt que d'essayer d'en incorporer d'autres mal identifiées, au risque de rendre l'attaque complètement inutile. Un exemple d'attaque observant m x et y est présenté dans la figure 3.15. Cette fois-ci, l'attaquant s'est trompé de point d'intérêt pour l'observation de x, simulé par l'observation d'une valeur aléatoire. Dans ce cas, l'attaque n'aboutit plus du tout.

Combinaisons de variables

Il est en fait tout à fait possible d'utiliser d'autres variables que x et y (toujours en plus de m qui est obligatoire). Dans le cas de l'AES, des fuites peuvent, par exemple, survenir dans le `MixColumns`. Ce dernier manipule des multiples de y, à savoir $1 \bullet y$, $2 \bullet y$ et $3 \bullet y$ lors du chiffrement, et $e \bullet y$, $b \bullet y$, $d \bullet y$ et $9 \bullet y$ lors du déchiffrement. L'ajout d'une ou plusieurs de ces variables augmente l'efficacité de l'attaque. Cela peut être combiné à l'observation de plusieurs instants manipulant la même variable comme expliqué précédemment. Il est important de remarquer que l'ajout d'une variable n'a pas toujours le même impact sur l'efficacité d'une attaque. Généralement, l'ajout d'une observation d'une variable déjà présente a un impact moindre que l'ajout d'une variable qui n'est pas déjà exploitée. D'autre part, une nouvelle variable aura un impact d'autant plus grand que la distinguabilité apportée sera importante. Pour illustrer ce dernier point, prenons l'exemple

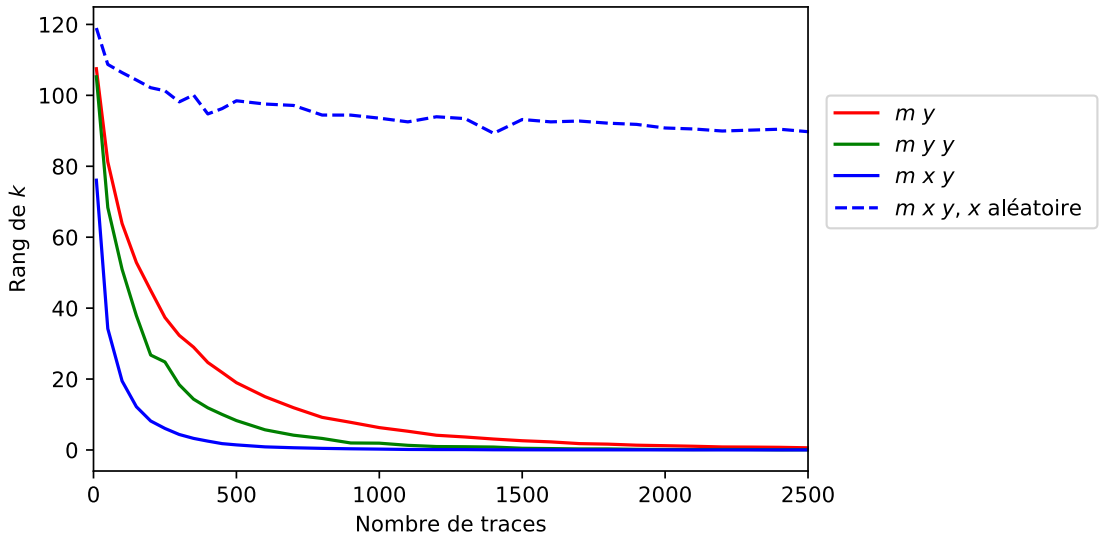


FIGURE 3.15 – Rang de k pour différentes attaques de plus de deux observations

de $2 \bullet y$. La distinguabilité apportée par $2 \bullet y$ à une attaque observant m et y est relativement restreinte puisque les deux variables y et $2 \bullet y$ partagent la moitié de leurs valeurs de poids de Hamming (Pour $y < 128$, $\mathrm{HW}(y) = \mathrm{HW}(2 \bullet y)$). $3 \bullet y$ sera par exemple un meilleur choix. Ajouter x s'avère encore plus efficace. La figure 3.16 présente le rang de k pour différentes attaques citées ci-dessus. Nous remarquons que l'observation de trois variables distinctes est plus efficace que deux, et que conjointement à m et y, l'observation de x est meilleure que $3 \bullet y$, elle-même légèrement meilleure que $2 \bullet y$.

3.3 Résultats expérimentaux

À notre connaissance, les attaques par distributions jointes n'avaient jamais été testées autrement qu'en simulation. Cette partie présente donc des résultats d'attaques expérimentales afin de valider leur faisabilité. Pour ce faire, nous avons implémenté un AES sans contre-mesure sur un microcontrôleur Arduino Uno 8 bits. Nous avons généré 1000 traces de consommation de courant d'un premier tour d'AES grâce à un oscilloscope digital Lecroy WaveRunner avec un taux d'échantillonnage de 5 GS/s. Les traces ne présentent pas de désynchronisation. Un simple traitement visant à centrer les consommations a été appliqué afin d'éliminer les éventuelles dérives de consommation au cours de la campagne d'acquisition, dérives qui peuvent être particulièrement importantes lorsque celle-ci est étalée dans le temps. Même si les traces résultent du premier tour du chiffrement, aucun avantage n'en a été retiré lors de l'attaque puisque les messages n'ont pas

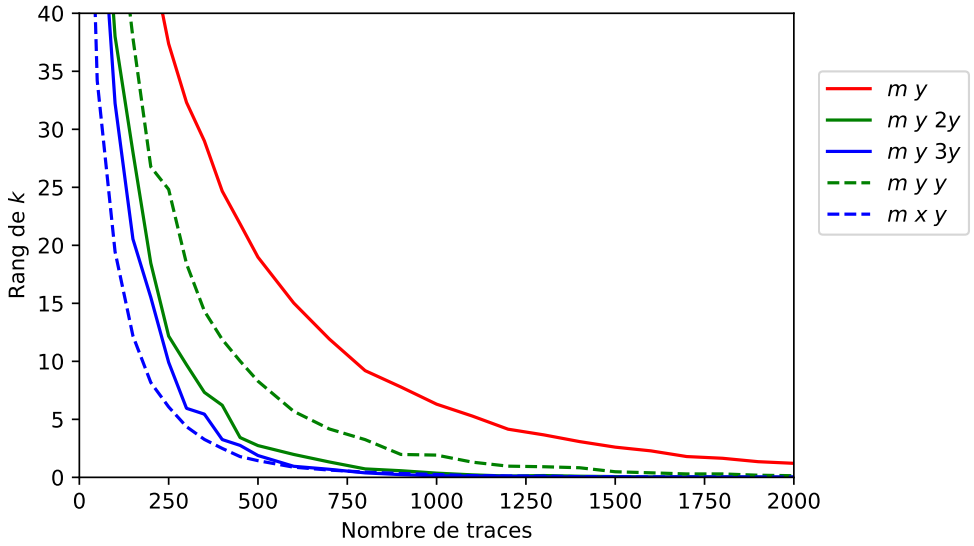


FIGURE 3.16 – Rang de k pour différentes attaques de plus de deux observations impliquant d'autres variables

été utilisés. Nous avons néanmoins profité de cette connaissance pour en déduire le sens de la pente de la fonction de consommation, comme il sera expliqué plus tard. L'Arduino présentant des fuites particulièrement prononcées, l'identification des points d'intérêt a été relativement facile. La trace d'écart-type correspondante est en fait celle présentée à la figure 3.9. Le point d'intérêt relatif à m pour l'octet i a été choisi comme le point maximisant la variance au sein du pic i du premier motif de 16 pics, aux alentours de l'instant 100000. Ceux relatifs à y ont été choisis de la même manière au sein du troisième motif, autour de l'instant 400000. Le second motif de 16 pics correspond à la manipulation de x, et le dernier correspond au début du `MixColumns`. Nous avons mené une attaque de type $m\ y$ sur chacun des 16 octets de clé.

Pour calculer les poids de Hamming en ces points, nous avons opté pour la méthode de la variance décrite en section 3.2.1. Grâce la distribution des valeurs de la courbe d'écart-type, nous avons approximé le bruit à une gaussienne centrée d'écart-type $\sigma_\omega = 2.0$. Cette valeur, ainsi que l'écart-type en chacun des points d'intérêt nous permettent de calculer un couple (α, β) pour chacun d'entre eux. Nous avons donc calculé 32 couples, 16 couples $((\alpha,\beta)_{m,i})_{1\leq i\leq 16}$ correspondant à autant de $(m_i)_{1\leq i\leq 16}$, et 16 couples $((\alpha,\beta)_{y,i})_{1\leq i\leq 16}$ correspondant à autant de $(y_i)_{1\leq i\leq 16}$. Nous sommes maintenant en mesure de calculer l'écart-type approximé du bruit en unité bit comme étant $\sigma = \frac{\sigma_\omega}{\alpha}$ en chacun des points. Notons que le calcul de α laisse la liberté du signe. Pour résoudre ce problème, nous avons dû étudier plus en détail le comportement de l'appareil. Grâce à une CPA, nous avons pu observer le comportement de la consommation en nos PoI. Le résultat est visible sur la figure 3.17. Nous remarquons que les pics de corrélation coïncident avec les pics d'écart-type, ce qui confirme la pertinence de notre sélection. Ensuite, nous remarquons que

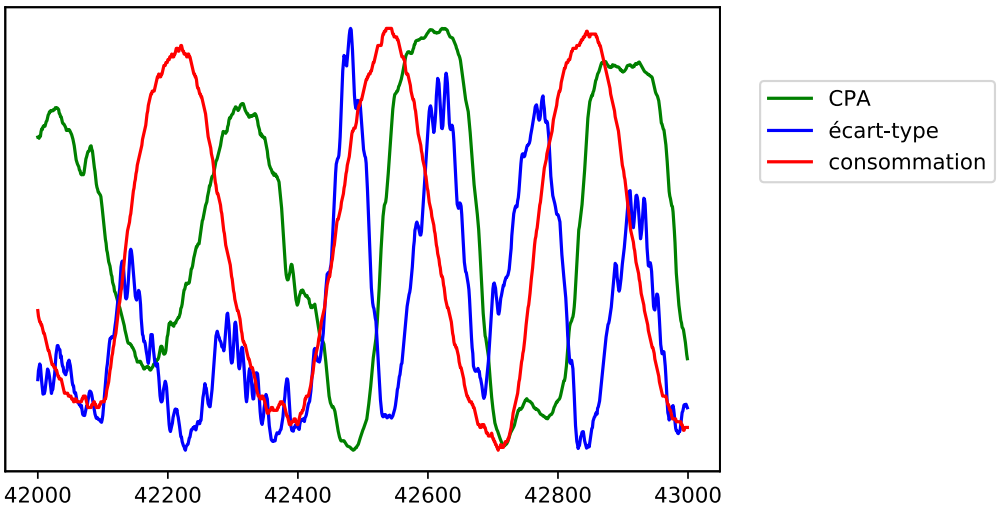


FIGURE 3.17 – Courbes de CPA sur m, d'écart-type et de consommation normalisées autour d'un point d'intérêt de m

les pics de corrélation coïncident au milieu de la pente de la consommation. Les pics de CPA sont positifs lors de la descente, et négatifs lors de la montée. Lors de nos essais, les valeurs les plus élevées d'écart-type étaient systématiquement sur des phases ascendantes, Le signe de la corrélation étant le sens de variation de la fonction de consommation, nous en avons déduit que le signe de α était toujours négatif.

Rappelons que cette étape de caractérisation visant à trouver le signe de α peut potentiellement être menée indépendamment de l'attaque, et sur d'autres jeux de courbes. Il est aussi possible de mener l'attaque avec chacun des signes. L'attaque en elle-même ne tire pas d'avantage de la connaissance des messages.

Suite à cela, nous avons attaqué l'ensemble des 16 octets de clé en utilisant le maximum de vraisemblance. Afin de le comparer aux distances, nous avons aussi utilisé l'Inner Product et la Distance euclidienne sur les observations arrondies. Les rangs des différents octets de clé pour les trois attaques sont exposés dans la table 3.1.

D'une part, il est clairement visible que l'attaque par distributions jointes fonctionne puisque près de la moitié des octets de clé sont retrouvés ou apparaissent dans les toutes premières positions. L'attaque trouve donc un intérêt à être utilisée dans des cas pratiques. D'autre part, nous remarquons que le maximum de vraisemblance est bien plus efficace que les distances dans cet exemple, et ce parfois de manière spectaculaire, confirmant l'intérêt d'utiliser cette méthode.

TABLE 3.1 – Rang des 16 octets de clé d'une attaque *m y* à clair inconnu d'un AES non protégé (1000 traces)

Numéro d'octet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Inner Product	**0**	**0**	167	29	45	187	192	45	77	108	36	124	5	104	64	147
euclidienne	1	80	210	106	3	62	186	17	38	68	194	48	27	120	21	116
MV	**0**	2	**6**	**1**	**1**	**17**	**1**	**0**	**1**	**19**	**5**	**15**	**4**	**40**	**19**	**13**

Jusqu'alors, cette attaque a été appliquée à l'ordre 1, c'est-à-dire dans le cadre d'implémentations non protégées. Dans la partie suivante nous nous intéressons à des implémentations protégées par masquage booléen d'ordre 1.

3.4 Attaques simples par distributions jointes d'implémentations masquées à l'ordre 1

Pour se protéger des attaques par canaux cachés, des contre-mesures sont souvent mises en place sur les appareils utilisant de la cryptographie. Comme nous l'avons expliqué dans le chapitre 1.5.2, une des plus communes est la randomisation par masquage booléen. Nous posons ici la question de comment attaquer des implémentations protégées par une telle contre-mesure à l'ordre 1. Tout d'abord nous rappelons rapidement quelques notions du masquage booléen et donnons quelques exemples. Nous étudions ensuite des moyens d'outrepasser cette contre-mesure.

3.4.1 Masquage booléen

Nous proposons au lecteur de se référer à la partie 1.5.3. Voici néanmoins un petit rappel.

Le masquage booléen d'ordre d consiste à "diviser" une variable x en $d+1$ shares tel que $x = \bigoplus_{i=1}^{d+1} x_i$. Nous appellerons $\mathbf{x} = \{x_1, ..., x_{d+1}\}$ le sharing d'une variable x. Une implémentation non masquée, dite d'ordre un, manipule une variable en une seule share. Une implémentation masquée à l'ordre 1, dite d'ordre 2, manipule une variable divisée en deux shares tel que $x = x_1 \oplus x_2$. Dans cette section, nous aborderons le problème du masquage d'ordre 1 uniquement. La sécurité repose sur l'uniformité du tirage aléatoire des masques, ce qui a pour effet de rendre l'observation de jusqu'à d shares parmi les $d+1$ de $\mathbf{x}$ inutile puisque la distribution de celles-ci est équiprobable pour toute valeur de x. Un exemple d'implémentation masquée d'ordre 1 consiste à tirer uniformément un masque aléatoire r puis de construire $\mathbf{x}$ tel que $\mathbf{x} = \{x \oplus r, r\}$. Dans ce cas, x ne peut pas être déduite par l'observation d'une seule share. Un algorithme protégé par masquage booléen peut consister, de manière simple, à masquer, c'est-à-dire diviser le message en deux

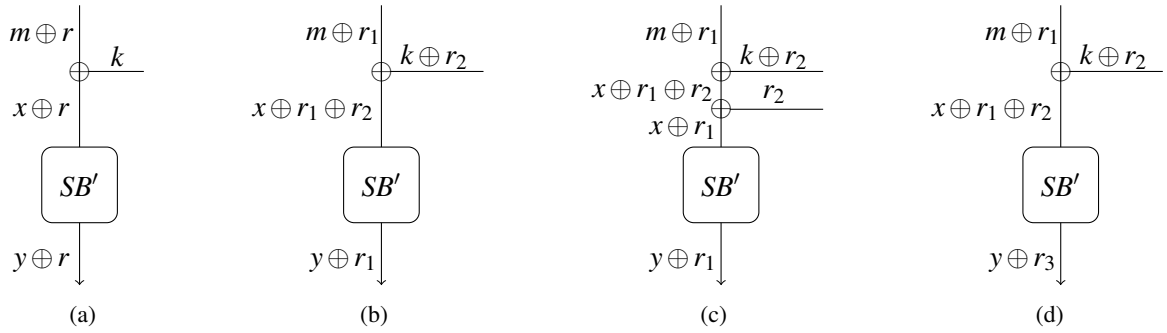


FIGURE 3.18 – Différents schémas de masquage d'ordre 1

shares grâce à une valeur aléatoire tirée à chaque exécution, effectuer les calculs sur l'une (ou les deux de manière indépendante), puis de démasquer, c'est-à-dire reconstruire la donnée. Le but est de faire en sorte que toute donnée manipulée par l'algorithme, en dehors du message et du chiffré, soit protégée à chaque instant par une valeur aléatoire différente à chaque exécution.

Dans ce cas de figure, les attaques par canaux cachés d'ordre un ne fonctionnent plus puisqu'elles se concentrent sur l'étude d'un seul instant temporel, et donc d'une seule share, dans le cadre de fuites multivariées tout du moins.

3.4.2 Attaque d'implémentations masquées à l'ordre 1

Comme nous avons pu le voir dans le chapitre 1.7, il existe des moyens d'attaquer de telles implémentations : les attaques d'ordre supérieur qui combinent plusieurs instants. Par exemple, la CPA second ordre consiste à effectuer une CPA classique sur une combinaison de chacun des couples de consommation de la trace.

Les attaques par distributions jointes sont, elles aussi, inefficaces lorsqu'elles sont appliquées à des implémentations masquées : l'attaquant observe la distributions des poids de Hamming $(\mathrm{HW}(m'), \mathrm{HW}(y'))$ de valeurs masquées, distribution différente de celle des valeurs non masquées, et que l'on peut même imaginer complètement uniforme.

La figure 3.18 présente différents schémas de masquages d'ordre 1 possibles lors du premier tour d'un AES. Nous y retrouvons les trois variables m x et y qui sont, cette fois, masquées.

La sous-figure 3.18a présente le schéma le plus simple où chacune des variables est masquée par le même masque tout au long de l'algorithme. La 3.18b présente un schéma où k est aussi masqué, et SB$'$ construit de manière à ce que son masque de sortie soit celui d'entrée de tour. La 3.18c est un peu similaire, à la différence d'un démasquage explicite avant l'entrée de SB$'$. Enfin, 3.18d

présente un schéma où m x et y sont masqués par trois masques différents. Les figures 3.18b et 3.18d utilisent une S-box construite de manière à ce que le masque d'entrée r_e soit différent du masque de sortie r_s. Cela est rendu possible par le calcul de SB′ tel que $\mathrm{SB}'(x) = \mathrm{SB}(x \oplus r_e) \oplus r_s$.

Particularité de l'attaque m x

Nous exposons ici un comportement tout particulier aux distributions jointes de m x. Dans le premier et troisième schémas, m et x sont masqués par le même masque, $m' = m \oplus r$ et $x' = m \oplus k \oplus r$. Rappelons que la distribution jointe s'exprime comme la distribution du nombre d'occurrences de chacun des couples (h_m, h_x) où h_m et h_x sont les poids de Hamming de m et x respectivement. Soit $\mathbf{c} = \{(m_1, x_1), ..., (m_{2^n}, x_{2^n})\}$ l'ensemble ordonné des couples des valeurs de m et de $x = m \oplus k$ dans $\mathrm{GF}(2^n)$ pour une valeur de clé k. Soit le masque $r \in \mathrm{GF}(2^n)$, alors $\mathbf{c} \oplus r = \{(m_1 \oplus r, x_1 \oplus r), ..., (m_{2^n} \oplus r, x_{2^n} \oplus r)\}$ est une permutation de $\mathbf{c}$. L'ensemble non ordonné des couples est inchangé et ce quelle que soit la valeur de r. La distribution jointe globale de (m', x') sous toutes les hypothèses de masques, qui est la somme de toutes ces distributions, est donc identique à celle de (m, x). Le raisonnement est identique lorsque l'on considère les distributions jointes de poids de Hamming. La conclusion que l'on peut tirer de ce raisonnement est que masquer m et x par la même valeur ne modifie en rien les distributions jointes des poids de Hamming, et qu'ainsi l'attaque portant sur m et x n'est donc pas affectée par ce type de masquage.

Observation de m et y masqués

Ici nous proposons une variante de l'attaque utilisant l'entrée de `AddRoundKey` et la sortie de `SubBytes` applicable à des implémentations masquées à l'ordre 1. Toute configuration n'est pas systématiquement vulnérable à cette proposition, et afin d'illustrer nos propos, nous nous intéressons d'abord aux trois premiers schémas de la figure 3.18. Leur particularité est de masquer les deux variables m et y par la même valeur de masque.

Nous étudions ici la possibilité de construire les distributions jointes des poids de Hamming des valeurs masquées m' et y'. Cette distribution δ'_k correspond à la somme des distributions sous chacune des hypothèses de masque, pondérées par la probabilité de la valeur de ce même masque :

$$\delta'_k = \sum_r \Pr(r)\delta'_{k,r}$$

où $\delta'_{k,r}$ est la distribution des poids de Hamming des valeurs masquées m' et y' correspondant au masque r. Cette dernière se calcule de la même manière que pour les données non masquées, à

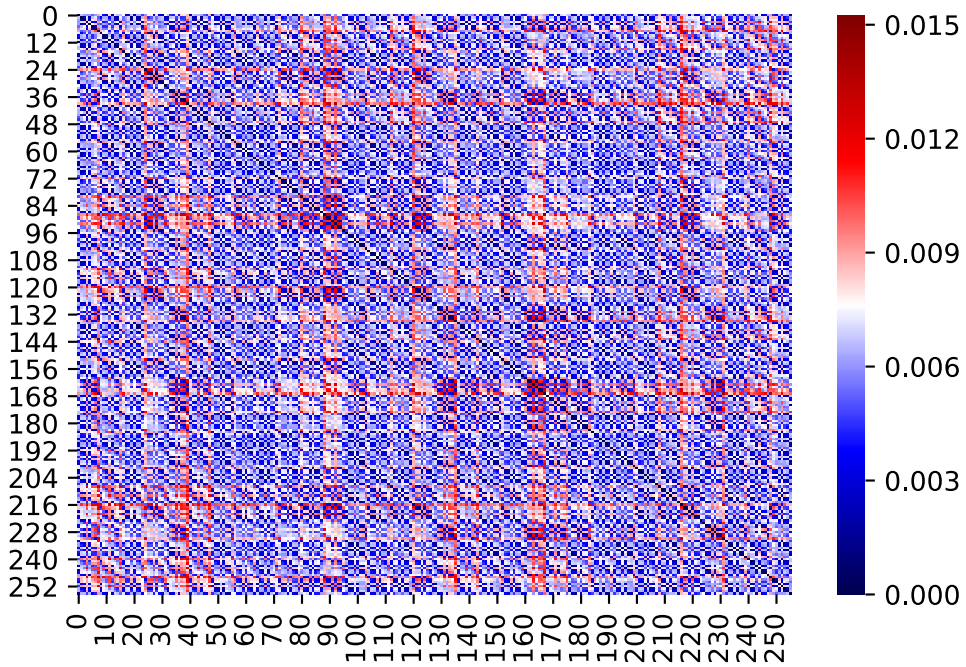


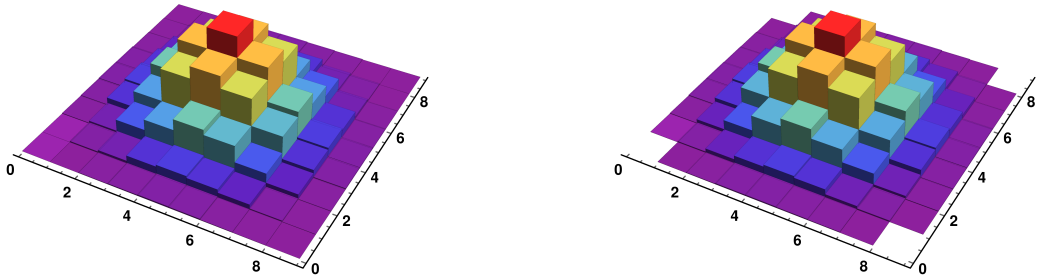
FIGURE 3.19 – Distance euclidienne entre les distributions jointes de m et y masqués par le même masque

l'ajout prés de la valeur de r :

$$\delta_{k,r} = \{p_{(h_1,h_2)}\}$$
$$p_{(h_1,h_2)} = \{\#m \mid m \in \mathrm{GF}(2^8),\ (\mathrm{HW}(m \oplus r), \mathrm{HW}(\mathrm{SB}(m \oplus k) \oplus r)) = (h_1,h_2)\}$$

Pour que l'attaque soit possible, il est nécessaire que les distributions jointes ne soient pas toutes égales pour chacune des hypothèses de clé. Pour répondre à cette question, nous avons calculé ces distributions jointes pour chacune des clés k dans le cas de l'AES. Il s'avère que les distributions sont effectivement toutes différentes. La figure 3.19 présente la distance euclidienne entre chacune. En la comparant à la figure 3.4, nous constatons que l'échelle est beaucoup plus petite, traduisant des différences beaucoup moins marquées. Cela explique en partie, comme nous allons le voir, que l'attaque est moins efficace qu'à l'ordre 1. La figure 3.20 présente les histogrammes de deux distributions jointes et permet de visualiser cela : les différences, qui étaient nettement visibles en figure 3.5, le sont beaucoup moins maintenant.

Contrairement à l'intuition qu'était la nôtre à la base, les distributions ne sont pas toutes équivalentes du fait de la non indépendance (et même ici l'égalité) des masques des deux variables. Comme nous allons le voir, cela permet l'application de l'attaque à ces cas de figure (résultats



(a) $k = \texttt{0x27}$ (b) $k = \texttt{0x7e}$

FIGURE 3.20 – Distributions jointes des HW de m et y

présentés en figure 3.21), et même plus généralement lorsque les deux variables ciblées sont masquées par des valeurs qui ne sont pas indépendantes. Au contraire, le schéma 3.18d utilise des masques indépendants sur m et y. Dans ce cas, les distributions sont effectivement toutes identiques et correspondent toutes à une distribution jointe de poids de Hamming de valeurs tirées aléatoirement. Il n'est donc pas possible d'attaquer un tel schéma de cette manière.
Notons que les autres étapes de l'attaque ne nécessitent aucune modification.

Observations d'autres variables

Déjà applicable au cas non masqué, l'utilisation de plus de variables et PoI est aussi possible dans le contexte d'implémentation masquée, et améliore aussi l'efficacité de l'attaque. Cependant, les contraintes liées à la non indépendance des masques sur deux variables s'appliquent toujours. De manière générale, les variables considérées doivent toutes être masquées par des valeurs non indépendantes entre elles. Il est par exemple possible d'attaquer une implémentation masquée au premier ordre grâce aux variables $m\ x$ et y dans les schémas 3.18a et 3.18c. À l'inverse, les deux autres schémas ne le permettent pas, le masque de x n'étant pas dépendant de celui de m et y. Dans le cas où l'observation de plus de variables est possible, les conclusions sont les mêmes qu'auparavant, à savoir l'amélioration de l'efficacité de l'attaque, ce que nous allons voir en simulations.

Simulations

Afin de valider le principe de l'attaque sur implémentations masquées à l'ordre 1, nous avons mené des simulations de la même manière que lors de l'attaque sur implémentations non masquée. Nous avons mené l'attaque à la fois sur $m\ y$ et sur $m\ x\ y$, dans le cas où les masques sont égaux pour toutes les variables, mais tirés aléatoirement à chaque nouvelle exécution. Lors de chaque run, correspondant à une attaque, une nouvelle valeur k est tirée. Lors de celui-ci, une trace

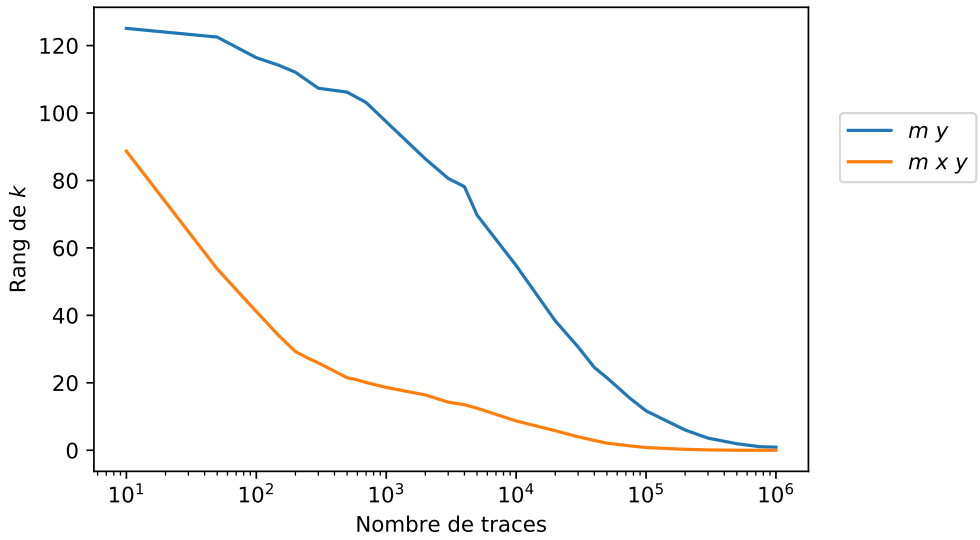


FIGURE 3.21 – Rang de k pour une implémentation masquée au premier ordre (1000 runs)

correspond au tirage aléatoire de m et du masque r. Les valeurs $m' = m \oplus r$, $x' = m \oplus k \oplus r$ et $y' = \mathrm{SB}(m \oplus k) \oplus r$ sont ensuite calculées, et leurs poids de Hamming bruités par un bruit gaussien centré d'écart-type $\sigma = 1.0$ forment l'observation. Nous avons ensuite utilisé le maximum de vraisemblance. Les résultats présentés sur la figure 3.21 sont issus d'une moyenne sur 1000 runs.

Dans nos simulations, le nombre de traces requises pour retrouver la valeur de k est de l'ordre du million pour l'attaque par observation des variables m et y. Ce nombre diminue énormément dès lors que x est aussi observé. Ce nombre est donc bien supérieur à celui requis lors de l'attaque d'une implémentation non masquée, mais la contre-mesure n'efface donc pas complètement la vulnérabilité de l'algorithme aux attaques par distributions jointes, tout du moins lorsque les masques ne sont pas indépendants sur chacune des variables visées. Afin d'aller plus loin, la partie suivante concerne des attaques applicables au schéma 3.18d, schéma non vulnérable à l'attaque décrite au-dessus.

3.5 Attaques quadrivariées

Dans cette partie, nous proposons d'aller plus loin dans l'étude des attaques sur implémentations masquées d'ordre 1 en solutionnant le problème où les variables visées sont masquées par des valeurs indépendantes, ce qui, jusqu'à présent, rend l'attaque présentée impossible.

3.5.1 Distributions quadrivariées

Des schémas de masquage tels que la figure 3.18d empêchent l'application de l'attaque telle que présentée jusqu'à présent puisque, par exemple, m et y sont masqués par deux valeurs de masques indépendantes. Notons que le tirage de valeurs aléatoires utilisées comme masques est en fait une opération coûteuse en milieu contraint, ce qui amène les implémenteurs à limiter ce nombre de tirages, ce qui peut en partie expliquer l'existence des schémas utilisant plusieurs fois le même masque. Une méthode classique consiste à utiliser les mêmes valeurs d'une position à une autre. Rappelons par exemple le cas de l'AES. L'AES met à jour un état constitué de 16 octets en y appliquant les quatre fonctions que sont l'`AddRoundKey`, le `SubBytes`, le `ShiftRows` et le `MixColumns`. Les trois premières sont appliquées indépendamment sur chacun des 16 octets. Un schéma de la figure 3.18 ne présente l'`AddRoundKey`, le `SubBytes` que d'un seul octet. Une méthode classique permettant d'économiser des tirages de valeurs consiste par exemple à tirer les valeurs nécessaires au masquage d'une position, puis de les réutiliser sur les 15 autres. Prises indépendamment, chacune des positions du schéma 3.18d est donc protégée, et ce avec le minimum de tirages possible.
Nous allons voir qu'il est tout de même possible de monter une attaque récupérant la clé dans le cas où les variables sont masquées par des valeurs indépendantes, mais que des astuces d'économie consistant en la réutilisation des masques à d'autres endroits ont été utilisées. Nous proposons pour cela la construction de nouvelles distributions jointes, les distributions que nous appellerons quadrivariées du fait qu'elles utilisent au minimum quatre variables, et cela de manière un peu particulière.

Quelques notations et termes avant d'aller plus loin. Nous utilisons l'exemple de l'AES. Le terme indice est utilisé pour désigner l'indice de position dans l'état, et il en existe 16. La clé étendue comporte 176 octets, et à chacun est associé un schéma 3.3. Par abus de langage, et afin de ne pas alourdir la lecture, nous utiliserons le terme position (ou position d'octet d'état/de clé) pour faire référence au traitement d'un de ces 176 octets, et donc l'une de ces 176 portions d'algorithme. Une position est définie par un numéro de tour, et un indice de position dans l'état. Les masques utilisés lors du traitement d'une position sont regroupés sous le terme jeu de masques. Le schéma 3.18d implique par exemple un jeu de masque $R = (r_1, r_1 \oplus r_2, r_3)$. Deux jeux de masques sont égaux si le masques sont égaux deux à deux.

Soient a et b deux positions d'octets d'état (ces deux positions sont donc deux couples (ix, rd) quelconques parmi les 176 où ix désigne l'indice et rd le tour), et k_a et k_b les octets de clé correspondants. Les variables considérées sont respectivement m_a, x_a, y_a pour la première position, et m_b, x_b, y_b pour la seconde. Considérons une implémentation masquée où le jeu de masques $R = (r_1, r_2, r_3)$ est le même pour ces deux positions, c'est-à-dire $m'_i = m_i \oplus r_1$, $x'_i = m_i \oplus k_i \oplus r_2$ et $y'_i = \mathrm{SB}(m_i \oplus k_i) \oplus r_3$ avec $i \in \{a, b\}$. Pour un couple (k_a, k_b) donné, nous proposons de construire

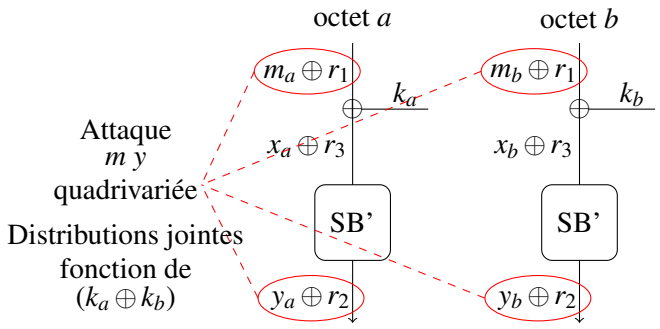


FIGURE 3.22 – Variables utilisées lors d'une attaque m y quadrivariée afin de retrouver $k_a \oplus k_b$

la distribution jointe $\delta_{(k_a,k_b)}$ des poids de Hamming des m'_i et y'_i, c'est-à-dire :

$$\begin{aligned}
\delta_{(k_a,k_b)} =& \{p_{\left(h_{m'_a},h_{y'_a},h_{m'_b},h_{y'_b}\right)}\} \\
p_{\left(h_{(a,m')},h_{(a,y')},h_{(b,m')},h_{(b,y')}\right)} =& \{\#(m_a,m_b) \mid m_a,m_b,r_1,r_3 \in \mathrm{GF}(2^8), \\
& \left(\mathrm{HW}(m'_a),\mathrm{HW}(y'_a),\mathrm{HW}(m'_b),\mathrm{HW}(y'_b)\right) = (h_{m'_a},h_{y'_a},h_{m'_b},h_{y'_b})\}
\end{aligned}$$

La figure 3.22 montre les variables dont les poids de Hamming sont utilisés dans la construction de cette distribution.

Nous avons calculé ces distributions jointes, et il en résulte que les 256 distributions sont différentes, chacune correspondant à une valeur de $k_a \oplus k_b$.
Démonstration : pour tout quadruplet de valeurs (m_a,m_b,r_1,r_3) et un couple d'octets de clé (k_a,k_b) est associé un couple $(y'_a = \mathrm{SB}(m_a \oplus k_a) \oplus r_3), y'_b = \mathrm{SB}(m_b \oplus k_b) \oplus r_3)$, et le quadruplet (m'_a,m'_b,y'_a,y'_b) où $m'_a = m_a \oplus r_1$ et $m'_b = m_b \oplus r_1$ est observé. Considérons maintenant le quadruplet de valeurs $(m^2_a = m_a \oplus x, m^2_b = m_b \oplus x, r^2_1 = r_1 \oplus x, r^2_3 = r_3 \oplus x)$. S l'on choisi $(k^2_a = k_a \oplus x, k^2_b = k_b \oplus x)$ comme couple d'octets de clé, alors le quadruplet d'observations

$$\begin{aligned}
(m'^2_a,m'^2_b,y'^2_a,y'^2_b) &= (m_a \oplus x \oplus r_1 \oplus x, m_b \oplus x \oplus r_1 \oplus x, \mathrm{SB}(m_a \oplus x \oplus k_a \oplus x) \oplus r_3, \mathrm{SB}(m_b \oplus x \oplus k_b \oplus x) \oplus r_3) \\
&= (m_a \oplus r_1, m_b \oplus r_1, \mathrm{SB}(m_a \oplus k_a) \oplus r_3, \mathrm{SB}(m_b \oplus k_b) \oplus r_3) \\
&= (m'_a,m'_{b_i},y'_a,y'_b)
\end{aligned}$$

est donc le même que le précédent. Pour un couple d'octets de clé (k_a,k_b) et un quadruplet (m_a,m_b,r_1,r_3), il existe alors exactement un quadruplet $(m_a \oplus x, m_b \oplus x, r_1 \oplus x, r_3 \oplus x)$ résultant en la même observation pour le couple d'octets de clé $(k_a \oplus x, k_b \oplus x)$. Il en résulte que les distributions jointes issues des poids de Hamming des valeurs masquées de m et de y sur deux positions d'octets d'état a et b sont invariantes à valeur de $k_a \oplus k_b$ fixée. C'est donc cette valeur qui sera déterminée lors d'une attaque quadrivariée.

Grâce à ces nouvelles distributions, il est possible de discriminer le xor de deux octets de clé même en présence d'un schéma de masquage fort où les deux variables d'une même position sont masquées par des valeurs indépendantes, avec pour seule condition que les jeux de masques soient identiques. Les autres étapes de l'attaque ne changent pas, seul le maximum de vraisemblance est modifié de sorte à être appliqué sur quatre variables.

Remarquons que les octets a et b n'ont pas comme contrainte d'être proches par la position ou le tour comme peut le laisser penser le schéma 3.22. Encore une fois, nous soulignons le fait que la seule condition nécessaire pour que les distributions permettent de discriminer les clés est que les jeux de masques de ces variables soient identiques. Prenons l'exemple d'un unique couple de masques (r_1, r_3) tiré en début d'exécution et utilisé à chaque tour et chaque indice, c'est-à-dire sur les 176 octets. Dans ce cas, il existe $\binom{176}{2}$ couples d'octets vulnérables à l'attaque, chacun livrant de l'information sur le xor du couple d'octets de clé. L'exploitation d'une telle information sera étudiée dans la partie 3.6.

De la même manière que dans les parties précédentes, il est tout à fait possible d'utiliser d'autres variables, par exemple la sortie de `AddRoundKey` x, toujours avec la contrainte de l'identité des jeux de masques sur les variables ciblées. Nous pouvons donc imaginer une attaque ciblant les trois variables m x et y dans le cas d'un schéma de type 3.18d par exemple, si le même triplet (r_1, r_2, r_3) est identique pour plusieurs positions d'octets d'état. Les distributions jointes sont dans ce cas basées sur des 6-uplets de poids de Hamming des variables, et les observations sont, elles aussi, des 6-uplets $(m'_a, m'_b, x'_a, x'_b, y'_a, y'_b)$ où $m'_i = m_i \oplus r_1$, $x'_i = x_i \oplus r_2$, et $y'_i = y_i \oplus r_3$. Dans ce cas l'attaque permet aussi de retrouver la valeur du xor des deux octets de clé. Le but est toujours d'améliorer l'efficacité de l'attaque de par des distributions plus distinctes les unes des autres grâce à l'ajout de couples de variables.

3.5.2 Le cas particulier de $m\ x$

Le cas m x est ici encore un cas particulier. Les distributions jointes sont issues des poids de Hamming des quadruplets $(m'_a = m_a \oplus r_1, m'_b = m'_b \oplus r_1, x'_a = x_a \oplus r_2, x'_b = x_b \oplus r_2)$. La figure 3.23 présente les variables utilisées lors de la construction des distributions jointes de l'attaque m x quadrivariée. De manière analogue au cas non masqué, seules 9 distributions différentes existent, chacune correspondant au poids de Hamming du xor des deux octets de clé, soit $\mathrm{HW}(k_a \oplus k_b)$. Nous noterons cette valeur la distance de Hamming entre ces deux octets $\mathrm{HD}(k_a, k_b)$.

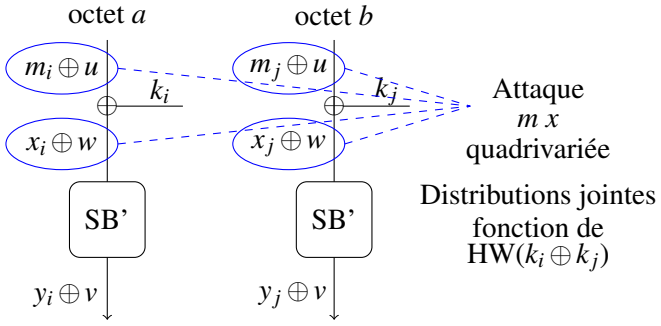


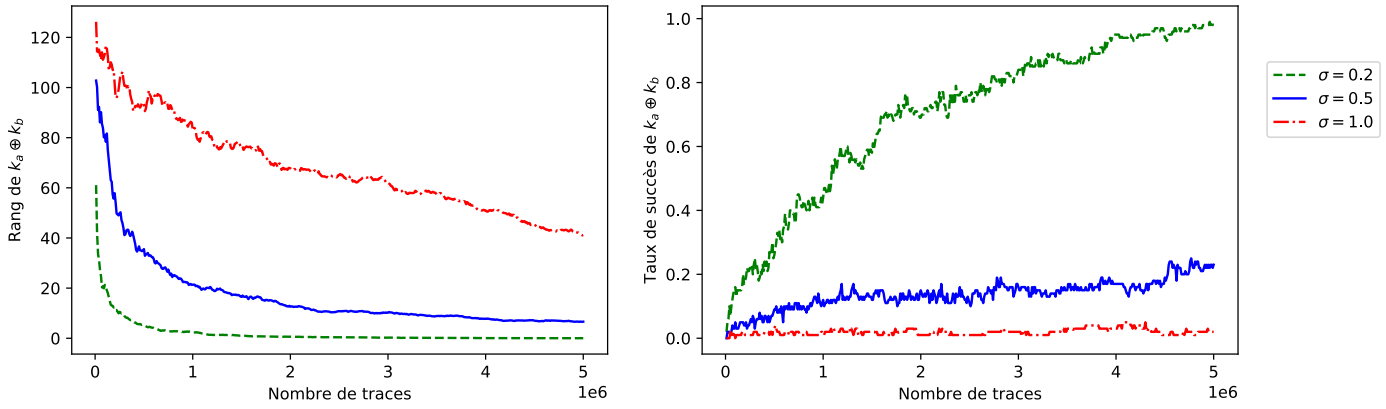
FIGURE 3.23 – Variables utilisées lors d'une attaque $m\ x$ quadrivariée afin de retrouver $k_a \oplus k_b$

3.5.3 Simulations

Suite à la construction de ces distributions jointes et à l'adaptation de l'attaque, nous avons mené des simulations sur les deux cas de figure $m\ y$ et $m\ x$ en utilisant le maximum de vraisemblance sur l'AES. Dans le cas de l'attaque m et y, chaque run consiste à tirer aléatoirement un couple d'octets de clé (k_a, k_b) puis pour chaque nouvelle trace quatre valeurs m_a m_b r_1 et r_3, à calculer les observations $(\mathrm{HW}(m_a \oplus r_1), \mathrm{HW}(m_b \oplus r_1), \mathrm{HW}(\mathrm{SB}(m_a \oplus k_a) \oplus r_3), \mathrm{HW}(\mathrm{SB}(m_b \oplus k_b) \oplus r_3))$, à bruiter ces derniers en leur ajoutant un bruit gaussien d'écart-type σ, puis d'appliquer le maximum de vraisemblance afin de déterminer la valeur de $k_a \oplus k_b$. Les simulations sont adaptées dans le cas où x est observé plutôt que y, et c'est la valeur $\mathrm{HD}(k_a, k_b)$ qui est déterminée. La figure 3.24 présente les résultats d'attaques $m\ y$ quadrivariées moyennés sur 100 runs, pour différents écart-types de bruit. Le rang et le taux de succès de détermination du xor des deux octets clés $k_a \oplus k_b$ y sont représentés. Ces courbes démontrent que l'attaque permet en effet de déterminer la valeur de $k_a \oplus k_b$. Il faut cependant compter le nombre de traces en millions afin de correctement déterminer cette valeur, même pour des bruits assez faibles.

L'attaque quadrivariée utilisant m et x est présentée sur la figure 3.25. Le taux de succès de détermination de $\mathrm{HD}(k_a, k_b)$ moyenné sur 1000 runs y est représenté pour différents niveaux de bruit. Sans surprise, beaucoup moins de courbes sont nécessaires comparé à l'attaque ci-dessus. Cependant, seule $\mathrm{HD}(k_a, k_b)$ est déterminée. Dans ce cas il est important de se poser la question de l'exploitation d'une telle information. Cette question sera étudiée plus loin.

Nous avons observé un comportement particulier dans les résultats que donnent les attaques quadrivariées $m\ x$. Selon le niveau de bruit, les distances de Hamming concurrentes à la vraie ne sont pas les mêmes. Dans le cas général, intuitivement, nous pouvons nous attendre à ce que les distances les plus fortes concurrentes à la bonne soient les plus proches. C'est effectivement le cas lorsque le bruit est assez élevé, comme nous pouvons le constater sur la figure 3.27a. Cette figure présente l'opposé du logarithme de la probabilité relative de chacune des distances à la meilleure. La vraie distance dans cet exemple est 3. Deux distances en particulier sont légèrement moins bonnes, et correspondent aux distances 2 et 4, les deux plus proches de 3. Notons que



(a) Rang

(b) Taux de succès

FIGURE 3.24 – Rang et taux de succès de l'attaque *m y* quadrivariée pour différents niveaux de bruit

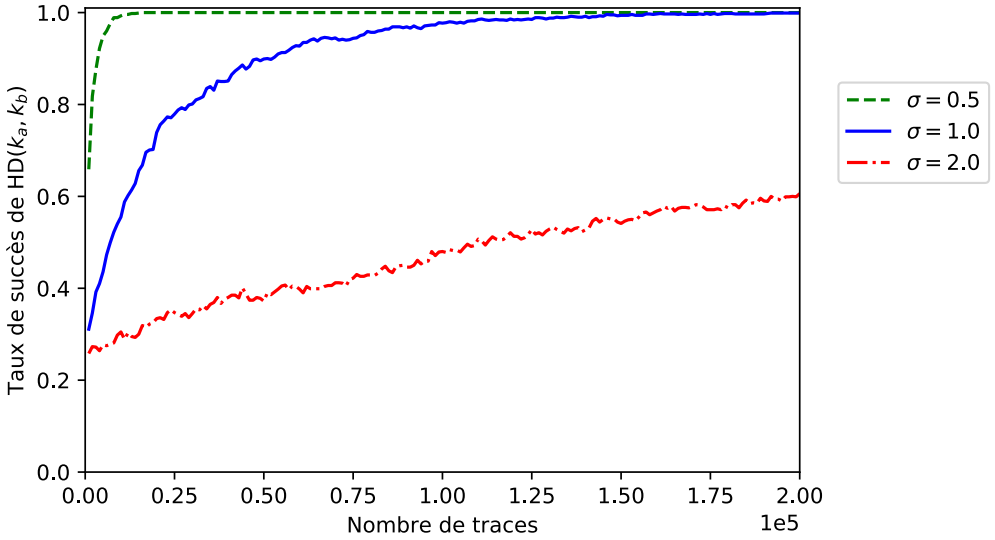


FIGURE 3.25 – Taux de succès d'attaque *m x* quadrivariée pour différents niveaux de bruit

dans les deux figures 3.27a et 3.27b, les distances 2 et 4 se chevauchent et sont donc difficiles à identifier. Viennent ensuite par groupe de deux les distances 1 et 5, puis 0 et 6, et enfin plus loin seules, la distance 7 puis la distance 8. Lorsque le bruit est faible, le comportement est différent. Comme nous pouvons le voir sur la figure 3.27b, les deux distances légèrement moins bonnes sont celles ayant la même parité, à savoir les distances 1 et 5 qui viennent en premier, bien avant celles attendues de 2 et 4. Notons que les distances euclidiennes entre les différentes distributions sont faibles à parités égales, ce qui explique que lorsque le bruit est très faible, ce sont plutôt les distances de même parité qui sont les plus vraisemblables.
Il est possible de faire une analogie avec les distributions *m x* non quadrivariées présentées en figure 3.6. À bruit très faible, les observations tombent dans les mêmes colonnes que les valeurs dont elles sont issues, et les distributions jointes expérimentales liées à des poids de Hamming de même parité sont donc très semblables puisqu'elles partagent les mêmes colonnes. De plus, elles sont très différentes d'une parité à l'autre puisqu'elles ne partagent aucune colonne en commun. Si l'on augmente progressivement le bruit lié aux mesures, arrive un moment où celles-ci peuvent tomber dans une colonne qui n'est pas celle des valeurs dont elles sont issues. À ce moment-là, ce sont plutôt les formes globales qui apparaissent. Dans ce cas de figure, les distributions jointes liées à des poids de Hamming proches sont plus semblables. Une illustration appuyant ces propos est présentée en figure 3.26. Des distributions jointes expérimentales pour les poids de Hamming de clé 0,1 et 4 y sont présentées, pour des niveaux croissants de bruit. Pour des bruits nuls, le maximum de vraisemblance favorise la proximité des poids de Hamming 0 et 4 [1]. À partir d'un écart-type de bruit de 0.75, le nombre de colonnes qu'ont les distributions des poids de Hamming 0 et 1 en commun semble supérieur à celui entre 0 et 4, et les colonnes semblent mieux correspondre.

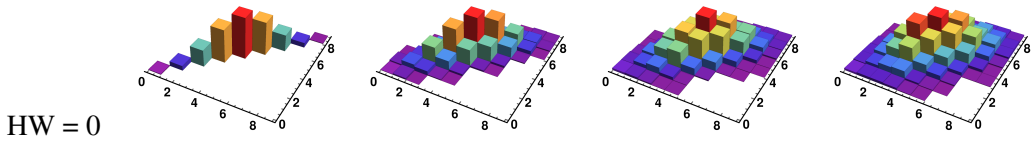
3.6 Exploitation de l'information des attaques *m x* et quadrivariées

Contrairement aux attaques observant *m* et *y* sur implémentations non protégées, ainsi que les attaques par canaux cachés classiques, les attaques que nous avons proposées, et notamment celles quadrivariées ne permettent pas de distinguer les valeurs des octets de clé. Ici nous proposons l'étude de l'exploitation des informations données par chacune de ces attaques. Celle-ci sera faite sous plusieurs hypothèses de schémas de masquage dans le cas quadrivarié, chacun se distinguant par son nombre de jeux de masques.

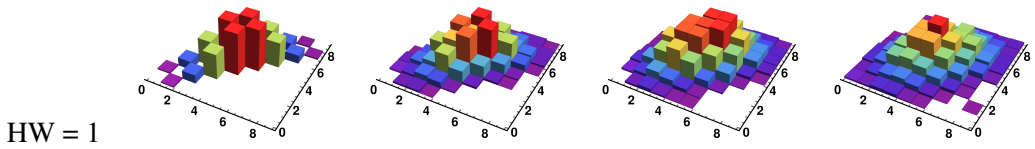
3.6.1 Les types de masques

Jusqu'à présent nous avons évoqué la possibilité d'attaquer tous les couples d'octets de clé sans contrainte sur les tours ni les positions de chacun. Cela nécessite des jeux de masques identiques à

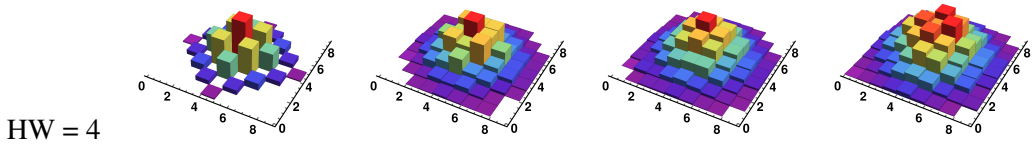
1. Les supports des distributions théoriques de HW pairs et impairs sont complètement disjoints



(a) $\sigma = 0.0$ (b) $\sigma = 0.5$ (c) $\sigma = 0.75$ (d) $\sigma = 1.0$

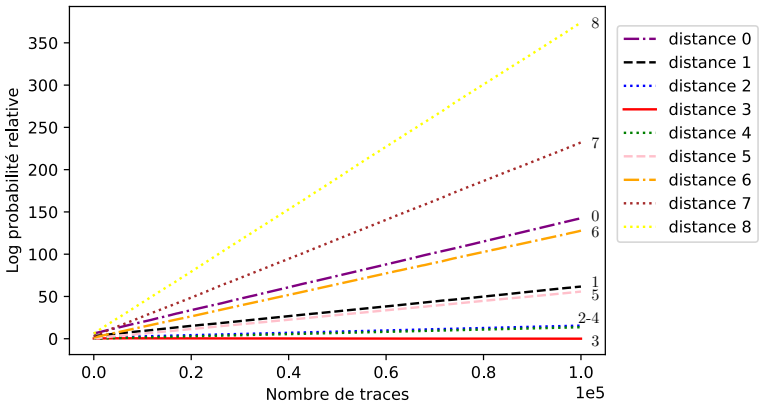


(e) $\sigma = 0.0$ (f) $\sigma = 0.5$ (g) $\sigma = 0.75$ (h) $\sigma = 1.0$

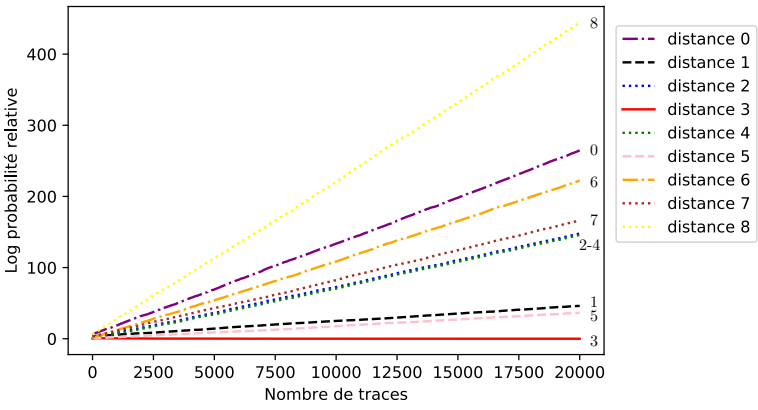


(i) $\sigma = 0.0$ (j) $\sigma = 0.5$ (k) $\sigma = 0.75$ (l) $\sigma = 1.0$

FIGURE 3.26 – Distributions jointes expérimentales de HW(m) et HW(x) pour trois poids de Hamming de clé et pour différents niveaux de bruits



(a) $\sigma = 1.0$



(b) $\sigma = 0.5$

FIGURE 3.27 – Log des probabilités relatives à la meilleure distance pour deux valeurs de bruit (bonne distance = 3)

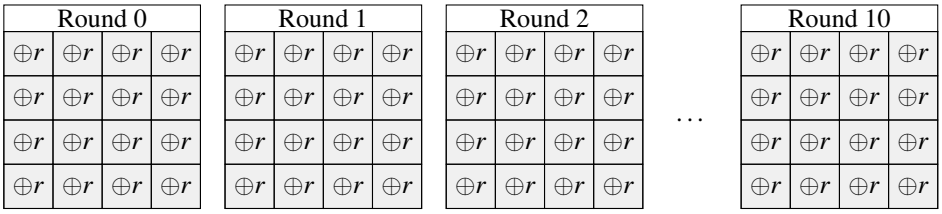


FIGURE 3.28 – Cas 1 : Jeu de masques commun pour tous les octets

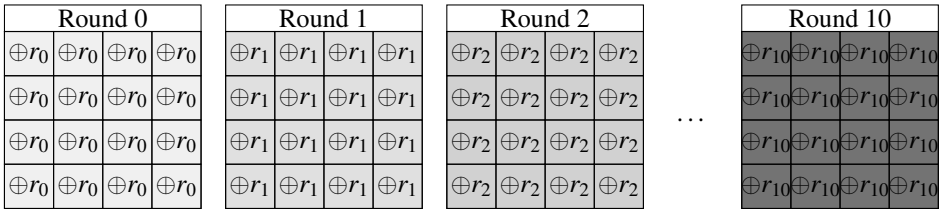


FIGURE 3.29 – Cas 2 : 11 jeux de masques

toutes les positions. Il existe des cas où ces jeux de masques sont différents d'un indice à un autre ou d'un tour à un autre. Nous étudions ici trois cas de masquages et la possibilité de récupérer la clé d'un AES 128 bits dans chacun d'entre eux. Rappelons que ce que nous appelons l'indice dans un AES signifie l'indice de l'octet dans l'état, il en existe 16. Dans le cas 128 bits, la clé étendue est constituée de 11 clés de tour de 16 octets. Un tour et un indice définissent une position d'octet de clé étendue et/ou d'octet d'état. Lorsqu'un jeu de masques est appliqué à une position, nous dirons que cette position utilise ce jeu.

Le premier type de masquage, le moins coûteux en termes de génération de nombres aléatoires, est le cas où le jeu de masques est identique en chacun des octets de clé étendue. Notons que la dernière clé de tour est un cas particulier puisque le `AddRoundKey` correspondant n'est pas suivi par un `SubBytes`. Pour attaquer cette dernière, il conviendrait de l'attaquer à partir du chiffré. Dans la suite des explications, nous omettons volontairement ce cas particulier. Dans le cas d'une attaque *m x*, la dernière clé de tour n'est plus un cas particulier. Ce premier type de masquage est illustré à la figure 3.28. Dans une telle configuration, il est possible d'attaquer n'importe lequel des $\binom{160}{2}$ ou $\binom{176}{2}$ couples d'octets de clé.

Le second type de masquage correspond à la situation où un jeu de masques est tiré de nouveau à chaque tour. Il existe donc 10, ou 11 lorsque l'on considère *m x*, jeux de masques, chacun étant identique pour les 16 positions d'un même tour. Ce cas est présenté à la figure 3.29, où chacun des r_i correspond à un jeu de masques. Dans ce cas, il n'est possible d'utiliser que des couples d'octets d'un même tour, soit $10 * \binom{16}{2}$ ou $11 * \binom{16}{2}$ couples.

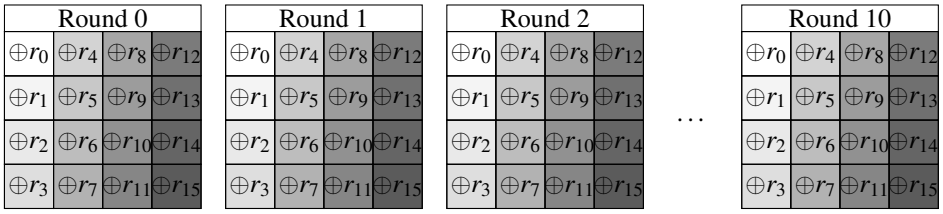


FIGURE 3.30 – Cas 3 : 16 jeux de masques

Enfin, le dernier type présenté à la figure 3.30 consiste à utiliser un jeu de masques identique à indice identique, et ce quel que soit le tour. Il existe donc 16 jeux de masques, chacun correspondant à un indice d'octet de clé, et commun à chacune des clés de tour. Dans cette configuration, les couples attaqués doivent être constitués d'octets de mêmes indices. Il existe donc $16 * \binom{10}{2}$ ou $16 * \binom{11}{2}$ possibilités.

Le cas où chacun des jeux de masques est différent n'est pas abordé puisqu'il empêcherait complètement l'attaque de fonctionner, n'ayant aucun couple partageant le même jeu à attaquer. Il s'agit donc d'une contre-mesure efficace face à ce type d'attaque.

3.6.2 Exploitation de l'information apportée par m et y

Nous nous intéressons ici à la manière de récupérer la clé grâce aux attaques décrites plus haut. Dans le cas d'observation quadrivariée des variables m et y, la distinction se fait au niveau du xor des deux octets de clé.

Cas 1 et cas 2

Si un attaquant est capable de récupérer les 16 couples de PoI correspondant chacun à la manipulation de m et de y de tous les octets d'une clé de tour, il lui est alors possible de mener 15 attaques, permettant, si elles sont fructueuses, de réduire à 256 le nombre de clés possibles. Les 15 attaques doivent être effectuées en chacun des couples impliquant un indice ix quelconque. Elles permettent ainsi de déterminer $(k_{ix} \oplus k_i)$ où $i = \{0,...,16\} \backslash \{ix\}$. À chaque valeur d'octet de clé d'indice ix correspond une unique clé de tour. Rappelons qu'obtenir une clé de tour est équivalent à obtenir la clé puisque le key schedule permet de calculer à la fois les clés de tour suivantes et précédentes.

Le second type de masquage se résout de la même manière que le cas précédent puisqu'il est possible d'attaquer n'importe quel couple d'octets d'une même clé de tour.

Cas 3

Dans le troisième type de masquage, chaque indice possède son propre jeu de masques. Supposons un attaquant capable de récupérer les 32 couples de PoI correspondant aux manipulations de m et y de deux clés de tour consécutives. Soient ces deux clés de tour $\mathbf{K_a} = \{k_{(a,0)}, ..., k_{(a,15)}\}$ et $\mathbf{K_{a+1}} = \{k_{(a+1,0)}, ..., k_{(a+1,15)}\}$. Il est alors capable de mener 16 attaques récupérant les valeurs $v_i = k_{(a,i)} \oplus k_{(a+1,i)}$ pour $i = \{0, ..., 15\}$. La clé de tour $\mathbf{K_a}$ est alors calculable comme suit :

$$
\begin{array}{lll}
k_{(a,0)} = v_0 \oplus v_4 & = k_{(a,0)} \oplus k_{(a+1,0)} \oplus k_{(a,4)} \oplus k_{(a+1,4)} & = k_{(a,0)} \oplus k_{(a+1,0)} \oplus k_{(a,4)} \oplus k_{(a,4)} \oplus k_{(a+1,0)} \\
k_{(a,1)} = v_1 \oplus v_5 & = k_{(a,1)} \oplus k_{(a+1,1)} \oplus k_{(a,5)} \oplus k_{(a+1,5)} & = k_{(a,1)} \oplus k_{(a+1,1)} \oplus k_{(a,5)} \oplus k_{(a,5)} \oplus k_{(a+1,1)} \\
k_{(a,2)} = v_2 \oplus v_6 & = k_{(a,2)} \oplus k_{(a+1,2)} \oplus k_{(a,6)} \oplus k_{(a+1,6)} & = k_{(a,2)} \oplus k_{(a+1,2)} \oplus k_{(a,6)} \oplus k_{(a,6)} \oplus k_{(a+1,2)} \\
k_{(a,3)} = v_3 \oplus v_7 & = k_{(a,3)} \oplus k_{(a+1,3)} \oplus k_{(a,7)} \oplus k_{(a+1,7)} & = k_{(a,3)} \oplus k_{(a+1,3)} \oplus k_{(a,7)} \oplus k_{(a,7)} \oplus k_{(a+1,3)} \\
k_{(a,4)} = v_4 \oplus v_8 & = k_{(a,4)} \oplus k_{(a+1,4)} \oplus k_{(a,8)} \oplus k_{(a+1,8)} & = k_{(a,4)} \oplus k_{(a+1,4)} \oplus k_{(a,8)} \oplus k_{(a,8)} \oplus k_{(a+1,4)} \\
k_{(a,5)} = v_5 \oplus v_9 & = k_{(a,5)} \oplus k_{(a+1,5)} \oplus k_{(a,9)} \oplus k_{(a+1,9)} & = k_{(a,5)} \oplus k_{(a+1,5)} \oplus k_{(a,9)} \oplus k_{(a,9)} \oplus k_{(a+1,5)} \\
k_{(a,6)} = v_6 \oplus v_{10} & = k_{(a,6)} \oplus k_{(a+1,6)} \oplus k_{(a,10)} \oplus k_{(a+1,10)} & = k_{(a,6)} \oplus k_{(a+1,6)} \oplus k_{(a,10)} \oplus k_{(a,10)} \oplus k_{(a+1,6)} \\
k_{(a,7)} = v_7 \oplus v_{11} & = k_{(a,7)} \oplus k_{(a+1,7)} \oplus k_{(a,11)} \oplus k_{(a+1,11)} & = k_{(a,7)} \oplus k_{(a+1,7)} \oplus k_{(a,11)} \oplus k_{(a,11)} \oplus k_{(a+1,7)} \\
k_{(a,8)} = v_8 \oplus v_{12} & = k_{(a,8)} \oplus k_{(a+1,8)} \oplus k_{(a,12)} \oplus k_{(a+1,12)} & = k_{(a,8)} \oplus k_{(a+1,8)} \oplus k_{(a,12)} \oplus k_{(a,12)} \oplus k_{(a+1,8)} \\
k_{(a,9)} = v_9 \oplus v_{13} & = k_{(a,9)} \oplus k_{(a+1,9)} \oplus k_{(a,13)} \oplus k_{(a+1,13)} & = k_{(a,9)} \oplus k_{(a+1,9)} \oplus k_{(a,13)} \oplus k_{(a,13)} \oplus k_{(a+1,9)} \\
k_{(a,10)} = v_{10} \oplus v_{14} & = k_{(a,10)} \oplus k_{(a+1,10)} \oplus k_{(a,14)} \oplus k_{(a+1,14)} & = k_{(a,10)} \oplus k_{(a+1,10)} \oplus k_{(a,14)} \oplus k_{(a,14)} \oplus k_{(a+1,10)} \\
k_{(a,11)} = v_{11} \oplus v_{15} & = k_{(a,11)} \oplus k_{(a+1,11)} \oplus k_{(a,15)} \oplus k_{(a+1,15)} & = k_{(a,11)} \oplus k_{(a+1,11)} \oplus k_{(a,15)} \oplus k_{(a,15)} \oplus k_{(a+1,11)} \\
k_{(a,12)} = S^{-1}(v_3) & = S^{-1}(k_{(a,3)} \oplus k_{(a+1,3)}) & = S^{-1}(k_{(a,3)} \oplus k_{(a,3)} \oplus S(k_{(a,12)})) \\
k_{(a,13)} = S^{-1}(v_0 \oplus 1) & = S^{-1}(k_{(a,0)} \oplus k_{(a+1,0)} \oplus 1) & = S^{-1}(k_{(a,0)} \oplus k_{(a,0)} \oplus S(k_{(a,13)}) \oplus 1 \oplus 1) \\
k_{(a,14)} = S^{-1}(v_1) & = S^{-1}(k_{(a,1)} \oplus k_{(a+1,1)}) & = S^{-1}(k_{(a,1)} \oplus k_{(a,1)} \oplus S(k_{(a,14)})) \\
k_{(a,15)} = S^{-1}(v_2) & = S^{-1}(k_{(a,2)} \oplus k_{(a+1,2)}) & = S^{-1}(k_{(a,2)} \oplus k_{(a,2)} \oplus S(k_{(a,15)}))
\end{array}
$$

Pour résumer, 16 attaques ou moins sont nécessaires afin de déduire la clé complète dans chacun des 3 cas de masquage si chacune d'entre elles a effectivement déduit la bonne valeur. Dans le cas contraire, il convient d'exhauster les clés proches de celle donnée par cette attaque. Puisque le résultat de chaque attaque est une liste des xors du plus probable, considéré comme la bonne valeur, au moins probable, il est possible d'utiliser des techniques d'énumération de clés comme présenté dans [LWWW17] par exemple.

3.6.3 Exploitation de l'information apportée par l'observation de m et x

De la même manière que pour le cas précédent, nous proposons ici un moyen de déduire la clé grâce à l'information donnée des attaques observant m et x. Nous apportons une solution à la

fois aux cas de figures correspondant au cas non masqué, au cas masqué avec même masque (contre-mesure inefficace), et au cas quadrivarié.

Cas simple

Dans le cas où m et x ne sont pas masqués, ou présentent le même masque, la situation est la même (voir 3.4.2). L'attaque permet de déterminer le poids de Hamming de l'octet de clé ciblé. Dans ce cas, il est possible d'utiliser des techniques telles que présentées dans [Man03, VBC05], thème que nous avons couvert en section 1.4. La seule différence est qu'une attaque par distributions jointes se substitue à l'attaque par SPA chargée de retrouver des poids de Hamming.

Cas quadrivarié

Dans ce cas, l'attaque permet seulement de déterminer les distances de Hamming entre couples d'octets. Une manière de calculer la clé dans de telles circonstances est étudiée dans [CMW14], et expliquée à la section 1.6.1. Dans la suite, nous proposons l'étude des cas 2 et 3 sans l'utilisation d'autant de données. Cela est motivé par le fait que les attaques quadrivariées sont difficiles, et qu'il est donc intéressant d'en mener le moins possible. Nous explorons donc la question de calculer la clé avec le moins d'attaques possible.

Cas 1 Dans le cas 1 où les jeux de masques sont identiques pour toutes les positions d'octets de clé étendue, l'étude est analogue à [CMW14], à la différence qu'aucune contrainte sur la valeur masquée n'est exercée, et que des contraintes supplémentaires peuvent s'appliquer entre les clés de tour. Il est donc tout à fait possible de récupérer la clé et ce dans un temps très court (moins d'une seconde).

Cas 2 Dans le cas 2, où les couples de masques sont les mêmes uniquement au sein d'une même clé de tour, il est possible d'attaquer et de récupérer $11 \cdot \binom{16}{2} = 1320$ distances de Hamming. Dans le cas où toutes ces distances sont récupérées, nous nous retrouvons exactement dans le cas de [CMW14]. Il suffit donc d'appliquer leur technique d'exploration après avoir déduit les distances de Hamming grâce à des attaques quadrivariées. Dans le cas où les distances sont correctement déterminées, retrouver la clé se fait en un temps très court (moins d'une seconde) et donne un unique candidat. Supposons maintenant que l'attaquant ne réalise que $\binom{16}{2} = 120$ attaques quadrivariées, déduisant ainsi la distance de Hamming de chacun des couples d'une seule clé de tour. Les résultats d'une exploration de type guess-compute-backtrack sont montrés

TABLE 3.2 – Résultats d'exploration utilisant les distances de Hamming d'une seule clé de tour

Nombre restant de clé candidates	Temps moyen d'exécution (s)	Nombre de cas
5 160 960	36.0	1
10 321 920	80.4	997
20 643 840	108.5	2

TABLE 3.3 – Résultats d'exploration utilisant les distances de Hamming de deux clés de tour

Nombre restant de clés candidates	1	2	3	4	6	8	12
Nombre de cas	784	189	1	17	7	1	1

dans la table 3.2. La grande majorité des cas résulte en 10 321 920 clés candidates trouvées en 80 secondes.

Si ce nombre est encore trop grand, il est possible d'ajouter les distances de Hamming des couples de n'importe quelle autre clé de tour. Dans la plupart des cas, l'ajout de telles contraintes résulte en une unique clé comme le montre la table 3.3. Afin d'optimiser le temps d'exploration dans un tel cas, il convient de considérer conjointement les contraintes, et non pas de sélectionner l'intersection des candidats donnés par chaque clé de tour. L'hypothèse de certains octets d'une clé de tour permet le calcul de façon déterministe de certains autres de la seconde, appliquant immédiatement des contraintes. Dans le cas où les clés sont consécutives, une telle exploration voit son temps moyen d'exécution tomber de 80 secondes à une seule.

Une remarque peut être faite sur le nombre de clés candidates restantes à l'issue de l'attaque n'utilisant qu'une seule clé de tour présenté à la table 3.2. Le nombre de candidates est multiple de 5 160 960. Expliquons tout d'abord le cas 10 321 920. Une clé de tour candidate respecte les contraintes des distances de Hamming entre couples d'octets déterminées par les attaques quadrivariées. Xorer une valeur à tous ses octets ne change pas les distances. Pour chaque clé candidate, il en existe 255 autres, chacune correspondant au xor d'une valeur sur tous les octets de l'originale. Les distances de Hamming ne sont pas non plus impactées par une permutation des bits appliquée à chacun des octets. Comme il existe 8! telles permutations de bits, il existe en fait une classe d'équivalence de $256 \cdot 8! = 10321920$ clés candidates. Le cas où ce nombre est divisé par 2 est la conséquence d'un couple de bits égaux sur chacun des 16 octets de la clé de tour. De la même manière, il est aussi possible que des classes d'équivalences plus petites existent lorsque plus de bits sont égaux sur les 16 octets de clé. Le cas de 20 643 840 candidates résulte vraisemblablement de deux classes d'équivalences.

L'attaque peut être adaptée dans le cas d'erreurs sur les distances de Hamming. Dans nos simulations d'attaques quadrivariées portant sur m et x avec un bruit d'écart-type $\sigma = 1.0$, les distances immédiatement candidates de la bonne sont les plus proches, à commencer par celles ayant un écart de 1 (comme vu à la partie 3.5.3). Grâce à cette propriété, il est possible de détecter des erreurs commises si elles ne sont pas trop nombreuses. Pour cela, considérons des triplets d'octets de clé de tour et leurs distances de Hamming respectives. La somme des distances de Hamming de ces trois octets est forcément paire. Par exemple, pour trois octets de clé k_1 k_2 et k_3, le cas $\mathrm{HD}(k_1,k_2) = 3$ $\mathrm{HD}(k_1,k_3) = 5$ $\mathrm{HD}(k_2,k_3) = 4$ peut exister sans erreur, mais en aucun cas celui où $\mathrm{HD}(k_1,k_2) = 3$ $\mathrm{HD}(k_1,k_3) = 5$ $\mathrm{HD}(k_2,k_3) = 7$. En appliquant cette méthode à tous les triplets d'octets attaqués, il est possible d'identifier des distances erronées. Une fois identifiées, un attaquant a le choix d'augmenter le nombre de traces utilisées lors des attaques dont ces distances sont issues si cela est possible, ou bien encore de les retirer complètement.

Voici la démonstration de cette propriété : soient les trois valeurs d'octets k_a, k_b et k_c et leurs trois distances de Hamming respectives $\mathrm{HD}(k_a,k_b)$, $\mathrm{HD}(k_a,k_c)$ et $\mathrm{HD}(k_b,k_c)$. Tout changement d'un bit de k_a résultant en k'_a implique l'une des situations suivantes :

$$\begin{cases} \mathrm{HD}(k'_a,k_b) = \mathrm{HD}(k_a,k_b)+1 \\ \mathrm{HD}(k'_a,k_c) = \mathrm{HD}(k_a,k_b)+1 \end{cases} \quad \begin{cases} \mathrm{HD}(k'_a,k_b) = \mathrm{HD}(k_a,k_b)-1 \\ \mathrm{HD}(k'_a,k_c) = \mathrm{HD}(k_a,k_b)-1 \end{cases}$$

$$\begin{cases} \mathrm{HD}(k'_a,k_b) = \mathrm{HD}(k_a,k_b)+1 \\ \mathrm{HD}(k'_a,k_c) = \mathrm{HD}(k_a,k_b)-1 \end{cases} \quad \begin{cases} \mathrm{HD}(k'_a,k_b) = \mathrm{HD}(k_a,k_b)-1 \\ \mathrm{HD}(k'_a,k_c) = \mathrm{HD}(k_a,k_b)+1 \end{cases}$$

Dans les quatre situations, la parité est inchangée, et ce quelles que soient les trois valeurs de départ. Pour le triplet de valeurs (0,0,0), la somme des distances est paire. Ainsi la somme des distances de trois valeurs est toujours paire.

Cas 3 De le cas où 16 jeux de masques sont tirés, chacun correspondant à un indice, l'attaque permet de déterminer les distances de Hamming entre les octets d'une même position au sein de la clé étendue. Pour une seule position, $\binom{11}{2} = 55$ telles distances peuvent être déterminées. En considérant les 16 indices, $55 \cdot 16 = 880$ distances peuvent être déterminées. Moins de contraintes s'appliquent en comparaison du cas 2, ce qui rend le calcul un peu plus long, passant de moins d'une seconde à 31 en moyenne au cours de nos simulations. Malgré cela, une seule clé candidate est trouvée dans tous les cas.

Si l'on ne considère cette fois qu'un seul indice, et que les 55 distances entre chacun des couples d'octets de cet indice sont déterminées, il n'est pas possible d'en déduire complètement la clé. Cela s'explique par la conception du `KeySchedule` qui ne permet, au mieux, que le calcul de 66 octets parmi les 176. Pour chaque candidat des 11 octets de clé d'un indice, il reste encore au mieux 5 octets dont les valeurs ne subissent aucune contrainte. Ceci est représenté à la figure 3.31.

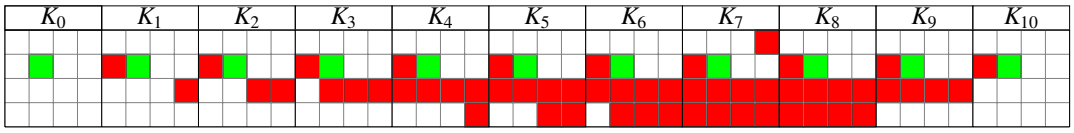


FIGURE 3.31 – Cas 3 : Octets calculables à partir des octets d'un indice

Les 176 octets de clé étendue y sont représentés. Ceux en vert sont ceux connus pour une clé candidate de l'indice 6. Les octets en rouges sont ceux calculables. Il existe donc pour chaque candidat des 11 octets verts 1 candidat de 66 octets. La clé de tour la plus déterminée par ce candidat est K_7 où 5 octets ne sont pas impliqués et sont donc complètement libres.

L'ajout d'un second indice bien choisi permet d'avoir des candidats pour l'ensemble de la clé étendue et donc la clé.

3.7 Conclusion

Dans ce chapitre, nous avons commencé par montrer que l'efficacité du maximum de vraisemblance, qui consiste à calculer les probabilités a posteriori de chacune des clé étant donné un jeu d'observation, était globalement meilleure que celle des distances employées plus traditionnellement. Ces résultats ont été validés par des attaques pratiques d'un Arduino Uno. Dans un second temps, nous avons proposé différentes variantes impliquant d'autres et/ou plus de variables/points d'intérêt.

Ensuite, nous nous sommes intéressés au cas masqué. Tout d'abord, nous avons montré qu'il est possible d'attaquer des implémentations masquées d'ordre 1 lorsque les variables ciblées partagent le même masque en modifiant les distributions jointes théoriques en conséquence. Nous avons notamment soulevé le cas de l'attaque observant m et x qui n'est pas impactée par un tel masquage.

Pour attaquer le cas plus complexe où les variables ciblées ne sont pas protégées par le même masque, mais que différentes positions le sont, nous avons introduit le nouveau concept des attaques quadrivariées. Grâce à ces dernières, il est possible de déterminer un certain type d'information vis-à-vis des couples de clé attaqués. Cette information étant différente selon les variantes proposées, nous avons aussi étudié la manière et la faisabilité de calculer la clé entière à partir de ces attaques. Le cas très particulier de l'attaque quadrivariée sur les variables m et x a notamment été étudié, et le calcul de la clé nécessite d'utiliser une autre attaque visant le key schedule, attaque déjà existante dans la littérature.

Chapitre 4

Recherche de S-boxes compactes

Les S-boxes sont des éléments non linéaires des schémas SPN (Substitution Permutation Network) et Feistel notamment, chargés d'assurer la confusion comme établie par Shannon. L'un des principaux buts de tels éléments est d'assurer la sécurité vis-à-vis de la cryptanalyse classique notamment. Ces dernières profitent de certaines propriétés des S-boxes, propriétés que nous avons introduites au chapitre 2 et que nous appelons critères dans ce document. Il est très important d'en sélectionner avec de bons critères cryptographiques afin que l'algorithme soit sécurisé. Cependant, dans un milieu contraint, les S-boxes doivent être implémentées / stockées de manière la plus légère possible, et un compromis doit généralement être trouvé entre compacité/coût et qualité. Dans cette partie nous présentons une étude visant à construire des S-boxes compactes d'un point de vue matériel, et faciles à protéger par une contre-mesure de type masquage, l'implémentation threshold (TI).

4.1 Motivation

Le projet PACLIDO (Protocoles et Algorithmes Cryptographiques Légers pour l'Internet Des Objets) a pour but de proposer des solutions pour sécuriser les objets connectés grâce à la cryptographie. Ces objets sont très souvent petits / peu chers et contraignent donc de ce fait en ressources et en taille les appareils embarqués spécifiques à la sécurité. Le projet a donc comme but d'implémenter des solutions *lightweight*. Un compromis doit être trouvé afin d'offrir un niveau de sécurité suffisant tout en limitant les ressources dont l'algorithme a besoin et le coût du matériel. Dans le cadre de ce projet, l'algorithme de chiffrement Lilliput-TBC, qui repose sur un schéma de Feistel étendu et qui présente la particularité d'être "tweakable", a été soumis au NIST Lightweight Cryptography Standardization Process [oST17].

Une des parties les plus coûteuses en matériel est la partie non linéaire, notamment lorsque l'implémentation est protégée. Une des tâches liées au développement de cet algorithme a donc été de décider de cette S-box afin qu'elle offre de bonnes propriétés cryptographiques tout en étant la plus compacte possible, même une fois protégée par TI.

Ce chapitre décrit en partie le travail réalisé au cours de la construction de la S-box de Lilliput-TBC [ABC+], et va plus loin en proposant une méthode de recherche de circuits compacts 8 bits grâce à une représentation qui n'était alors utilisée que pour des circuits 4 bits. Nous proposons aussi une nouvelle utilisation des algorithmes génétiques dans le cadre de la recherche de S-box.

4.2 Préliminaires

Toutes ces notions ont déjà été introduites dans le chapitre 2. Nous ne faisons que rappeler quelles sont celles que nous utilisons dans cette partie.

4.2.1 Propriétés cryptographiques

Parmi les propriétés cryptographiques que présentent les S-boxes, nous utilisons principalement le degré deg, l'uniformité différentielle δ et la linéarité $\mathcal{L}$.

4.2.2 Classes d'équivalences

L'équivalence affine définit des classes de S-boxes qui ont comme particularité de ne contenir que des représentants ayant les mêmes triplets de critères $(deg, \delta, \mathcal{L})$. D'un point de vue cryptographique, si l'on ne considère que les attaques utilisant ces trois critères, tous les éléments d'une classe sont équivalents.

4.2.3 Implémentation threshold

L'implémentation threshold d'une fonction est une contre-mesure de type masquage qui a pour particularité de se baser sur du calcul multi-parties, c'est-à-dire de diviser les variables en shares, et la fonction en plusieurs sous fonctions dont la somme revient au résultat de la fonction initiale.

4.3 Construction de S-boxes 8 bits dans la littérature

Du fait de leur importance, la construction/génération des S-boxes est un sujet intensivement exploré par la communauté depuis des années. Le but est principalement de trouver les plus résistantes possibles à la cryptanalyse, c'est-à-dire celles aux meilleures propriétés cryptographiques, et les plus petites, celles qui requièrent le moins de place en code, mémoire ou surface. En ce qui concerne les propriétés cryptographiques, il existe trois moyens principaux de générer des S-boxes. Le premier consiste à en tirer de manière aléatoire. Cette méthode ne fonctionne pas du fait de la taille de l'espace à explorer, résultant en une très faible probabilité de tomber sur de bonnes propriétés cryptographiques comme expliqué par Perrin dans sa thèse [Per17, Table 9.2]. La seconde méthode consiste à en générer grâce à des méthodes algébriques. L'exemple le plus connu est celui de la S-box de l'AES qui est pour l'instant la meilleure connue, possédant une différentielle $\delta = 4$ et une linéarité $\mathcal{L} = 32$, et qui est notamment construite grâce à une inversion. Enfin, la troisième méthode consiste à guider les recherches grâce à des heuristiques. Cette dernière englobe par exemple les recherches par des algorithmes génétiques, de la programmation génétique [PJ19] (voir par exemple [Kne17] pour une vision d'ensemble sur l'utilisation des techniques évolutives), le "hill climbing" [Mil98], la descente de gradient [KKO13], ou le recuit simulé [CJS05].

4.4 Représentation d'un circuit

Tout ce chapitre utilisera la notion de circuit. Nous définissons ici ce que nous comprenons par circuit.

Une instruction est équivalente à une porte. Par instruction en dehors d'un circuit nous désignons une opération telle que le XOR, OR ou AND par exemple. Mais plus généralement au sein d'un circuit, nous désignons par instruction à la fois le type d'opération, mais aussi ses indices de bits d'entrée, et son bit de sortie. Un circuit C est constitué d'une série ordonnée de n_g instructions/portes g_i telle que $C = \{g_1, g_2, ..., g_{n_g}\}$. Le résultat en sortie d'une valeur x donnée en entrée de ce circuit est noté $C(x)$. Le circuit est résolu en appliquant la série d'instructions de manière itérative à x. Un autre moyen de le représenter est de calculer les termes liés à chacun des bits de sortie en fonction des bits d'entrée. Voici un exemple d'un circuit 4 bits ($b_3 b_2 b_1 b_0$) où b_0

désigne le bit de poids faible (Least Significant Bit - LSB), et b_3 désigne le bit de poids fort (Most Significant Bit - MSB) : soit C un circuit constitué de deux instructions g_1 et g_2, correspondant respectivement à une porte XOR ayant pour entrées les bits b_0 et b_1, et pour sortie b_1, et à une porte AND ayant pour entrées les bits b_1 et b_3, et pour sortie le bit b_0. Le bit b_1 a pour équation $b_1 \oplus b_0$, et le bit b_0 a pour équation $(b_0 \oplus b_1) \wedge b_3$. Le circuit entier a pour équations :

$$\begin{cases} b_0 = & (b_0 \oplus b_1) \wedge b_3 \\ b_1 = & b_1 \oplus b_0 \\ b_2 = & b_2 \\ b_3 = & b_3 \end{cases}$$

où les termes à gauche des égalités sont les bits de sortie, et ceux à droite ceux d'entrée. Dans cet exemple, $C(5) = C(0b0101) = 0b0110 = 6$. Appliquer ce circuit à toutes les valeurs de $GF(2^4)$ correspond à construire la table correspondant au circuit.

4.5 S-box Lilliput-TBC

Dans cette partie, nous décrivons une partie de l'étude qui nous a permis de sélectionner la S-box de Lilliput-TBC. L'étude complète faite conjointement avec d'autres auteurs est présentée en [ABC+]. Nous nous concentrons principalement sur l'aspect recherche de S-boxes 4 bits optimales et facilement "thresholdables" qui a été notre tâche au sein de ce projet.

4.5.1 Recherche en largeur de S-boxes 4 bits

Nous avions pour but de rechercher une S-box 4 bits compacte qui, une fois insérée dans un schéma de type Feistel, donnerait une S-box 8 bits ayant de bonnes propriétés. De plus, la S-box 4 bits doit être de bonne qualité afin que la 8 bits le soit aussi. Nous avons donc décidé de procéder de la même manière que Ullrich et al. [UDCI+11], c'est-à-dire réaliser une exploration exhaustive en partant "du bas". Partant du circuit vide, nous avons progressivement ajouté des instructions. Pour que cette recherche soit exhaustive, toutes les instructions possibles, c'est-à-dire opération et bits d'entrée/sortie, sont testées avant d'en ajouter une autre. Cela correspond à une exploration en largeur d'un arbre dont les nœuds correspondent à un circuit, et sa profondeur au nombre d'instructions qui les composent. Pendant l'exploration, la classe d'équivalence affine de chaque noeud est testée. De cette manière, nous garantissons l'optimalité de la taille du circuit en nombre de portes pour chaque classe atteinte. Notons cependant que ces résultats garantissent qu'au moins une instance particulière d'une classe est atteignable en un certain nombre de portes, mais en aucun cas que toute instance de cette classe l'est en ce nombre de portes. Afin de rendre ces circuits faciles à thresholder, nous avons choisi de restreindre les opérations possibles au jeu

{XOR,AND}. Contrairement à [UDCI+11], nous ne comptons pas l'opération MOV puisque dans le contexte matériel cela correspond simplement à un fil dont le coût n'est généralement pas comptabilisé. Le nombre de registres disponibles a été fixé à cinq. Cela correspond aux quatre registres d'entrée, plus un registre temporaire, ce qui correspond à cinq fils verticaux dans nos schémas. Nous avons en fait remarqué une nette amélioration des résultats lors du passage de quatre à cinq registres, mais aucun lors du passage à six, raison de ce choix.
Le nombre maximum d'enfants que chaque noeud peut avoir est $N_g \cdot \binom{N_r}{2} \cdot N_r$ où N_g est le nombre d'opérations possibles, et N_r le nombre de registres. Dans nos conditions, le nombre maximum d'enfants que peut avoir un nœud est $2 \cdot \binom{5}{2} \cdot 5 = 100$. Afin de diminuer ce nombre, et ainsi la complexité d'exploration, nous utilisons quelques astuces qui empêchent l'ajout de certaines portes, à savoir :

— Une instruction ne doit pas réécrire (et donc effacer) un registre qui n'a pas été utilisé en entrée de cette ou une autre porte. Si c'était le cas, l'instruction ayant servi à calculer ce registre n'ayant pas servi est inutile.

— Le résultat d'une instruction ne doit pas être systématiquement 0 pour toutes les entrées. Les deux instructions XOR et AND utilisant une telle entrée sont inutiles puisque donnent toutes deux en sortie le même résultat qu'une entrée.

— Une instruction doit changer au moins un registre. Si ce n'est pas le cas, elle est inutile.

Nous utilisons aussi la notion d'équivalence par permutation de bits afin de restreindre l'exploration. Deux circuits C_1 et C_2 ayant respectivement pour tables T_1 et T_2 sont dit équivalents par permutation de bits si il existe une permutation P_1 des bits d'entrée et P_2 des bits de sortie telles que :

$$T_1(x) = P_2 \circ T_2 \circ P_1(x) \qquad \forall x \in \mathbb{F}_2^n$$

Pendant l'exploration, nous gardons un seul représentant de chaque classe d'équivalence par permutation de bits. Notons que cette équivalence ne change pas l'équivalence affine. Pour cela, nous avons défini une représentation canonique ayant pour propriété d'être la même pour deux circuits si et seulement s'ils font partie de la même classe. Chaque nouveau noeud est donc testé, et si cette classe a déjà un représentant, alors ce noeud est écarté et ne produira pas d'enfants.

Le très grand nombre de noeuds à explorer pose un problème de mémoire. Ce nombre étant lié à la profondeur de l'arbre, la recherche ne peut pas dépasser un certain nombre de portes. Pour aller plus loin, nous avons choisi de diviser l'exploration en plusieurs processus, chacun ne commençant pas par un circuit vide, mais par un ensemble de circuits résultant d'une recherche antérieure. L'inconvénient est de ne pas profiter pleinement de l'astuce permettant de réduire l'exploration par détection d'équivalence par permutation de bits puisque chaque processus est indépendant et ne communique pas aux autres sa propre liste de classes atteintes.

4.5.2 Résultats

Grâce à cette méthode, nous avons réussi à construire les circuits possédant jusqu'à 8 portes. Parmi ceux-ci, nous avons gardé exclusivement ceux réalisant une permutation. 62 classes d'équivalence affine ont été trouvées parmi les 302 (voir [DC07]). Plus particulièrement, les six classes quadratiques ont naturellement été atteintes, ainsi que trois des seize classes dites optimales, c'est-à-dire présentant les meilleurs triplets de critères possibles. En reprenant les notations de [BNN+15], les trois optimales atteintes sont $\mathcal{C}_{223}$ $\mathcal{C}_{296}$ et $\mathcal{C}_{297}$. Grâce à la division de la recherche en plusieurs processus, nous avons été capables d'explorer partiellement jusqu'à 9 portes, ce qui nous a permis d'atteindre 108 classes, dont l'optimale supplémentaire $\mathcal{C}_{266}$.

Pour minimiser également le coût de l'implémentation threshold de la S-box de Lilliput-TBC, nous n'avons pas choisi directement un représentant parmi les classes optimales trouvées ci-dessus. Nous avons utilisé la même astuce que [BNN+15], à savoir construire une S-box cubique en composant deux S-boxes quadratiques afin de diminuer le nombre de shares nécessaires. Leur étude montre que quatre classes optimales sont atteignables par composition de deux quadratiques, à savoir $\mathcal{C}_{223}$ $\mathcal{C}_{266}$ $\mathcal{C}_{296}$ $\mathcal{C}_{297}$. Plusieurs représentants de chacune des classes quadratiques ont été atteints avec un nombre optimal de portes. La table 4.1 résume ces résultats :

TABLE 4.1 – Nombre de portes requises des représentants optimaux des classes quadratiques

Classe	Nbr portes
Q_4	2
Q_{12}	4
Q_{294}	4
Q_{293}	6
Q_{299}	6
Q_{300}	6

Une composition de deux de ces classes résulte en 4, 6, 8, 10 ou 12 portes. Toujours d'après l'étude de [BNN+15], nous en déduisons qu'aucune classe optimale n'est atteignable en 4 ou 6 portes. Le minimum d'une composition pour atteindre une classe optimale est 8, ce qui est le cas pour les quatre classes.

Un autre critère permettant de minimiser le coût d'implémentation threshold de ce circuit est de choisir, si possible, des circuits uniformes lorsque thresholdés directement. Seules des représentants des classes Q_4 Q_{294} et Q_{299} peuvent présenter cette particularité. D'après les deux dernières observations, nous avons décidé de créer toutes les compositions des représentants optimaux de la classe Q_{294}. Afin d'optimiser au maximum le nombre de portes, la composition des

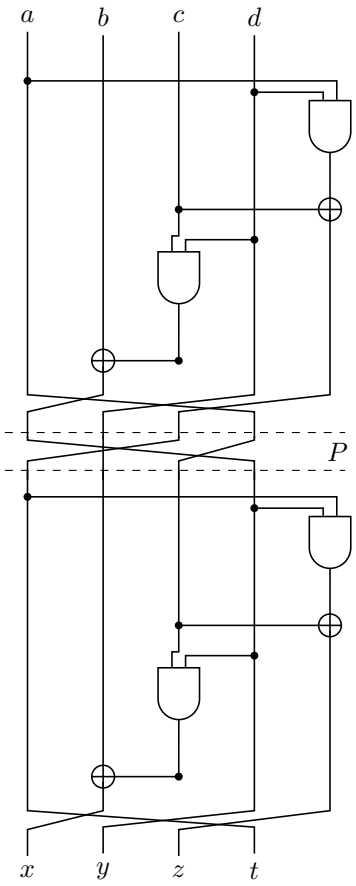


FIGURE 4.1 – S-box 4 bits interne du schéma de Feistel utilisé pour la construction de la S-box 8 bits de Lilliput-TBC

représentants comporte uniquement des permutations de bits au lieu des traditionnels mappages linéaires proposés. Après sélection des représentants fournissant les meilleurs critères lors de la construction de S-boxes 8 bits incluant notre 4 bits, nous avons choisi un représentant de la classe $\mathcal{C}_{223}$ ayant la particularité d'être la composition de deux circuits quadratiques identiques. Ce circuit est représenté en figure 4.1.

Les circuits quadratiques étant optimaux, nous sommes assurés d'avoir atteint une classe optimale avec le plus petit nombre de portes permis par ce jeu d'instructions et ce nombre de registres. De plus, le circuit est optimisé pour nécessiter le plus petit nombre de shares possible lors du threshold. Notons que nous atteignons cette classe avec autant de portes que [UDCI+11] sans compter les MOV.

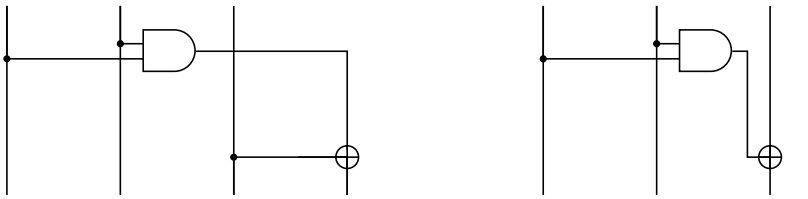
4.6 S-boxes 8 bits

La section précédente décrit une exploration systématique des circuits 4 bits en ajoutant progressivement des portes afin de trouver les circuits optimaux de chacune des classes d'équivalence affine. Nous avons réussi à atteindre une dizaine de portes grâce à cette méthode. L'adaptation directe à 8 bits pose deux problèmes. Premièrement, puisque le nombre de registres augmente, la complexité d'exploration augmente elle aussi. Et deuxièmement, le nombre de portes requis afin d'atteindre des propriétés cryptographiques intéressantes augmente lui aussi. Il est donc impossible de prétendre explorer de la même manière les circuits 8 bits. Comme nous l'avons déjà mentionné, des constructions sont étudiées afin de construire des circuits 8 bits à partir de 4 bits dans le but d'être compacts. Nous posons ici la question de créer des circuits compacts sans utiliser ce genre de technique. Dans ce but, nous proposons deux méthodes destinées à chercher des circuits compacts atteignant de bonnes propriétés pour leur taille, et facilement thresholdable, c'est-à-dire en conservant le jeu réduit d'opérations {XOR,AND}. La première est une recherche purement aléatoire dans l'espace des circuits, et la seconde une méthode génétique dont les individus sont des circuits.

4.6.1 Porte ANDXOR

Avant toute chose, nous introduisons ici une instruction particulière qui permet d'accélérer le processus de recherche dans les deux méthodes. Un problème lié à la construction de circuits par ajouts de portes est la très faible proportion d'entre-eux réalisant une permutation. Cette proportion a même tendance à diminuer entre les circuits 4 bits et les 8 bits. Grâce aux observations de nos circuits optimaux 4 bits, et à l'observation de S-boxes compactes proposées dans la littérature, nous avons remarqué que les opérations OR et AND sont souvent suivis d'un XOR, la seconde prenant en entrée la sortie de la première. Nous avons donc choisi d'utiliser des instructions doubles portes ANDXOR comportant trois bits d'entrée, correspondant aux deux entrées du AND, et à la seconde du XOR, et un bit de sortie. En choisissant le registre de sortie égal à la seconde entrée du XOR, cette instruction composite, que nous appellerons ANDXOR avec résultat en place, a pour propriété de transformer une permutation en une autre lorsqu'elle est ajoutée au circuit, au même titre que le XOR. Grâce à cela, nous avons un moyen d'explorer systématiquement des circuits de permutations. L'inconvénient de cette méthode est de ne pas couvrir l'ensemble des circuits possibles, mais il est très largement compensé par l'avantage énorme qu'il procure en temps de calcul. La figure 4.2 présente une instruction composite ANDXOR quelconque à gauche, et une porte ANDXOR avec résultat en place à droite.

Preuve :
La table de la porte ANDXOR est représentée en 4.2, où b_3 et b_2 sont les bits d'entrée du AND, b_1 la seconde entrée du XOR, et b_s le bit de sortie. Puisque le ANDXOR est avec résultat en place,



(a) ANDXOR

(b) ANDXOR resultat en place

FIGURE 4.2 – Instructions ANDXOR

b_s est écrit par dessus b_1. Rappelons que la composition de deux permutations résulte en une permutation. Nous remarquons que la porte ANDXOR avec résultat en place est une permutation sur 3 bits. Dans un circuit 8 bits vide, les 5 autres fils ne sont pas modifiés, et représentent donc l'identité. Dans ces conditions, un circuit 8 bits composé d'une porte ANDXOR avec résultat en place est une permutation. Ainsi, rajouter une telle porte à un circuit réalisant une permutation donne aussi une permutation.

bits d'entrée $b_3b_2b_1$	bits de sortie $b_3b_2b_s$
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

TABLE 4.2 – Table ANDXOR

De manière analogue, xorer deux bits avec "résultat en place" a les mêmes propriétés.

4.6.2 Recherche aléatoire

La première méthode consiste à construire des circuits en ajoutant des instructions de manière aléatoire en partant du circuit vide. Nous utilisons des portes ANDXOR avec résultat en place comptant pour deux portes, et une petite proportion de XOR. Nous avons décidé, de par la nature des portes ANDXOR avec résultat en place, de n'utiliser que huit registres. Le calcul

des propriétés cryptographiques d'un circuit est effectué à chaque nouvelle instruction lors de la construction du circuit. Une liste des circuits les plus petits réalisant chaque triplet de critères est tenue à jour. Après quelques essais, nous avons remarqué que la probabilité de trouver des circuits ayant de bonnes propriétés cryptographiques diminue avec le nombre de portes. Plus précisément, si l'on considère deux circuits de même taille, l'un ayant de bonnes qualités cryptographiques, et l'autre de mauvaises, il est très peu probable que l'ajout de portes aux deux circuits ne rende le second meilleur que le premier. Nous avons donc opté pour une recherche par étapes. Premièrement, des recherches aléatoires avec un petit nombre maximum de portes sont lancées. Les meilleurs circuits sont sélectionnés et forment la base des recherches suivantes pour un petit nombre de portes en plus. Ce processus est ensuite répété jusqu'à avoir un grand nombre de portes.
Nous avons lancé beaucoup de recherches avec différents jeux de paramètres, et il s'avère que la progression par étapes donne effectivement de meilleurs résultats. Après des semaines de calculs sur une douzaine de postes différents, nous avons obtenu des "records" (qui nous sont propres) pour certaines propriétés sur des circuits allant de 10 à 50 portes environ. Ces records sont présentés en table 4.3 dans la colonne "recherche aléatoire". Chaque ligne correspond à un nombre de portes, et la colonne "recherche aléatoire" présente les meilleures propriétés différentielles et linéaires atteintes lors de notre recherche pour ce nombre de portes. Certaines lignes présentent plusieurs propriétés lorsqu'elles ne sont pas comparables entre elles. Le degré algébrique est, lui, toujours égal à 7 à partir de 22 portes.

4.6.3 Recherche génétique

Les algorithmes génétiques (GA) sont des méthodes heuristiques qui se basent sur le principe de l'évolution telle que présentée par Darwin. Ils permettent généralement de trouver de bonnes solutions à des problèmes d'optimisation où des méthodes déterministes peinent à y arriver du fait du trop grand espace à explorer.

Algorithmes génétiques

Les GA font évoluer une population d'individus représentés par des chromosomes, et chacun représente une solution au problème à optimiser. Ils se composent généralement de plusieurs phases :

— Création de la population initiale, dite première génération

— Calcul du score de la qualité (fitness) de chaque individu

— Sélection, sur la base de leur qualité, d'individus parents amenés à se reproduire

— Création d'enfants par croisement des parents, et transformation en individus solutions si nécessaire

— Mutations aléatoires des enfants

— Renouvellement de la diversité en incluant des individus aléatoires (optionnel)

— Imitation de la sélection naturelle en éliminant des individus de mauvaise qualité, gardant des individus de bonne qualité à la génération suivante

— Continuer jusqu'à avoir de bonnes solutions

Application à la recherche de circuits 8 bits

Les GA supposent que croiser des individus de moyenne ou bonne qualité peut donner d'encore meilleures solutions en combinant des sous-parties de chacun des parents. Dans ces conditions, la qualité des individus augmente au cours des générations. Un aspect particulièrement important des GA est la manière dont les chromosomes encodent les solutions du problème. Les GA ont déja été utilisés afin de rechercher de bonnes S-boxes, mais à notre connaissance seulement pour optimiser leurs critères cryptographiques, sans considérer le circuit associé. Nous pouvons citer la représentation par table des individus, un chromosome représentant la liste ordonnée des sorties. Dans notre cas, nous préférons encoder nos S-boxes par leur circuit. Un chromosome est une liste ordonnée d'instructions, définissant entièrement un circuit. Cet encodage procure deux avantages : de cette manière il est possible de compter la taille du circuit dans la fonction de qualité, ce qui permet d'en optimiser la compacité. Deuxièmement, il est possible que cela préserve des structures internes au circuit, favorisant de bonnes propriétés cryptographiques.

Pour qu'un individu soit solution, le circuit encodé par son chromosome doit être une permutation. Un obstacle majeur à l'application des GA avec cet encodage particulier est la transformation des enfants en solution. Heureusement, comme nous avons pû le voir, les portes hybrides ANDXOR avec résultat en place permettent d'obtenir systématiquement une permutation si ajoutée à une permutation. Puisque c'est le cas n'importe où dans le circuit, le croisement de portions de circuits solutions résulte systématiquement en une solution.

Les algorithmes génétiques dans la littérature sont nombreux, utilisent un grand nombre de paramètres et de méthodes, et aucune n'est supérieure aux autres dans le cas général. Nous avons donc choisi d'en utiliser certaines avec différents jeux de paramètres afin de trouver la meilleure configuration possible. Voici les différentes méthodes que nous avons utilisés. Remarquons que chaque phase, comme décrite dans le paragraphe 4.6.3, est indépendante. Plusieurs méthodologies peuvent exister pour une phase et généralement peuvent se substituer sans qu'il ne soit nécessaire de changer les autres. Nous nous sommes attachés à tester toutes les combinaisons possibles des méthodes que nous avons implémentées.

Nous détaillons ici les méthodes, options et paramètres que nous avons utilisées. Le but final de

cette recherche est de minimiser la taille en portes du circuit, son uniformité différentielle et sa linéarité, et de maximiser son degré.

Première génération La première génération consiste en un tirage aléatoire de circuits tous de la même taille. Plus tard, nous avons aussi choisi de sélectionner de bons circuits issus de recherches précédentes.

Fonction de qualité La fonction de qualité est un des aspects clé des GA. Elle est notamment utilisée lors de la sélection des parents et de la sélection des individus survivants qui feront partie de la génération suivante. L'optimisation multicritère rend la définition de cette fonction délicate. Nous avons choisi deux méthodes :

— Méthode 1 : Fonction pondérée des critères

— Methode 2 : Méthode de classement par rang

La fonction la plus simple est une fonction pondérée des critères :

$$fit = \omega_1 \cdot n + \omega_2 \cdot deg + \omega_3 \cdot \delta + \omega_4 \mathcal{L}$$

avec fit le score de qualité du circuit, n le nombre de portes, deg le degré, δ l'uniformité différentielle et $\mathcal{L}$ la linéarité. Le problème d'une telle méthode est d'établir le jeu de pondérations. Un moyen d'y parvenir est de lancer des exécutions ayant des pondérations aléatoires, puis après observation des résultats et par essais successifs, de se rapprocher des valeurs donnant de bons résultats. Souvent jugée non satisfaisante, la communauté a cherché à utiliser d'autres méthodes.

La seconde méthode consiste à classer les individus par rang, puis d'attribuer un score de qualité dégressif en fonction de ce dernier. L'une des premières propositions de ce principe introduit la notion de rang de dominance. Dans notre cas, cette dernière était mal adaptée et faisait converger les populations vers des circuits ayant peu de portes, mais de très mauvaises qualités. Nous donnons ici notre propre définition de dominance. Soient deux individus I_1 et I_2, et leurs jeux de n_c critères respectifs $C = \{c_1, c_2, ..., c_{n_c}\}$. Chacun possède donc son vecteur de critères, respectivement $\{c_1^1, c_2^1, ...c_{n_c}^1\}$ et $\{c_1^2, c_2^2, ...c_{n_c}^2\}$. Nous noterons que le critère c_i^1 est meilleur que c_i^2 $c_1^i > c_2^i$. Si c^i doit être minimisé, meilleur signifie de valeur inférieure, et inversement. I_1 domine I_2 noté $I_1 > I_2$ ssi $\#\{i | c_i^1 > c_i^2\} > \frac{n_c}{2}$, à savoir que le nombre de critères où I_1 est meilleur que I_2 est supérieur à la moitié. Nous ne retiendrons que cette nouvelle définition de dominance à présent. Les individus de rang 1 sont ceux qui ne sont dominés par aucun autre. Ceux de rang 2 sont ceux dominés par aucun autre n'étant pas de rang 1, etc. Cette définition permet de réduire la pression exercée par le nombre de portes et de garder des populations plus diversifiées.
Chaque individu d'un rang se voit attribuer le même score de qualité, le score le plus élevé étant

attribué au rang 1. Nous avons utilisé deux fonctions afin d'établir la qualité de chacun des rangs. La première consiste à attribuer un score proportionnel à l'inverse du rang. Le score de qualité d'un individu x de rang r_x est :

$$fit_x = \frac{\frac{1}{r_x}}{\sum_{i=1} \frac{1}{r_i}}$$

Cette méthode exerce une très forte pression de sélection sur les individus et avantage grandement ceux de rang très faible.
La seconde méthode de calcul de la qualité en fonction du rang est la suivante :

$$fit_x = \frac{r_{max} - r_x + 1}{\sum_{i=1} r_{max} - r_i + 1}$$

Avec r_{max} le rang maximum atteint par un/des individu(s) de la population. Cette méthode présente beaucoup moins de disparités entre les scores de qualité puisque la différence entre deux scores de rangs consécutifs est la même, à savoir $\frac{1}{\sum_{i=1} r_{max} - r_i + 1}$.

Sélection des parents Pour sélectionner les individus qui se reproduiront, nous avons utilisé deux méthodes. La première est le classique tirage de type roulette. Chaque individu x se voit attribuer une probabilité de tirage $p(x)$ proportionnelle à son score de qualité telle que

$$p(x) = \frac{fit_x}{\sum_{i=1} fit_i}$$

Les individus sont tirés au hasard en suivant cette distribution de probabilités.
La seconde méthode est celle du tournoi. Nous avons opté pour le tournoi à deux. Dans ce tournoi, deux individus sont tirés uniformément, puis celui dont le score de qualité est le plus élevé gagne le droit de se reproduire. Des tournois sont joués jusqu'à ce que suffisamment de parents soient tirés. Cette méthode permet généralement d'exercer moins de pression de sélection que la roulette dans le cas où quelques individus possèdent des scores beaucoup plus élevés que les autres. Dans les deux méthodes, le choix peut être fait de permettre ou pas à un individu d'être parent plusieurs fois.

Le croisement, ou reproduction Suite à la sélection des parents, des couples sont formés. Chaque couple de parents donne deux enfants grâce à un croisement de portions de leurs chromosomes respectifs. Comme expliqué plus haut, les chromosomes sont les instructions ordonnées constituant le circuit. Nous avons choisi trois méthodes de croisement. La première, présentée en figure 4.3, découpe les individus en trois portions (potentiellement seulement deux lorsque l'un des indices est celui d'une instruction à l'extrémité d'un chromosome) en tirant au hasard deux indices qui délimitent respectivement le début et la fin de la seconde. Ce tirage a pour contrainte

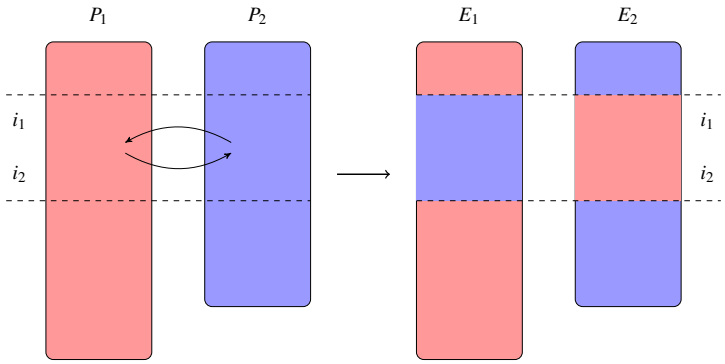


FIGURE 4.3 – Croisement : méthode 1

de former au moins deux portions, c'est-à-dire que les indices ne peuvent pas être à la fois 1 et n_c. Ces indices sont les mêmes pour les deux parents et le croisement interchange les secondes portions des deux individus, c'est-à-dire effectue l'échange horizontal des secondes portions. Soient deux individus parents P_1 et P_2 constitués respectivement des instructions $\{g_1^1...,g_{n_c^1}^1\}$ et $\{g_1^2...,g_{n_c^2}^2\}$. Les indices i_1 et i_2 sont tirés aléatoirement tels que :

$$0 \leq i_1 < min(n_c^1, n_c^2)$$
$$i_1 < i_2 \leq min(n_c^1, n_c^2)$$
$$0 < (i_2 - i_1 + 1) < min(n_c^1, n_c^2)$$

Ces deux parents donnent deux enfants E_1 et E_2 respectivement constitués de
$\{g_1^1,...,g_1^{i_1-1},g_2^{i_1},g_2^{i_1+1},...,g_2^{i_2},g_1^{i_2+1},...,g_1^{n_c^1}\}$ et
$\{g_2^1,...,g_2^{i_1-1},g_1^{i_1},g_1^{i_1+1},...,g_1^{i_2},g_2^{i_2+1},...,g_2^{n_c^2}\}$.

La seconde méthode, présentée en figure 4.4, est assez similaire à la première puisqu'elle découpe, elle aussi, les parents en 3 portions. Cependant, au contraire de la méthode précédente, celle-ci n'utilise pas les mêmes indices pour les deux individus. La seconde portion est de même taille pour les deux individus. Soient les deux mêmes circuits que précédemment P_1 et P_2. Une taille de portion t et les deux indices i_1^1 et i_1^2 de début de seconde portion sur chacun des circuits sont tirés tels que :

$$1 \leq t < min(n_c^1, n_c^2)$$
$$1 \leq i_1^1 \leq (n_c^1 - t + 1)$$
$$1 \leq i_1^2 \leq (n_c^2 - t + 1)$$

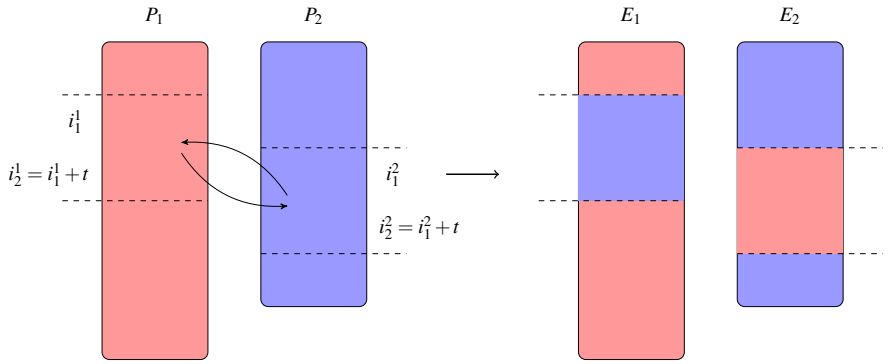


FIGURE 4.4 – Croisement : méthode 2

et les indices de fin de secondes portions sont déduits tels que :

$$i_2^1 = i_1^1 + t$$
$$i_2^2 = i_1^2 + t$$

Les deux enfants sont donc constitués respectivement de
$\{g_1^1, ..., g_1^{i_1^1-1}, g_2^{i_1^2}, g_2^{i_1^2+1}, ..., g_2^{i_2^2}, g_1^{i_2^1+1}, ..., g_1^{n_c^1}\}$ et
$\{g_2^1, ..., g_2^{i_1^2-1}, g_1^{i_1^1}, g_1^{i_1^1+1}, ..., g_1^{i_2^1}, g_2^{i_2^2+1}, ..., g_2^{n_c^2}\}$

Ces deux méthodes ont pour particularité de conserver des séries d'instructions puisque les enfants sont constitués de blocs d'instructions des parents. La seconde a pour avantage de permettre à des instructions de "monter" ou "descendre" dans la série d'instruction de l'enfant.

La troisième méthode, présentée en figure 4.5, consiste en un tirage de 2 masques binaires complémentaires qui associent respectivement à chaque enfant l'instruction d'un parent ou de l'autre. Sans perte de généralité, considérons que I_1 soit plus long que I_2. Pour des raisons de clarté dans l'explication suivante, les masques binaires sont tirés dans l'espace $\{1,2\}$ et non pas $\{0,1\}$. Soit le masque binaire aléatoire de taille n_c^2, $M_1 = \{m_1^1, m_2^1, ..., m_{n_c^2}^1\}$ et son complémentaire $M_2 = \{m_1^2, m_2^2, ..., m_{n_c^2}^2\}$. Le premier enfant est constitué de $\{g_1^{m_1^1}, g_2^{m_2^1}, ..., g_{n_c^2}^{m_{n_c^2}^1}, g_{n_c^2+1}^1, ..., , g_{n_c^1}^1\}$, et le second de $\{g_1^{m_1^2}, g_2^{m_2^2}, ..., g_{n_c^2}^{m_{n_c^2}^2}\}$.

Mutation Les enfants subissent généralement une étape de mutation afin de diversifier et de créer de nouveaux gènes. La probabilité de mutation est généralement faible. Nous en avons implémenté quatre.

La première méthode de mutation est la modification aléatoire d'une instruction. Soit un individu I constitué des instructions $\{g_1, g_2, ..., g_i, ..., g_{n_c}\}$. La première mutation modifie la série d'instructions en $\{g_1, g_2, ..., g_i', ..., g_{n_g}\}$ avec $i \in [1; n_g]$ un indice aléatoire et g_i' une instruction quelconque avec résultat en place.

La seconde méthode consiste à insérer une instruction aléatoire g' à l'indice aléatoire $i \in [1, n_c + 1]$.

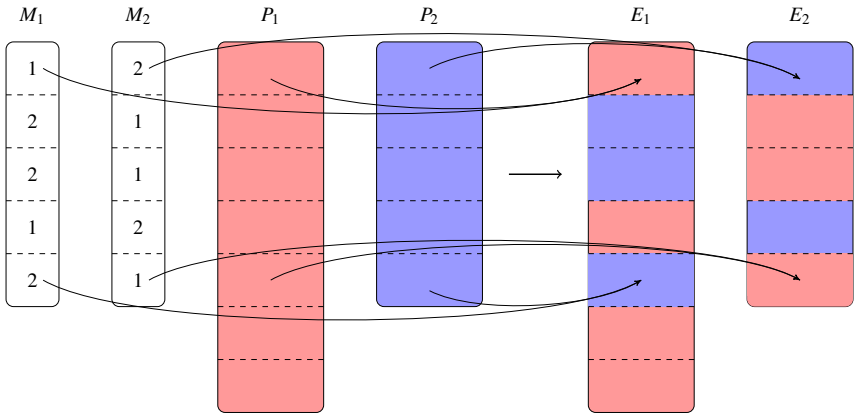


FIGURE 4.5 – Croisement : méthode 3

Soit I comme défini ci-dessus, sa nouvelle série d'instructions est $\{g_1, g_2, ..., g_{i-1}, g'_i, g_i ..., g'_{n_c}\}$.
La troisième méthode élimine une instruction au hasard. L'individu I voit sa série modifiée en $\{g_1, g_2, ..., g_{i-1}, g_{i+1}, ..., g_{n_c}\}$ avec $i \in [1; n_c]$ un indice aléatoire. Cette mutation est le seul moyen de diminuer le nombre de portes d'un circuit.
La dernière mutation permet de modifier l'ordre des instructions d'un individu sans modifier ses critères. Notons d'abord qu'il existe certaines instructions consécutives qui ne s'influencent pas les unes les autres et sont dites indépendantes. Deux instructions consécutives $g_1 = \{p_1, i1_1^1, i1_1^2, i1_1^3, o_1\}$ et $g_2 = \{p_2, i1_2^1, i1_2^2, i1_2^3, o_2\}$ avec p le type de porte, i les registres d'entrée et o le registre de sortie, sont indépendantes, noté $g_1 \perp\!\!\!\perp g_2$ si et seulement si aucun des deux registres de sortie des deux instructions ne font partie des registres d'entrée de l'autre, autrement dit :

$$\begin{cases} \{i1_1, i2_1, i3_1\} \cap \{o_2\} = \varnothing \\ \{i1_2, i2_2, i3_2\} \cap \{o_1\} = \varnothing \end{cases}$$

Dans ce cas, puisqu'aucune instruction ne modifie les entrées de l'autre, leur ordre n'influence pas les équations finales. Il est donc possible de les interchanger sans aucune conséquence sur le circuit qui sera complètement équivalent au niveau de la table et donc de ses critères. Deux exemples de telles instructions sont présentés en figure 4.6. Nous avons implémenté une mutation d'un individu qui consiste en la recherche de deux instructions consécutives indépendantes, puis si deux telles instructions existent, les interchange. Soit $G = \{(g_i, g_{i+1}) \mid g_i \perp\!\!\!\perp g_{i+1}\}$ l'ensemble des couples d'instructions consécutives indépendantes d'un individu I défini comme précédemment. Si $G \neq \varnothing$, alors un couple $(g_x, g_{x+1}) \in G$ est tiré au hasard, puis l'individu voit sa série d'instructions modifiée en $\{g_1, g_2, ..., g_{x+1}, g_x, ..., g_{n_c}\}$.

Cette mutation a pour but de changer l'ordre des instructions afin de permettre d'autres possibilités de croisement, notamment lorsque celui-ci se fait horizontalement (méthodes 1 et 3).

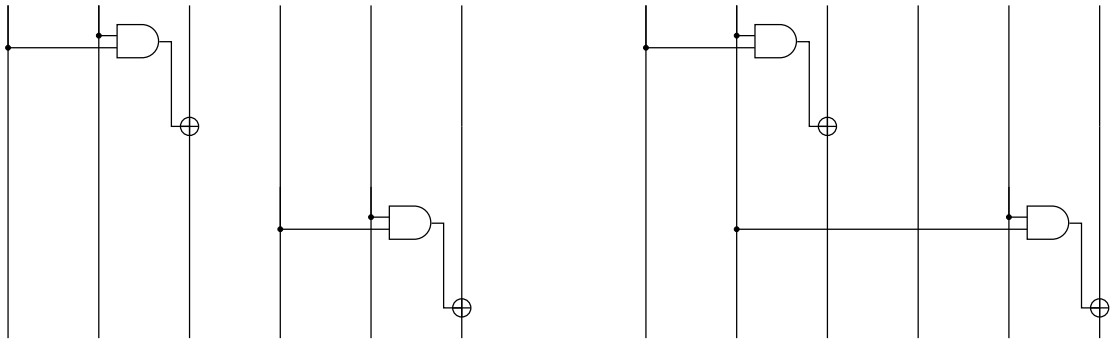


FIGURE 4.6 – Exemples de portes indépendantes

Diversification L'étape suivante est optionnelle mais permet à l'algorithme de ne pas converger trop vite vers une population qui n'évolue plus, et qui n'apporte donc aucune nouvelle solution. Elle consiste à injecter de nouveaux gênes dans la population, par l'ajout de nouveaux individus par exemple. Nous donnons donc la possibilité à l'algorithme de supprimer les individus doublons afin de ne garder qu'un seul représentant d'un même circuit, et de les remplacer par des individus créés aléatoirement. Ainsi la population ne peut pas se réduire à quelques individus tous identiques qui empêcheraient l'algorithme de progresser. De plus, ce but peut aussi être atteint lors de l'étape suivante et sera expliqué dans le prochain paragraphe.

Selection naturelle Un cycle se termine par la selection naturelle, étape qui sélectionne les individus qui survivront et qui constitueront la génération suivante. Différentes méthodes existent, et la première était de sélectionner uniquement des individus parmi la population d'enfants créés. Afin de ne jamais écarter les meilleurs circuits, nous avons choisi une méthode élitiste qui semble le mieux convenir à l'optimisation multicritère. Nous considérons donc à la fois la génération en cours et les enfants créés comme candidats à la génération suivante. La survie des individus est assurée par un classement selon le fitness puis sélection des meilleurs dans l'ordre. Ainsi, les meilleurs sont assurés de survivre et de participer à la génération suivante. Dans notre cas, le nombre de survivants est égal au nombre d'individus au début de la génération, la taille de la population de chaque génération restant identique tout au long de la recherche. Pour assurer plus de diversité comme énoncé dans le paragraphe précédent, il est possible de réduire légèrement la taille de la population survivante, puis de la compléter avec des individus crées aléatoirement. Une fois cette étape terminée, un nouveau cycle peut démarrer.

L'algorithme calcule les propriétés cryptographiques de tous les enfants à chaque génération et tient à jour une liste des meilleurs critères atteints pour un nombre de portes donné.

4.6.4 Résultats de la recherche par algorithme génétique

Nous avons lancé beaucoup de recherches en utilisant un grand nombre de combinaisons de paramètres. Nous avons observé que procéder par étapes, à savoir fournir comme population initiale des circuits issus de recherches précédentes ayant de bonnes propriétés cryptographiques et compactes en plus d'individus aléatoires, permet de trouver d'encore meilleures solutions. Nous avons alloué moins de temps à cette méthode qu'à celle aléatoire, néanmoins, de bons résultats ont quand même été trouvés.

4.7 Résultats / Comparaison

Les résultats trouvés sont présentés dans la table 4.3. Nous y présentons les critères cryptographiques atteints pour un nombre de portes donné pour les deux méthodes présentées. Les critères en gras indiquent que les meilleurs atteints pour un certain nombre de portez ne l'ont été qu'avec une seule des deux méthodes.

Nous exprimons la qualité d'une S-box par le triplet $(deg, \delta, \mathcal{L})$. Nous essayons de nous comparer à la littérature en ne prenant en compte que le nombre de portes dont les circuits sont constitués, indépendamment du fait que ce soit des portes XOR ou AND.

Dans [CDL15], Canteaut et al. proposent une S-box $(6,8,64)$ construite à partir d'un schéma de Feistel qui requiert 38 portes. Nous avons trouvé une S-box presque équivalente, dont le circuit est présenté en figure 4.7 et sa table en 4.4, composée de 40 portes et qui atteint les qualités $(7,8,64)$. Notons tout de même que leur construction présente moins de portes non-linéaires, alors que nous avons un nombre égal de XOR et de AND, ce qui, vraisemblablement, augmente le coût du circuit une fois thresholdé. Cependant, ils utilisent une porte OR qui, à notre connaissance, n'est pas thresholdable de manière triviale.

Comparé à la proposition de Lilliput-TBC, la conclusion est à peu près équivalente. La S-box Lilliput atteint les qualités $(6,8,64)$, est composée de 39 portes et est optimisée pour le threshold. La S-box présentée en 4.7 a dans les deux cas un degré supérieur pour une ou deux portes de plus.

Grâce à la table 1 de [CDL15], nous pouvons aussi nous comparer aux S-boxes de Robin et Fantomas. Ces algorithmes utilisent des S-boxes $(6,16,64)$ composées de 36 portes, alors que nous avons atteint une S-box $(7,16,64)$ en seulement 30 portes. Dans [BGG+17], Boss et al. proposent plusieurs S-boxes issues de constructions. Malheureusement pour nous, ils ne proposent que des valeurs de surface. Nous essayons tout de même de faire des comparaisons avec leur *raw implementations* grâce aux indications qu'ils donnent concernant la composition de leurs S-boxes ainsi que les tailles minimum des S-boxes quadratiques (voir 4.1). $\mathbf{SB}_1$ utilise 8 itérations de Q_{294} dont la taille minimum est quatre portes. Un minimum de 32 portes leur est donc nécessaire

# portes	degré algébrique	uniformité différentielle (δ) - linéarité ($\mathcal{L}$)	
		recherche aléatoire	recherche génétique
8	5	128 - 256	128 - 256
9	5	128 - 256	128 - 256
10	6	128 - 256	128 - 256
11	6	128 - 256	128 - 256
12	7	128 - 256	128 - 256
13	7	128 - 256	128 - 256
14	7	128 - 256	128 - 256
15	7	128 - 256	128 - 256
16	7	64 - 128	64 - 128
17	7	64 - 128	64 - 128
18	7	64 - 128	64 - 128
19	7	64 - 128	64 - 128
20	7	64 - 128	64 - 128
21	7	64 - 128	64 - 128
22	7	**48 - 128**	64 - 128
23	7	48 - 128	48 - 128
24	7	32 - 128	32 - 128
25	7	32 - 128	32 - 128
26	7	32 - 128	32 - 128
27	7	32 - 96	**32 - 68**
28	7	32 - 76	**32 - 64**
29	7	16 - 128 20 - 76 32 - 68	**16 - 76 24 - 68 32 - 64**
30	7	16 - 76 18 - 72 32 - 64	**16 - 64**
31	7	16 - 72 20 - 68 32 - 64	**16 - 64**
32	7	16 - 64	16 - 64
33	7	16 - 64	16 - 64
34	7	16 - 64	16 - 64
35	7	16 - 64	16 - 64
36	7	12 - 64	12 - 64
37	7	12 - 64	**10 - 80** 12 - 64
38	7	10 - 68 12 - 64	**10 - 64**
39	7	10 - 64	10 - 64
40	7	10 - 64	**8 - 64**
41	7	10 - 64	**8 - 64**
42	7	10 - 64	**8 - 64**
43	7	10 - 64	**8 - 64**
44	7	10 - 64	**8 - 64**
45	7	**10 - 60**	**8 - 64**
46	7	**10 - 60**	**8 - 64**
47	7	8 - 64 **10 - 60**	8 - 64 12 - 60
48	7	8 - 64 10 - 60	8 - 64 10 - 60
...	...	...	...
51	7	**10 - 56**	8 - 64 10 - 60
...	...	...	...
54	7	**8 - 60**	8 - 64 10 - 56
...	...	...	...
57	7	8 - 60	8 - 60

TABLE 4.3 – Nos meilleures S-boxes 8 bits en fonction de la taille du circuit pour les recherches aléatoires et génétiques

pour atteindre les qualités $(6,16,64)$. Encore une fois, nous avons trouvé une S-box $(7,16,64)$ composée de 30 portes. $\mathbf{SB}_3$ est composée de Q_{293}, Q_{299} et une matrice de multiplication dans un schéma de type SPN de 4 itérations. Ne comptant que le coût des S-boxes quadratiques, et en omettant celui de la multiplication (qui est néanmoins non négligeable), ils atteignent les qualités $(7,8,60)$ en 48 portes. Nous obtenons les mêmes qualités en 54. $\mathbf{SB}_6$ atteint les qualités $(7,10,60)$ grâce à une construction similaire et utilise Q_{293} et Q_{294}, en 40 portes. Encore une fois, nous ne comptons pas le potentiellement fort coût de la multiplication. Nous atteignons les mêmes qualités en 45 portes. $\mathbf{SB}_5$ atteint les mêmes qualités grâce à 9 itérations composées de Q_4 et Q_{294}, et sans multiplication, donc 54 portes au minimum. Selon leur table 1, la surface utilisée par $\mathbf{SB}_5$, composée de 54 portes, est inférieure à celle utilisée par $\mathbf{SB}_6$ qui est, elle, composée de 40 portes et de la multiplication. Nous pouvons donc faire l'hypothèse que la multiplication a effectivement un coût assez élevé.
Notons tout de même que la comparaison avec [BGG+17] n'est pas complètement juste puisque les comparaisons sont faites sur des implémentations non masquées, alors que leurs travaux cherchent en grande partie à optimiser l'implémentation masquée. Cependant, nous avons remarqué que dans le cas non masqué, les S-boxes trouvées au cours de nos recherches sont assez comparables, et même parfois meilleures en nombre de portes, à des schémas de construction dédiés. Nous pensons donc que ces recherches ne sont pas dénuées d'intérêt et peuvent conduire à chercher de nouvelles méthodes de construction de S-box. Nous pensons aussi qu'il est possible d'adapter ces méthodes à l'optimisation du coût d'implémentation une fois thresholdée.

Nous présentons aussi un circuit nous paraissant intéressant : une S-box ayant les propriétés $(7,10,80)$ a été atteinte en 37 portes. Il s'agit de notre circuit le plus court atteignant une uniformité différentielle de 10. Le circuit est visible en figure 4.8, et la table en 4.5.

Conclusion

Dans ce chapitre, nous avons implémenté et étudié différentes méthodes afin de chercher des S-boxes compactes ayant de bonnes propriétés cryptographiques.
Pour les S-boxes 4 bits, nous avons adapté une approche systématique utilisée précédemment dans [UDCI+11, CDL15] en restreignant le jeu de portes au AND et XOR afin d'obtenir des circuits facilement thresholdables. Nous avons atteint des circuits de classes optimales en le même nombre de portes minimum que ces travaux. Un de ces circuits a été utilisé dans la construction de la permutation 8 bits de Lilliput-TBC.
En ce qui concerne les S-boxes 8 bits, nous avons choisi d'explorer l'espace des circuits plutôt que d'utiliser des constructions. En plus de nous donner de bonnes chances de trouver de bonnes solutions, cela nous a permis de trouver facilement des circuits de degré algébrique maximum. La méthode aléatoire construit des circuits en partant du bas, et la recherche génétique cherche à optimiser les différents critères pour des implémentations non protégées. Après des semaines de

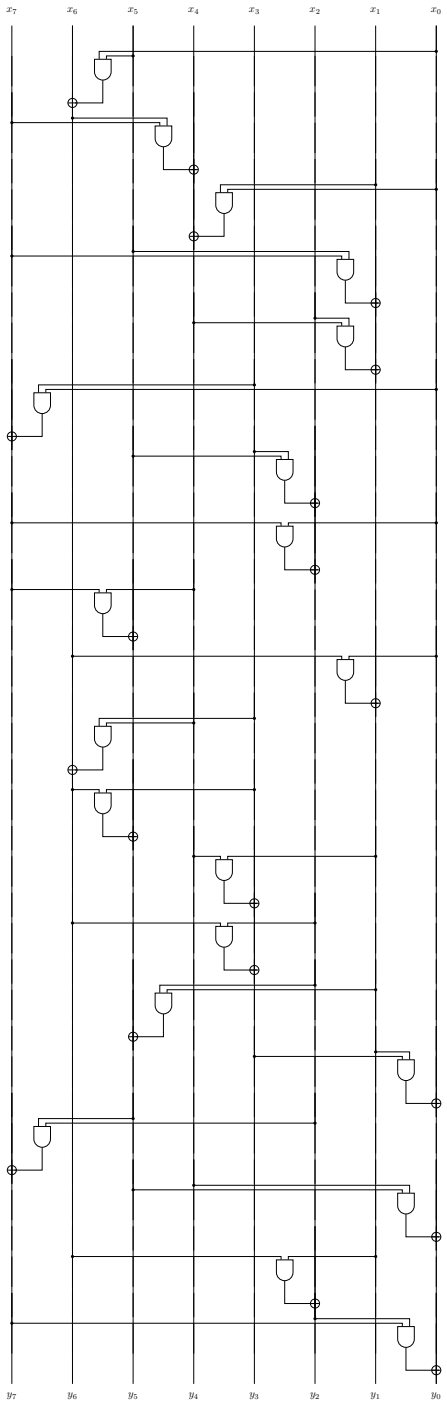


FIGURE 4.7 – Circuit 40 portes de la S-box $(7, 8, 64)$ de la table 4.4

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	01	02	1A	04	05	A7	15	08	8C	0B	7B	0C	89	AE	D9
10	10	11	1B	03	BF	BE	14	A6	79	D4	77	2E	5B	D6	F4	8A
20	20	67	22	70	A5	4A	06	53	AD	CF	0F	99	28	63	2B	36
30	31	7F	3A	61	1F	FD	B4	EC	54	93	FA	C9	56	9C	58	C4
40	40	47	46	51	4C	EA	EB	F2	68	C3	6F	3C	E5	EF	42	B2
50	50	5E	5F	41	F3	5D	5C	4D	18	96	12	65	B6	B8	1C	E9
60	60	21	66	3B	ED	A4	4B	B5	44	A9	E2	F7	48	2D	4F	74
70	71	30	7E	23	52	1E	FC	07	BC	F8	16	AA	33	DA	39	8F
80	80	84	82	9F	85	81	26	B0	88	09	8B	76	8D	0D	2F	F5
90	B1	34	BA	27	9E	BB	35	83	D8	78	D7	0A	7A	5A	D5	AF
A0	A2	DC	A0	CA	87	DF	24	E6	8E	BD	2C	E3	AB	32	A8	4E
B0	9B	6D	90	72	95	E1	3E	D1	DB	45	75	17	F9	49	F6	38
C0	F1	D3	FF	CC	D2	F0	7D	C1	B9	13	B3	69	97	1D	3D	E4
D0	C0	6A	C7	7C	CD	C6	6B	FE	E8	6E	EE	19	64	43	C2	B7
E0	DE	86	D0	94	DD	A3	73	9A	37	0E	9D	55	98	2A	92	57
F0	E7	3F	E0	25	CB	91	6C	A1	62	FB	C5	AC	CE	59	C8	29

TABLE 4.4 – Table correspondant à la S-box 4.7 avec $deg = 7$, $\delta = 8$ et $\mathcal{L} = 64$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	63	02	11	06	BD	04	93	0B	09	08	5A	85	14	87	6F
10	30	13	52	21	DB	91	B8	DE	5B	39	38	0A	AD	2C	EE	16
20	20	01	64	FD	EF	07	C3	A2	6B	6A	AF	37	5E	BC	40	4E
30	31	10	ED	74	EA	92	7B	2A	1A	1B	47	FF	84	17	23	5C
40	44	BE	CC	27	22	43	E1	C2	CF	75	46	CD	A3	58	60	6D
50	54	CE	DC	77	7F	2E	49	3B	BF	65	36	DD	4A	C0	7C	41
60	EC	45	62	53	42	61	AC	A9	26	AE	28	19	C1	78	3D	86
70	55	FC	03	12	4D	3F	05	90	FE	76	29	18	E0	69	DF	15
80	99	A8	9A	1F	1E	E6	1D	9C	83	96	81	A0	8C	8A	8F	25
90	2D	1C	6E	EB	57	9F	35	A4	D7	E2	B5	94	B0	67	D2	89
A0	DA	88	3E	C4	A5	0F	4B	59	A1	D6	E5	BB	24	B1	F9	70
B0	3C	0E	D0	CA	56	8D	F5	B6	82	95	4F	F1	9D	98	E8	51
C0	7D	E4	C5	6C	D9	CB	2B	7A	C7	FB	7E	F2	68	D5	BA	50
D0	E9	F0	D1	F8	71	32	F6	B7	B3	2F	AA	A6	F4	79	73	C8
E0	A7	4C	B9	5F	FA	AB	E7	34	5D	B2	E3	80	48	F7	66	9E
F0	D8	F3	8B	0D	72	33	0C	8E	C6	C9	B4	97	3A	D4	D3	9B

TABLE 4.5 – Table correspondant à la S-box 4.8 avec $deg = 7$, $\delta = 10$ et $\mathcal{L} = 80$.

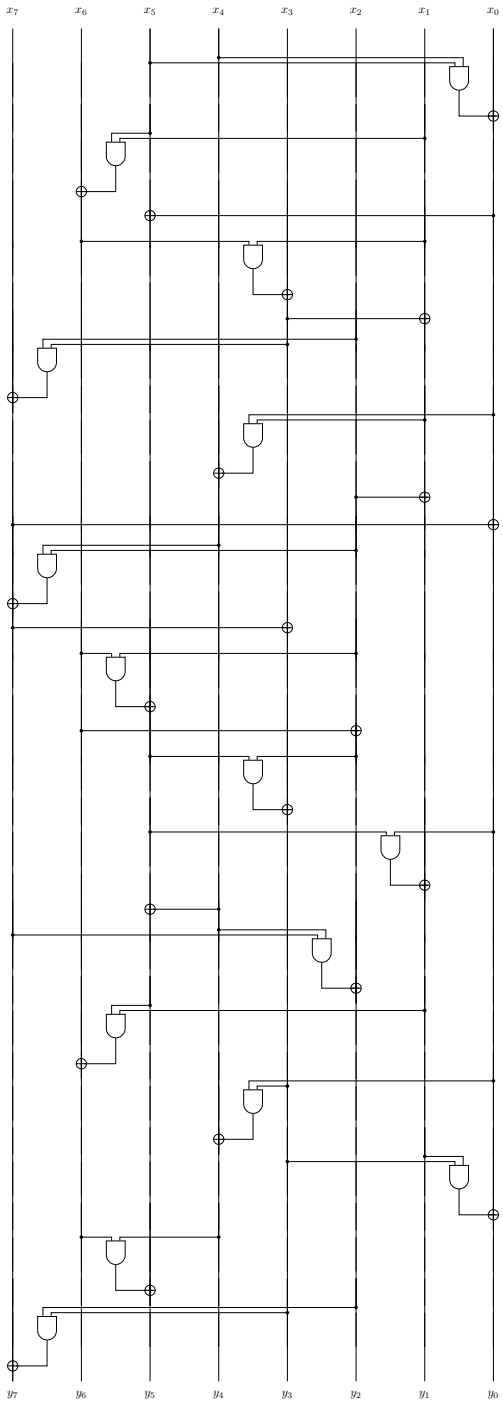


FIGURE 4.8 – Circuit 37 portes de la S-box $(7, 10, 80)$ de la table 4.5

calculs, nous avons atteint des circuits compétitifs avec des S-boxw connues.
Alors que nous nous sommes restreints à optimiser les qualités et la compacité d'implémentations non protégées mais facilement thresholdables, nous pensons qu'il est possible de mener l'étude dans le cas protégé à l'ordre 1. Nous pensons que ce travail peut ouvrir la voie à d'autres travaux similaires.

Chapitre 5

Conclusion

Au cours de cette thèse, deux axes principaux ont été étudiés.
Premièrement, dans le chapitre 3, une étude plus approfondie des attaques par canaux cachés portant sur les distributions jointes a été faite, ce qui a mené au développement d'une nouvelle technique de calcul des poids de Hamming issus d'observations de consommations testée expérimentalement, mais aussi à de nouvelles variantes d'attaques pouvant utiliser à la fois de nouvelles variables, ainsi que l'utilisation simultanée de plus de deux variables. Nous nous attardons notamment sur le comportement particulier de l'observation de m et de x qui informe seulement sur le poids de Hamming de la clé, mais dont l'efficacité est très intéressante comparée à l'observation de y. Nous avons ensuite étudié des schémas d'implémentations masquées d'ordre 1 et avons proposé des attaques pouvant les compromettre, tout d'abord dans un cas simple où le masque est identique (mais plus généralement lorsqu'il est dépendant) pour les variables observées, puis plus loin dans le cas plus complexe où les masques sont différents d'une variable à l'autre, mais que les jeux sont identiques sur plusieurs positions de la clé étendue.
Le chapitre 4 quant à lui résulte de la réflexion que nous avons initiée avec le projet PACLIDO. Nous avons tout d'abord voulu créer une S-box 4 bits compacte lorsque thresholdée, puis nous avons ensuite cherché à créer des circuits 8 bits en explorant directement dans l'espace des circuits formés des portes AND et XOR. Grâce à l'astuce des portes hybrides avec résultat en place, deux algorithmes de recherche, un aléatoire et un autre génétique, nous ont permis d'établir une liste des qualités cryptographiques que nous avons atteintes en fonction du nombre de portes utilisées pour des circuits non protégés mais facilement thresholdable. Une suite possible à ces travaux est de rechercher des circuits compacts une fois thresholdés.

Bibliographie

[ABC+] Alexandre Adomnicai, Thierry P. Berger, Christophe Clavier, Julien Francq, Paul Huynh, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud, and Gaël Thomas. Lilliput-AE : a New Lightweight Tweakable Block Cipher for Authenticated Encryption with Associated Data. Submission to the NIST Lightweight Cryptography Standardization Process.

[AVFM07] Frederic Amiel, Karine Villegas, Benoit Feix, and Louis Marcel. Passive and Active Combined Attacks : Combining Fault Attacks and Side Channel Analysis. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 92–102. IEEE, 2007.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis With a Leakage Model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[BGG+17] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit Sboxes with Efficient Masking in Hardware (extended version). *J. Cryptographic Engineering*, 7(2) :149–165, 2017.

[Bil15] Begül Bilgin. Threshold Implementations : as Countermeasure Against Higher-Order Differential Power Analysis. 2015.

[Blo11] Céline Blondeau. *La cryptanalyse différentielle et ses généralisations. (Differential cryptanalysis and its generalizations)*. PhD thesis, Pierre and Marie Curie University, Paris, France, 2011.

[BNN+15] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold Implementations of Small S-Boxes. *Cryptography and Communications*, 7(1) :3–33, 2015.

[BP09] Joan Boyar and René Peralta. New logic minimization techniques with applications to cryptology. *IACR Cryptology ePrint Archive*, 2009 :191, 2009.

[BS91] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of CRYPTOLOGY*, 4(1) :3–72, 1991.

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[Can05] David Canright. A Very Compact S-Box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 441–455, 2005.

[Can16] Anne Canteaut. Lecture Notes on Cryptographic Boolean Functions. *Inria, Paris, France*, 2016.

[CDL15] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of Lightweight S-Boxes using Feistel and MISTY structures (Full Version). *IACR Cryptology ePrint Archive*, 2015 :711, 2015.

[CG00] Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking Against Differential Power Analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 231–237. Springer, 2000.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Wound Approaches to Counteract Power-Analysis Attacks. In *Annual International Cryptology Conference*, pages 398–412. Springer, 1999.

[CJS05] John A. Clark, Jeremy L. Jacob, and Susan Stepney. The Design of S-Boxes by Simulated Annealing. *New Generation Comput.*, 23(3) :219–231, 2005.

[CMW14] Christophe Clavier, Damien Marion, and Antoine Wurcker. Simple Power Analysis on AES Key Expansion Revisited. In *CHES*, 2014.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[DC07] Christophe De Cannière. Analysis and Design of Symmetric Encryption Algorithms. *Doctoral Dissertaion, KULeuven*, 2007.

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

[FIP01] PUB FIPS. 197. *Advanced encryption standard (AES)*, 26, 2001.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 426–442. Springer, 2008.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis the "Duplication" Method. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 158–172. Springer, 1999.

[HTM09] Neil Hanley, Michael Tunstall, and William P Marnane. Unknown plaintext template attacks. In *International Workshop on Information Security Applications*, pages 148–162. Springer, 2009.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing Implementations of Lightweight Building Blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4) :130–168, 2017.

[KJJ+98] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. Introduction to Differential Power Analysis and Related Attacks, 1998.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[KKO13] Oleksandr Kazymyrov, Valentyna Kazymyrova, and Roman Oliynykov. A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent. *IACR Cryptology ePrint Archive*, 2013 :578, 2013.

[Kne17] Karlo Knezevic. Combinatorial Optimization in Cryptography. In *40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017, Opatija, Croatia, May 22-26, 2017*, pages 1324–1330, 2017.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[Kor16] Roman Korkikian. *Side-channel and fault analysis in the presence of countermeasures : tools, theory, and practice*. Theses, PSL Research University, October 2016.

[LB14] Hélène Le Bouder. *A formalism for physical attacks on cryptographic devices and its exploitation to compare and reasearch news attacks*. Theses, Ecole Nationale Supérieure des Mines de Saint-Etienne, October 2014.

[LDL13] Yanis Linge, Cécile Dumas, and Sophie Lambert-Lacroix. Using the joint distributions of a cryptographic function in side channel analysis. *IACR Cryptology ePrint Archive*, 2013 :859, 2013.

[Lin13] Yanis Linge. *Cryptographic and statistical side channel analysis*. Theses, Université de Grenoble, November 2013.

[LP07] Gregor Leander and Axel Poschmann. On the Classification of 4 bit S-Boxes. In *International Workshop on the Arithmetic of Finite Fields*, pages 159–176. Springer, 2007.

[LWWW17] Yang Li, Shuang Wang, Zhibin Wang, and Jian Wang. A Strict Key Enumeration Algorithm for Dependent Score Lists of Side-Channel Attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 51–69. Springer, 2017.

[Man03] Stefan Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In Pil Joong Lee and Chae Hoon Lim, editors, *Information*

Security and Cryptology — ICISC 2002, pages 343–358, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[Mes00a] Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *International Workshop on Fast Software Encryption*, pages 150–164. Springer, 2000.

[Mes00b] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 238–251. Springer, 2000.

[Mil98] William Millan. How to Improve the Nonlinearity of Bijective S-Boxes. In *Information Security and Privacy, Third Australasian Conference, ACISP'98, Brisbane, Queensland, Australia, July 1998, Proceedings*, pages 181–192, 1998.

[MM17] Thorben Moos and Amir Moradi. On the Easiness of Turning Higher-Order Leakages Into First-Order. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 153–170. Springer, 2017.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *Cryptographers' Track at the RSA Conference*, pages 351–365. Springer, 2005.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2) :292–321, 2011.

[oST17] National Institute of Standards and Technology. Lightweight Cryptography, January 2017. https://csrc.nist.gov/Projects/Lightweight-Cryptography.

[Per17] Léo Perrin. *Cryptanalysis, Reverse-Engineering and Design of Symmetric Cryptographic Algorithms*. PhD thesis, University of Luxembourg, 2017.

[Pha02] Raphael Chung-Wei Phan. Mini advanced encryption standard (mini-AES) : a testbed for cryptanalysis students. *Cryptologia*, 26(4) :283–306, 2002.

[PJ19] Stjepan Picek and Domagoj Jakobovic. On the design of S-box constructions with genetic programming. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 395–396. ACM, 2019.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking Against Side-Channel Attacks : A Formal Security Proof. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 142–159. Springer, 2013.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Transactions on computers*, 58(6) :799–811, 2009.

[Sha49] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell system technical journal*, 28(4) :656–715, 1949.

[Sto16] Ko Stoffelen. Optimizing S-Box Implementations for Several Criteria Using SAT Solvers. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 140–160, 2016.

[UDCI+11] Markus Ullrich, Christophe De Canniere, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4× 4-bit S-Boxes. In *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark*, pages 16–17, 2011.

[VBC05] Joel VanLaven, Mark Brehob, and Kevin J. Compton. A Computationally Feasible SPA Attack on AES VIA Optimized Search. In Ryoichi Sasaki, Sihan Qing, Eiji Okamoto, and Hiroshi Yoshiura, editors, *Security and Privacy in the Age of Ubiquitous Computing*, pages 577–588, Boston, MA, 2005. Springer US.

Résumé

La cryptographie est largement utilisée de nos jours, et les attaques par analyse de canaux auxiliaires sont une menace avérée. Afin de développer des contre-mesures adéquates, ces attaques sont constamment étudiées et améliorées. Ce document s'inscrit dans cette démarche et étudie les attaques par distributions jointes qui ont pour particularité de ne nécessiter ni le clair ni le chiffré contrairement aux attaques classiques. Premièrement, de nouvelles variantes qui améliorent cette attaque sont proposées. Elles consistent notamment à observer plus ou différentes variables intermédiaires, et à adapter les distributions théoriques en conséquence. Ensuite, une expérimentation sur microcontrôleur est proposée et valide l'efficacité de l'attaque en général hors simulations. Enfin, plusieurs solutions visant à compromettre des implémentations masquées sont proposées. Elles consistent à tirer profit de la réutilisation de certaines valeurs de masques sur plusieurs variables internes, ce qui résulte en la proposition des attaques quadrivariées.
Un autre axe principal ayant pour sujet la construction de S-boxes compactes 8 bits est exploré dans ce document et a pour origine la participation au projet PACLIDO, dont l'un des buts est le développement d'un algorithme de cryptographie légère. La S-box est un composant essentiel des algorithmes de chiffrement et a pour rôle de les rendre résistants à la cryptanalyse classique. Malheureusement, ce composant est souvent très coûteux en nombre de portes logiques. Nous proposons deux méthodes de recherche de circuits compacts 8 bits composés de portes hybrides ANDXOR, et montrons quelques résultats proches de ceux qui se trouvent dans la littérature.

Abstract

Cryptography is widely used nowadays, and side channel attacks are known threats. In order to develop suitable countermeasures, these attacks are constantly studied and improved. This document is part with this approach and studies joint distributions attacks which do not need the plaintext nor the ciphertext, contrary to classical attacks. First, new variants which improve this attack are proposed. They mainly consist in observing more or different intermediate variables, and adapt theoretical distributions accordingly. Then, a practical experiment is done on a microcontroller which validates the attack apart from simulations. Finally, several solutions that compromise masked implementations are proposed. They consist in using the fact that some masks are reused on several internal variables, ending in the quadrivariate attacks proposal.
An other main theme about 8 bits S-boxes construction is explored in this document, and originated from the participation to the PACLIDO project which aims to develop a lightweight cryptography algorithm. The S-box is a critical component of cryptographic algorithms and acts to make them resilient to classical cryptanalysis. Unfortunately, this component is often costly in terms of logical gates. We propose two methods for searching compact 8 bit circuits composed of hybrid gates ANDXOR, and show some results close to the literature.