Université de Limoges

ÉCOLE DOCTORALE SCIENCES ET INGÉNIERIE POUR L'INFORMATION, MATHÉMATIQUES

Étude de la sécurité d'algorithmes de cryptographie embarquée vis-à-vis des attaques par analyse de la consommation de courant

THÈSE

présentée et soutenue publiquement le 23 octobre 2015

pour l'obtention du

Doctorat de l'Université de Limoges

(spécialité informatique)

par

Antoine Wurcker

Composition du jury

Rapporteurs :	Louis Goubin Sylvain Guilley
Examinateurs :	Benoît Feix Julien Francq
Directeur :	Christophe Clavier

Laboratoire de recherche XLIM UMR CNRS 7252
Thèse financée par la région Limousin

Xlim institut de recherche

Région LIMOUSIN

Mis en page avec la classe thesul.

Remerciements

Je suis heureux de profiter de l'occasion qui m'est donnée ici pour remercier les gens qui m'ont guidé, aidé, soutenu, qui m'ont donné ma chance et fait confiance, et sans qui je n'aurais pas trouvé cette passion qu'est pour moi la recherche et qui me permet de travailler avec plaisir. Ne pouvant, hélas, citer tout le monde, je prie ceux qui ne seraient pas dans cette page de ne pas m'en tenir rigueur.

Je souhaiterai commencer par Christophe qui m'a communiqué sa passion pour la recherche bien avant de diriger ma thèse, qui s'est battu pour que je puisse la réaliser, qui m'a accompagné et fait me sentir comme un collègue de travail tout au long des années et qui m'a contaminé au virus des séquences aliquotes. Je ne vois pas cette thèse que comme la conclusion de 6 années de collaborations entre nous mais plutôt sur la promesse de bien belles années à venir.

Pour mon parcours je repense à beaucoup d'enseignants qui, depuis mon plus jeune age, m'ont permis d'aimer apprendre et de devenir curieux de comprendre les choses. Même si parfois les noms m'échappent je ne vous oublie pas, vous et ce que vous avez pu me transmettre. Je citerai notamment Mr Verrier qui m'a transmis bien des valeurs à travers l'apprentissage de l'athlétisme et Mr Lauron pour son enseignement des mathématiques en classes préparatoires qui m'a permis de mieux voir a quel point cette matière peut être belle.

Je n'oublie pas Jacques-Arthur Weil et Karim Tamine qui m'ont fait confiance pour intégrer leurs formations malgré mon parcours hétéroclite. Je remercie la région Limousin pour m'avoir financé pendant cette thèse et l'université de Limoges, où il a été plaisant d'apprendre mais aussi d'enseigner. À Marc Rybowicz, Jean-Louis Lanet, Julien Iguchi-Cartigny, Benoît Crespin et bien d'autres pour leurs enseignements où chacun, par sa touche personnelle, faisait vivre ses cours et les rendait agréables et passionnants. Pour mon stage de master je remercie Julien Francq pour m'avoir proposé ce sujet passionnant et toute l'équipe d'EADS qui m'a accueilli.

Je remercie également tout le personnel de la faculté et du laboratoire XLIM qui se démène pour que toute cette folle entreprise qu'est une université puisse fonctionner, en particulier Hubert, Julie et Guilhem, Odile et Débora. Un mot aussi pour l'équipe en charge du calculateur CALI qui m'a servi pour les simulations présentes dans mes travaux.

Je remercie aussi tous les collaborateurs que j'ai pu côtoyer pendant ma thèse, Damien en premier lieu mais aussi tous les membres du groupe de travail du projet Marshal+ avec qui les réunions étaient intéressantes et créatives. Je pense particulièrement à Mylène, Georges et Vincent avec qui il a toujours été agréable d'échanger. D'autres doctorants, compagnons dans la même aventure, Aymerick, Guillaume, Youssef, Gaël, Hélène et bien d'autres. Je pense aussi aux étudiants que j'ai pu encadrer pour leurs stages : Quentin, Cuong (qui me pardonnera j'espère cette simplification de son nom), Bathie et bien sûr Romain et Gregory. Enfin je pense à tous ceux que j'ai pu côtoyer au cours des conférences où il est toujours très agréable de partager et de faire germer les idées, mention spéciale à Céline dont les pistes se sont toujours révélées fructueuses.

Sur un plan plus personnel je remercie grandement Cécile, qui a eu la patience de me supporter et dont l'aide m'a été essentielle pendant toutes ces années. Benoît pour tous les bons moments passés, mais il comprendra que je ne m'étende pas plus ici par manque de place. Ianto qui a toujours eu du chien et dont la bonne humeur est contagieuse. Je pense aussi aux bons moments passés avec Olivier, Hugo, Thibaut, Mélie, Vivi, Laure et bien d'autres.

Je pense à ma mère qui m'a montré le goût de réfléchir, discuter, être curieux et autonome, mon frère avec qui il a toujours été agréable de vivre et de partager, et à mon père aussi pour tout ce qu'il a pu m'apporter.

Je termine ces remerciements par les rapporteurs qui ont patiemment relu cette thèse et dont les remarques m'ont permis d'en améliorer la qualité. Louis Goubin qui aura relu ma thèse mais aussi encadré celle de Christophe continuant ainsi une chaîne qu'il faudra tenter de poursuivre si je viens un jour à en encadrer une moi même. Sylvain Guilley qui m'a suivi depuis ma participation à son *DPA contest* et que je remercie pour sa gentillesse pendant toutes ces années. Je remercie aussi les autres membres du jury pour leurs avis et remarques, ainsi que pour l'aide qu'ils ont pu m'apporter depuis le début de mon aventure dans la recherche. Benoît Feix pour son humour mordant et ses points de vue précieux. Julien Francq pour les nombreuses opportunités offertes et la bonne humeur toujours présente.

Glossaire

Les entrées de ce glossaire sont indiquées lors de leur première apparition dans le corps de cette thèse par un astérisque : mot*.

6502	Modèle de microprocesseur 8 bits datant de 1975. Devenu populaire grâce à son prix bas, il sera la base de développement d'ordinateurs et de consoles. Toujours utilisé aujourd'hui dans des systèmes embarqués.
AES	***A**dvanced **E**ncryption **S**tandard* Algorithme de chiffrement symétrique établi comme standard en 2001 par le NIST pour remplacer le DES.
ANF	***A**lgebraic **N**ormal **F**orm* *Forme normale algébrique.* Forme de réécriture d'expressions de logique booléenne sous forme normalisée à base de ET et de OU-Exclusifs (XOR) logiques.
ARM	Modèle d'architecture matérielle développé par la société du même nom.
ASIC	***A**pplication-**S**pecific **I**ntegrated **C**ircuit* *Circuit intégré spécifique pour une application.* Puce fondue spécifiquement dans le but de réaliser une implémentation matérielle d'un algorithme.
AT91	Famille de microcontroleurs de marque Atmel basés sur une architecture ARM.
Backtrack	De l'anglais "marche arrière". Action de retourner en arrière dans un arbre d'exploration afin de choisir une autre branche au dernier embranchement croisé. On effectue un *backtrack* lorsque la branche en cours s'avère ne pas contenir de solution et qu'on veut continuer l'exploration.
Bruit	Dans le domaine de l'analyse de canaux auxiliaires le bruit va être la représentation des perturbations présentes sur la mesure physique observée. Cela peut être environnemental : instruments de mesures (sonde de mauvaise qualité, précision maximale insuffisante, ...), perturbations autour du dispositif attaques (ondes wifi, lumières, températures, ...), ou bien interne au composant : contre-mesures ayant pour objectif de perturber les mesures de l'attaquant.

Brute force — La méthode brute force consiste à tester toutes les clefs possibles d'un algorithme jusqu'à trouver la bonne.

Canaux auxiliaires — Canaux d'information non prévus par les concepteurs de dispositifs électroniques, ce sont des données physiques telles que la consommation électrique à travers lesquelles de l'information peut être retrouvée.

CNF — ***C****onjunctive* ***N****ormal* ***F****orm*
Forme normale conjonctive. Réécriture d'expressions de logique booléenne sous forme normalisée à base de ET et de OU logiques.

Contre-mesure — Méthode de protection face à des attaques. Mise en place au moment de la création d'un système, elles en augmentent le coût.

Coupons — Le problème du collecteur de coupons est un problème de probabilité où l'on doit compléter une collection d'élément en piochant des éléments aléatoires.

CPA — ***C****orrelation* ***P****ower* ***A****nalysis*
Méthode d'analyse par utilisation de formules de corrélation, permettant d'extraire de l'information d'un lot de mesures d'une donnée physique prélevées sur un dispositif électronique.

CPU — ***C****entral* ***P****rocessing* ***U****nit*
Composant électronique de calcul, aussi appelée processeur, permettant d'exécuter un ensemble de tâches simples dont la combinaison successive permet d'effectuer des tâches complexes.

DES — ***D****ata* ***E****ncryption* ***S****tandard*
Algorithme de chiffrement symétrique établi comme standard en 1975 par le NIST. Ayant une taille de clef devenue trop faible il a été remplacé par l'AES comme nouveau standard en 2001. Une version avec taille de clef augmentée, le triple-DES, est cependant toujours en service.

DFA — ***D****ifférential* ***F****ault* ***A****nalysis*
Méthode d'attaque permettant d'obtenir de l'information par analyse des différences entre une exécution normale et une exécution où une faute est injectée.

DPA — ***D****ifférential* ***P****ower* ***A****nalysis*
Méthode d'analyse permettant d'extraire de l'information d'un lot de mesures d'une donnée physique prélevées sur un dispositif électronique par analyse de différentielles.

DPA Contest	Successions de concours organisés par l'école Télécom ParisTech où un lot de courbes est fourni (sauf pour la troisième édition) et où tous les participants ont pour objectif de retrouver une clef de chiffrement en minimisant le nombre de courbes utilisées. Le but étant de tester toutes les méthodes sur une même base afin de pouvoir les comparer.
Drapeaux	Dans un processeur les drapeaux (flags) sont des signaux spécifiques, souvent booléens, indiquant une information telle que le signe du calcul ou la retenue d'une opération.
FIRE	***F**ault **I**njection for **R**everse-**E**ngineering* Application des méthodes d'analyse de fautes dans un but de rétro-conception.
Hamming	* Poids de Hamming : Nombre de bits à 1 dans un mot binaire. * Distance de Hamming : Nombre de bits qui différent entre deux mots binaires.
IFA	***I**neffective **F**ault **A**nalysis* Méthode d'attaque permettant d'obtenir de l'information lorsqu'une faute injectée dans un dispositif est sans effet.
Ingénierie Sociale	L'ingénierie sociale concerne tout subterfuge (usurpation d'identité, espionnage, chantage, ...) utilisé pour obtenir des informations sensibles auprès de personnes les connaissant.
Kerckhoffs	La loi de Kerckhoffs est un principe de sécurité qui prétend que la sécurité doit résider dans le secret de la clef et non dans le fait de tenir secret une méthode de chiffrement.
Logiciel	Se dit d'une implémentation destinée à être exécutée sur un processeur qui découpera une tâche en un grand nombre de petites instructions élémentaires. Plus lente que l'implémentation matérielle mais plus polyvalente et moins volumineuse.
MARSHAL+	***M**echanisms **A**gainst **R**everse-engineering for **S**ecure **H**ardware and **AL**gorithms* *Mécanismes contre la rétro-conception pour les matériels et algorithmes sécurisés.* Projet de recherche de deux ans impliquant plusieurs acteurs académiques et industriels avec pour but de créer de nouvelles attaques et contre-mesures appliquant les méthodes d'analyse de canaux auxiliaires et d'injection de fautes au domaine la rétro-conception.
Matériel	Se dit d'une implémentation destinée à être exécutée sur un matériel dédié qui découpera la tâche à exécuter en un petit nombre de partie. Plus rapide que l'implémentation logicielle mais très peu polyvalente et plus volumineuse.

Mode d'adressage	Le mode d'adressage est l'ensemble de règles qui indiquent comment est réalisé le calcul des adresses mémoires par un processeur.
MODELSIM	Logiciel utilisé pour simuler et visualiser tous les états intermédiaires d'une implémentation matérielle réalisée en VHDL ou Verilog.
NIST	***N**ational **I**nstitute of **S**tandards and **T**echnology* Institut national américain de standardisation qui organise notamment la création de standards de cryptographie.
Obscurité	La sécurité par l'obscurité est un principe qui va à l'encontre de la loi de Kerckhoffs et qui consiste à baser la sécurité de son système sur le fait que l'attaquant ne connaît pas l'algorithme utilisé.
Pay-TV	*Services de télévisions payantes* Ces services sont souvent diffusés chiffrés à tout le monde et seul les détenteurs de cartes à puce permettant de décoder le flux peuvent le regarder. Le but des cartes utilisées dans ce cadre est donc de garantir que seuls les abonnés puissent accéder au service.
RAM	***R**amdom **A**ccess **M**emory* *Mémoire à accès aléatoire*, type de mémoire vive rapide utilisée pour stocker les informations le temps d'une exécution. Les données disparaissent une fois que la mémoire n'est plus alimentée électriquement.
Registres	Dans un processeur les registres sont les mémoires dont il dispose pour effectuer les calculs.
Rétro-conception	(*Reverse-Engineering* en anglais) Toute méthode visant à retrouver tout ou partie d'une spécification d'un algorithme tenue secrète, telles que les différentes tables constantes, le nombre de tours, etc.
ROM	***R**ead **O**nly **M**emory* *Mémoire à lecture seule*, famille de mémoires non volatiles plus lentes que la RAM, utilisées pour stocker les informations pour une longue durée. Contrairement à la RAM, les données y restent inscrites même lorsque la mémoire n'est plus alimentée électriquement.
RSA	***R**ivest **S**hamir **A**dleman* Algorithme de chiffrement asymétrique le plus connu, contraction des initiales de ses trois inventeurs.
SCA	***S**ide **C**hannel **A**nalysis* Analyse de canaux auxiliaires ayant pour but de retrouver des informations secrètes.

SCARE	***S**ide **C**hannel **A**nalysis for **R**everse-**E**ngineering* Application des méthodes d'analyse de canaux auxiliaires dans un but de rétro-conception.
SIM	***S**ubscriber **I**dentity **M**odule* Puce utilisée en téléphonie mobile, insérée dans un terminal, tel un téléphone, elle permet l'identification de l'utilisateur auprès de l'opérateur et son accès au réseau.
SPA	***S**imple **P**ower **A**nalysis* Méthode d'analyse permettant d'extraire de l'information par analyse directe d'une mesure d'une donnée physique prélevée sur un dispositif.
Verilog	Le Verilog est un langage où l'on décrit des opérations à effectuer et qui peut ensuite être synthétisé en un schéma de placement de portes logiques pour effectuer la tâche prévue.
VHDL	***VHSIC H**ardware **D**escription **L**anguage* Le VHDL est un *alter ego* du langage Verilog.
XLIM	Laboratoire de recherche en région Limousin.
XOR	Opération de ou-exclusif bit à bit.

Sommaire

Partie III Attaques physiques appliquées à la rétro-conception

Avant-propos

Cadre de cette thèse

La présente thèse s'est déroulée en trois ans au sein de l'institut de recherche XLIM* sur le site de la faculté de sciences et techniques de Limoges et a été financée par la région Limousin. Ce laboratoire réunit plus de 480 personnes (chercheurs, enseignants-chercheurs, post-doctorants, doctorants, ...) réparties dans des pôles variés tels que l'informatique, les mathématiques, l'électronique haute fréquence, la photonique, l'étude des ondes, le traitement du signal, de l'image, ...

Le travail de recherche a été effectué dans le département mathématiques et informatique (DMI), dans son équipe *Smart Secure Device* effectuant de la recherche dans le domaine de la robustesse des composants électroniques embarqués de sécurité.

Une part importante des publications détaillées ici concernent des recherches réalisées dans le cadre du projet MARSHAL+*, un projet de recherche national de deux ans (2012-2014) ayant pour but d'évaluer les menaces de rétro-conception* et impliquant de nombreux acteurs industriels et académiques.

Sujet de la thèse

Le sujet exact de la thèse est : "Étude de la sécurité d'algorithmes de cryptographie embarquée vis-à-vis des attaques par analyse de la consommation de courant.".

Les algorithmes de cryptographie servent à sécuriser les flux d'information de nombreux domaines de nos vies (banque, santé, armée, vie privée, ...), et ont besoin d'être portables (embarqués) pour faciliter leur utilisation (cartes à puce, badges, ...).

Le domaine de la cryptanalyse consiste en l'étude de la robustesse mathématique de ces algorithmes, afin de minimiser les risques de failles qui permettraient à des attaquants de récupérer des informations qui ne leurs sont pas destinées. Cette analyse était suffisante tant que les dispositifs de chiffrement étaient rares et accessibles uniquement par un nombre réduit d'individus. Mais depuis que les chiffrement se démocratisent dans les poches d'une majorité de la population (cartes bancaires, cartes SIM*, ...) un nouveau domaine d'étude est devenu nécessaire : l'analyse de canaux auxiliaires*.

L'analyse des canaux auxiliaires, qui concerne cette thèse, consiste en l'étude de phénomènes physiques mettant en danger la robustesse d'un algorithme dès lors qu'il est implémenté dans un dispositif matériel*. Ainsi même un algorithme mathématiquement éprouvé peut révéler ses secrets – si l'on n'y prend pas garde – à cause de fuites d'informations dues au matériel. On parle ici, par exemple, de la consommation de courant du matériel, qui reste le modèle de canal

auxiliaire le plus couramment étudié même si d'autres phénomènes physiques peuvent révéler de l'information (temps, émissions électromagnétiques, son, émissions lumineuses, ...).

De la même manière que la cryptanalyse évalue la robustesse mathématique des algorithmes, notre domaine évalue la robustesse physique des algorithmes et de leurs implémentations, jouant l'analogie de l'épée et du bouclier, où les chercheurs proposent d'une part de nouvelles attaques pour pointer les failles et d'autre part de nouvelles protections en cherchant à combiner efficacité et limitation des coûts. Cette thèse est un recueil de nos travaux de recherche, présentant de nouvelles attaques et contre-mesures*, qui ont été publiées durant ces trois dernières années.

Structure du document

Cette thèse est structurée comme suit :

— La Partie I introduit la thèse et les notions utiles à la compréhension des travaux qui y sont présentés (Chapitre 1) ;

— La Partie II présente la portion des travaux publiés au cours de la thèse concernant des attaques physiques dites *classiques* car elles visent à retrouver une clef secrète. Le Chapitre 2 présente une version revisitée d'une attaque type SPA* sur le *key schedule* de l'AES*, qui permet de retrouver la clef de chiffrement en se basant sur les mesures des poids de Hamming* des octets de la clef étendue, réalisées pendant une exécution protégée. Le Chapitre 3 quant à lui détaille deux propositions d'attaques faites au concours *DPA Contest V2** où nous exploitons le comportement atypique d'un bit ;

— La Partie III va, à la différence de la Partie II traitant d'attaques *classiques*, chercher une autre utilisation de ces techniques d'attaques (et de protections face à ces attaques), en ayant pour but la rétro-conception de spécifications d'algorithmes ou de codes sources. Elles utiliseront les méthodes d'analyse de canaux auxiliaires et d'analyse de fautes, on parlera alors respectivement d'attaques SCARE* et FIRE*. Le Chapitre 4 compile trois attaques, une FIRE et deux SCARE, visant toutes à retrouver une spécification secrète d'un algorithme de chiffrement du type AES modifié. Le Chapitre 5 présente une contre-mesure basée sur une implémentation alternative des algorithmes et permettant de se protéger de certaines méthodes de rétro-conception.

En fin de document nous conclurons de manière globale cette thèse. On pourra également y trouver la liste des publications qui ont été faites au cours de la thèse, les listes des figures et tableaux, ainsi que la bibliographie.

Première partie

Introduction

Chapitre 1

Notions de base et état de l'art

Sommaire

1.1 Chiffrement

Un chiffrement est une méthode permettant d'assurer la confidentialité en rendant un message inintelligible afin de le transférer à un destinataire de telle sorte que s'il est intercepté il ne puisse être lu. Le destinataire possède, quant à lui, un moyen de retrouver le message originel. Un example simple est le chiffrement de César où l'on décale simplement les lettres de l'alphabet, il suffit alors à l'interlocuteur de connaître le nombre de lettres à décaler pour retrouver le message originel. Le problème de cette méthode est qu'une personne interceptant le message

et connaissant la méthode utilisée n'a que 25 décalages non triviaux à tester pour retrouver le message. De nombreuses méthodes plus complexes ont vu le jour et à présent on considère qu'un attaquant doit faire face à au moins 2^{80} possibilités pour considérer le chiffrement comme sûr. Cette limite va certainement passer à 2^{100} à cause de l'augmentation des capacités calculatoires des ordinateurs.

1.1.1 Chiffrement symétrique

Dans un chiffrement symétrique, aussi appelé chiffrement à clef secrète, deux utilisateurs voulant communiquer vont tout d'abord convenir d'une clef K à utiliser qu'ils gareront secrète. Lorsque l'un d'entre eux souhaite communiquer le message M il lui appliquera tout d'abord la fonction de chiffrement $E()$ en utilisant la clef K pour produire le chiffré $C = E(M, K)$. En envoyant le chiffré C sur le réseau l'utilisateur sait que s'il est intercepté par un tiers ce dernier ne pourra pas en comprendre le sens, seul son interlocuteur connaissant K pourra effectuer la transformation inverse et obtenir $M = D(C, K)$, où $D()$ est la fonction de déchiffrement inverse de $E()$.

1.1.2 Chiffrement asymétrique

Dans un chiffrement asymétrique, aussi appelé chiffrement à clef publique, chaque utilisateur génère une paire de clefs, l'une appelée clef secrète qu'il est le seul à connaître, l'autre appelée clef publique qu'il diffuse sur un annuaire et qui peut servir aux autres pour le contacter.

Lorsqu'un utilisateur veut envoyer un message M il va aller chercher la clef publique du destinataire $K_{\text{pub,dest}}$ et l'utilise pour créer un chiffré $C = E(M, K_{\text{pub,dest}})$. Contrairement au chiffrement symétrique, il sera alors lui-même dans l'incapacité d'effectuer la transformation inverse ne possédant pas la clef privée de son interlocuteur. L'interlocuteur sera le seul à pouvoir récupérer le message originel $M = D(C, K_{\text{priv,dest}})$, grâce à sa clef privée $K_{\text{priv,dest}}$.

1.1.3 Utilisations du chiffrement

Un chiffrement peut donc être utilisé pour chiffrer un message et le rendre inintelligible pendant un transfert. Dans certains cadres un chiffrement peut aussi servir à d'autres buts. Il peut être utilisé par une personne pour s'authentifier, certifier une intégrité des données ou bien signer un message afin de prouver qu'il en est l'auteur.

Ce qui nous intéresse ici est l'utilisation du chiffrement dans un système embarqué (cartes bleues, cartes SIM, ...) où il doit être réalisé dans un environnement aux contraintes fortes : temps limité, performances réduites, mémoire limitée, environnement changeant et non sécurisé, attaquant pouvant être le possesseur officiel du dispositif. Dans ce type de fonctionnement, un attaquant aura souvent connaissance – voire le choix – des messages qui sont envoyés à la carte et obtiendra souvent la connaissance du chiffré.

1.1.4 Cryptanalyse

La méthode basique pour retrouver une clef de chiffrement consiste à tester toutes les clefs jusqu'à trouver la clef correcte et s'appelle la méthode *brute force**. Toute méthode permettant de retrouver la clef en explorant un espace plus réduit que celui de la méthode brute force est une cryptanalyse.

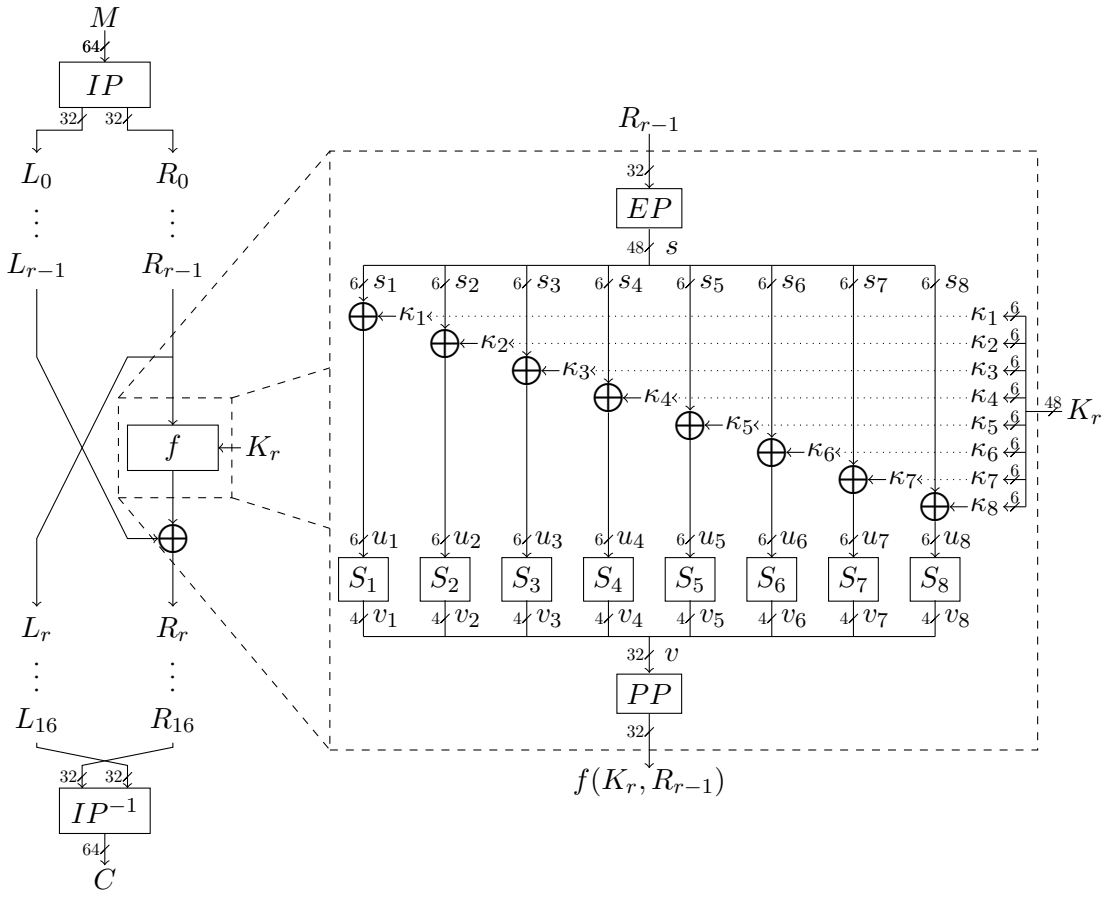

FIGURE 1.1 – Illustration du fonctionnement général du DES avec le détail de la fonction f

La cryptanalyse dite *classique* utilise un raisonnement mathématique pour réduire le nombre de clefs possibles en trouvant une faille dans la construction de l'algorithme.

Les attaques par canaux auxiliaires ont le même but de réduire le nombre de clefs possibles mais utilisent cette fois-ci des fuites d'informations produites par le matériel sur lequel l'algorithme de chiffrement est exécuté.

1.2 Algorithmes

1.2.1 DES

Les DES* pour ***D**ata **E**ncryption **S**tandard* est un algorithme qui a été le standard de chiffrement symétrique entre 1977 et 2001. Le DES est basé sur un schéma de Feistel. On pratique un découpage des données en deux parties : gauche (L) et droite (R), on utilise la même fonction de transformation f à chaque tour. Un tour de DES est détaillé en Figure 1.1. Il est à remarquer que la fonction f n'a pas nécessité d'être inversible. Elle est ici composée d'une permutation expansive (EP), de l'ajout d'une clef de tour (K_r), du passage par 8 S-Boxs (S_i) et enfin d'une permutation simple (PP). Un autre avantage des schémas de Feistel réside dans le fait que le déchiffrement utilise le même circuit que le chiffrement, seul l'ordre des clefs doit être inversé.

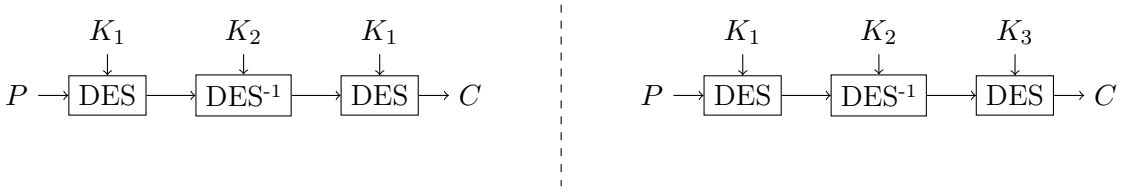

FIGURE 1.2 – Variantes du triple-DES à deux clefs (gauche) et à trois clefs (droite)

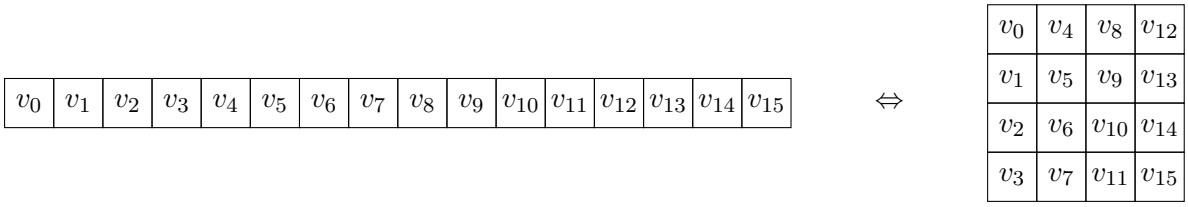

FIGURE 1.3 – Transposition entre les visions vectorielle et matricielle d'un état intermédiaire d'AES-128.

Les 56 bits de sa clef devenant trop peu, une version nommée triple-DES a été conçue pour augmenter la taille de clef en permettant l'utilisation de deux ou trois clefs indépendantes, illustrée en figure 1.2. Cette version est encore utilisée aujourd'hui.

L'algorithme DES a été remplacé par l'AES en 2001 car sa clef de 56 bits devenait trop faible. À titre d'exemple, des machines dédiées (Copacobana, Rivyera, ...) peuvent pour moins d'une centaine de milliers d'euros tester toutes les clefs du DES en moins de trois jours.

1.2.2 AES

L'AES, pour ***A**dvanced **E**ncryption **S**tandard*, est le remplaçant du DES (voir Section 1.2.1) depuis 2001 comme standard de chiffrement symétrique. Il est basé sur l'algorithme RIJNDAEL (contraction du nom des créateurs Vincent Rijmen et Joan Daemen) vainqueur du concours organisé par le NIST* pour désigner un successeur au DES.

Cet algorithme est basé sur un réseau de substitutions/permutations et a été conçu pour une implémentation facilitée en logiciel* ou en matériel. Il possède trois versions où il chiffre un clair de 128 bits pour rendre un chiffré de 128 bits à l'aide d'une clef de 128, 192 ou 256 bits – selon la version utilisée – en respectivement 10, 12 et 14 tours.

Dans le processus de chiffrement un octet est considéré comme un élément du corps fini $GF(2^8)$ et chaque bloc interne de 16 octets peut être représenté par une matrice carrée de 4×4 octets. La transposition entre les représentations vectorielle et matricielle est faite en numérotant les éléments colonne par colonne. Cette transposition est détaillée en Figure 1.3.

Nous décrivons ici le fonctionnement de la version 128 bits de l'AES. Soit un clair de 128 bits $M = (m_0, \ldots, m_{15})$ et une clef de chiffrement de 128 bits $K = (k_0, \ldots, k_{15})$ l'AES-128 calcule un chiffré $C = (c_0, \ldots, c_{15})$, par la méthode représentée en Figure 1.4, en débutant par une première opération XOR* entre M et K puis en en injectant le résultat, S_0, dans un cycle de 10 tours. Pour $r = 1, \ldots, 9$, chaque tour d'AES-128 r calcule son état de sortie S_r en appliquant successivement quatre transformations : `SubBytes`, `ShiftRows`, `MixColumns` et `AddRoundKey` à son état d'entrée S_{r-1}. Le chiffré est finalement défini comme étant la sortie du dixième et dernier tour qui a pour différence de ne pas comporter d'opération `MixColumns`. Le processus de chiffrement est résumé

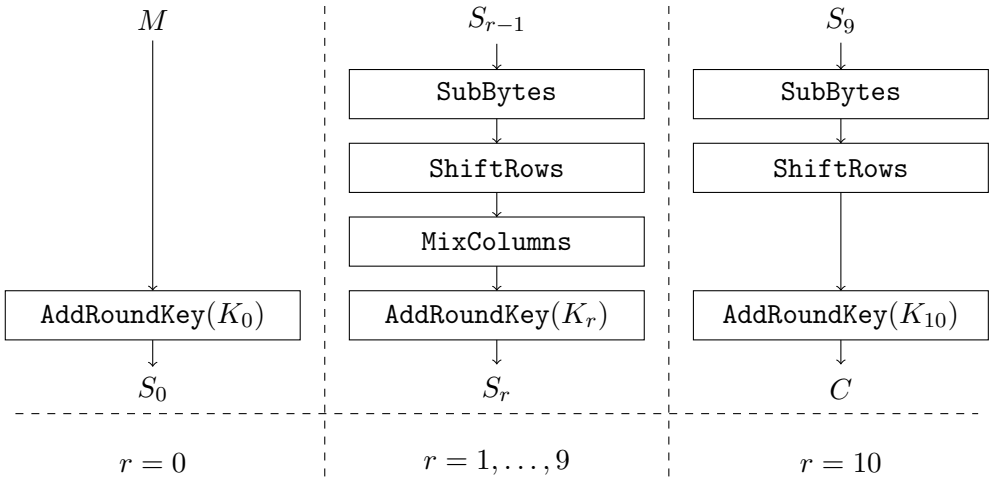

FIGURE 1.4 – Processus de chiffrement de l'AES

comme suit :

$$\begin{aligned} S_0 &\leftarrow M \oplus K_0 \qquad (K_0 = K) \\ S_r &\leftarrow \texttt{MixColumns}(\texttt{ShiftRows}(\texttt{SubBytes}(S_{r-1}))) \oplus K_r \quad (r = 1, \ldots, 9) \\ C &\leftarrow \texttt{ShiftRows}(\texttt{SubBytes}(S_9)) \oplus K_{10} \end{aligned}$$

La transformation `SubBytes` est une permutation de $GF(2^8)$ définie par une table S-Box notée S. L'opération `ShiftRows` consiste en une rotation de chaque ligne i ($i = 0, \ldots, 3$) de la matrice de i octets vers la gauche. Le `MixColumns` calcule chaque colonne de sortie comme le produit d'une matrice constante et de la colonne d'entrée correspondante. En fin de tour, l'opération `AddRoundKey` combine par opération XOR (addition dans $GF(2^8)$) l'état courant et la clef de tour K_r.

Les différentes clefs de tours K_r impliquées dans le processus de chiffrement sont dérivées de la clef principale K à travers la fonction *key schedule*. Le *key schedule* de l'AES-128 est décrit sur la Figure 1.5. L'opération `RotWord` consiste en une rotation d'une colonne d'une position vers le haut, `SubWord` applique la S-Box à chaque octet de la colonne, enfin `Rcon`$[r]$ est une constante dépendante du tour égale à $(\rho^{r-1}, 0, 0, 0)$ avec $\rho = 2$ ajoutée à la colonne.

Pour plus d'informations sur l'algorithme AES nous renvoyons vers sa spécification officielle [Nat01].

1.3 Canaux auxiliaires

Les canaux auxiliaires sont des canaux d'informations, accessibles à un attaquant, qui n'étaient pas prévus par les concepteurs d'un dispositif de cryptographie. Cela peut être le temps d'une exécution, sa consommation électrique, ses émissions électromagnétiques, ses émissions lumineuses, les sons qui sont produits, ... Ces données physiques peuvent donc être mesurées et utilisées pour obtenir de l'information supplémentaire sur le déroulement des opérations dans le dispositif de cryptographie. Ces mesures étant sensibles à des perturbations, nommés bruits parasites, il peut être nécessaire de les effectuer dans un environnement dédié et surtout elles sont traditionnellement appliquées sur des dispositifs de taille et de fonctionnalités réduites, désignant les

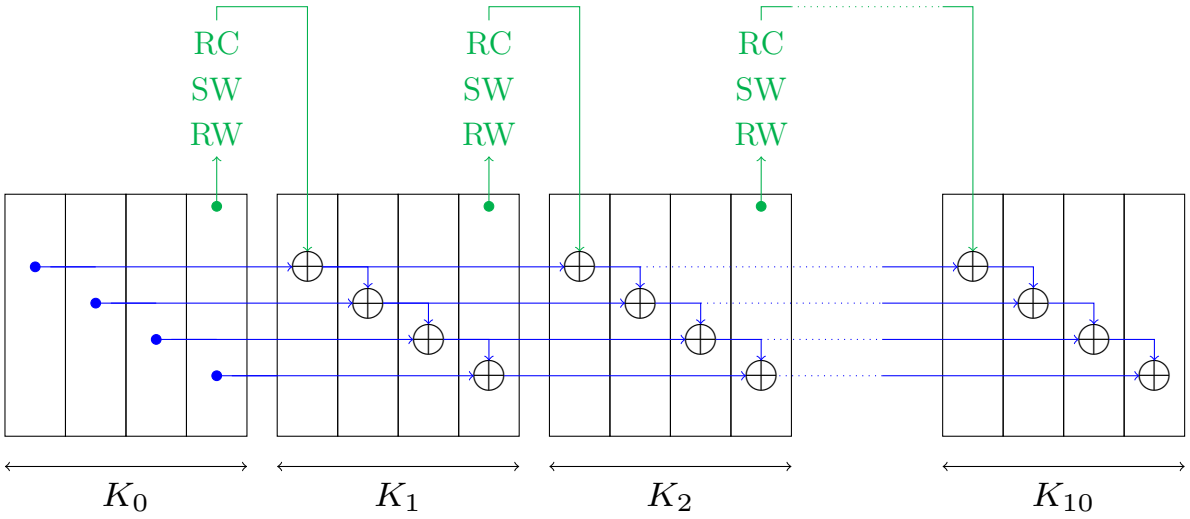

FIGURE 1.5 – Key schedule de l'AES-128

dispositifs du type des cartes à puce comme des cibles privilégiées. A contrario, certaines publications récentes montrent que même sur des systèmes plus complexes comme des ordinateurs classiques des attaques sont possibles. Un exemple de ces nouvelles attaques est décrit par Genkin et al. dans [GPT14] où ils parviennent à attaquer une exécution cryptographique réalisée sur un ordinateur par les fluctuations du courant émis sur le câble réseau connecté à celui-ci.

1.3.1 Implémentations logicielles et matérielles

Il existe deux façons d'exécuter un algorithme, soit en passant par un processeur, généraliste et plus lent, soit par un matériel dédié, spécialisé et rapide. Les deux fonctionnements produisant des signaux différents sur les canaux auxiliaires nous allons détailler ici leurs différences.

1.3.1.1 Fonctionnement général d'un processeur générique

Un processeur est une puce de taille réduite possédant un jeu d'instructions varié, qui sera capable d'exécuter n'importe quel algorithme. Il suffit au créateur de l'algorithme de *compiler* son implémentation pour ce processeur, c'est à dire découper le travail à effectuer en un grand nombre de petites opérations que peut effectuer ce processeur. L'ensemble des opérations disponible sur un processeur est appelé son jeu d'instructions. Le compilateur produira alors un plan de route qui indiquera au processeur où aller chercher les données en mémoire, quelles opérations effectuer sur ces données et enfin où placer le résultat en mémoire. Le processeur peut donc exécuter tout algorithme pouvant être compilé dans son jeu d'instruction, en revanche le grand nombre d'opérations élémentaires nécessaires pour effectuer un algorithme le rend coûteux en temps.

1.3.1.2 Fonctionnement général d'un matériel dédié

Un matériel dédié est une puce de taille plus importante et qui, contrairement au processeur, ne peut servir qu'à un seul but mais peut alors le réaliser beaucoup plus vite. Sa taille plus importante vient de la multiplications des portes logiques nécessaires à la conception et des liens entre celles-ci.

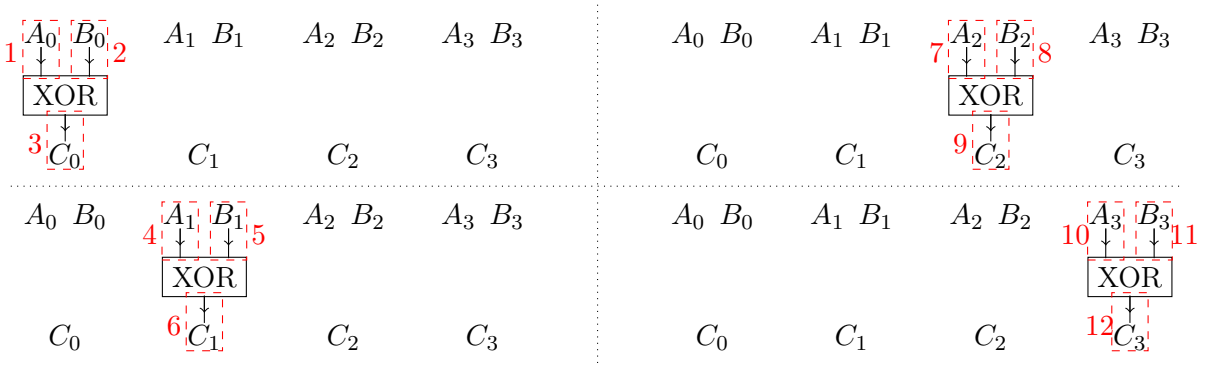

FIGURE 1.6 – Illustration simplifiée du nombre élevé d'étapes pour le calcul du XOR de deux mots de 32 bits sur un processeur ayant un bus de 8 bits.

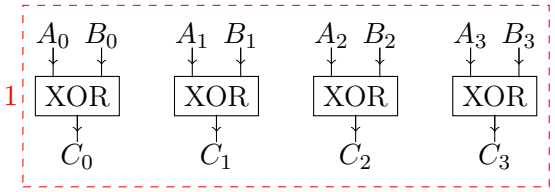

FIGURE 1.7 – Illustration simplifiée de l'exécution en une seule étape du calcul du xor de deux mots de 32 bits sur un matériel dédié.

Des puces programmables (FPGA, ASICS*, ...) peuvent être utilisées pour tester des implémentations matérielles avant leur mise en fonderie mais peuvent aussi être utilisées directement comme un matériel en production.

1.3.1.3 Comparaison des implémentations

On examine ici les différences entre les implémentations logicielle et matérielle d'un XOR de deux mots A et B de 32 bits : $C = A \oplus B$. On note ici la décomposition d'un mot de 32 bits X en quatre mots de 8 bits $X = X_0|X_1|X_2|X_3$.

Les Figures 1.6 et 1.7 présentent une version très simplifiée d'une exécution du XOR de deux mots de 32 bits respectivement sur un processeur 8 bits et sur un matériel dédié.

On peut constater que le processeur ne peut exécuter qu'un seul XOR 8 bit à la fois et que cela lui demande de nombreuses opérations comparé au matériel dédié qui n'en demande qu'une seule grâce à ses cellules en parallèle.

L'avantage d'un matériel dédié est bien plus important que sur cette schématisation imprécise, en effet avec une surface suffisante un matériel dédié peut exécuter un tour entier d'AES en une seule étape là où il en faut des centaines pour un processeur.

Dans le cadre des attaques physiques un processeur aura des fuites d'information liées à des données plus petites, fonction de la taille de son bus de données, et plus nombreuses au vu de son nombre plus élevé d'étapes nécessaires pour effectuer une tâche. Un matériel dédié, quant à lui, pourra effectuer un tour entier en un seul cycle et la consommation ne traduira, pour sa grande majorité, que les données finales générées. Par exemple les données de fin de tour d'AES produiront une fuite mais pas les sorties des S-Boxs car elles font partie des données intermédiaires.

1.3.2 Analyse de canaux auxiliaires

Les attaques de type SCA*, pour **S**ide **C**hannel **A**nalysis, (Analyse de canaux auxiliaires) ont été introduites par Kocher et al. [Koc96, KJJ99] comme un moyen efficace de retrouver les clefs d'algorithmes cryptographiques sur des systèmes embarqués. Depuis, de nombreuses techniques d'attaques de ce type ont été découvertes, ainsi que de nouvelles contre-mesures pour s'en protéger.

Ces attaques consistent à utiliser des canaux auxiliaires, décrits en Section 1.3, pour acquérir des informations sur ce qui est en train d'être exécuté sur un composant.

1.3.2.1 SPA

Les attaques de type SPA, pour ***S**imple **P**ower **A**nalysis*, (Analyse simple du courant) sont des attaques qui reposent sur une inspection "visuelle" d'une seule courbe (dans les cas favorables). Deux types d'informations peuvent être retrouvées par SPA : à haut niveau on peut repérer des instructions ou des blocs d'instructions exécutées sur le dispositif, à bas niveau on peut obtenir de l'information sur les valeurs des données traitées. Des attaques utilisent le lien entre la donnée manipulée et le niveau de consommation électrique qu'elle implique [MDS99, MS00, Mes00].

La SPA *timing attack* (attaque par le temps) est la première attaque du domaine, conçue par Kocher [Koc96]. Elle se base sur l'analyse des fluctuations du temps d'exécution d' algorithmes où des instructions conditionnelles sont exécutées en fonction des valeurs de clefs. Cela ouvre un lien entre la mesure physique du temps et la valeur de la clef.

1.3.2.2 DPA

Les attaques de type DPA*, pour ***D**ifferential **P**ower **A**nalysis*, (Analyse différentielle de la Consommation) ont été introduites par Kocher et al [KJJ99]. La DPA consiste à réaliser plusieurs acquisitions où le message change à chaque exécution. En se basant sur la supposition qu'un bit à 1 passant sur le bus de données consomme différemment d'un bit à 0, l'attaquant va pouvoir, selon les suppositions sur une partie de clef, classer les courbes en deux catégories et en faire respectivement les deux moyennes. Enfin il effectuera une différence de ces deux moyennes afin d'obtenir une courbe de DPA propre à la valeur supposée de la clef en cours d'évaluation. Si l'une des courbes de DPA ainsi générées présente un pic élevé l'attaquant conclura alors que c'est cette supposition qui est le meilleur candidat à la valeur de clef.

Une schématisation de la DPA est présentée en Figure 1.8, où l'on peut voir une analyse de 8 acquisitions, avec la différentiation de la supposition correcte qui produit un pic de DPA (en haut) et des suppositions incorrectes qui ne produisent pas de pics (en bas).

1.3.2.3 CPA

Les attaques de type CPA*, pour ***C**orrelation **P**ower **A**nalysis*, ont été introduites par Brier et al. [BCO04] pour palier au fait que la DPA n'analyse qu'un seul bit de la donnée intermédiaire alors que la consommation peut dépendre de plusieurs bits à la fois. Comme la DPA, elle consiste à analyser un lot de courbes de consommation, mais elle prend en compte plus d'un seul bit en essayant de faire correspondre un modèle de consommation d'une donnée intermédiaire, liée à une supposition sur la clef, avec les consommations mesurées. Le résultat de cette analyse est un coefficient de corrélation entre les données intermédiaires et la consommation obtenue grâce au coefficient de Pearson :

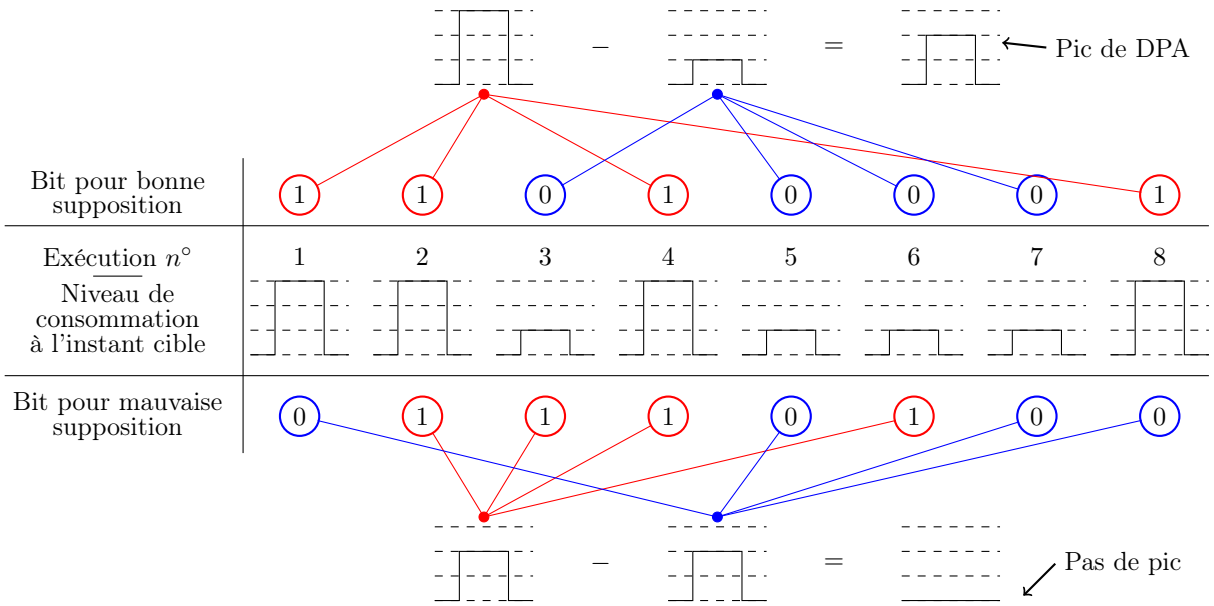

FIGURE 1.8 – Shématisation de la méthode DPA : Obtention d'un pic de DPA pour la bonne supposition de valeur de clef (en haut) et non-obtention d'un pic lors des mauvaises suppositions (en bas)

$$\text{Corrélation} = \frac{\text{Covariance}(\text{Modèle}(\text{donnée}), \text{Consommation})}{\text{écart-type}(\text{Modèle}(\text{donnée})) * \text{écart-type}(\text{Consommation})}$$

Les modèles de consommation classiques sont les modèles en poids de Hamming (nombre de bits à 1 dans un mot binaire) ou en distance de Hamming (nombre de bits différents entre deux mots binaires). Le premier modèle représente un système où la consommation est dépendante des données, avec les bits à 1 consommant différemment des bits à 0. Le deuxième modèle représente un système où la consommation est dépendante de l'étape de remplacement d'une donnée par une nouvelle. Dans ce second modèle, un changement de bit de 0 à 1 ou de 1 à 0 va consommer plus intensément que lorsque un bit reste inchangé.

Une schématisation de la CPA est présentée en Figure 1.9, où l'on peut voir une analyse de 8 acquisitions. On considère un composant dont la consommation dépend linéairement du poids de Hamming, noté HW(), de la donnée traitée : $\alpha * \text{HW} + \beta$. La supposition correcte, dont les poids de Hamming s'accordent avec la consommation, produit une corrélation forte (en haut). Cette dernière se différenciant des suppositions incorrectes, dont les poids de Hamming ne s'accordent que parfois par hasard, qui produisent une corrélation faible (en bas).

1.3.2.4 Impact du bruit

Toutes les méthodes d'attaques par canaux auxiliaires sont dépendantes de la qualité de la mesure effectuée. En effet un bruit de mesure peut apparaître, lié au matériel de mesure, des perturbations extérieures ou à des contre-mesures spécifiques. L'intensité d'un bruit prends de l'importance car les analyses décrites ci-dessus se basent sur des fluctuations parfois fines de la donnée physique mesurée, un bruit pouvant alors retarder, voire empêcher, la récupération des données recherchée.

Un bruit étant aléatoire, on peut améliorer la qualité du signal en effectuant la moyenne de

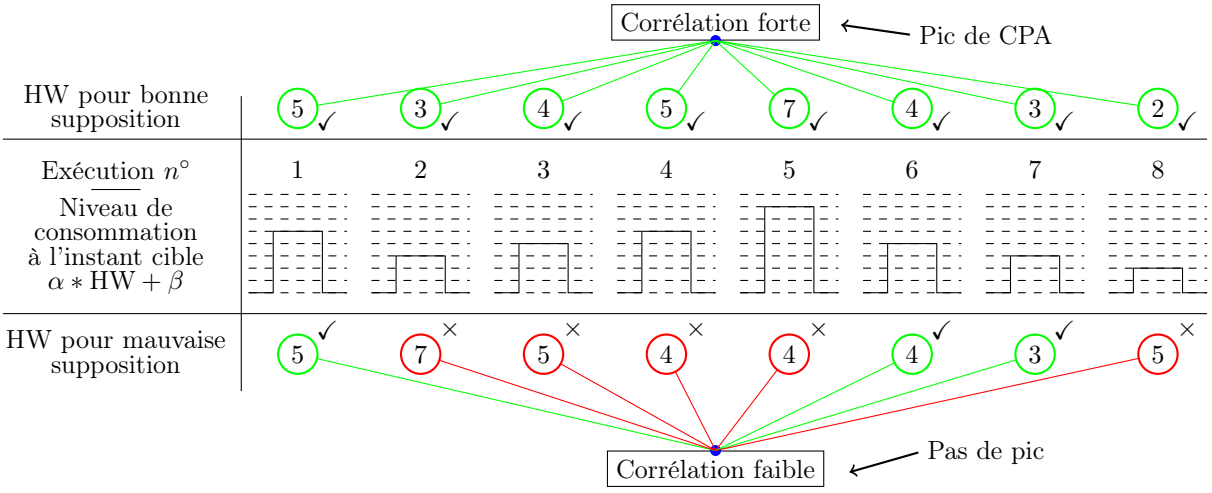

FIGURE 1.9 – Shématisation de la méthode CPA : Obtention d'un pic de CPA pour la bonne supposition de valeur de clef (en haut) et non-obtention d'un pic lors des mauvaises suppositions (en bas)

plusieurs mesures du chiffrement du même message, le bruit s'atténuant alors avec l'augmentation du nombre de courbes incluses dans la moyenne. Cette méthode nécessite en revanche une bonne synchronisation temporelle des courbes.

1.3.3 Analyse de fautes

L'analyse de fautes sur des systèmes embarqués a été introduite pour la première fois fin 1996 [BS97, BDL97] dans des attaques visant les deux plus célèbres algorithmes cryptographiques de l'époque : le DES et le RSA. Depuis, de nombreuses contributions ont été publiées indiquant comment retrouver des clefs cryptographiques de nombreux algorithmes par l'injection de fautes produisant des résultats erronés.

1.3.3.1 DFA

La DFA*, pour ***D**iffrential **F**ault **A**nalysis*, a été décrite par Biham et Shamir [BS97] et consiste à analyser les différentielles induites par des fautes afin de retrouver des informations sur la clef de chiffrement. On y compare les chiffrés donnés en sortie par le composant lors de l'exécution sans faute et des chiffrés du même message mais avec injection d'une faute en cours d'exécution. De ces différences l'attaquant peut alors déduire des informations.

1.3.3.2 SEA

La SEA [YJ00, JQYY02], pour ***S**afe **E**rror **A**nalysis*, considère des fautes injectées provoquant un effet sur le système cible tel qu'il ignore des instructions (e.g. une des deux branches d'un embranchement conditionnel). Si l'exécution se déroule pour autant correctement, cela signifie à l'attaquant que l'instruction ignorée n'a pas été utilisée et il obtient ainsi de l'information. Cette méthode peut, de la même manière, être appliquée sur des données au lieu d'instructions.

1.3.3.3 IFA

L'IFA*, pour ***I**neffective **F**ault **A**nalysis*, décrite par Clavier [Cla07b], est une analyse qui se base sur le fait qu'une faute injectée n'a pas d'effet pour donner de l'information à l'attaquant. Ressemblant à la SEA, elle diffère dans le fait de s'attaquer aux données et non aux étapes du chiffrement. Elle sont utilisées avec un modèle de faute dont l'effet est de fixer à une valeur précise une donnée intermédiaire.

Par exemple, si l'on sait injecter une faute forçant à zéro la valeur de l'octet ciblé, on peut comparer avec le chiffré de la même exécution sans injection de faute et observer si le chiffré est resté inchangé ou pas. Ce booléen indique en réalité si la valeur ciblée valait zéro ou non, en effet le fait de remettre à zéro une valeur déjà à zéro n'aura aucun effet.

Ce type d'analyse ne donne que des informations booléennes mais permet d'analyser n'importe quelle étape de l'algorithme.

1.4 Rétro-conception

La rétro-conception est une famille d'attaques visant à retrouver, non pas une clef de chiffrement, mais les spécifications d'un algorithme. Certaines pratiques de *sécurité par l'obscurité** prônent l'utilisation d'algorithmes tenus secrets afin de se protéger des attaques, cela contrevenant au principe de Kerckhoffs* (voir Section 1.4.1).

Il se trouve que l'on peut adapter les méthodes d'analyse des canaux auxiliaires afin de retrouver les spécifications d'algorithmes secrets. Au-delà de l'ingénierie sociale*, des méthodes utilisant les canaux auxiliaires existent pouvant mettre en péril ce type d'implémentations, on parle alors de rétro-conception par analyse de mesures physiques : *SCARE* (voir Section 1.4.2) ou par injection de fautes : *FIRE* (voir Section 1.4.3).

1.4.1 Principe de Kerckhoffs

Le principe de Kerckhoffs est une idée de bonne pratique de la cryptographie qui dit que la sécurité d'un système de chiffrement ne doit reposer que sur la clef. Il vient s'opposer aux pratiques de *sécurité par l'obscurité* qui consistent à garder secrètes les spécifications d'un algorithme, pensant que garder l'attaquant dans l'ignorance de la méthode va garantir la sécurité de son implémentation.

De nombreux acteurs de la cryptographie peuvent être tentés de concevoir un algorithme, ou modifier un algorithme connu, et de taire son implémentation et ainsi réduire drastiquement son coût en n'ayant pas à le protéger de la plupart des attaques nécessitant de connaître l'implémentation (e.g. DPA, CPA, ...). Cette pratique demande cependant de limiter fortement le nombre de personnes connaissant les spécifications car une seule fuite d'information peut alors compromettre tous les systèmes en cours d'utilisation implémentant cette solution. Ce choix comporte aussi le risque de subir une attaque de rétro-conception révélant les spécifications.

1.4.2 SCARE

Les attaques de type SCARE, pour ***S**ide-**C**hannel **A**nalysis for **R**everse **E**ngineering* (Analyse des canaux auxiliaires appliquée à la rétro-conception), ont pour but de retrouver tout ou partie d'un algorithme tenu secret, cela peut concerner sa structure générale ou ses paramètres.

Novak [Nov03a] fut le premier à décrire une attaque SCARE qui permet de retrouver le contenu d'une des deux tables de substitution d'un algorithme secret d'authentification conçu

pour la téléphonie mobile. Son attaque considère le modèle d'un attaquant capable de détecter, par analyse des canaux auxiliaires, si deux données intermédiaires collisionnent ou non, sans pour autant connaître leur valeurs. En gardant le même modèle, Clavier [Cla07a] a amélioré cette attaque en retrouvant les deux tables sans connaissance *a priori* de la clef de chiffrement. Dans [DLMV03] Daudigny et al. proposent une attaque SCARE appliquée sur le DES. Ils utilisent une attaque de type DPA afin de détecter l'ordre d'utilisation des bits et ainsi retrouver les fonctions de permutation (voir Section 1.2.1). Deux autres travaux ont étendu l'utilisation des méthodes SCARE à des familles de chiffrement partageant la même structure : Real et al. [RDG+08] ont proposé une méthode révélant la fonction de tour de n'importe quelle implémentation matérielle d'un schéma de Feistel. Rivain et al. quant à eux ont montré dans [RR13] comment exploiter des collisions dans les S-Boxs afin d'obtenir une représentation équivalente de n'importe quelle fonction de chiffrement basée sur un réseau de substitution-permutation. Enfin d'autres publications [Nov03b, GSM+10] traitent aussi de l'utilisation des canaux auxiliaires afin de découvrir des informations à propos d'algorithmes secrets.

Nous avons contribué par deux nouvelles attaques SCARE publiées dans [CIW13, CIMW15] s'attaquant à toute implémentation d'AES où les constantes ont été modifiées et tenues secrètes en utilisant la détection de collisions au niveau des S-Boxs. Elles sont décrites en Sections 4.5 et 4.6 du Chapitre 4.

1.4.3 FIRE

Les attaques de type FIRE, pour ***F**ault **I**njection for **R**everse **E**ngineering* (Injection de fautes appliquée à la rétro-conception), ont le même but que les attaques SCARE (voir Section 1.4.2) mais utilisent cette fois-ci les méthodes classiques d'analyse des effets des fautes afin de les adapter à la rétro-conception.

Pedro et al. ont développé dans [PSG11] une attaque de type FIRE s'attaquant à une S-Box inconnue sur des algorithmes de type DES et AES. Ces attaques laissent une recherche exhaustive parmi 2^{32} candidats après 1000 fautes sur le DES et parmi 2^{57} candidats après 400 fautes sur l'AES.

La littérature sur les attaques de type FIRE sur un algorithme complet était, à notre connaissance, inexistante avant la publication de notre attaque [CW13] (ainsi que [CIMW15] pour sa version étendue), qui est décrite en Section 4.4 du Chapitre 4. Elle utilise une analyse de fautes de type IFA sur toute implémentation d'AES où les constantes ont été modifiées et tenues secrètes.

1.5 Contre-Mesures

Les contre-mesures sont les réponses aux nouvelles attaques, elles modifient une implémentation, parfois de manière mineure mais souvent avec un surcoût élevé, afin de la rendre insensible aux nouvelles menaces. On différenciera les contre-mesures dites *logicielles* (Section 1.5.1) qui consistent à modifier l'algorithme et les contre-mesures dites *matérielles* (Section 1.5.2) où des éléments physiques du composant électronique ont été modifiés afin de protéger les données traitées pendant l'exécution.

Les contre-mesures évoquées dans cette thèse peuvent se classer en plusieurs catégories détaillées ci-après. Les différentes familles sont illustrées par les Figures 1.11 à 1.14. À titre de comparaison une illustration d'une exécution sans contre-mesure est donnée en Figure 1.10. On peut y voir trois exécutions chacune exécutant la même série de 5 opérations. Vu que ces fonctions sont alignées temporellement (verticalement) les attaques classiques (DPA, CPA , ...) peuvent s'appliquer.

1.5.1 Logicielles

Exécution en ordre aléatoire

Ces contre-mesures consistent à rendre aléatoire l'ordre d'exécution d'opérations indépendantes, cela provoque une difficulté pour l'attaquant d'appliquer les méthodes classiques car ainsi les opérations ne sont plus alignées temporellement. Ce phénomène est illustré en Figure 1.11 où on a permuté aléatoirement les calculs des 5 opérations (en supposant qu'elles sont indépendantes). L'attaquant ne pouvant prédire quelle fonction a été exécutée, il ne peut pas prédire les données intermédiaires.

Masquage

Le masquage est une famille de contre-mesures utilisant une valeur aléatoire – nommée le masque – différente à chaque exécution qui est combinée avec les valeurs sensibles afin que les canaux auxiliaires reflètent la valeur masquée plutôt que la valeur originelle. L'aléa apporte une protection contre les attaques classiques, car si l'attaquant peut toujours supposer les valeurs intermédiaires théoriques ce sont des valeurs différentes qui vont passer physiquement sur les bus de données. Ces nouvelles données ont une complexité à tester exhaustivement qui croît à chaque nouvelle courbe et il devient alors très vite impossible de s'y attaquer. Grâce à la connaissance du masque qu'il possède, le dispositif peut, en fin d'exécution de la partie sensible, démasquer la donnée et récupérer la donnée correcte.

Une illustration de cette contre-mesure est donnée en Figure 1.13 où l'on peut voir trois exécutions masquées. Les cinq opérations sont toujours alignées temporellement mais à chaque exécution une nouvelle valeur aléatoire r_i est utilisée, obligeant un attaquant à supposer les valeurs de tous les r_i pour mener une attaque classique. L'exemple fourni ici serait un masquage dit au premier ordre car on utilise la même valeur de masque pendant toute l'exécution. Des attaques pouvant parfois être possibles en utilisant plusieurs points de la même courbe, ce masquage ne serait donc plus efficace, on peut alors utiliser des masquages où le masque change en cours d'exécution que nous appellerons dans la suite de ce mémoire des *masquages d'ordre plus élevé.*

Délais aléatoires logiciels

Une contre-mesure à base de délais aléatoires va effectuer de manière aléatoire plusieurs fausses opérations en cours de calcul, de sorte que la consommation de cette fausse opération ressemble à la consommation d'une opération normale (e.g. appliquer une des opérations f_i sur des données aléatoires inutiles au calcul). Cette méthode est illustrée en Figure 1.12.

Une combinaison des différentes méthodes présentées jusqu'ici peut être utilisée comme l'illustre la Figure 1.14.

1.5.2 Matérielles

Délais aléatoires matériels

Un composant électronique peut être conçu pour insérer des délais aléatoires répartis au cours d'une exécution. Apparaissant de manière aléatoire et pour une durée aléatoire ces délais viennent réaliser un effet de désynchronisation temporelle des différentes opérations de manière plus fine que les délais aléatoires logiciels vues plus haut. Un attaquant devra alors mettre en

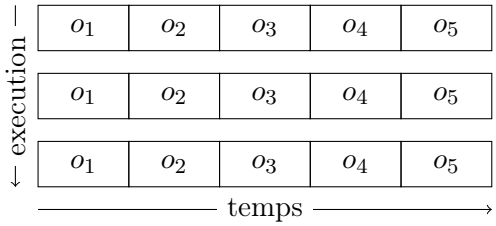

FIGURE 1.10 – Illustration de trois exécutions d'une même série de 5 opérations sans contre-mesure

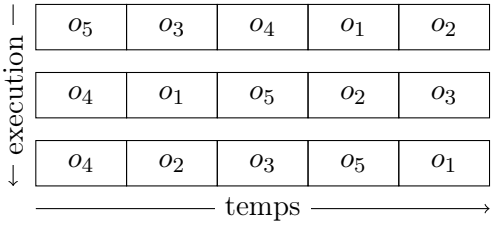

FIGURE 1.11 – Illustration de la contre-mesure d'exécution en ordre aléatoire

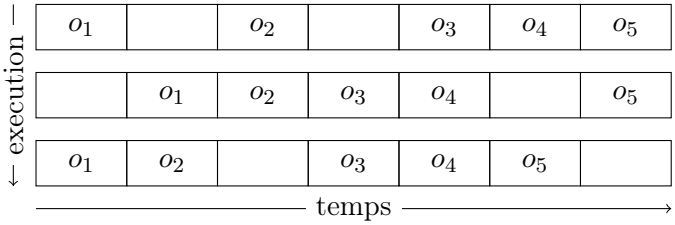

FIGURE 1.12 – Illustration de la contre-mesure introduisant des délais aléatoires

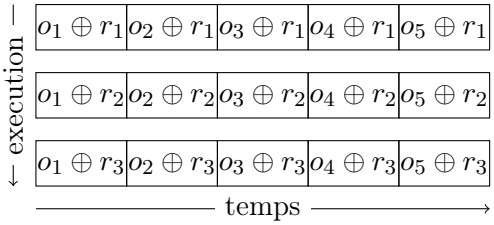

FIGURE 1.13 – Illustration de la contre-mesure utilisant des masques

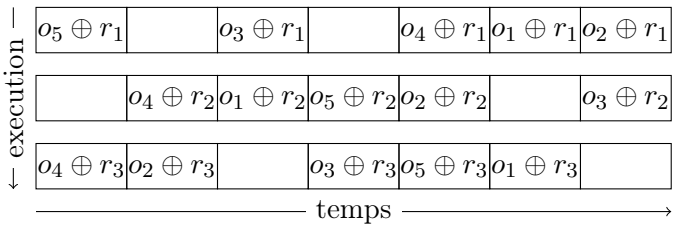

FIGURE 1.14 – Illustration de la combinaison des trois contre-mesures

place une méthode pour détecter, si seulement cela lui est possible, ces portions de courbes et les enlever avant de pouvoir mener ses attaques à bien.

Bruit additionnel

Un bruit* sur la mesure peut grandement impacter la réussite d'une attaque comme expliqué en Section 1.3.2.4. Un composant peut alors être conçu afin d'ajouter volontairement un bruit sur les mesures physiques qui pourraient être effectuées.

Camouflage matériel

Certains attaquants disposant de moyens suffisants peuvent ouvrir une puce, enlever des couches du composant pour pouvoir accéder à certaines parties plus profondes, notamment pour l'observer au microscope et en déduire des informations. Des contre-mesures de camouflage matériel peuvent alors être mises en place afin de contrer ce genre d'observations en concevant de manière particulière les composants de sorte que tout soit semblable et donc non différentiable. Une image illustrant cette contre-mesure est présentée en Figure 5.1 dans le Chapitre 5.

Comme précédemment il est possible de combiner des contre-mesures matérielles entre elles mais aussi des matérielles avec des logicielles.

1.6 DPA Contest

Le DPA contest est une famille de concours créés par Télécom ParisTech visant à faire travailler les participants dans un cadre commun afin de pouvoir comparer les différentes propositions et faire ainsi ressortir les méthodes les plus efficaces. Il y a eu à ce jour quatre éditions (la quatrième étant toujours en cours à la date d'édition de ce mémoire). La plupart des éditions ont pour but l'analyse d'un grand lot de courbes fourni par les organisateurs, tous les paramètres – messages, clefs, variables de sécurités – étant fournis afin de permettre aux participants d'avoir accès à toutes les données intermédiaires pour calibrer leurs attaques. L'objectif étant de concevoir une attaque réussissant à retrouver la clef de chiffrement en utilisant le moins de courbes possibles. Seule la troisième édition fait exception, l'objectif consistait alors à acquérir les meilleures courbes sur un matériel donné. Toutes les informations et résultats sont disponibles sur le site officiel `http://www.dpacontest.org`.

DPA Contest V1

La première édition s'est tenue entre août 2008 et août 2009, les participants devaient retrouver une clef de DES dans un lot de courbes mesurées sur une implémentation matérielle de l'algorithme. Le lot d'évaluation des attaques était le lot donné aux participants ce qui fut changé les années suivantes, l'évaluation étant alors faite sur un lot différent uniquement connu des organisateurs.

DPA Contest V2

La deuxième édition s'est tenue entre janvier 2010 et février 2011, elle portait sur des courbes mesurées sur une implémentation matérielle de l'AES.

Notre proposition d'attaque à ce concours est détaillée en Chapitre 3.

DPA Contest V3

La troisième édition est donc un concours visant à obtenir la meilleure qualité de courbes. Les participants devaient posséder une parte-forme définie et devait y charger une implémentation d'AES fournie par les organisateurs. Chaque participant pouvait acquérir ses propres courbes et les soumettre aux organisateurs pour évaluation. Cette édition courait jusqu'à fin juillet 2012.

DPA Contest V4

La quatrième édition a débuté en juillet 2013, elle concerne plusieurs implémentations logicielle d'AES et qui sont sécurisées par différentes méthodes. De nouvelles versions seront ajoutées au cours du concours.

1.7 Notations

Dans la suite du document nous utiliserons les notations suivantes relatives à l'algorithme de chiffrement AES-128 :

- $K_r = (k_{r,0}, \ldots, k_{r,15})$, la clef du $r^{\text{ème}}$ tour (pour $r = 0, \ldots, 10$)
- $X_r = (x_{r,0}, \ldots, x_{r,15})$, les entrées de l'opération `SubBytes` du tour r (pour $r = 1, \ldots, 10$)
- $Y_r = (y_{r,0}, \ldots, y_{r,15})$, les sorties de l'opération `SubBytes` du tour r (pour $r = 1, \ldots, 10$)
- $Z_r = (z_{r,0}, \ldots, z_{r,15})$, les entrées de l'opération `MixColumns` du tour r (pour $r = 1, \ldots, 9$)
- $T_r = (t_{r,0}, \ldots, t_{r,15})$, les sorties de l'opération `MixColumns` du tour r (pour $r = 1, \ldots, 9$)

Deuxième partie

Attaques physiques classiques

Chapitre 2

Attaque de type SPA sur le *key schedule* de l'AES revisitée

Sommaire

Les travaux réalisés dans ce chapitre ont été publiés en 2014 [CMW14]. Ils consistent en des attaques visant à retrouver une clef d'AES-128 en se basant sur la mesure des poids de Hamming des octets de la clef étendue. De précédents travaux ayant traité le cas sans contre-mesure nous avons envisagé plusieurs scenarii d'implémentations protégées.

2.1 Introduction

Dans ce chapitre nous considérons un attaquant capable de récupérer les poids de Hamming des données manipulées par la cible : une implémentation logicielle de l'AES sur un microprocesseur de taille de bus 8 bits. Les données obtenues par l'attaquant sont les poids de Hamming des octets des différentes clefs de tours, ciblés lors de leur calcul dans le *key schedule* ou bien lors de leur utilisation dans le chiffrement à l'étape `AddRoundKey` où ils sont ajoutés par opération XOR à l'état courant. Ce problème a été évoqué pour la première fois dans [BS99], mais c'est Mangard [Man03] qui fut le premier à décrire une attaque de ce type, puis VanLaven et al. [VBC05] l'améliorèrent en 2005. Ces travaux ne considérant que des implémentations non protégées, nous proposons dans ce chapitre une analyse de l'adaptation de ces attaques à des implémentations comportant des contre-mesures. Nous envisageons trois scenarii : deux comportant un masquage

booléen sur les clefs de tours et un comportant une exécution en ordre aléatoire d'opérations indépendantes du *key schedule*. Les contre-mesures à base de masquage empêchent l'attaquant de lire les poids de Hamming des octets et l'exécution en ordre aléatoire l'empêche de savoir précisément à quel octet correspond chacun des poids de Hamming observés.

Le chapitre est organisé comme suit : La définition du problème et les travaux déjà effectués sur le sujet sont détaillés en Section 2.2. La Section 2.3 répond une question laissée ouverte par Mangard sur l'existence de clefs présentant exactement la même séquence de poids de Hamming sur toute leur clef étendue. La Section 2.4 détaille notre contribution principale où nous décrivons nos attaques face à trois contre-mesures. À la lumière de ces attaques nous proposons des recommandations d'implémentations en Section 2.5 et la Section 2.6 conclut ce chapitre.

2.2 Description de la problématique et travaux précédents

Comme vu en Section 1.2.2, le *key schedule* de l'AES dérive onze clefs, une par tour, d'une taille de 16 octets chacune à partir d'une clef de chiffrement K composée de 16 octets. Elles sont notées K_r $(r = 0, \ldots, 10)$ avec $K_0 = K$.

La clef étendue $\{K_0, \ldots, K_{10}\}$ est calculée colonne après colonne par le biais de deux types de relations, une linéaire et une non-linéaire :

$$k_{r,i} = k_{r-1,i} \oplus k_{r,i-4} \quad (\text{pour } i = 4, \ldots, 15) \tag{2.1}$$

$$k_{r,i} = k_{r-1,i} \oplus S(k_{r-1,12+((i+1) \bmod 4)}) \oplus c'_r \quad (\text{pour } i = 0, \ldots, 3) \tag{2.2}$$

où S correspond à la table de substitution S-Box et où c'_r est une constante de tour spécifique égale à $\{\texttt{02}\}^{r-1}$ si $i = 0$ et égale à 0 si $i \in \{1, 2, 3\}$.

Le problème considéré dans ce chapitre est : comment identifier la clef de chiffrement K en se basant sur un ensemble $\{\mathrm{HW}(k_{r,i})\}_{r,i}$ composé de tout ou partie des poids de Hamming des octets de la clef étendue.

Mangard [Man03] a été le premier à donner une solution à ce problème. Il propose de construire des listes de valeurs pour un lot de 5 octets de clef qui soient à la fois compatibles avec les poids de Hamming observés pour ces octets mais aussi avec les poids de Hamming de 9 autres octets de clef (ainsi que plusieurs valeurs intermédiaires) calculables à partir de ces 5 octets.

Dans [VBC05] VanLaven et al. considèrent le même problème et donnent une analyse élégante des liens existant entre les octets de la clef étendue, ce qui leur permet de concevoir un algorithme efficace de type *guess-compute-and-backtrack*[1] où l'on suppose la valeur d'une séquence d'octets successifs dans un ordre optimal afin de maximiser le nombre d'autres octets qui sont calculables et que l'on confronte alors avec leurs poids de Hamming mesurés. Si une incohérence est observée l'algorithme considérera alors la prochaine valeur possible et pourra même remonter d'un niveau dans la séquence si toutes les valeurs pour l'octet du niveau courant ont été considérées. Par la suite nous utiliserons le terme *marche arrière* pour décrire ce retour en arrière dans l'arbre des possibles. La dernière contribution de ce papier consiste en une intéressante analyse du fait que leur algorithme peut aboutir à un succès même en cas de mesures (faiblement) bruitées au prix d'un effort calculatoire plus important.

1. lit. *supposition-calcul-et-marche-arrière** : Algorithme où l'on met en place des suppositions successives de manière à ce que si une supposition s'avère entrer en contradiction avec une contrainte il soit aisé de revenir en arrière pour essayer une autre combinaison sans avoir à refaire toute la succession de suppositions.

2.3 Travail préliminaire sur les clefs indistinguables

2.3.1 Clefs indistinguables : une question ouverte

Nous commençons par l'étude de la question ouverte à propos de l'existence de paires de clefs (appelées clefs jumelles) – ou plus généralement d'un ensemble de clefs – qui soient indistinguables car ayant la même signature en poids de Hamming [2]. Nous nous sommes donc intéressés à l'existence ou la non-existence de deux clefs K et K' telles que leurs clefs étendues possèdent les mêmes 176 poids de Hamming.

Si le *key schedule* de l'AES dérivait les clefs de tour K_1 à K_{10} avec un comportement aléatoire idéal, la probabilité qu'il existe deux clefs indistinguables serait très basse. En effet, la probabilité que deux octets aléatoires aient le même poids de Hamming est $p = 2^{-2.348}$ donc la probabilité que deux signatures de clefs aléatoires soient identiques est $q = p^{176} \simeq 2^{-413.3}$. Cela induit que la probabilité d'avoir au moins une collision de signature sur tout l'espace des clefs possibles est d'environ $1 - e^{-\frac{q}{2} \cdot 2^{2*128}} \simeq 2^{-158.3}$.

Même si le *key schedule* est loin d'avoir un comportement aléatoire, il a été considéré dans [Man03] que de telles clefs jumelles n'existent probablement pas ou devraient être très rares [3]. Nous montrons dans cette section que cette croyance est fausse en proposant une méthode constructive permettant de générer facilement des millions de telles clefs.

On peut remarquer que l'existence de clefs jumelles a un intérêt au niveau de la théorie car il apporte une nouvelle illustration du caractère imparfait de la diffusion dans le *key schedule*. En revanche il n'a pas d'impact concret dans les attaques considérées dans ce chapitre car sa seule conséquence est qu'un attaquant ciblant une clef d'une telle paire obtiendra deux clefs au lieu d'une seule à la fin de son attaque. L'identification de la bonne clef pourra alors être réalisée à l'aide d'un couple clair/chiffré connu.

2.3.2 Générer des clefs indistinguables

L'idée centrale de notre méthode vient de l'observation qu'avec une permutation τ de $\{0, \ldots, 7\}$ et une transformation d'octet $\pi : b = (b_7 \ldots b_0) \mapsto \pi(b) = (b_{\tau(7)} \ldots b_{\tau(0)})$ nous obtenons $\mathrm{HW}(\pi(b)) = \mathrm{HW}(b)$. De plus, une condition suffisante pour K et K' pour être jumelles est que $k'_j = \pi(k_j)$ pour tout $j = 0, \ldots, 175$. Notre but est de trouver une clef K telle qu'en définissant K' par $k'_{0,i} = \pi(k_{0,i}), i = 0, \ldots, 15$ la condition suffisante se propage jusqu'à la fin du *key schedule*, ou du moins presque jusqu'à la fin. Comme π est linéaire, la condition se propage automatiquement sur toutes les relations linéaires. La seule difficulté est de s'assurer de la propagation pour les relations non-linéaires. En notant $c_r = \{\mathtt{02}\}^{r-1}$ la constante impliquée dans la première relation non linéaire du tour $r = 1, \ldots, 10$, et en supposant que la condition suffisante a été propagée jusqu'à la clef K_{r-1}, elle se propagera alors à K_r si les équations suivantes sont respectées :

$$k'_{r,0} = \pi(k_{r,0}) \quad \Leftrightarrow \quad S(\pi(k_{r-1,13})) \oplus c_r = \pi(S(k_{r-1,13})) \oplus \pi(c_r) \tag{2.3}$$

$$k'_{r,1} = \pi(k_{r,1}) \quad \Leftrightarrow \quad S(\pi(k_{r-1,14})) = \pi(S(k_{r-1,14})) \tag{2.4}$$

$$k'_{r,2} = \pi(k_{r,2}) \quad \Leftrightarrow \quad S(\pi(k_{r-1,15})) = \pi(S(k_{r-1,15})) \tag{2.5}$$

$$k'_{r,3} = \pi(k_{r,3}) \quad \Leftrightarrow \quad S(\pi(k_{r-1,12})) = \pi(S(k_{r-1,12})) \tag{2.6}$$

2. Par *signature en poids de Hamming* d'une clef K nous parlons de l'ensemble de tous les poids de Hamming des octets de la clef étendue.

3. La phase exacte de l'auteur est : *The high diffusion of the AES key expansion suggests that there are only very few keys of this kind, if there are such keys at all.* qui peut être traduit par : *La diffusion élevée du key schedule suggère qu'il n'existe que peu de clefs de ce type, si seulement il en existe.*

Notre première tâche est donc de trouver une permutation de bits maximisant la probabilité que ces conditions soient conservées par hasard. La probabilité de garder cette condition pour les équations (2.4) à (2.6) est d'environ $\frac{1}{4}$ lorsque τ est une permutation de seulement deux bits[4]. Cela est dû au fait que $S(\pi(x)) = \pi(S(x))$ dès que $\pi(x) = x$ et $\pi(y) = y$ pour $y = S(x)$ où la double collision de point fixe a lieu avec la probabilité $\frac{1}{2}$. Trouver une paire de clef jumelles nécessite seulement que toutes les valeurs des $k_{r-1,i}$ ($r = 1, \ldots 10$ and $i = 12, \ldots, 15$) appartiennent aux ensembles suivants :

$$\begin{aligned} \Delta_r &= \{x : S(\pi(x)) \oplus c_r = \pi(S(x)) \oplus \pi(c_r)\} \quad (\text{pour } i = 13) \\ \Delta &= \{x : S(\pi(x)) = \pi(S(x))\} \quad (\text{pour } i \in \{12, 14, 15\}) \end{aligned}$$

Il est important que chacun des ensembles Δ et Δ_1 contiennent des valeurs x vérifiant la condition sans être un point fixe pour π sans quoi K' sera égal K. Nous avons choisi τ comme étant la permutation des bits 0 et 6. On peut noter que c'est la seule transposition de bit qui possède une valeur n'étant pas un point fixe dans Δ.

La deuxième tâche consiste en la génération d'un grand nombre de clef candidates vérifiant par construction autant de conditions que possible. Nous avons conçu une méthode qui génère efficacement un grand nombre de clefs candidates qui vérifient les conditions pour tous les tours $r \leqslant 5$. En premier lieu nous faisons varier les douze octets $k_{1,12+n}$, $k_{2,12+n}$ et $k_{3,12+n}$ ($n = 0, \ldots, 3$) qui sont libres à partir du moment où ils appartiennent à leurs ensembles Δ,Δ_2, Δ_3 ou Δ_4 respectifs. La précédente remarque donne un nombre de combinaisons minorée par $(256/4)^{12} = 2^{72}$. Nous utilisons aussi les relations suivantes sur les octets de clef :

$$k_{4,12+n} = k_{0,12+n} \oplus S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \tag{2.7}$$

$$k_{0,8+n} = k_{0,12+n} \oplus S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{3,12+n} \tag{2.8}$$

$$k_{0,4+n} = k_{0,8+n} \oplus \begin{cases} S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{2,12+n} \\ S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{3,12+n} \end{cases} \tag{2.9}$$

$$k_{0,0+n} = k_{0,8+n} \oplus \begin{cases} S(k_{0,12+(n+1) \bmod 4}) \oplus c'_1 \oplus k_{1,12+n} \\ S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{2,12+n} \end{cases} \tag{2.10}$$

où c'_r est défini comme étant c_r si $n = 0$ et 0 dans les autres cas. Pour le lecteur intéréssé l'obtention des Équations (2.7) à (2.10) est détaillé en annexe (voir l'Annexe A). En considérant l'Équation (2.7), et sachant que chacun des $k_{3,12\ldots15}$ a été choisi dans son ensemble Δ, on est capable de choisir des valeurs pour $k_{0,12+n}$, dans son ensemble Δ, de telle façon que $k_{4,12+n}$ soit aussi dans son propre ensemble Δ. Par exemple, avec une valeur de $k_{3,14} \in \Delta$ on peut trouver deux valeurs $k_{0,13}$ et $k_{4,13}$ qui seront respectivement dans Δ_1 et Δ_5. Il existe toujours plusieurs choix que nous avons tabulés bien qu'un seul soit suffisant dans notre implémentation. Choisir $k_{0,12+n}$ de cette manière nous assure que les conditions suffisantes seront vérifiées même pour les relations non linéaires impliquées dans le calcul de K_5.

Le processus de génération de clefs candidates se résume comme suit : on choisit arbitrairement les valeurs de $k_{1,12+n}$, $k_{2,12+n}$ et $k_{3,12+n}$ ($n = 0, \ldots, 3$) dans leurs ensembles Δ respectifs, nous choisissons alors $k_{0,12+n}$ comme expliqué ci-dessus, et nous terminons le calcul de $K = K_0$ en utilisant successivement les équations (2.8) à (2.10). Pour chaque clef K générée nous calculons K' en appliquant la transposition π à tous ses octets. Notre méthode nous assure par construction que $k'_{r,i} = \pi(k_{r,i})$ – et donc que $\mathrm{HW}(k'_{r,i}) = \mathrm{HW}(k_{r,i})$ – pour tout $r = 0, \ldots, 5$.

En générant suffisamment de clefs candidates, on peut espérer en trouver une pour laquelle le hasard propagera les conditions suffisantes à travers les relations non-linéaires jusqu'à la fin

4. C'est aussi vrai pour l'équation (2.3) pour une raison similaire.

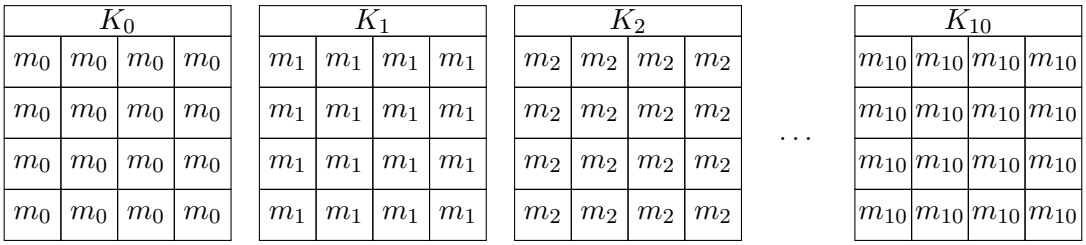

FIGURE 2.1 – Partie du masquage à 11 octets d'entropie

du *key schedule*. Après avoir trouvé une première paire de clef indistinguables égales à :

$$\begin{cases} K = \texttt{B3 65 58 9D B4 EB 57 72 1F 51 F7 58 02 0C 00 17} \\ K' = \texttt{F2 65 19 DC B4 EB 57 33 5E 51 F7 19 02 0C 00 56} \end{cases}$$

nous avons exploré le voisinage de cette solution et, de manière surprenante, avons pu générer un grand nombre d'autres paires de clefs indistinguables bien plus facilement que pour trouver la première paire. Par exemple en gardant les valeurs $k_{1,13}$, $k_{1,14}$, $k_{2,12}$ et $k_{2,13}$ impliquées dans la première paire, nous avons été capables de générer plus de 23 millions d'autres paires indistinguables en quelques jours de calcul. Cela tend à démonter que les paires de clefs partageant la même signature en poids de Hamming sont loin d'être uniformément distribuées, mais nous n'avons pas étudié plus en détail ce comportement.

2.4 Retrouver la clef face à des implémentations protégées

Dans cette section nous étudions trois contre-mesures différentes pouvant être implémentées pour protéger le *key schedule* des attaques par SPA.

Les deux premières contre-mesures sont deux façons d'appliquer un masquage booléen sur le *key schedule*. Elles utilisent respectivement 11 et 16 octets de masque afin de s'adapter à l'environnement réduit en mémoire vive et/ou à la difficulté à extraire suffisamment d'entropie des générateurs d'aléa des dispositifs embarqués. La troisième contre-mesure consiste en une modification aléatoire de l'ordre de calcul des octets à l'intérieur d'une colonne car ils sont indépendants.

2.4.1 Masquage booléen à 11 octets d'entropie

Nous considérons ici que, pour chaque exécution, les clefs de tour sont masquées par 11 octets aléatoires spécifiques m_r. Ainsi l'attaquant n'a plus accès aux fuites individuelles des octets $k_{r,i}$ de chaque K_r mais plutôt à leur version masquée $K'_r = (k'_{r,i})_i$ avec $k'_{r,i} = k_{r,i} \oplus m_r$. La Figure 2.1 détaille le schéma de masquage appliqué sur les octets du *key schedule*.

L'attaque de base ne peut s'appliquer directement car les poids de Hamming mesurés sont reliés aux octets de masques qui ne vérifient aucune relation, qu'elle soit linéaire ou non, du processus du *key schedule*.

Pour pouvoir appliquer une stratégie similaire de type *guess-compute-and-backtrack* nous devons à présent faire des suppositions sur les valeurs des masques appliqués sur tous les octets impliqués dans la recherche de la clef. Comme chaque octet de masque à supposer induit une multiplication par 2^8 de la taille de l'espace de recherche, nous utilisons deux astuces afin de contenir l'augmentation calculatoire de travail à effectuer.

La première idée est d'exploiter une information supplémentaire qui peut être acquise à propos des octets de clefs $k_{r,i}$ en considérant des mesures provenant de plusieurs acquisitions. Plus précisément, lorsqu'on a deux octets x et y masqués par la même valeur aléatoire m. Les poids de Hamming des octets masqués, notés respectivement $x' = x \oplus m$ et $y' = y \oplus m$ vérifient les deux propriétés suivantes :

$$|\,\mathrm{HW}(x') - \mathrm{HW}(y')| \leqslant \mathrm{HD}(x, y) \leqslant \min(8, \mathrm{HW}(x') + \mathrm{HW}(y')) \tag{2.11}$$

$$\mathrm{HD}(x, y) \equiv \mathrm{HW}(x') + \mathrm{HW}(y') \pmod 2 \tag{2.12}$$

Ces deux équations donnent de l'information sur la distance de Hamming entre les valeurs non masquées x et y. Supposons par exemple que $x = 30$ et $y = 121$ (i.e. $\mathrm{HD}(x, y) = 5$). Avec une première acquisition, pour laquelle $m = 70$, nous mesurons $\mathrm{HW}(x') = \mathrm{HW}(30 \oplus 70) = 3$ et $\mathrm{HW}(y') = \mathrm{HW}(121 \oplus 70) = 6$. Nous obtenons de l'équation (2.11) que $3 \leqslant \mathrm{HD}(x, y) \leqslant 8$, et l'imparité donnée par l'équation (2.12) nous indique que $\mathrm{HD}(x, y) \in \{3, 5, 7\}$. Avec une deuxième acquisition, pour laquelle $m = 24$, nous mesurons $\mathrm{HW}(x') = \mathrm{HW}(30 \oplus 24) = 2$ et $\mathrm{HW}(y') = \mathrm{HW}(121 \oplus 24) = 3$. Cette seconde mesure permet de contraindre plus précisément $\mathrm{HD}(x, y)$ qui à présent appartient à $\{3, 5\}$. L'exploitation de plus en plus d'acquisitions permet de réduire le nombre de candidats pour à terme identifier précisément la distance de Hamming entre les octets non masqués. Il est intéressant de noter que l'équation indiquant la parité peut être utilisée pour détecter des mesures erronées. Par exemple, si les mesures de dix acquisitions indiquent une valeur impaire pour $\mathrm{HW}(x') + \mathrm{HW}(y')$ à huit reprises et une valeur paire seulement deux fois, on peut alors fortement présumer que soit $\mathrm{HW}(x')$ soit $\mathrm{HW}(y')$ a été mal mesuré sur ces deux acquisitions.

Dans une première phase de l'attaque, plusieurs courbes sont analysées afin d'obtenir autant d'information que possible à propos de la distance de Hamming $\mathrm{HD}(k_{r,i}, k_{r,i'})$ entre chaque couple d'octets masqués par la même valeur et donc appartenant à la même clef de tour. Dans une seconde phase une exploration intelligente de l'espace de clef est réalisée à partir des poids de Hamming obtenus sur une seule courbe et l'information accumulée sur les distances de Hamming obtenues en première phase.

La deuxième idée permettant de réduire l'effort calculatoire est de limiter le processus de suppositions et de calcul des octets à seulement deux clefs adjacentes K_r et K_{r+1}. Cette méthode permet de n'avoir à supposer que deux octets de masque. Pour chaque candidat au couple (m_r, m_{r+1}) nous pratiquons une recherche de la clef en supposant successivement des valeurs d'octets de K_r et en dérivant les valeurs successives des octets de K_{r+1}. Par exemple, considérons que nous commençons la recherche par $k_{r,12}$ (de manière équivalente nous pourrions commencer par les positions 13, 14 ou 15). L'étape suivante sera alors de supposer la valeur de $k_{r,3}$ et calculer $k_{r+1,3}$. Ensuite supposer $k_{r,7}$ et calculer $k_{r+1,7}$. Continuer par supposer $k_{r,11}$ et calculer $k_{r+1,11}$, et ainsi de suite. La Figure 2.2 montre l'ordre dans lequel les octets successifs de K_r et K_{r+1} sont respectivement supposés et calculés. Comme dans l'attaque de base, à chaque fois qu'un octet est supposé ou calculé nous confrontons sa valeur à la mesure du poids de Hamming de sa valeur masquée. Le contrôle le plus efficace consiste en la vérification que chaque octet nouvellement supposé ou calculé possède une distance de Hamming compatible avec tous les autres octets déjà obtenus dans la même clef de tour. Par exemple, lorsque $k_{r,11}$ est supposé lors de la troisième étape, quatre contraintes sur $\mathrm{HW}(k_{r,11} \oplus m_r)$, $\mathrm{HD}(k_{r,11}, k_{r,7})$, $\mathrm{HD}(k_{r,11}, k_{r,3})$ et $\mathrm{HD}(k_{r,11}, k_{r,12})$ sont vérifiées, et lorsque $k_{r+1,11}$ est calculé, ce sont trois contraintes impliquant $\mathrm{HW}(k_{r+1,11} \oplus m_{r+1})$, $\mathrm{HD}(k_{r+1,11}, k_{r+1,7})$ et $\mathrm{HD}(k_{r+1,11}, k_{r+1,3})$. Comme on a pu le voir, plus on avance dans l'exploration plus on a d'opportunités d'invalider de mauvaises suppositions et ainsi effectuer une marche arrière.

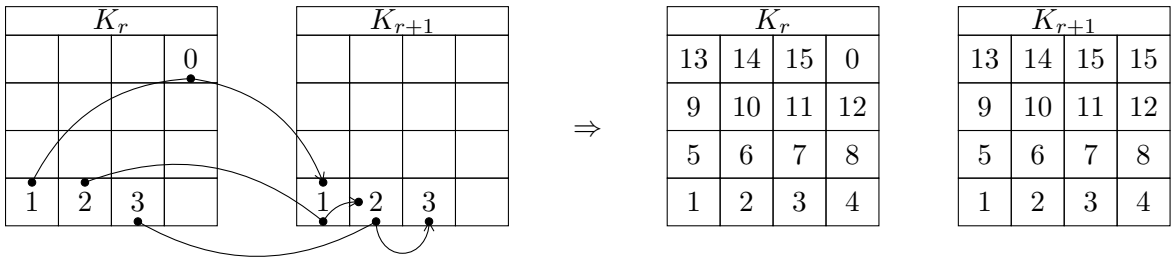

FIGURE 2.2 – Ordre des suppositions et calculs dans le schéma de masquage à 11 octets d'entropie

Nous avons effectué de nombreuses simulations de cette attaque en générant un ensemble de mesures parfaites de poids de Hamming. Pour différents nombres de courbes exploitées T ($T \in \{5, 10, 15, 20, 30\}$) – ce nombre influence l'écart des bornes de distances de Hamming dérivées des courbes – nous avons lancé N simulations de l'attaque ($N = 1000$ dans la plupart des cas). Pour chaque simulation nous avons sélectionné aléatoirement une clef et pour chacune des T exécutions nous avons calculé une dérivation de clef masquée par un ensemble aléatoire de masques $(m_0, \ldots, m_{10})$. De cette exécution nous avons dérivé un ensemble de poids de Hamming supposés êtres accessibles à l'attaquant. Ayant fixé un numéro de tour r, nous avons établi l'ensemble des candidats aux distances de Hamming pour l'ensemble des couples $(k_{r,i}, k_{r,i'})$ et $(k_{r+1,i}, k_{r+1,i'})$. Nous avons alors choisi une courbe en particulier (un ensemble de poids de Hamming donc) parmi les T disponibles ainsi qu'une position de départ des suppositions [5], à la suite de quoi nous exécutons la seconde phase de l'attaque (le processus d'exploration).

La Table 2.1 présente les résultats de simulations obtenus sur un PC classique équipé d'un processeur I5 à 2.4GHz avec 4GB de RAM*. Pour chaque nombre de courbes exploitées nous indiquons le temps moyen d'exécution et l'entropie résiduelle moyenne de la clef (le $\log_2$ du nombre de clefs compatibles retournées par l'attaque). À cause d'une variance élevée dans le temps d'exécution nous avons choisi d'interrompre les simulations au bout d'un *temps maximal autorisé*. Sa valeur et le pourcentage d'exécutions s'étant terminées avant cette limite sont aussi indiqués. Les valeurs présentes dans les deuxième et troisième colonnes sont calculées avec les simulations qui sont arrivées à terme avant l'écoulement du temps maximal.

L'attaque proposée est efficace, même dans le cas où le nombre de courbes exploitées est réduit à cinq. Dans ce cas 45% des exécutions se terminent en moins de 30 minutes et l'entropie de clef restante à explorer n'est que de cinq bits.

Remarque 1. *D'un point de vue pratique lié à la capacité de l'attaquant à inférer les poids de Hamming depuis les fuites d'informations des courbes, nous notons que cette attaque n'utilise pas l'ensemble des 176 poids de Hamming mais seulement 32. De même, l'opportunité laissée à l'attaquant de choisir quelle clef de tour sera attaquée peut être utilisée pour se placer sur la portion de courbe où il est le plus confiant en ses mesures de poids de Hamming.*

5. Il est à noter qu'un attaquant peut choisir : la courbe exploitée durant la recherche de clef, le tour r entre 0 et 9 dans lequel la recherche va avoir lieu et la position de départ entre 12 et 15. Nous exploitons ces opportunités de choix pour sélectionner la combinaison qui minimise le nombre de valeurs possibles du triplet de départ – e.g. $(k_{r,12}, k_{r,3}, k_{r+1,3})$ dans l'exemple ci-dessus – compatibles avec les poids de Hamming mesurés.

TABLE 2.1 – Résultats de simulation de l'attaque sur la version masquée à 11 octets d'entropie

Nombre de courbes (T)	Temps moyen (s)	Entropie résiduelle moyenne (bits)	Temps maximum autorisé (s)	Pourcentage sous le temps maximum	Nombre d'exécutions (N)
5	398	5.9	1800	47.0	83
10	40.6	0.66	300	93.4	500
15	10.0	0.29	60	94.7	1000
20	5.9	0.24	60	98.2	1000
30	3.0	0.24	60	100.0	1000

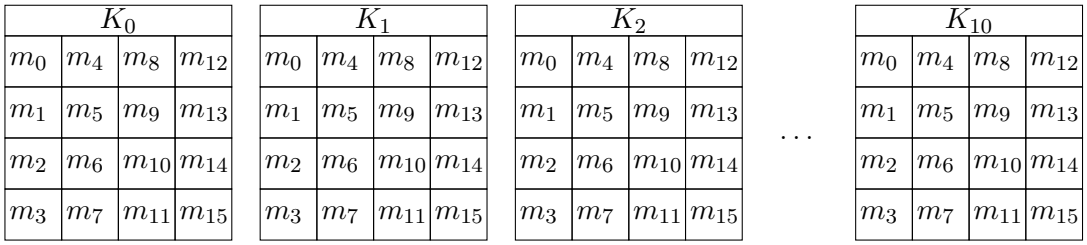

FIGURE 2.3 – Partie du masquage à 16 octets d'entropie

2.4.2 Masquage booléen à 16 octets d'entropie

La deuxième contre-mesure que nous avons étudiée consiste en un masquage de tous les octets d'une clef de tour par un octet aléatoire potentiellement différent, et on répète ces mêmes 16 masques pour tous les tours. Plus précisément chaque clef est définie comme $K'_r = (k'_{r,i})_i$ avec $k'_{r,i} = k_{r,i} \oplus m_i$ $(i = 0, \ldots, 15)$. La Figure 2.3 détaille ce schéma de masquage appliqué sur la clef étendue.

Comme dans le cas de l'attaque sur le schéma de masquage à 11 octets d'entropie, nous allons en premier lieu exploiter plusieurs courbes afin d'obtenir des informations sur les distances de Hamming entre les octets de clef partageant le même masque. Nous souhaitons aussi limiter à deux le nombre de valeurs de masques à supposer simultanément dans la partie la plus sujette à l'explosion combinatoire (là où le nombre de contraintes est le plus bas). Il en découle que l'ordre de supposition des valeurs des octets sera plus avantageux en s'étendant horizontalement plutôt qu'en restant dans une seule clef du tour r.

En prenant une position de départ $a \in \{0, 1, 2, 3\}$, nous définissons la position reliée à cette première par $b = 12 + ((a + 1) \bmod 4)$. Pour chaque supposition sur le couple de masques (m_a, m_b), nous réalisons une exploration de l'espace de clef possibles comme suit. En premier lieu nous supposons $k_{0,a}$. Alors, de manière successive pour $r = 0, \ldots, 9$ nous supposons la valeur de $k_{r,b}$ et nous calculons $k_{r+1,a}$. Comme pour l'attaque décrite en Section 2.4.1, chaque octet nouvellement supposé ou calculé est vérifié vis-à-vis de son poids de Hamming et ses distances de Hamming avec d'autres octets de clefs déjà connus. Nous avons alors effectué la partie la plus ardue de l'exploration car nous avons eu à faire de nouvelles suppositions pour chaque $k_{r,b}$. Il nous reste alors un petit nombre de clef candidates encore compatibles pour lesquelles nous connaissons toutes les valeurs aux postions a et b (excepté pour $k_{10,b}$). Nous supposons alors la valeur de $k_{10,b}$ qui est très contrainte par les distances avec toutes les autres valeurs à la position

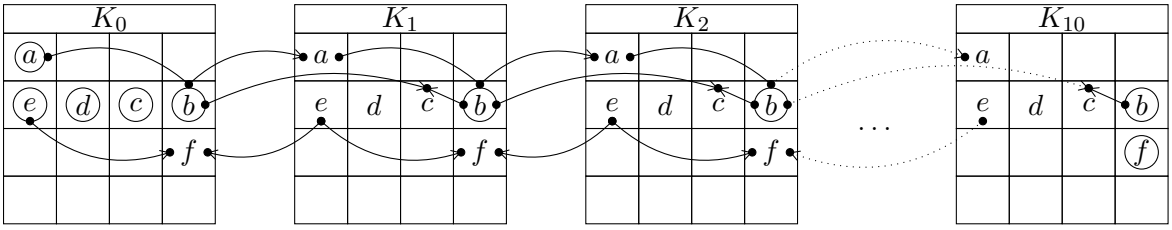

FIGURE 2.4 – Ordre des suppositions et calculs dans le schéma de masquage à 16 octets d'entropie

b ce qui n'augmente pas beaucoup la taille de l'exploration. Connaissant $k_{10,b}$, nous pouvons alors successivement calculer les octets aux positions $c = b - 4$ depuis $k_{10,c}$ jusqu'à $k_{1,c}$. Il est à noter que m_c est maintenant la seule valeur que nous ayons à supposer pour étendre la remontée jusqu'à $k_{0,c}$. À présent, en supposant le masque m_d en position $d = c - 4$ nous pouvons, de la même manière, calculer la ligne d de $k_{10,d}$ à $k_{1,d}$, et terminer la ligne en supposant la valeur $k_{0,d}$. Nous pouvons poursuivre le processus avec une ligne supplémentaire en position $e = d - 4$ et ainsi la ligne $f = 12 + ((b+1) \bmod 4)$ est calculée de $k_{0,f}$ à $k_{9,f}$ en terminant sur une supposition pour $k_{10,f}$. Successivement nous pouvons déterminer tous les octets de la clef étendue, ligne après ligne, aux positions représentées par $a, b, c, \ldots$ sur la Figure 2.4.

Il est intéressant de noter une propriété de la ligne a qui permet de grandement accélérer l'attaque. Pour chaque solution trouvée pour les lignes a et b par supposition sur le couple de masques (m_a, m_b) nous avons une autre solution reliée, pour une autre valeur m'_a, où tous les $k_{r,b}$ sont identiques et où chaque valeur $k_{r,a}$ est remplacée par $k_{r,a} \oplus (m_a \oplus m'_a)$. Du fait que les octets en position a n'influencent pas ceux retrouvés sur les lignes successives b, c et d,... nous n'avons pas à connaître la valeur exacte de m_a et pouvons donc la fixer arbitrairement. À la fin de l'attaque nous sommes capables de calculer la valeur correcte de la ligne a en retrouvant l'erreur faite sur m_a en se basant, par exemple, sur la différence entre $k_{10,a}$ et sa valeur exacte obtenue par $k_{9,p} \oplus k_{10,p}$ avec $p = a + 4$. En procédant ainsi, la première partie de l'exploration, qui consiste à connaître les valeurs aux positions a et b, peut être réalisée en ne faisant des suppositions que sur un octet de masque : m_b. Un gain en rapidité de 2^8 est ainsi obtenu, ce qui rend cette attaque particulièrement efficace.

La Table 2.2 présente les résultats de simulation pour cette attaque de la même manière qu'en Section 2.4.1. Il est surprenant de constater que l'attaque retrouvant la clef en présence d'une contre-mesure à 16 octets d'entropie est bien plus efficace que celle à 11 octets d'entropie, alors que l'entropie aurait suggéré la situation inverse. Par exemple la clef est retrouvée ici en moins d'une seconde en moyenne lorsque 10 courbes sont utilisées face à 40 secondes pour le masque à 11 octets d'entropie. Il est alors possible de réduire le nombre de courbes utilisées à 3, tout en gardant un temps d'exécution et une entropie résiduelle faibles dans la majorité des cas.

2.4.3 Modification aléatoire de l'ordre des exécutions

La troisième contre-mesure consiste à calculer des octets indépendants dans un ordre aléatoire. La structure de calcul par colonnes du *key schedule* fait que les quatre octets de chaque colonne peuvent être calculés indépendamment. La Figure 2.5 présente un exemple d'une des séquences de permutations possibles.

Cette contre-mesure camoufle une partie de l'information. Nous pouvons toujours supposer que l'attaquant est capable d'identifier correctement les 176 poids de Hamming mais pour chaque

TABLE 2.2 – Résultats de simulation de l'attaque sur la version masquée à 16 octets d'entropie

Nombre de courbes (T)	Temps moyen (s)	Entropie résiduelle moyenne (bits)	Temps maximum autorisé (s)	Pourcentage sous le temps maximum	Nombre d'exécutions (N)
3	77.3	7.3	600	60.7	28
5	25.3	4.2	300	88.5	1000
10	1.09	1.7	60	100.0	1000
15	0.24	0.93	60	100.0	1000
20	0.12	0.55	60	100.0	1000
30	0.07	0.24	60	100.0	1000

K_0			
$k_{0,2}$	$k_{0,4}$	$k_{0,9}$	$k_{0,14}$
$k_{0,3}$	$k_{0,6}$	$k_{0,11}$	$k_{0,12}$
$k_{0,1}$	$k_{0,5}$	$k_{0,10}$	$k_{0,13}$
$k_{0,0}$	$k_{0,7}$	$k_{0,8}$	$k_{0,15}$

K_1			
$k_{1,0}$	$k_{1,7}$	$k_{1,11}$	$k_{1,15}$
$k_{1,1}$	$k_{1,6}$	$k_{1,10}$	$k_{1,14}$
$k_{1,3}$	$k_{1,5}$	$k_{1,9}$	$k_{1,12}$
$k_{1,2}$	$k_{1,4}$	$k_{1,8}$	$k_{1,13}$

K_2			
$k_{2,3}$	$k_{2,7}$	$k_{2,8}$	$k_{2,13}$
$k_{2,2}$	$k_{2,5}$	$k_{2,9}$	$k_{2,15}$
$k_{2,0}$	$k_{2,6}$	$k_{2,11}$	$k_{2,14}$
$k_{2,1}$	$k_{2,4}$	$k_{2,10}$	$k_{2,12}$

...

FIGURE 2.5 – Partie d'une exécution avec contre-mesure donnant un ordre aléatoire aux opérations indépendantes

colonne il peut seulement obtenir un ensemble non ordonné de 4 valeurs. Par exemple, en prenant la clef d'exemple représentée sur la Figure 2.6, où le poids de Hamming des octets est représenté dans le coin droit des cases, l'information à laquelle un attaquant peut accéder est détaillée dans la Figure 2.7. Les octets de clef de chaque colonne ont été permutés de manière aléatoires donc l'attaque ne peut inférer qu'un quadruplet non ordonné de poids de Hamming.

Au vu du fait que les 24 permutations possibles d'un quadruplet peuvent être considérées comme valides a priori [6], l'effort calculatoire afin de considérer la combinaison de toutes les permutations de toutes les colonnes rend la recherche de clef inaccessible. Afin de réduire le coût exploratoire nous utilisons ce que nous nommons un système de réservation. Pendant notre attaque nous allons réserver des poids de Hamming à des positions fixes, soit par choix lorsqu'on fera une supposition pour une valeur, soit par contrainte lorsqu'elle sera calculée à partir d'autres octets de clefs déjà fixés. Une fois réservé à une position donnée, le poids de Hamming en question est alors indisponible aux autres positions de sa colonne à moins qu'une marche arrière, liée à la modification de la valeur du dernier octet ayant été supposé, ne l'ait libéré de sa réservation.

Par exemple, si nous avons à supposer la valeur de $k_{1,15}$ nous allons commencer par supposer son poids de Hamming dans la liste $\{2,3,4,5\}$ des poids de Hamming disponibles pour sa colonne. En supposant $\mathrm{HW}(k_{1,15}) = 4$, nous fixons la liste des valeurs possibles pour $k_{1,15}$ à toutes celles ayant un poids de Hamming égal à 4 et la liste des poids de Hamming restant disponibles pour le reste de la colonne est alors réduite à $\{2,3,5\}$. Lorsqu'un autre octet de la même colonne sera supposé (ou calculé), lors d'une étape plus profonde du processus d'exploration, il devra

6. Ce nombre est réduit dans le cas où des poids de Hamming se répètent, donc certaines colonnes peuvent présenter moins de 24 permutations possibles.

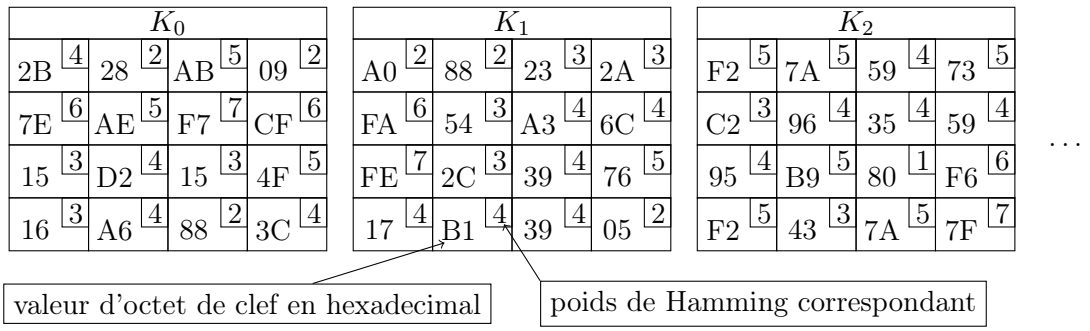

FIGURE 2.6 – Trois premières clefs de tour dérivées d'une clef d'exemple avec leurs poids de Hamming correspondant

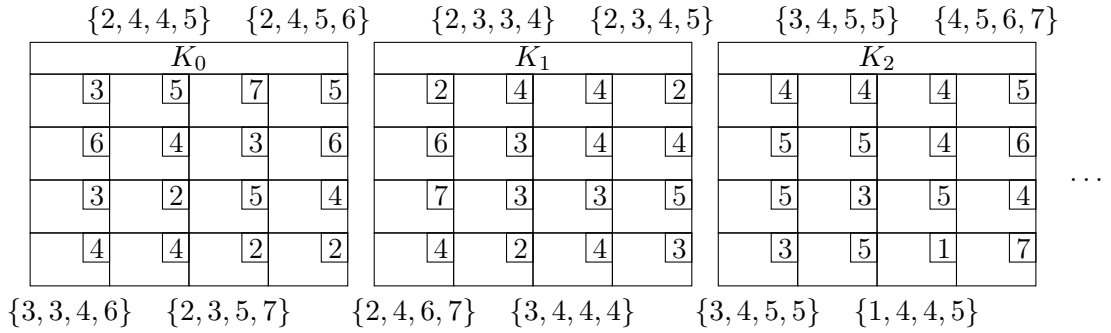

FIGURE 2.7 – Information récupérée par l'attaquant, réduite à un quadruplet de poids de Hamming par colonne

TABLE 2.3 – Résultats de l'attaque sur l'implémentation modifiant aléatoirement l'ordre des opérations, sans utilisation d'injections de fautes

Temps de l'exécution	$\leqslant$ 30 min	$\leqslant$ 1h	$\leqslant$ 2h	$\leqslant$ 3h	$\leqslant$ 4h	$\leqslant$ 5h	$\leqslant$ 6h	+ 6h
# sur 100 exécutions	6	25	41	55	66	71	73	27

nécessairement appartenir à cet ensemble réduit. Si à un moment de l'exploration une marche arrière a lieu en incluant la valeur $k_{1,15}$ alors le poids de Hamming 4 est relâché et sera donc à nouveau disponible pour d'autres octets de cette colonne.

Nous détaillons ci-après deux versions de cette attaque, une utilisant l'information donnée par une seule acquisition, ce qui peut prendre un temps non négligeable, et une version plus rapide qui exploite des exécutions fautées afin de collecter plus d'information.

2.4.3.1 Attaque sans fautes

Dans cette version basique de notre attaque nous suivons un schéma d'exploration équivalent à celui présenté dans [VBC05] pour une implémentation non protégée. La seule différence réside dans le fait que chaque octet peut avoir plusieurs poids de Hamming au lieu d'un seul. Comme expliqué ci-dessus, avant de faire la supposition de la valeur de l'octet à la position en cours d'étude nous avons à supposer la valeur de poids de Hamming non encore réservée qui va être utilisée à cette position et nous allons la réserver pendant que l'on teste de manière exhaustive les valeurs d'octet correspondantes. Lorsque nous avons effectué suffisamment de suppositions pour pouvoir calculer d'autres octets de clef nous vérifions que le poids de Hamming de ces valeurs est disponible dans la colonne correspondante et nous les réservons aussi. Si le poids de Hamming d'une valeur calculée n'est pas disponible, car absente ou bien déjà réservée, cette solution est donc invalide et nous effectuons une marche arrière sur le dernier octet dont on a supposé la valeur. À noter que si un poids de Hamming apparaît n fois dans une colonne il peut aussi y être réservé n fois.

Nous avons effectué des simulations de cette attaque en considérant des clefs aléatoires et les quadruplets non ordonnés de poids de Hamming pour chacune des colonnes. La Table 2.3 présente la proportion de 100 exécutions qui se terminent avant un temps limité variant de 30 minutes à 6 heures. Comme ces exécutions peuvent prendre un temps non déterminé nous avons fait le choix de l'interrompre après 6 heures (27 % des cas). On peut remarquer que le temps moyen pour les exécutions qui n'ont pas été interrompues est d'environ 2 heures, donc le temps moyen sur toutes les exécutions est potentiellement plus long.

2.4.3.2 Attaque en fautes

Nous décrivons ici une version plus efficace de l'attaque utilisant l'injection de fautes afin de réduire significativement le temps d'exécution pour la recherche de clef.

Nous supposons que l'attaquant est capable d'induire des fautes dans un des octets d'une colonne choisie, nous prenons l'exemple de la première colonne dans les explications qui suivent. Le modèle de faute considéré suppose une modification aléatoire de l'octet ciblé par la faute.

La propriété du *key schedule* de l'AES utilisée dans cette attaque est qu'une différence induite sur un octet de clef de la première colonne va se propager selon un schéma fixe d'octets actifs. Par exemple, si la faute modifie la valeur de $k_{0,0}$ alors on obtient les positions des octets actifs tels

K_0			
$k_{0,0}$	$k_{0,4}$	$k_{0,8}$	$k_{0,12}$
$k_{0,1}$	$k_{0,5}$	$k_{0,9}$	$k_{0,13}$
$k_{0,2}$	$k_{0,6}$	$k_{0,10}$	$k_{0,14}$
$k_{0,3}$	$k_{0,7}$	$k_{0,11}$	$k_{0,15}$

K_1			
$k_{1,0}$	$k_{1,4}$	$k_{1,8}$	$k_{1,12}$
$k_{1,1}$	$k_{1,5}$	$k_{1,9}$	$k_{1,13}$
$k_{1,2}$	$k_{1,6}$	$k_{1,10}$	$k_{1,14}$
$k_{1,3}$	$k_{1,7}$	$k_{1,11}$	$k_{1,15}$

K_2			
$k_{2,0}$	$k_{2,4}$	$k_{2,8}$	$k_{2,12}$
$k_{2,1}$	$k_{2,5}$	$k_{2,9}$	$k_{2,13}$
$k_{2,2}$	$k_{2,6}$	$k_{2,10}$	$k_{2,14}$
$k_{2,3}$	$k_{2,7}$	$k_{2,11}$	$k_{2,15}$

...

FIGURE 2.8 – Partie du schéma induit par une faute sur le premier octet de la première colonne de K_0

que représentés sur la Figure 2.8, où seules les trois premières clefs de tour sont représentées [7]. L'exécution aléatoire d'opérations rend impossible à l'attaquant de savoir quel octet parmi $k_{0,0}$, $k_{0,1}$, $k_{0,2}$ ou $k_{0,3}$ a été touché par la faute injectée dans la première colonne, mais l'important est que les positions relatives des octets actifs sont fixes (données par le schéma en Figure 2.8) et sont connues de l'attaquant.

Comme dans l'attaque de base décrite ci-dessus, l'attaquant peut exploiter une exécution sans injection de faute afin d'obtenir les quadruplets de poids de Hamming de référence, un pour chaque colonne.

Lorsqu'il exploite une exécution avec injection de faute, l'attaquant peut comparer, pour chaque colonne, le quadruplet potentiellement modifié avec celui de référence. Il est de même capable d'identifier quels poids de Hamming ont été modifiés et donc ceux qui concernent des octets actifs. Considérons un exemple où l'octet fauté est $k_{0,2}$ dont la valeur aurait été modifiée de `0x15` en `0xB1`. Cet exemple est décrit dans la Figure 2.9 où on peut voir tous les octets actifs. Il est à noter que dans cet exemple certains octets actifs ($k_{2,5}$, $k_{2,9}$ et $k_{2,13}$) on été modifiés mais que leur poids de Hamming est pourtant resté inchangé.

La contre-mesure donne à l'attaquant une vision telle que celle de la Figure 2.10 où chaque colonne a subit une permutation aléatoire. Rappelons que l'attaquant ne connaît ni la valeur, ni la position des octets actifs (colorés en rouge sur la figure) mais il obtient seulement les quadruplets de poids de Hamming. En comparant par exemple le quadruplet original ({2,4,6,7} de la Figure 2.7) et le fauté ({2,4,4,6} de la Figure 2.10) de la cinquième colonne, il est capable d'inférer que 7 est le poids de Hamming du seul octet actif dans cette colonne. De manière similaire il peut inférer que le poids de Hamming de la seule cellule active de la huitième colonne est 5. Si on considère la onzième colonne, l'attaquant ne peut inférer qu'une information partielle qu'un des deux octets actifs a la valeur 1 pour poids de Hamming.

Même si l'information retrouvée à propos des poids de Hamming de chaque colonne est incomplète, nous pouvons pourtant les exploiter durant l'algorithme de recherche de la clef. Par exemple, dans le processus *guess-compute-and-backtrack*, lorsqu'on fera la supposition que la valeur de $k_{1,3}$ a la valeur 7 pour poids de Hamming (donc que $k_{1,3}$ est un octet actif), alors dans la huitième colonne l'octet actif sera nécessairement placé dans la dernière case de la colonne (c.f. le schéma d'octets actifs de la Figure 2.8), nous savons alors que $\mathrm{HW}(k_{1,15}) = 5$.

Comme on peut le voir, le principe de l'attaque en faute est d'exploiter durant la recherche de la clef l'information à propos du poids de Hamming des octets actifs (dont la position verticale relative est fixe) qui a été acquise en comparant les quadruplets de poids de Hamming d'exécu-

7. Le schéma ne se limite évidemment pas à ces trois clefs de tour mais s'étend sur l'intégralité de la clef étendue.

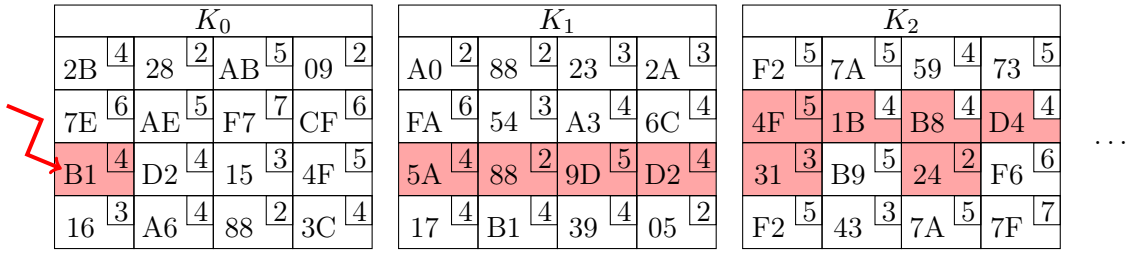

FIGURE 2.9 – Détails des effets d'une faute sans prendre en compte l'effet de la contre-mesure (Rouge/Gris foncé : Modifié, Blanc : Inchangé)

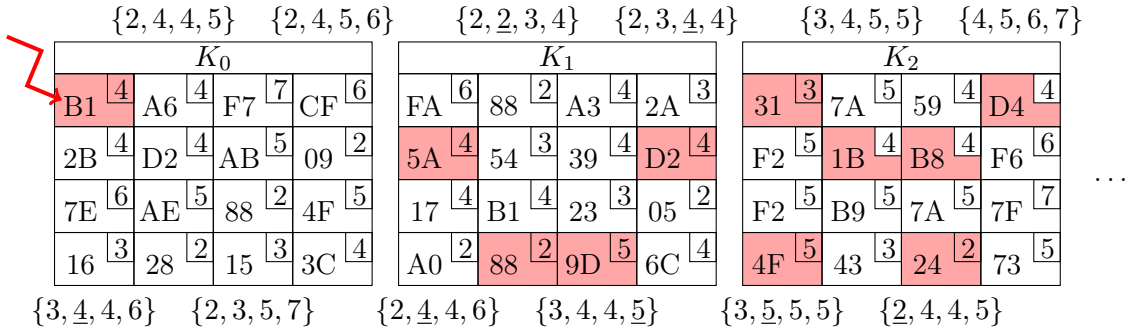

FIGURE 2.10 – Point de vue de l'attaquant de l'exécution avec faute, les valeurs soulignées sont celles que l'attaquant reconnaît comme modifiées par la faute

tions avec injection d'une faute et de ceux de référence. Même si les explications sont complexes il est possible d'utiliser plusieurs exécutions avec injections de faute pour augmenter la rapidité de la phase de recherche de la clef.

Nous avons effectué des simulations de l'attaque en faute en exploitant autant d'informations données par les fautes que possible. Nous détaillons dans la Table 2.4 les temps d'exécutions moyens de la phase de recherche en fonction du nombre de fautes exploitées. À noter que même avec une seule faute le temps moyen de la recherche est très fortement réduit, de plusieurs heures à seulement 20 minutes.

Remarque 2. *Il est intéressant de remarquer que si la faute n'a pas lieu sur la première colonne l'attaque reste possible même si elle devient potentiellement moins efficace. En effet le schéma d'octets actifs induit par une faute dans n'importe quelle colonne est toujours un sous ensemble*

TABLE 2.4 – Résultats de l'attaque sur l'implémentation modifiant aléatoirement l'ordre des opérations, avec utilisation d'injections de fautes

Nombre de fautes	Temps (min)
1	20
5	5
10	3
20	2
30	2

du schéma induit par une faute dans la première colonne. En conséquence ce schéma plus court possède la même forme générale que celui commençant en première colonne et peut donc être exploité de la même manière bien qu'il fournisse de l'information uniquement sur les colonnes calculées après l'injection de la faute. Cela permet d'effectuer cette attaque même dans le cas où l'attaquant n'a pas un contrôle précis de l'instant de l'injection de la faute.

2.5 Recommandations pour implémentations sécurisées

La contribution originelle [Man03] concernant le recouvrement de clef par analyse des poids de Hamming des octets de clefs du *key schedule* proposait plusieurs contre-mesures parmi lesquelles les masquages booléens du *key schedule*. Nous avons montré ici que deux versions de cette contre-mesure avec 11 et 16 octets d'entropie de masque n'empêchent pas un attaquant de retrouver la clef lorsqu'il a la capacité d'obtenir précisément les poids de Hamming. Contrairement à [Man03], nos attaques sur le masquage booléen s'appliquent aussi si le calcul des clefs de tours est effectué une fois pour toutes et n'est jamais réalisé sur le dispositif embarqué. Dans cette configuration les poids de Hamming ne peuvent pas être mesurés durant leur calcul mais pendant leurs transferts en RAM et/ou leur utilisation dans la fonction `AddRoundKey`.

L'utilisation d'une implémentation de l'AES matérielle ou logicielle sur processeur 16 ou 32 bits permet de se protéger de nos attaques qui ne s'appliquent que sur les implémentations logicielles sur processeur 8 bits. Sur ces dernières nous suggérons d'implémenter (si cela est possible) un masquage complet à 176 octets d'entropie où chaque octet est masqué par des valeurs aléatoires indépendantes, ou bien de combiner un masquage de taille inférieure à d'autres contre-mesures afin de renforcer sa sécurité. Par exemple, la combinaison d'une des deux méthodes de masquage étudiées dans ce chapitre avec celle rendant aléatoires l'ordre de calcul des opérations indépendantes devait être suffisante pour empêcher l'attaquant d'obtenir suffisamment d'information exploitable.

De la même manière, pour la manipulation des octets de clef lors du processus de chiffrement, la combinaison de masquage et de modification aléatoire de l'ordre des étapes indépendantes devrait être suffisante. Elle possède de plus l'avantage que l'entropie de la modification aléatoire est ici plus élevée car tous les 16 octets peuvent être utilisés dans un ordre aléatoire au lieu de ne pouvoir le faire que par lots de 4 octets. Une autre protection évidente consiste à utiliser des méthodes rendant difficile la découverte des points d'intérêt sur la courbe – par exemple des délais aléatoires – ou rendant difficile l'interprétation des fuites en termes de poids de Hamming – ajout de bruit sur les données mesurables –, ces méthodes ajoutant une sécurité supplémentaire aux implémentations de l'AES.

2.6 Conclusion

Dans ce chapitre nous avons revisité une attaque de type SPA sur le *key schedule* de l'AES. Les travaux précédents ne s'appliquant que sur des implémentations non protégées, nous avons étudié trois contre-mesures différentes et avons détaillé des attaques efficaces pour chacun des scenarii. Pour les deux masquages booléens (masques d'entropies de 11 et 16 octets respectivement) nos attaques retrouvent la clef en quelques secondes lorsque quelques courbes sont utilisées. Dans le cas d'une contre mesure à exécution en ordre aléatoire des opérations indépendantes nous avons détaillé une attaque prenant plusieurs heures en moyenne et avons proposé une version plus efficace qui tire avantage d'une information supplémentaire fournie par une analyses des

effets d'injections de fautes. Cette amélioration réduit le temps de recherche à quelques minutes seulement.

Nos attaques font la supposition que l'attaquant est capable d'obtenir les valeurs correctes des poids de Hamming des octets de clef. Comme travail futur il serait intéressant d'étudier la difficulté de travailler avec des observations erronées.

Annexe A : Preuves des équations (2.7) à (2.10)

Cette annexe détaille l'obtention de quatre équations utilisées en Section 2.3.2.

Équation (2.7)

$$k_{4,12+n} = k_{0,12+n} \oplus S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4$$

Démonstration.

$$\begin{aligned}
k_{4,12+n} &= k_{4,8+n} \oplus k_{3,12+n} \\
&= k_{4,4+n} \oplus k_{3,12+n} \oplus k_{3,8+n} \\
&= k_{4,0+n} \oplus k_{3,12+n} \oplus k_{3,8+n} \oplus k_{3,4+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{3,12+n} \oplus k_{3,8+n} \oplus k_{3,4+n} \oplus k_{3,0+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{3,4+n} \oplus k_{3,0+n} \oplus k_{2,12+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{2,12+n} \oplus k_{2,4+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{2,8+n} \oplus k_{2,4+n} \oplus k_{1,12+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{1,12+n} \oplus k_{1,8+n} \\
&= S(k_{3,12+(n+1) \bmod 4}) \oplus c'_4 \oplus k_{0,12+n}
\end{aligned}$$

□

Équation (2.8)

$$k_{0,8+n} = k_{0,12+n} \oplus S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{3,12+n}$$

Démonstration.

$$\begin{aligned}
k_{3,12+n} &= k_{3,8+n} \oplus k_{2,12+n} \\
&= k_{3,4+n} \oplus k_{2,12+n} \oplus k_{2,8+n} \\
&= k_{3,0+n} \oplus k_{2,12+n} \oplus k_{2,8+n} \oplus k_{2,4+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus k_{2,8+n} \oplus k_{2,4+n} \oplus k_{2,0+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,4+n} \oplus k_{2,0+n} \oplus k_{1,12+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{1,12+n} \oplus k_{1,4+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{1,8+n} \oplus k_{1,4+n} \oplus k_{0,12+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{0,12+n} \oplus k_{0,8+n}
\end{aligned}$$

□

Équation (2.9)

$$k_{0,4+n} = k_{0,8+n} \oplus S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{2,12+n} \oplus S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{3,12+n}$$

Démonstration.

$$\begin{aligned}
k_{3,12+n} &= k_{3,8+n} \oplus k_{2,12+n} \\
&= k_{3,4+n} \oplus k_{2,12+n} \oplus k_{2,8+n} \\
&= k_{3,0+n} \oplus k_{2,12+n} \oplus k_{2,8+n} \oplus k_{2,4+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus k_{2,8+n} \oplus k_{2,4+n} \oplus k_{2,0+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus k_{2,0+n} \oplus k_{1,8+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{1,8+n} \oplus k_{1,0+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{1,4+n} \oplus k_{1,0+n} \oplus k_{0,8+n} \\
&= S(k_{2,12+(n+1) \bmod 4}) \oplus c'_3 \oplus k_{2,12+n} \oplus S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{0,8+n} \oplus k_{0,4+n}
\end{aligned}$$

□

Équation (2.10)

$$k_{0,0+n} = k_{0,8+n} \oplus S(k_{0,12+(n+1) \bmod 4}) \oplus c'_1 \oplus k_{1,12+n} \oplus S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{2,12+n}$$

Démonstration.

$$\begin{aligned}
k_{2,12+n} &= k_{2,8+n} \oplus k_{1,12+n} \\
&= k_{2,4+n} \oplus k_{1,12+n} \oplus k_{1,8+n} \\
&= k_{2,0+n} \oplus k_{1,12+n} \oplus k_{1,8+n} \oplus k_{1,4+n} \\
&= S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{1,12+n} \oplus k_{1,8+n} \oplus k_{1,4+n} \oplus k_{1,0+n} \\
&= S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{1,12+n} \oplus k_{1,0+n} \oplus k_{0,8+n} \\
&= S(k_{1,12+(n+1) \bmod 4}) \oplus c'_2 \oplus k_{1,12+n} \oplus S(k_{0,12+(n+1) \bmod 4}) \oplus c'_1 \oplus k_{0,8+n} \oplus k_{0,0+n}
\end{aligned}$$

□

Chapitre 3

Participation au *DPA Contest V2*

Sommaire

Les attaques de ce chapitre sont une description de deux propositions que nous avons soumises lors du concours *DPA Contest V2*. Elles ont été publiées dans un article collaboratif [CDD+14] résumant les différentes méthodes utilisées lors du concours, détaillées par les concepteurs de ces dernières.

3.1 Introduction

Comme détaillé en Section 1.6 le DPA Contest est un concours organisé par Télécom ParisTech où un lot de courbes est fourni aux participants et où ils doivent concevoir une attaque qui devra retrouver la clef de chiffrement en utilisant le moins de courbes possible.

Lors de la deuxième édition, qui nous intéresse dans ce chapitre, les courbes constituaient des mesures de la consommation d'un composant exécutant une implémentation matérielle de l'AES sans contre-mesure. Cela signifie que, contrairement aux implémentations logicielles où nous avons un découpage du calcul à effectuer en instructions élémentaires et où l'on met simultanément à jour un nombre réduit de bits (8, 32, ...), nous sommes ici en présence d'un composant dédié au calcul de l'AES où un tour de chiffrement ne prend qu'un seul cycle d'horloge et où 128 bits d'état intermédiaire sont mis à jour simultanément à chaque tour.

Pour cette édition, les données sont réparties en deux bases publiques et une base privée. La base privée est conservée par les organisateurs afin d'évaluer les attaques proposées, tandis que les bases publiques sont données aux concurrents afin de leur permettre de développer et tester leurs attaques. La première base publique est similaire, dans sa forme et sa taille, à la base privée : elle comporte 32 lots de 20 000 courbes, chaque lot n'utilisant qu'une seule clef constante et des messages choisis aléatoirement pour chacune des 20 000 acquisitions. Le troisième lot,

nommé *template*, est une base de 1 000 000 courbes pour lesquelles le message et la clef ont été tirés aléatoirement. Cette dernière base a pour but de fournir une grande variété de saisies afin que les attaquants puissent étudier des comportements spécifiques. Nous ne l'avons pas utilisée pour concevoir nos attaques présentées ci après.

Ce chapitre présente les deux attaques que nous avons proposées, nommées – comme cela dans le concours – *Attaque A* (Section 3.2.2) et *Attaque B* (Section 3.2.3). La Section 3.3 présente les résultats de nos attaques dans le concours. Et nous concluons en Section 3.4.

3.2 Description des attaques

3.2.1 Partie commune

Comme expliqué en introduction nous faisons face à une implémentation matérielle de l'AES, ce qui implique que l'état courant après l'opération `AddRoundKey`(K_i) est écrasé en mémoire par l'état courant après `AddRoundKey`(K_{i+1}). Nous utilisons cela pour concevoir une attaque de type CPA où les courbes de consommation de courant sont corrélées à des distances de Hamming entre ces deux valeurs intermédiaires.

Comme il n'y a pas d'opération `MixColumns` lors du dernier tour, chaque octet de sortie $C[i] = \mathrm{AES}_{10}[i]$ $(i = 0, \ldots, 15)$ ne dépend que d'un octet de clef $K_{10}[i]$ et d'un octet $\mathrm{AES}_9[\sigma(i)]$ de la sortie du tour précédent, où $\sigma(i)$ est l'indice d'entrée d'un octet étant déplacé en indice i après l'étape `ShiftRows`.

Étant donné une supposition g sur la valeur de l'octet de clef $K_{10}[i]$ nous calculons la courbe de CPA définie par le coefficient de corrélation de Pearson entre la fuite $W[t]$ et la distance de Hamming prédite entre deux valeurs successives de registre, défini tel que :

$$\rho_{i,g}[t] = \frac{\mathrm{Cov}(W[t], D_{i,g})}{\sqrt{\mathrm{Var}(W[t])\,\mathrm{Var}(D_{i,g})}}$$

où la distance de Hamming $D_{i,g}$ s'exprime par :

$$D_{i,g} = \mathrm{HD}(\texttt{SubBytes}^{-1}(C[i] \oplus g), C[\sigma(i)])$$

3.2.2 Attaque A

Les 256 valeurs envisagées pour l'octet de clef sont alors triées par hauteur maximum – en valeur absolue – de pic de corrélation obtenu sur la courbe de corrélation. Dans l'*Attaque A* nous améliorons la CPA basique décrite ci-dessus avec trois astuces additionnelles :

Dans un premier temps, afin d'éviter des pics pouvant apparaître à des instants non reliés aux deux derniers tours, nous avons décidé de nous concentrer uniquement sur une partie réduite des courbes de courant en sélectionnant une fenêtre temporelle comprise entre $t_1 = 2\,300$ et $t_2 = 2\,900$ où nous savons que la corrélation pertinente est plus encline à apparaître.

Ensuite, tel un filtre passe-bas, nous appliquons une moyenne mobile afin d'augmenter le rapport signal sur bruit [8]. Nous avons choisi une fenêtre de 20 points.

Pour illustrer le gain apporté par ces deux astuces, il est possible de retrouver 13 octets de clef sur 16 en utilisant 5000 courbes avec la première astuce seule, 15 sur 16 avec la combinaison des deux astuces, contre seulement 11 sur 16 avec une CPA classique.

8. Augmenter la taille des pics de corrélation utiles vis-à-vis du bruit de mesure.

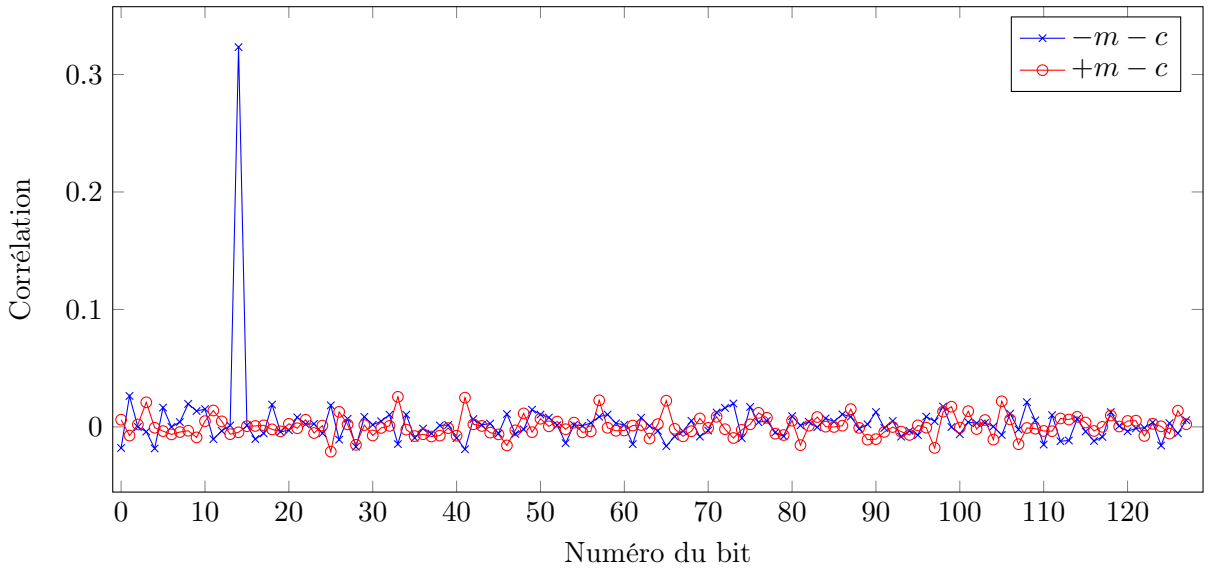

FIGURE 3.1 – Corrélation mono-bit à l'instant 327 pour la clef numéro 30 ($k_{14} = 0$)

La dernière astuce consiste en un traitement particulier réservé à l'octet numéro 13 de K_{10}. Dans une phase de caractérisation des fuites, préalable à la construction de nos attaques, nous avons remarqué qu'un bit en particulier avait un comportement singulier, différent de ceux des autres. En effet le bit numéro 14 [9] de la fin du neuvième tour s'avère corréler significativement vis-à-vis de la consommation de courant. Nous avons choisi de profiter de cette observation et ainsi retrouver la valeur de l'octet 13 de K_{10} comme celui montrant la meilleure corrélation mono-bit entre ce bit et les courbes de consommation.

3.2.3 Attaque B

Pour cette attaque nous exploitons un autre comportement particulier du bit numéro 14 qui se produit en début d'AES. En notant respectivement m_{14}, c_{14} et k_{14} les bits numéro 14 du clair, du chiffré et de la clef, on observe que les courbes de consommation de courant sont fortement corrélées avec $-(m_{14} + c_{14})$ lorsque $k_{14} = 0$ et avec $m_{14} - c_{14}$ lorsque $k_{14} = 1$. Les Figures 3.1 et 3.2 font respectivement référence aux clefs numéro 30 et 31 de la base de courbes publique, pour lesquelles k_{14} est respectivement égal à 0 et 1. Elles illustrent les niveaux de corrélation obtenus, pour 20 000 courbes, entre la consommation électrique à l'instant $t = 327$ et les valeurs de $(m - c)$ et $-(m + c)$. Un pic de corrélation à 0.3 pour le bit numéro 14 permet de clairement identifier k_{14}, alors qu'il est proche de zéro pour les autres bits. Le comportement du bit 14 – et seulement celui là – est étrange et doit être causé par une raison matérielle à bas niveau que nous n'avons pas identifiée.

Nous avons donc conçu l'*Attaque B* – qui est simplement censée être une amélioration mineure de l'attaque A – en utilisant l'avantage offert par ce comportement anormal afin d'identifier le bit 14 de K. En effet, cette corrélation est tellement forte que nous pouvons commencer n'importe quelle attaque par l'action de retrouver la valeur de ce bit avec un niveau de confiance élevé dès qu'on atteint quelques centaines de courbes.

9. Nous numérotons ici les bits d'un état d'AES de 0 à 127, en démarrant du bit de poids fort de l'octet 0 jusqu'au bit de poids faible de l'octet 15.

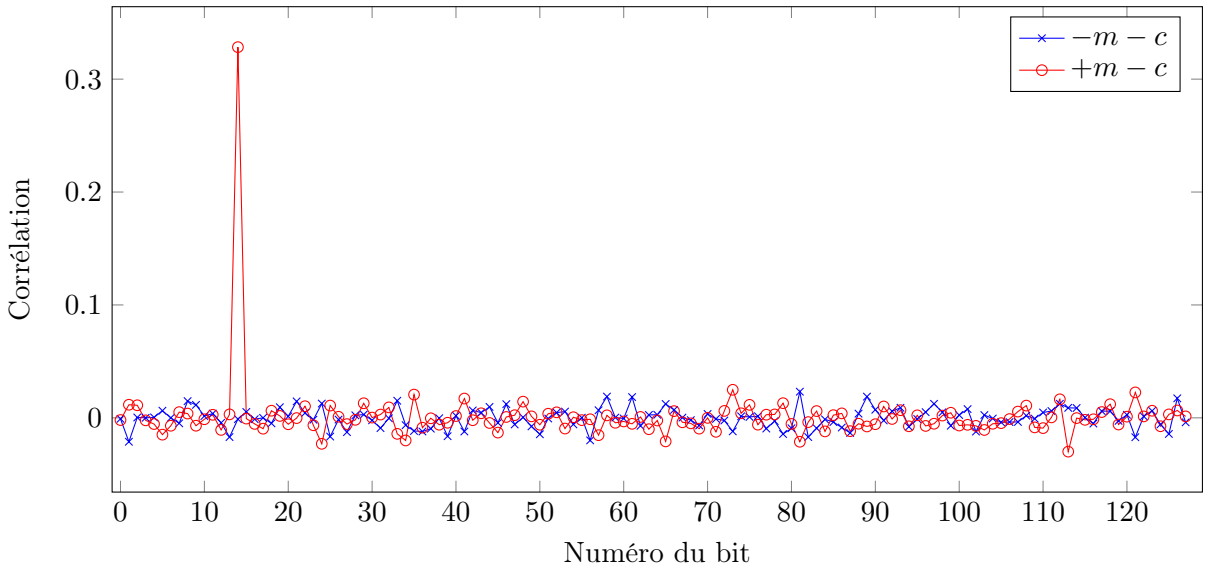

FIGURE 3.2 – Corrélation mono-bit à l'instant 327 pour la clef numéro 31 ($k_{14} = 1$)

Nous exécutons l'attaque A qui nous fournit un classement des 256×16 suppositions des octets de K_{10}. De la valeur de K_{10} donnée par les 16 octets en tête des listes nous pouvons dériver la clef K correspondante et vérifier si la valeur de son bit numéro 14 correspond à la valeur obtenue précédemment. Si oui l'attaque se termine et la clef K est alors retournée. Si non K_{10} est légèrement modifiée en remplaçant l'un de ses octets par le second dans le classement, et on vérifie le bit numéro 14 du nouveau K ainsi dérivé. Ce processus continue jusqu'à ce qu'un candidat de K_{10}, proche de celui de départ, soit établi comme entraînant la bonne valeur du bit numéro 14 de sa clef K correspondante.

Remarque 3. *Lors de la phase de caractérisation du composant, nous avons remarqué un fort niveau de corrélation entre la consommation du début de l'AES et la distance de Hamming des 128 bits du clair et du chiffré. De prime abord, il semble surprenant d'observer une corrélation impliquant le chiffré au début de l'algorithme mais on peut expliquer ce phénomène par le fait que les courbes du concours sont en fait la moyenne de* 128 *acquisitions d'exécutions ayant exactement les mêmes entrées et que le registre contenant le chiffré d'une exécution est donc probablement écrasé par le clair au début de celle qui suit.*

3.3 Résultats

En premier lieu il est à noter que les organisateurs ont permis des soumissions d'attaques sur trois périodes de temps, la *première période*, s'arrête à la date originelle du concours, la *deuxième période* a été rajoutée par la suite par les organisateurs pour récupérer de nouvelles contributions et enfin la *troisième période* est encore en cours à l'heure où cette thèse est rédigée, car les organisateurs acceptent maintenant sans délai les nouvelles propositions d'attaques. Les résultats sont ainsi donnés selon la période de temps dans laquelle l'attaque a été soumise.

Les attaques proposées sont évaluées selon trois critères principaux :

— **GSR stable $>$ 80%** : Nombre de courbes nécessaires pour que le taux de succès global (**G**lobal **S**uccess **R**ate) soit supérieur à 80% et le reste. Le taux de succès global étant le ratio de clefs complètement retrouvées sur les 32 clefs de la base.

TABLE 3.1 – Nom et nombre de courbes des trois meilleures propositions d'attaques soumises au *DPA Contest V2* triées par critère et par période

Critère	Rang	1ère période		1ère et 2ème périodes		Toutes périodes	
GSR	1	Walle	7061	Walle	7061	Gérard+1	439
stable	2	Berthier	15943	Lomné	10666	Li+2	2256
$> 80\%$	3	Bonnecaze	18458	Berthier+1	10,796	Heuser+3	3589
Min PSR	1	Walle	5890	Walle	5890	Gérard+1	439
stable	2	Bonnecaze	12318	Berthier+1	7510	Li+2	2155
$> 80\%$	3	**Attq. A**	**12621**	Meynard	8835	Heuser+3	2748
Max PGE	1	Walle	3388	Berthier+1	2767	Gérard+1	479
stable	2	**Attq. A&B**	**4192**	Walle	3388	Heuser+3	2356
< 10	3	Berthier	4706	**Attq. A&B**	**4192**	Berthier+1	2767

— **Min PSR stable $> 80\%$** : Nombre de courbes nécessaires pour que le minimum du taux de succès partiel (**P**artial **S**uccess **R**ate) soit supérieur à 80% et le reste. Le taux de succès partiel étant le ratio d'octets de clefs retrouvés pour chacune des 32 clefs de la base.

— **Max PGE stable < 10** : Nombre de courbes nécessaires pour que l'entropie maximum des suppositions partielles (**P**artial **G**uessing **E**ntropy) soit inférieure à 10 et le reste. L'entropie des suppositions partielles étant le nombre, pour chacune des 32 clefs, de bit nécessaires pour représenter l'exploration permettant de trouver la bonne clef dans les listes triées d'octets.

La Table 3.1 présente donc les trois meilleures propositions d'attaques selon les trois critères et pour chacune des trois périodes de soumissions. Nos trois apparitions dans ce tableau sont indiquées en gras.

3.4 Conclusion

Dans ce chapitre nous avons présenté les deux propositions d'attaques que nous avons soumises au concours *DPA Contest V2*. Afin d'améliorer la CPA classique nous avons réduit la fenêtre temporelle autour de la zone d'intérêt pour limiter les pics non pertinents, puis nous avons appliqué en pré-traitement une moyenne mobile pour augmenter le rapport signal sur bruit. Enfin, nous avons utilisé une propriété particulière du bit numéro 14, seul bit à corréler très fortement pour une raison inexpliquée mais possiblement due à une particularité au niveau matériel, pour accélérer la récupération d'un des octets de clef.

Dans la seconde version de l'attaque nous avons ajouté une autre utilisation de la propriété du bit 14 consistant à obtenir, avec confiance élevée, le bit numéro 14 de la clef K recherché et ainsi pouvoir invalider une clef K_{10}, proposée par la CPA, si elle n'induit pas la bonne valeur de ce bit.

Ces attaques nous ont permis de nous placer au rang de deuxième et troisième dans deux catégories respectives lors de la première période de soumission.

Nos attaques mettent en avant l'intérêt d'analyser le comportement du composant attaqué à différentes tailles de mots et en variant les modèles de consommation. Par exemple l'analyse

classique au niveau octet ne permet pas de détecter le phénomène particulier attaché au bit numéro 14 de ce lot de courbes.

Troisième partie

Attaques physiques appliquées à la rétro-conception

Chapitre 4

Méthodes FIRE et SCARE permettant la retro-conception complète d'un algorithme de type AES modifié

Sommaire

Les travaux présentés dans cette section réunissent trois attaques sur le même thème. La première, une attaque de type FIRE décrite en Section 4.4, a été publiée en 2013 [CW13]. La deuxième attaque de type SCARE par collisions de valeurs décrite en Section 4.5 a été publiée en 2013 [CIW13]. Ces deux publications on été étendues et associées à une troisième attaque pour être publiées dans un journal en 2015 [CIMW15], cette dernière attaque décrite en Section 4.6 est aussi de type SCARE et étend le modèle aux collisions de poids de Hamming.

4.1 Introduction

En dépit du principe de Kerckhoffs certains acteurs du domaine de la sécurité utilisent encore la sécurité par l'obscurité, par le biais d'algorithmes, propriétaires ou dont les spécifications sont tenues secrètes, utilisés dans des applications militaires ou bien civiles telles que dans la téléphonie mobile ou pour les chaînes de télévision payantes. Dans le but de simplifier la création d'un algorithme de cryptographie les concepteurs peuvent opter pour une stratégie consistant à modifier des paramètres d'un algorithme public déjà existant et largement étudié. Ce choix réduit les coûts de conception et de développement mais permet aussi d'hériter en partie de la robustesse d'un algorithme largement étudié tel que l'AES. Dans ce chapitre nous nous concentrons sur la difficulté de retrouver – au moyen d'attaques physiques – la spécification complète d'un algorithme de type AES modifié dont tout ou partie des paramètres ont été remplacés par des valeurs non-standard et tenues secrètes. Afin de montrer qu'un large panel d'attaques peut mettre en péril ce type de sécurité par l'obscurité nous proposons trois méthodes, basées sur trois modèles différents, permettant de retrouver ces paramètres secrets. Nous montrerons pour les trois modèles qu'un attaquant ignorant les paramètres d'un AES modifié, ainsi que la clef utilisée, peut retrouver toutes ces informations, et que pour certains modèles il en est capable même en présence de certaines contre-mesures de type masquage booléen ou de type mélange de l'ordre d'opérations indépendantes.

Nous supposons ici que l'attaquant possède la connaissance *a priori* du fait que l'implémentation concerne un AES modifié. Cette connaissance peut souvent être déduite d'une analyse de type SPA où l'attaquant repère visuellement une forme dans la consommation se répétant, lui révélant le nombre de tours effectués. Puis de plus petites formes se répétant dans un tour lui révèlent le nombre de sous-étapes différentes et leurs occurrences. A noter que les injections de fautes ou les collisions de consommations de courant utilisées dans les attaques de ce chapitre nécessitent un repérage de ces étapes pour pouvoir injecter à l'endroit désiré ou comparer les portions de consommation.

Ce chapitre est organisé comme suit : La Section 4.2 décrit ce que nous appelons un AES modifié qui est la cible de nos attaques. Nous détaillons en Section 4.3 les trois modèles d'attaquants considérés. La Section 4.4 décrit l'attaque retrouvant tous les paramètres d'un AES modifié par la méthode FIRE. La Section 4.5 décrit l'attaque réalisant le même objectif mais avec le modèle SCARE par collisions de valeurs, ainsi que l'extension de cette attaque à des implémentations protégées. La Section 4.6 présente l'attaque dans le cadre du troisième modèle : SCARE par collisions de poids de Hamming, qui est un modèle plus souple que le précédent car les collisions sont plus faciles à détecter. Des contre-mesures à nos attaques sont proposées en Section 4.7 et nous concluons en Section 4.8.

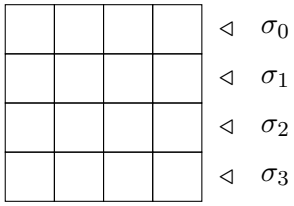

FIGURE 4.1 – Paramètres de `ShiftRows`

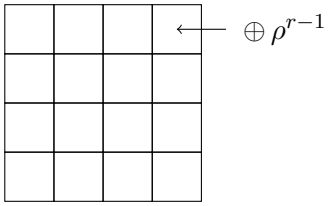

FIGURE 4.2 – Paramètre de `Rcon`$[r]$

4.2 Définition d'un AES modifié

Dans cette section nous allons détailler les modifications qui peuvent être apportées à l'AES sans modifier sa structure. Pour de plus précises informations sur l'AES standard voir la section d'introduction sur l'AES (Section 1.2.2) et le document officiel du NIST [Nat01]. De cette spécification stricte décrivant l'AES il est possible de dériver une collection fournie de fonctions différentes mais possédant néanmoins la même structure en réseau de substitution-permutation et le même nombre et tailles de données internes.

Nous définissons un AES modifié comme étant une fonction dérivée de l'AES par modification des paramètres suivants :

1. la table S-Box notée S, peut être remplacée par n'importe quelle autre table préservant la propriété que l'opération `SubBytes` est une permutation sur $GF(2^8)$,
2. la transformation opérée par `ShiftRows` est une rotation de la ligne i de σ_i octets vers la gauche, où σ_i peut être n'importe quelle valeur entre 0 et 3 (i.e. $\sigma_i = i$ pour l'AES standard),
3. la matrice constante qui définit l'opération `MixColumns` peut être remplacée par différentes configurations, selon si on considère le modèle FIRE ou SCARE :
   3.1. attaque FIRE : n'importe quelle matrice circulante basé sur un quadruplet d'octets $(\gamma_0, \ldots, \gamma_3)$ dans $GF(2^8) \setminus \{0\}$,
   3.2. attaque SCARE : n'importe quel 16-uplet d'octets $(\alpha_0, \ldots, \alpha_{15})$ dans $GF(2^8)$,
4. l'opération `RotWord` du *key schedule* applique une rotation de η octets vers le haut à la dernière colonne de chaque clef de tour, où η peut être n'importe quelle valeur entre 0 et 3 (i.e. $\eta = 1$ pour l'AES standard),
5. la constante dépendante du tour `Rcon`$[r]$ impliquée dans le *key schedule* pour le calcul de la clef de tour K_r est définie par $(\rho^{r-1}, 0, 0, 0)$ où ρ peut prendre n'importe quelle valeur non nulle (i.e. $\rho = 2$ pour l'AES standard).

Les Figures 4.1 à 4.4 montrent respectivement les degrés de libertés des différents paramètres : `ShiftRows`, `RotWord`, `Rcon` et `MixColumns`.

Pour alléger le propos dans la suite de ce chapitre nous nommerons *AES* une modification secrète de l'AES dont l'attaquant cherche à retrouver les paramètres. Nous utiliserons la notation *AES standard* lorsqu'il sera nécessaire de faire référence à l'AES standard tel que défini par la norme du NIST.

4.2.1 Notations

Nous introduisons ici les notations spécifiques à ce chapitre :

1. notations utilisées en Section 4.4 (attaque FIRE) :

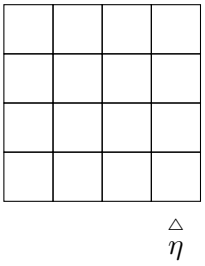

FIGURE 4.3 – Paramètre de `RotWord`

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\ \gamma_3 & \gamma_0 & \gamma_1 & \gamma_2 \\ \gamma_2 & \gamma_3 & \gamma_0 & \gamma_1 \\ \gamma_1 & \gamma_2 & \gamma_3 & \gamma_0 \end{bmatrix} \qquad \begin{bmatrix} \alpha_0 & \alpha_4 & \alpha_8 & \alpha_{12} \\ \alpha_1 & \alpha_5 & \alpha_9 & \alpha_{13} \\ \alpha_2 & \alpha_6 & \alpha_{10} & \alpha_{14} \\ \alpha_3 & \alpha_7 & \alpha_{11} & \alpha_{15} \end{bmatrix}$$

FIGURE 4.4 – Matrice du `MixColumns` (Gauche : FIRE | Droite : SCARE)

- $\lambda_i = k_{0,i} \oplus S^{-1}(0)$ (pour $i = 0, \ldots, 15$)
- $mk_{i,j}$ la valeur d'octet vérifiant l'équation : $\gamma_j * S(mk_{i,j} \oplus k_{0,0}) = k_{1,i} \oplus S^{-1}(0)$ (pour $i = 0, \ldots, 15$ et $j = 0, \ldots, 3$)
- $\beta_{i,j} = \gamma_i / \gamma_j$ (pour $i, j = 0, \ldots, 3$)

2. notations utilisées en Section 4.5 and 4.6 (attaques SCARE) :
   - $\mu_{r,i,j} = k_{r,i} \oplus k_{r,j}$ (pour $r = 0, \ldots, 10$ et $i, j = 0, \ldots, 15$)

4.3 Les modèles d'attaquant

Nous nous plaçons dans le scénario à clair choisi où l'attaquant possède un dispositif (e.g. une carte à puce) embarquant une implémentation logicielle d'un AES tenu secret. Il a la possibilité de soumettre des clairs choisis par lui et reçoit en retour le chiffré correspondant. Il ignore la valeur de la clef K utilisée par le dispositif et son but est de procéder à une rétro-conception de tous les paramètres secrets de la fonction de chiffrement, grâce à l'analyse d'effets de fautes, pour l'attaque de type FIRE, ou grâce à l'analyse de courbes de canaux auxiliaires acquises à chaque chiffrement, pour les attaques de type SCARE.

Il paraît clair que la robustesse cryptographique d'un AES tel que défini en Section 4.2.1 peut aller de très faible à raisonnablement forte. Donc uniquement une petite partie de ce type de fonctions pourrait être éligible pour une utilisation cryptographique. Malgré cette possible simplification, nous avons choisis un modèle d'attaquant ne disqualifiant pas une fonction candidate par rapport à la pertinence de ses paramètres et qui considère donc tous les ensembles de paramètres – $(S, \{\sigma_i\}_i, \{\gamma_i\}_i, \eta, \rho)$ pour le modèle FIRE et $(S, \{\sigma_i\}_i, \{\alpha_i\}_i, \eta, \rho)$ pour les modèles SCARE – modifiables selon notre définition.

4.3.1 Modèle d'attaque FIRE : octet fixé à 0

Pour l'attaque de type FIRE décrite en Section 4.4 nous utilisons un modèle où un octet est fixé à 0 durant la lecture en mémoire d'un octet correspondant à une sortie de S-Box. Cela

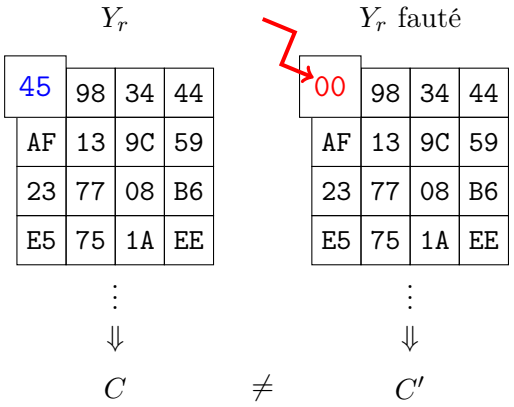

FIGURE 4.5 – Exemple de faute avec effet

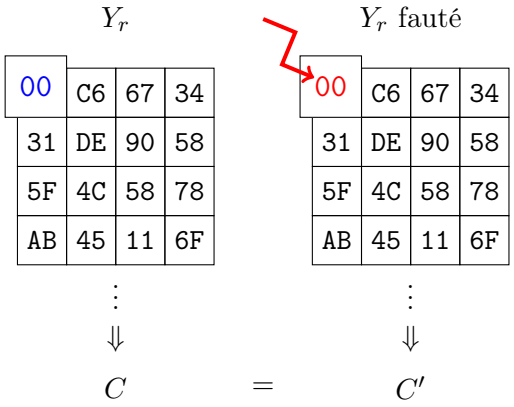

FIGURE 4.6 – Exemple de faute sans effet

signifie que lorsqu'une faute est injectée pendant la lecture d'une S-Box ciblée la valeur de l'octet en sortie de celle-ci est alors forcée à zero quelle que soit sa valeur originelle.

Dans ce modèle l'attaquant peut obtenir l'information booléenne suivante : la valeur originelle (non fautée) en sortie de la S-Box ciblée est égale à zero ou non. Il lui suffit de chiffrer deux fois le même clair : une fois sans faute et une fois avec une faute et comparer les chiffrés correspondants. Comme décrit dans les Figures 4.5 et 4.6, la tentative de faute n'a aucun effet si la valeur ciblée est déjà égale à zéro (on parle de faute sans effet) alors qu'elle aura un effet si la valeur originelle n'est pas égale à zero. Un attaquant peut donc identifier une faute sans effet lorsque les deux chiffrés sont identiques et une faute avec effet lorsque les deux chiffrés sont différents.

Il est à noter que lorsque une contre-mesure de type double exécution est implantée sur le dispositif les fautes sans effet peuvent être détectée avec une seule exécution au lieu de deux. En effet cette contre-mesure procède deux fois au chiffrement et ne retourne le chiffré que si les deux exécutions ont produit exactement le même résultat. Le dispositif refusera de donner le chiffré dans le cas d'une faute ayant un effet alors qu'il acceptera de donner un chiffré si la faute est sans effet. L'information booléenne est donc obtenue plus facilement si cette contre-mesure est implantée.

4.3.2 Modèle d'attaque SCARE : collision entre appels de S-Box

Pour la première attaque de type SCARE décrite en Section 4.5 nous faisons la supposition d'un attaquant capable d'identifier, grâce à une observation de canaux auxiliaires, une collision de valeurs entre deux appels à la table S-Box. Plus précisément, utilisant deux segments de courbe de consommation électrique T et T' correspondant respectivement à deux appels à la table S-Box $y = S(x)$ et $y' = S(x')$ durant un calcul d'AES [10], l'attaquant peut décider si $(x, y) = (x', y')$ ou pas en se basant sur un distingueur basé sur les canaux auxiliaires. Un tel modèle a déjà été utilisé dans un grand nombre d'attaques visant à retrouver des clefs [SWP03, SLFP04, BBKK07, Bog07, Bog08, CFG+11] et pour certaines ayant pour but la retro-conception de paramètres secrets [Nov03a, Cla07a]. À noter que Bogdanov [Bog08] a utilisé exactement le même modèle que nous appliqué sur les S-Boxs de l'AES.

Une objection qui pourrait être soulevée est que la détection de collision entre S-Boxs provenant de deux exécutions différentes est plus difficile que lorsqu'elles viennent de la même

10. Les deux appels à la table S-Box peuvent provenir de deux tours différents voire même de deux courbes distinctes avec des clairs différents.

exécution et peut mener à des décisions moins efficaces à cause de différences dans les conditions expérimentales (température, ...). Comme nous considérons que cette question est sujette à débat, nous proposons cette attaque dans deux modes : le scenario *inter-courbes* où l'attaquant peut détecter des collisions entre des courbes différentes, et le scénario *intra-courbe* où il ne le peut pas.

Pour la seconde attaque SCARE présentée en Section 4.6 nous avons adopté un modèle de collisions plus réaliste, moins difficiles à obtenir pour l'attaquant mais lui donnant moins d'information : les collisions en poids de Hamming sur les S-Boxs. Au lieu de devoir détecter la condition $(x, y) = (x', y')$ l'attaquant n'a plus que la nécessité de détecter la situation $(\mathrm{HW}(x), \mathrm{HW}(y)) = (\mathrm{HW}(x'), \mathrm{HW}(y'))$. Le choix de ce modèle moins contraint est basé sur le comportement physique à bas niveau d'un microprocesseur 8-bit où les huit lignes du bus de données contribuent supposément de manière additive à la consommation électrique. Cela suggère directement le modèle en poids de Hamming qui s'est montré pertinent et efficace, en particulier pour de nombreuses SPA [MS00, Mes00, MDS99] et CPA [BCO04].

4.4 Attaque FIRE utilisant l'IFA

Dans cette section nous décrivons étape par étape comment retrouver les paramètres secrets d'une implémentation d'AES qui ne possède pas de contre-mesures contre les attaques par canaux auxiliaires. Le modèle de faute introduit en Section 4.3.1 permet à l'attaquant de détecter par IFA si $y_{r,i} = S(x_{r,i}) = 0$ ou non lorsqu'il introduit une faute durant le calcul ou la lecture de la S-Box i du tour r.

La recouvrement des coefficients de la matrice de `MixColumns` fait intervenir l'ordre multiplicatif des éléments de type $\beta_{i,j} = \gamma_i/\gamma_j$. Nous faisons la supposition que au moins l'un d'entre eux a pour ordre 255 (ce qui signifie qu'il est générateur de $GF(2^8)\backslash\{0\}$). Nous avons compté que cette condition était satisfaite dans 95.28% des 255^4 quadruplets possibles.

Nous introduisons ci-après en Section 4.4.1 des résultats préliminaires qui seront utilisés dans la description de l'attaque.

4.4.1 Resultats préliminaires

Lemme 1. *La connaissance de $(\lambda_0, \ldots, \lambda_{15})$ permet de choisir des octets de clair m_i et m_j tels que $x_{1,i} = x_{1,j}$ (pour tout $i, j = 0, \ldots, 15$).*

Démonstration. Quel que soit m_i nous avons $x_{1,i} = m_i \oplus k_{0,i}$. Si on choisit $m_j = m_i \oplus \lambda_i \oplus \lambda_j$ cela implique :

$$\begin{aligned} x_{1,j} &= m_i \oplus \lambda_i \oplus \lambda_j \oplus k_{0,j} \\ &= m_i \oplus k_{0,i} \oplus S^{-1}(0) \ \oplus k_{0,j} \oplus S^{-1}(0) \ \oplus k_{0,j} \\ &= m_i \oplus k_{0,i} \\ &= x_{1,i} \end{aligned}$$

□

Corollaire 1. *La connaissance de $(\lambda_0, \ldots, \lambda_{15})$ et des paramètres de l'étape `ShiftRows` permet à l'attaquant de retrouver n'importe quelle valeur $mk_{i,j}$.*

Démonstration. En choisissant $m_k = \lambda_k$ on peut forcer à zéro la sortie de la S-Box correspondante. Connaissant les paramètres de l'étape `ShiftRows` un attaquant est donc capable de construire un clair qui induit trois valeurs à zéro à des emplacements choisis dans une même colonne d'entrée de `MixColumns`. Il peut alors tester de façon exhaustive les valeurs du dernier octet de la colonne par modification de la valeur de l'octet de clair correspondant, et attendre qu'une IFA survienne sur une S-Box du deuxième tour choisie parmi celles utilisant un octet provenant de la sortie de cette opération `MixColumns`. Pour un i et un j choisis, où i est l'indice de l'octet impliqué dans l'IFA sur la ligne $\ell = i \bmod 4$ et la colonne $c = \lfloor i/4 \rfloor$, et où γ_j est le coefficient de `MixColumns` impliqué, la position de l'octet de clair impliqué peut s'exprimer comme $i' = \ell' + 4c'$ où $\ell' = (\ell + j) \bmod 4$ et $c' = (c + \sigma_{\ell'}) \bmod 4$, et nous obtenons les équations équivalentes successives suivantes :

$$\begin{aligned} y_{2,i} &= 0 \\ x_{2,i} &= S^{-1}(0) \\ \gamma_j * S(m_{i'} \oplus k_{0,i'}) \oplus k_{1,i} &= S^{-1}(0) \\ \gamma_j * S(m_{i'} \oplus \lambda_0 \oplus \lambda_0 \oplus k_{0,i'}) &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_j * S\left((m_{i'} \oplus \lambda_0 \oplus \lambda_{i'}) \oplus k_{0,0}\right) &= k_{1,i} \oplus S^{-1}(0) \end{aligned}$$

et par définition de $mk_{i,j}$ cela induit que $mk_{i,j} = m_{i'} \oplus \lambda_0 \oplus \lambda_{i'}$.

□

Lemme 2. *Pour $i \in \{0, 4, 1, 5, 2, 6, 3, 7\}$, on vérifie que $k_{1,i} \oplus k_{1,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}$.*

Démonstration.

$$\left\{ \begin{array}{l} k_{1,i+4} = k_{1,i} \oplus k_{0,i+4} \\ k_{1,i+8} = k_{1,i+4} \oplus k_{0,i+8} \end{array} \right\}$$

$$\Downarrow$$

$$k_{1,i} \oplus k_{1,i+8} = k_{0,i+4} \oplus k_{0,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}$$

□

4.4.2 Description de l'attaque

4.4.2.1 Retrouver K_0 à une constante près

Dans le but de retrouver $k_{0,i}$ à une valeur d'octet constante près, qui se trouve être égale à $S^{-1}(0)$, nous fautons la sortie de la S-Box numéro i du premier tour tout en testant de façon exhaustive la valeur de l'octet m_i. Lorsqu'une IFA survient nous savons que $y_{1,i} = 0$, et nous pouvons alors déduire que $m_i \oplus k_{0,i} = S^{-1}(0)$. Cette valeur particulière de m_i est donc égale à $\lambda_i = k_{0,i} \oplus S^{-1}(0)$. En appliquant le même procédé sur chaque position d'octet de clair nous sommes capables de retrouver $(\lambda_0, \ldots, \lambda_{15})$ qui est K_0 à un XOR avec $S^{-1}(0)$ près.

4.4.2.2 Retrouver les paramètres de `ShiftRows`

Pour retrouver les paramètres de `ShiftRows` il nous faut en premier lieu obtenir un clair de référence produisant une IFA sur la première S-Box du deuxième tour. Dans cette optique nous chiffrons successivement des clairs où les octets m_0, m_4, m_8 and m_{12} sont simultanément

testés de manière exhaustive. Lorsqu'une IFA survient nous savons que $y_{2,0} = 0$. La valeur de cet octet dépend uniquement de quatre octets de clair, un sur chaque ligne de la matrice d'état, et les positions de ces quatre octets sont directement reliées aux paramètres de `ShiftRows`. Nous pouvons identifier les positions de ces quatre octets par le fait que changer la valeur de n'importe lequel d'entre eux modifie la valeur $y_{2,0}$ et fait alors disparaître l'IFA, alors que modifier n'importe lequel des 12 autres octets laisse $y_{2,0}$ inchangé. En chiffrant le clair de référence avec un seul octet modifié à chaque fois nous obtenons les quatre positions qui font disparaître l'IFA. Chacune de ces positions révèle un des paramètres de `ShiftRows`. En effet, si modifier m_{r+4c} fait disparaître l'IFA sur $y_{2,0}$ cela implique que cet élément, qui appartient à la ligne r a été décalé de la colonne c à la colonne 0 par l'opération `ShiftRows`. On peut en déduire que $\sigma_r = c$.

4.4.2.3 Réduire l'entropie de `MixColumns` en retrouvant les ordres des $\beta_{i,j}$

Pour obtenir les ordres multiplicatifs des valeurs $\beta_{i,j}$ nous essayons d'obtenir des IFAs sur la première S-Box du deuxième tour. Pour une question de clarté nous allons décrire ici la méthode pour retrouver l'ordre de $\beta_{1,2}$ mais cette méthode reste valide pour tous les $\beta_{i,j}$. En premier lieu il faut remarquer qu'une IFA sur $y_{2,0}$ implique que l'équation suivante est vérifiée :

$$\gamma_0 * z_{1,0} \oplus \gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} \oplus k_{1,0} = S^{-1}(0)$$

Cette équation peut être simplifiée si on assigne à $m_{4\sigma_0}$ la valeur $mk_{0,0} \oplus (\lambda_0 \oplus \lambda_{4\sigma_0})$ où $mk_{0,0}$ est déterminé par la méthode du Corollaire 1. Ce choix implique que $\gamma_0 * z_{1,0} = k_{1,0} \oplus S^{-1}(0)$ et que l'équation de l'IFA ci-dessus se réduit alors à :

$$\gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} = 0$$

Une deuxième astuce consiste à forcer $z_{1,3} = 0$ en assignant à $m_{3+4\sigma_3}$ la valeur $\lambda_{3+4\sigma_3}$. L'équation de l'IFA se réduit à :

$$\gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} = 0$$

Maintenant nous entrons dans la phase itérative. À chaque étape $k \geqslant 0$ la valeur inconnue $z_{1,1}$ est gardée fixe, notée $\tau^{(k)}$, pendant que la valeur inconnue $z_{1,2}$ est testée de manière exhaustive (concrètement en testant $m_{2+4\sigma_2}$ de manière exhaustive) jusqu'à ce qu'une IFA survienne. Cette valeur particulière de $z_{1,2}$ va être utilisée à l'étape $(k+1)$ pour remplacer $z_{1,1}$ elle sera donc notée $\tau^{(k+1)}$. L'équation de l'IFA implique qu'à l'étape k nous avons :

$$\begin{aligned} & \gamma_2 * \tau^{(k+1)} = \gamma_1 * \tau^{(k)} \\ \Rightarrow \quad & \tau^{(k+1)} = \beta_{1,2} * \tau^{(k)} \end{aligned}$$

donc la combinaison de toutes les équations des étapes 0 à k nous donne :

$$\tau^{(k+1)} = \beta_{1,2}^{k+1} * \tau^{(0)}$$

La génération d'une telle séquence de valeurs inconnues τ en cours d'algorithme requiert l'utilisation d'une sequence de valeurs connues notées θ en entrée de l'algorithme, la succession des valeurs est décrite sur la Figure 4.7. À chaque étape k, la valeur $\theta^{(k)}$ est définie telle que $z_{1,1} = \tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$ – et donc telle que $z_{1,2} = \tau^{(k+1)} = S(\theta^{(k+1)} \oplus k_{0,0})$ – par la méthode suivante :

— à l'étape $k = 0$, l'octet de clair $m_{1+4\sigma_1}$ prend une valeur arbitraire et nous avons $\theta^{(0)} = m_{1+4\sigma_1} \oplus (\lambda_0 \oplus \lambda_{1+4\sigma_1})$,

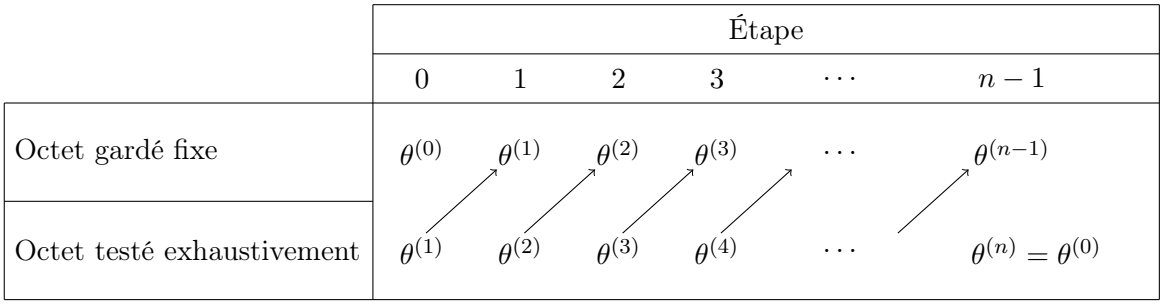

FIGURE 4.7 – Construction de la séquence θ basée sur un $\beta_{i,j}$ d'ordre n

— aux étapes $k > 0$ la valeur de $\theta^{(k)}$ est définie comme $\theta^{(k)} = m_{2+4\sigma_2} \oplus (\lambda_0 \oplus \lambda_{2+4\sigma_2})$ où $m_{2+4\sigma_2}$ est la valeur ayant produit l'IFA à l'étape $(k-1)$. La valeur de l'octet de clair assurant que $z_{1,1} = S(\theta^{(k)} \oplus k_{0,0})$ à l'étape k est donnée par $m_{1+4\sigma_1} = \theta^{(k)} \oplus (\lambda_0 \oplus \lambda_{1+4\sigma_1})$.

Au cours du processus nous allons observer à une certaine itération $k = n-1$ que $\theta^{(n)} = \theta^{(0)}$ impliquant que :

$$\tau^{(n)} = \beta_{1,2}^{n} * \tau^{(0)} = \tau^{(0)}$$

d'où nous déduisons que l'ordre multiplicatif de $\beta_{1,2}$ est égal à n.

Nous pouvons à présent changer les rôles des indices afin de générer les séquences $\theta_{i,j}$ et $\tau_{i,j}$ reliées à l'ordre de $\beta_{i,j}$ (pour $0 \leqslant i < j \leqslant 3$). À remarquer qu'il est toujours possible – et nous supposons que l'attaquant procède ainsi – de choisir les valeurs d'octet des premières étapes de sorte que les six séquences $\theta_{i,j}$ partagent la même valeur initiale $\theta_{i,j}^{(0)}$.

Ayant retrouvé les ordres des six $\beta_{i,j}$, nous pouvons les utiliser comme contraintes pour réduire le nombre de candidats au quadruplet $(\gamma_0, \gamma_1, \gamma_2, \gamma_3)$.

Une fois que nous avons observé que l'ordre d'au moins un $\beta_{i^\star,j^\star}$ est égal à 255, nous sommes capables d'utiliser les deux séquences $\theta_{i^\star,j^\star}$ et $\tau_{i^\star,j^\star}$ comme séquences de références contenant toutes les valeurs non nulles. Par la suite nous allons noter θ et τ ces séquences de référence de longueur 255, et par $\beta = \beta_{i^\star,j^\star}$ le quotient les ayant générées. Pour rappel ces séquences vérifient les deux propriétés suivantes :

$$\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$$
$$\tau^{(k)} = \beta^{k} * \tau^{(0)}$$

Remarque 4. *Retrouver les ordres des $\beta_{i,j}$ est assez coûteux en terme de nombre de fautes requises. Pour obtenir une IFA à l'étape k nous devons tester de manière exhaustive un ensemble de (256-k) valeurs, cela requiert en moyenne (256-k)/2 fautes. En notant $n_{i,j}$ l'ordre de $\beta_{i,j}$, une moyenne de $\Sigma_{k=0}^{n_{i,j}-1}(256-k)/2 \approx n_{i,j}(256-n_{i,j}/2))/2$ fautes sont nécessaires pour déterminer la longueur de la séquence. Par exemple, environ 2^{14} fautes sont requises si $\beta_{i,j}$ a un ordre de 255.*

Pour optimiser nous avons remarqué que lorsque les séquences de référence θ and τ ont été obtenues, il n'est alors plus nécessaire de générer les séquences complètes pour les autres $\beta_{i,j}$. En effet il suffit alors d'obtenir la première IFA à l'étape $k = 0$, donnant la valeur $\theta_{i,j}^{(1)}$. Il est alors possible d'identifier $\theta_{i,j}^{(1)}$ comme $\theta^{(x)}$ dans la séquence de référence. et d'en déduire que $n_{i,j} = 255/\gcd(255, x)$. À titre d'exemple si plus d'un des six ordres est égal à 255, l'obtention du premier coûtera 2^{14} fautes mais les suivants ne coûteront que 128 fautes.

4.4.2.4 Réduire l'entropie de `MixColumns` avec les relations d'ordres croisés

Les ordres $n_{i,j}$ des $\beta_{i,j}$ appartiennent tous à l'ensemble $\{1, 3, 5, 15, 17, 51, 85, 255\}$ des diviseurs de 255. Si il existe des paires d'indices (i_1, j_1) et (i_2, j_2) pour lesquels on a $n_{i_1,j_1} \mid n_{i_2,j_2}$ et $n_{i_1,j_1} > 1$ [11] alors, vu qu'elles partagent le même premier élément, la séquence θ_{i_1,j_1} est entièrement inclue dans la séquence θ_{i_2,j_2}. On peut ainsi identifier l'index t pour lequel $\theta^{(1)}_{i_1,j_1} = \theta^{(t)}_{i_2,j_2}$ impliquant que $\tau^{(1)}_{i_1,j_1} = \tau^{(t)}_{i_2,j_2}$ et nous donnant la relation $\beta_{i_1,j_1} = \beta^t_{i_2,j_2}$.

Toutes ces relations d'ordres croisés sont inférées sans besoin de réaliser des fautes supplémentaires, et elles peuvent être utilisées comme nouvelles contraintes pour réduire encore l'ensemble des candidats au quadruplet $\{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}$.

4.4.2.5 Réduire l'entropie de `MixColumns` avec les relations de K_1

Nous continuons à cibler la première S-Box du deuxième tour et utilisons les relations obtenues dans le Lemme 2 ($k_{1,i} \oplus k_{1,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}$) pour déterminer de nouvelles contraintes sur les paramètres de `MixColumns`. Premièrement on détermine les valeurs de $mk_{i,1}$ et $mk_{i+8,2}$ par la méthode décrite dans le Corollaire 1 afin d'être capable de choisir des octets de clair induisant $\gamma_1 * z_{1,1} = k_{1,i} \oplus S^{-1}(0)$ et $\gamma_2 * z_{1,2} = k_{1,i+8} \oplus S^{-1}(0)$. Nous utilisons aussi la connaissance de $mk_{0,0}$, déjà déterminé plus tôt, pour faire disparaître $k_{1,0}$ de l'équation d'IFA. Dans un second temps on teste de manière exhaustive l'octet de clair $m_{3+4\sigma_3}$ relié à $z_{1,3}$ jusqu'à l'apparition d'une IFA. On est capable de reconnaître $m_{3+4\sigma_3} \oplus (\lambda_0 \oplus \lambda_{3+4\sigma_3})$ comme étant un $\theta^{(p)}$, donc nous apprenons l'indice p tel que $z_{1,3} = \tau^{(p)}$, de cela nous dérivons :

$$
\begin{aligned}
y_{2,i} &= 0 \\
\gamma_0 * z_{1,0} \oplus \gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} \oplus k_{1,0} &= S^{-1}(0) \\
k_{1,0} \oplus S^{-1}(0) \ \oplus k_{1,i} \oplus S^{-1}(0) \ \oplus k_{1,i+8} \oplus S^{-1}(0) \ \oplus \gamma_3 * \tau^{(p)} \oplus k_{1,0} &= S^{-1}(0) \\
k_{1,i} \oplus k_{1,i+8} \oplus \gamma_3 * \beta^p * \tau^{(0)} &= 0 \\
\frac{\lambda_{i+4} \oplus \lambda_{1,i+8}}{\gamma_3 * \beta^p} &= \tau^{(0)}
\end{aligned}
$$

Les éléments inconnus dans la dernière équation sont $\tau^{(0)}$ et des paramètres de `MixColumns` : γ_3 et β. Deux équations de ce type peuvent être obtenues et combinées pour faire disparaître $\tau^{(0)}$ et créer une équation n'impliquant que des paramètres de `MixColumns`, ainsi cette dernière peut être utilisée comme contrainte réduisant encore l'ensemble des quadruplets.

Lemme 3. *À cette étape de l'attaque les informations acquises sont suffisantes pour réduire l'ensemble des candidats pour $\{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}$ à 255 éléments.*

Démonstration. En exploitant deux relations de K_1 du Lemme 2 on obtient deux équations qui sont :

$$
\left.
\begin{aligned}
\tau^{(0)} &= \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\gamma_3 * \beta^{p_1}} \\
\tau^{(0)} &= \frac{\lambda_{i+8} \oplus \lambda_{i+12}}{\gamma_3 * \beta^{p_2}}
\end{aligned}
\right\} \Rightarrow \ \beta^{p_1 - p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}}
$$

11. Dans le cas où $n_{i_1,j_1} = 1$, nous savons déjà que $\gamma_{i_1} = \gamma_{j_1}$.

$$\Rightarrow \quad (\frac{\gamma_{i^\star}}{\gamma_{j^\star}})^{p_1-p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}}$$

$$\Rightarrow \quad \gamma_{i^\star}^{p_1-p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}} * \gamma_{j^\star}^{p_1-p_2}$$

Comme $\gamma_{i^\star}$ et $\gamma_{j^\star}$ sont les seules valeurs inconnues, un choix pour l'une d'entre elles va déterminer la valeur de la seconde. Donc il reste 255 paires valides pour $(\gamma_{i^\star}, \gamma_{j^\star})$.

L'étape précédente nous donne des relations d'ordres croisés, en particulier des équations du type :

$$\beta_{i^\star,l}^{n_1} = \beta_{i^\star,j^\star}^{n_2}$$

$$\Rightarrow \left(\frac{\gamma_{i^\star}}{\gamma_l}\right)^{n_1} = \left(\frac{\gamma_{i^\star}}{\gamma_{j^\star}}\right)^{n_2}$$

$$\Rightarrow \quad \gamma_l^{n_1} = \gamma_{j^\star}^{n_2} * \gamma_{i^\star}^{n_1-n_2}$$

Donc pour chaque paire valide $(\gamma_{i^\star}, \gamma_{j^\star})$ il n'existe qu'une seule valeur pour γ_l qui soit valide, et nous pouvons appliquer la même méthode pour le quatrième paramètre. Finalement nous obtenons un ensemble réduit à 255 candidats valides pour être les paramètres de `MixColumns`. □

Remarque 5. *Pour chacun des quadruplets candidats, les équations obtenues durant cette étape peuvent aussi être utilisées pour calculer $\tau^{(0)}$, et donc toute la séquence $(\tau^{(k)})_{k=0,\ldots,254}$.*

4.4.2.6 Retrouver les paramètres de `MixColumns` et de `RotWord`

Cette étape permet de retrouver les paramètres de `MixColumns` et de `RotWord` et utilise principalement des données déjà acquises durant les étapes précédentes. Nous utilisons des relations présentes dans le processus de *key schedule* qui mettent en jeu η, le paramètre de l'opération `RotWord`, que nous combinons avec des équations obtenues précédemment liant $k_{1,i}$ et $mk_{i,j}$ (pour $i = 0, \ldots, 3$ et pour tout j). Nous sommes capables de reconnaître $mk_{i,j}$ comme une valeur $\theta^{(q)}$. Nous développons ici les équations pour $i = 0$, des équations pour d'autres i peuvent êtres établies de la même façon :

$$\begin{cases} k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho^0 \\ k_{1,0} = \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \oplus S^{-1}(0) \end{cases}$$

$$\begin{aligned} \Rightarrow \qquad S(k_{0,12+\eta}) &= k_{0,0} \oplus S^{-1}(0) \oplus 1 \oplus \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \\ &= \lambda_0 \oplus 1 \oplus \gamma_j * S(\theta^{(q_1)} \oplus k_{0,0}) \\ &= \lambda_0 \oplus 1 \oplus \gamma_j * \tau^{(q_1)} \end{aligned}$$

D'après la Remarque 5, pour tout candidat aux paramètres de `MixColumns`, la séquence des valeurs τ est connue permettant que la partie droite de la dernière équation soit connue et reconnaissable en tant que $\tau^{(q_2)}$ qui est directement reliée à une valeur connue $\theta^{(q_2)}$:

$$\Rightarrow \quad S(k_{0,12+\eta}) = \tau^{(q_2)}$$

$$\Rightarrow \quad S(k_{0,12+\eta}) = S(\theta^{(q_2)} \oplus k_{0,0})$$

$$\Rightarrow \quad k_{0,12+\eta} = \theta^{(q_2)} \oplus k_{0,0}$$

$$\Rightarrow \quad \theta^{(q_2)} = \lambda_0 \oplus \lambda_{12+\eta}$$

La partie gauche de la dernière équation ne dépend que du quadruplet candidat au `MixColumns` en cours d'évaluation sur les 255 possibles. La partie droite ne dépend que du paramètre de `RotWord` qui ne peut prendre que quatre valeurs possibles. On a à disposition quatre équations de ce type (pour $i = 0, \ldots, 3$), mais dans la majorité des cas deux sont suffisantes. L'utilisation d'une des équations comme contrainte ne va laisser que 4 quadruplets en lice, un par valeur de η. La probabilité est élevée que l'intersection avec les 4 quadruplets sélectionnés par une autre équation va laisser une seule solution possible pour la paire $((\gamma_0, \gamma_1, \gamma_2, \gamma_3), \eta)$.

4.4.2.7 Retrouver $S^{-1}(0)$

Nous sommes à présent capables de calculer $k_{1,4}$:

$$\begin{cases} k_{1,0} = k_{0,0} \oplus \tau^{(q_2)} \oplus 1 \\ k_{1,4} = k_{1,0} \oplus k_{0,4} \end{cases}$$

$$\begin{aligned} \Rightarrow \qquad k_{1,4} &= k_{0,0} \oplus \tau^{(q_2)} \oplus 1 \oplus k_{0,4} \\ &= \tau^{(q_2)} \oplus 1 \oplus \lambda_0 \oplus S^{-1}(0) \ \oplus \lambda_4 \oplus S^{-1}(0) \\ &= \tau^{(q_2)} \oplus 1 \oplus \lambda_0 \oplus \lambda_4 \end{aligned}$$

On utilise alors $k_{1,4}$ pour dériver $S^{-1}(0)$ d'une équation obtenue par le Corollaire 1 :

$$\begin{aligned} k_{1,4} &= \gamma_j * S(mk_{4,j} \oplus k_{0,0}) \oplus S^{-1}(0) \\ \Rightarrow \qquad S^{-1}(0) &= \gamma_j * S(\theta^{(q_3)} \oplus k_{0,0}) \oplus k_{1,4} \\ &= \gamma_j * \tau^{(q_3)} \oplus k_{1,4} \end{aligned}$$

Une fois $S^{-1}(0)$ obtenu nous pouvons déduire K_0 des valeurs λ_i. Sachant $k_{0,0}$ il nous est possible d'inférer la table S-Box depuis les 255 équations $\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$. Ce niveau de connaissance nous permet de calculer le premier tour de *key schedule* et obtenir K_1.

4.4.2.8 Retrouver les paramètres de `Rcon`

Maintenant que nous connaissons K_0, K_1 ainsi que tous les paramètres de l'AES excepté ρ, nous avons le contrôle sur la valeur T_2. Nous sommes donc capables de tester de manière exhaustive les valeurs de $t_{2,0}$ jusqu'à provoquer une IFA en sortie de la première S-Box du troisième tour. L'équation d'IFA $y_{3,0} = 0$ permet de retrouver $k_{2,0} = t_{2,0} \oplus S^{-1}(0)$. La valeur de $k_{2,0}$ qui nous permet de simplement calculer ρ :

$$\begin{aligned} k_{2,0} &= k_{1,0} \oplus S(k_{1,12+\eta}) \oplus \rho^1 \\ \Rightarrow \qquad \rho &= k_{1,0} \oplus S(k_{1,12+\eta}) \oplus k_{2,0} \end{aligned}$$

4.4.3 Résultats Expérimentaux

Dans l'objectif d'estimer le nombre de fautes nécessaires pour retrouver les spécifications complètes d'un AES secret nous avons réalisé des simulations sur PC. Pour chaque étape de l'attaque décrite ci-dessus nous avons développé un programme qui simule cette portion de l'attaque afin de pouvoir évaluer le coût de chacune d'elles individuellement. Tous ces programmes ont été exécuté sur un grand nombre de tours de simulations. Chaque tour comprend les étapes suivantes :

TABLE 4.1 – Résultats expérimentaux de l'attaque FIRE sur une implémentation non protégée

Étape	# fautes
4.4.2.1 - Retrouver les valeurs λ_i	2 056
4.4.2.2 - Retrouver `ShiftRows`	138
4.4.2.3 - Retrouver l'ordre des $\beta_{i,j}$	22 340
4.4.2.4 - Retrouver les relations d'ordres croisés	0
4.4.2.5 - Retrouver les relations K_1	916
4.4.2.6 - Retrouver `MixColumns` et `RotWord`	64
4.4.2.7 - Retrouver $S^{-1}(0)$	0
4.4.2.8 - Retrouver `Rcon`	128
Total	**25 642**

1. Un AES modifié est généré par sélection aléatoire de ses différents paramètres conformément aux latitudes définies en Section 4.2, comme par exemple la propriété supplémentaire que au moins un des $\beta_{i,j}$ a 255 pour ordre multiplicatif,
2. une clef secrète K est générée aléatoirement,
3. tous les paramètres (ou connaissance sur les clefs) supposés avoir été retrouvés dans des étapes ultérieures sont considérés comme connus,
4. un oracle simule une expérimentation parfaite d'IFA : il prend en entrée tous les paramètres de l'AES, la clef, un clair et une position de faute (exprimée comme un numéro de tour et un indice de S-Box) et retourne un booléen qui indique si la valeur de sortie de cette S-Box est zéro (cas d'occurrence d'une IFA) ou pas.
5. l'étape de l'attaque est réalisée telle que décrite dans la section correspondante et le nombre d'appels à l'oracle est compté.

La Table 4.1 donne le nombre de fautes (tentative d'obtenir une IFA) requises pour chaque étape moyenné sur 10^6 simulations. Par souci de clarté nous avons aussi mentionné les quelques étapes ne nécessitant aucune faute sachant que nous ne les avons évidemment pas simulées. Comme on peut le voir l'étape la plus coûteuse est celle de la détermination des ordres des $\beta_{i,j}$ qui requiert environ 22 300 fautes même en ayant implémenté l'astuce d'optimisation décrite dans la Remarque 4. Les autres étapes sont bien moins coûteuses menant à ce qu'en moyenne environ 25 600 fautes soient nécessaires pour recouvrer complètement une spécification d'un algorithme de type AES modifié. Comme cela peut sembler un grand nombre de fautes nous tenons à mettre en avant que l'entropie de l'information secrète à retrouver est souvent beaucoup plus importante dans le cadre d'une attaque FIRE que dans le cadre d'attaques en fautes plus classiques visant à simplement retrouver une clef. Dans le cas d'AES modifié présenté ici l'entropie totale à recouvrer s'élève à environ 1 734 bits [12] d'information en plus des 128 bits de la clef.

4.4.4 Cas particuliers

Dans cette section nous étudions deux définitions étendues d'un AES modifié et décrivons comment adapter l'attaque FIRE présentée ci-dessus à ces nouvelles variantes. La première variante que nous avons étudiée consiste en un choix plus large de paramètres de `MixColumns`, par

12. Cela prend en compte tous les éléments secrets : S-Box ($\log_2(256!) \simeq 1684$ bits), `ShiftRows` (4 x 2 bits), `MixColumns` (4 x 8 bits), `RotWord` (2 bits), `Rcon` (8 bits).

$$\begin{bmatrix} \gamma_0 & \gamma_4 & \gamma_8 & \gamma_{12} \\ \gamma_1 & \gamma_5 & \gamma_9 & \gamma_{13} \\ \gamma_2 & \gamma_6 & \gamma_{10} & \gamma_{14} \\ \gamma_3 & \gamma_7 & \gamma_{11} & \gamma_{15} \end{bmatrix}$$

FIGURE 4.8 – Matrice de `MixColumns` étendue

l'autorisation de n'importe quelle matrice inversible de 16 coefficients à la place des matrices seulement circulantes. L'entropie est augmentée d'environ 32 à environ 128 bits. La seconde variante concerne le paramètre de `Rcon` en autorisant un ensemble de 10 octets indépendants ρ'_r au lieu des valeurs reliées successives ρ^{r-1} (pour $r = 1, \ldots, 10$). Cela augmente l'entropie des paramètres de `Rcon` de 8 à 80 bits.

4.4.4.1 Matrice de `MixColumns` de pleine entropie

La matrice de `MixColumns` n'est pas nécessairement circulante et a 16 coefficients "indépendants" au lieu de 4. La nouvelle matrice est représentée en Figure 4.8 où on peut voir la nouvelle numérotation colonne par colonne.

Les Étapes 4.4.2.1 et 4.4.2.2 ne sont pas impactées par ce changement et permettent de retrouver les valeurs λ_i et les paramètres de `ShiftRows` comme décrit précédemment. Le Corollaire 1 ne permet plus d'obtenir n'importe quelles valeurs $mk_{i,j}$ mais seulement celles dont les indices vérifient $i \equiv j \pmod 4$.

Corollaire 2. *La connaissance de* $(\lambda_0, \ldots, \lambda_{15})$ *et des paramètres de* `ShiftRows` *permet de retrouver n'importe quel* $mk_{i,j}$ *où* $i \equiv j \pmod 4$.

Démonstration. Soit r et c représentant respectivement la ligne et la colonne d'un élément i. Une IFA sur la valeur $y_{2,i}$ donne une équation impliquant les coefficients γ_{0+r}, γ_{4+r},γ_{8+r} et γ_{12+r} d'une même ligne r :

$$\gamma_{0+r} * z_{1,4c} \oplus \gamma_{4+r} * z_{1,4c+1} \oplus \gamma_{8+r} * z_{1,4c+2} \oplus \gamma_{12+r} * z_{1,4c+3} \oplus k_{1,i} = S^{-1}(0)$$

Comme indiqué dans le Corollaire 1, l'attaquant est capable de forcer à zéro trois octets d'une colonne d'entrée d'un `MixColumns`, et tester de manière exhaustive le dernier jusqu'à obtention d'une IFA. Chaque position $4c + k$ (pour $k = 0, \ldots, 3$) de l'octet actif nous donne une des équations suivantes :

$$\begin{aligned} \gamma_{0+r} * z_{1,4c} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{4+r} * z_{1,4c+1} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{8+r} * z_{1,4c+2} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{12+r} * z_{1,4c+3} &= k_{1,i} \oplus S^{-1}(0) \end{aligned}$$

Comme on peut le voir on ne peut obtenir de valeurs $mk_{i,j}$ que pour $j = 4k + r$ ce qui revient à $j \equiv i \pmod 4$. □

À cause du Corollaire 2, l'Étape 4.4.2.3 doit être appliquée séparément sur chaque ligne r, permettant de retrouver six valeurs $\beta_{i,j}$ par ligne avec $(i, j) \in \{0 + r, 4 + r, 8 + r, 12 + r\}$. Nous voulons mettre en avant que lorsque le premier ordre $\beta_{i,j}$ égal à 255 est découvert, les ordres

suivants (même pour d'autres lignes) peuvent être identifiés rapidement grâce à l'optimisation décrite en Remarque 4. En conséquence le coût total de cette étape n'est pas significativement plus élevé que pour le cas d'une matrice circulante.

L'Étape 4.4.2.4 dérive des relations d'ordres croisés entre deux $\beta_{i,j}$ lorsque l'ordre de β_{i_1,j_1} divise celui de β_{i_2,j_2}. Il est intéressant de remarquer que de telles relations peuvent toujours être dérivées, que les deux $\beta_{i,j}$ proviennent d'une même ligne ou pas. En effet, le Lemme 1 qui permet de forcer à une même valeur les sorties de n'importe quel couple de S-Box du premier tour, nous permet faire débuter toutes les séquences par une même valeur $\theta^{(0)}$. Comme toutes les séquences sont basées sur la même valeur initiale nous pouvons identifier l'index t tel que $\theta^{(1)}_{i_1,j_1} = \theta^{(t)}_{i_2,j_2}$, révélant des relations d'ordres croisés.

En Section 4.4.2.5, nous utilisions deux équations – sans compter les huit apportées par le Lemme 2 – réduisant à 255 le nombre de candidats aux paramètres de `MixColumns`. La même méthode peut être appliquée pour le cas de la matrice étendue puisque nous avons toujours deux équations disponibles par ligne. Nous utilisons alors toutes les huit équations – deux par ligne – pour réduire à 255 valeurs chaque ensemble de candidats pour une ligne de `MixColumns`.

La méthode décrite en Section 4.4.2.6 doit être réadaptée et ne permet plus de retrouver complètement les paramètres de `MixColumns` et `RotWord` mais crée en revanche des relations exclusives entre les paramètres passant de 255^4 possibilités à seulement 255. Dans notre attaque de base nous utilisions une des quatre équations disponibles ressemblant à :

$$\begin{cases} k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho^0 \\ k_{1,0} = \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \oplus S^{-1}(0) \end{cases}$$

pour réduire le nombre de candidats de `MixColumns` à seulement quatre, exactement un par candidat pour η. Cette information était alors combinée avec une seconde équation (e.g. une utilisant $k_{1,1}$ et $mk_{1,j}$) afin de déterminer le couple correct de paramètres de `MixColumns` et `RotWord`. Dans le cas de la matrice étendue chaque équation impliquant $k_{1,i}$ est nécessairement reliée à la ligne i. Donc nous ne pouvons utiliser qu'une seule équation par ligne et ne pouvons que réduire le nombre de candidats à 4 par ligne, un par candidat pour η. Nous obtenons donc un candidat de matrice de `MixColumns` pour chaque candidat de `RotWord`.

Pour finir cette attaque on peut remarquer qu'il nous reste 2^2 candidats pour le couple `MixColumns` et `RotWord`, 2^8 candidats pour $S^{-1}(0)$ et 2^8 candidats pour ρ, le paramètre de `Rcon`. On est en mesure de considérer chacune des 2^{18} combinaisons de ces candidats, chacune nous permettant de retrouver la table S-Box correspondante et ainsi de calculer tout le *key schedule* En effet, pour chaque combinaison de candidats la séquence τ est connue car nous connaissons la matrice de `MixColumns` et $k_{0,0}$ est aussi connu car nous connaissons $S^{-1}(0)$. On peut alors retrouver la table S-Box par l'exploitation des 255 relations $\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$.

On est alors capable d'appliquer la méthode de la Section 4.4.2.1 aux S-Boxs du dernier tour et retrouver ainsi n'importe quel octet de K_{10}. Par exemple lorsqu'une IFA survient sur la S-Box numéro i du dernier tour nous apprenons que $y_{10,i} = 0$ ce qui implique que $k_{10,i'} = c_{10,i'}$ où $i' = (i - 4\sigma_{i \bmod 4}) \bmod 16$. Ayant obtenu des valeurs d'octets de K_{10}, on peut alors les comparer avec la prédiction de valeur de K_{10} que chacun des 2^{18} candidats aux paramètres manquants vont générer pour déterminer quelle combinaison est la bonne.

Remarque 6. *Comparativement avec le cas de la matrice circulante, la proportion de matrices pour lesquelles il existe au moins un ordre de $\beta_{i,j}$ valant 255 passe de 95.28% à 99.99%, cela étant dû au fait qu'il peut apparaître sur n'importe quelle ligne.*

4.4.4.2 Paramètre de `Rcon` étendu

À la place de définir les paramètres `Rcon`$[r] = \rho^{r-1}$ comme étant tous dépendants du même octet ρ, nous étudions dans cette section le cas où ils sont tous indépendants. On les note ρ'_r (pour $r = 1, \ldots, 10$) et l'équation qui définit le premier octet $k_{r,0}$ de chaque tour devient :

$$k_{r,0} = k_{r-1,0} \oplus S(k_{r-1,12+\eta}) \oplus \rho'_r$$

Les paramètres de `Rcon` ne sont impliqués dans des équations que lors des étapes des Sections 4.4.2.6, 4.4.2.7 et 4.4.2.8.

Pour l'Étape 4.4.2.6 nous retrouvons les paramètres de `MixColumns` et `RotWord` en utilisant deux équations parmi les quatre disponibles :

$$\begin{aligned}
k_{1,0} &= k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho'_1 \\
k_{1,1} &= k_{0,1} \oplus S(k_{0,12+(1+\eta) \bmod 4}) \\
k_{1,2} &= k_{0,2} \oplus S(k_{0,12+(2+\eta) \bmod 4}) \\
k_{1,3} &= k_{0,3} \oplus S(k_{0,12+(3+\eta) \bmod 4})
\end{aligned}$$

Comme une seule d'entre elles contient un paramètre de `Rcon` on peut simplement l'éviter et terminer cette étape sans autre modification.

Pour l'Étape 4.4.2.7 nous récupérons $S^{-1}(0)$ grâce au calcul d'une des quatre valeurs $k_{1,4}$, $k_{1,5}$, $k_{1,6}$ et $k_{1,7}$. Nous avions pris $k_{1,4}$ comme exemple, il est relié aux paramètres de `Rcon` par sa relation avec $k_{1,0}$:

$$\begin{cases}
k_{1,4} = k_{1,0} \oplus k_{0,4} \\
k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho'_1
\end{cases}$$

Comme dans l'étape précédente nous pouvons simplement éviter le problème en choisissant n'importe laquelle des trois autres valeurs qui, elles, ne sont pas reliées à ρ'_1.

La dernière étape détaillée en Section 4.4.2.8 a pour but de retrouver le dernier paramètre inconnu : ρ. Nous avions utilisé le fait qu'on avait le contrôle sur l'état T_2 qui permettait de calculer le paramètre `Rcon` du tour 2. Nous n'avons plus le contrôle sur T_2 mais ce raisonnement peut être tenu à n'importe quel tour puisqu'il dépend de la structure du *key schedule* qui est identique à chaque tour.

Dans le cas de `Rcon` étendu, les seuls paramètres manquant sont les dix valeurs ρ'_r. Nous avons le contrôle sur l'état T_1 qui nous permet de découvrir la valeur ρ'_1. À présent nous pouvons calculer la valeur de K_1 nous donnant le contrôle sur l'état T_2, nous permettant de la même manière de retrouver itérativement toutes les valeurs ρ'_r.

Le surcoût engendré par l'extension du paramètre de `Rcon` n'est présent que dans le fait que la méthode décrite en Section 4.4.2.8 doit être réalisée pour chaque tour au lieu d'un seul. Le coût moyen théorique de cette étape devient donc de 1275 fautes au lieu de 127.5. C'est une faible augmentation au regard du coût total de l'attaque.

4.5 Attaque SCARE utilisant le modèle de collisions en valeurs

Dans cette section nous décrivons l'attaque SCARE réalisée dans le modèle de collisions en valeurs. Nous rappelons que dans ce modèle l'attaquant a la capacité de détecter lorsque deux accès à la table S-Box, $y = S(x)$ et $y' = S(x')$, vérifient $x = x'$ et $y = y'$. Nous avions aussi mis en place deux types d'attaquants : en inter-courbes il est capable de détecter les collisions

entre deux appels à la table S-Box provenant de la même exécution ou bien de deux exécutions différentes, lorsqu'en intra-courbe il n'est capable de détecter les collisions qu'entre deux appels à l'intérieur d'une même exécution.

La Section 4.5.1 décrit une attaque dans ce modèle lorsque aucune contre-mesure n'est implémentée, la Section 4.5.2 quant à elle décrit une attaque valide même en présence de certaines contre-mesures classiques. En Section 4.5.3 nous détaillons une extension de la Section 4.5.1 à un paramètre de `Rcon` à entropie augmentée. Enfin, en supposant que l'attaquant possède *a priori* les valeurs relatives des octets de la clef, nous proposons en Section 4.5.4 l'extension de l'attaque de la Section 4.5.2 à une implémentation protégée par un masquage d'ordre supérieur (128 bits).

4.5.1 Attaque SCARE par collisions en valeurs sans contre-mesure

Dans cette section nous décrivons comment retrouver les paramètres secrets d'une implémentation d'AES qui ne comporte pas de contre-mesure face aux attaques par canaux auxiliaires. Nous procédons étape par étape, où l'ordre de ces étapes est important car chacune dépend des données acquises par les précédentes.

Lorsque cela est pertinent, nous détaillons deux méthodes, une pour l'attaquant inter-courbes, l'autre pour l'attaquant intra-courbe.

4.5.1.1 Retrouver les paramètres de `ShiftRows`

Pour le cas de l'attaquant inter-courbes nous pouvons facilement retrouver les paramètres σ_i. Dans un premier temps nous faisons l'acquisition d'une courbe avec un clair choisi aléatoirement, nous comparons ensuite cette courbe avec quatre autres correspondantes à la modification d'un seul octet de clair m_i ($i = 0, \ldots, 3$). Pour chaque ligne i, l'observation du quadruplet de S-Boxs consécutives qui ne collisionnent pas au deuxième tour nous révèle la valeur de σ_i.

Dans le scénario intra-courbe, les choses sont un peu plus complexes :

Lemme 4. *Soit une collision entre la S-Box i du premier tour et la S-Box j du deuxième tour. Il existe deux façons de détruire cette collision par modification d'un seul octet de clair : (i) si l'octet actif est en position i ou, (ii) si l'octet actif est impliqué dans le calcul de $x_{2,j}$.*

Selon si une des quatre positions impliquées dans le calcul de $x_{2,j}$ est égale à la valeur i ou pas, il y a respectivement 4 ou 5 octets actifs détruisant la collision.

Définition 1. *Nous noterons 4-`Collision` et 5-`Collision` les collisions pouvant être détruites par, respectivement, 4 et 5 octets de clair.*

Lemme 5. *Les quatre octets impliqués dans le calcul de la même valeur $x_{2,j}$ appartiennent tous à des lignes différentes de la matrice d'état et sont alignés sur une même colonne en sortie de l'opération `ShiftRows`.*

Pour retrouver les paramètres de `ShiftRows` nous chiffrons en premier lieu des clairs aléatoires jusqu'à obtenir une collision unique entre la S-Box i du premier tour et la S-Box j du deuxième. Une fois ce clair particulier obtenu nous allons chiffrer, pour tout $k \neq i$, ce même clair où seulement m_k aura été modifié, et nous identifions alors les 3 ou 4 positions détruisant la collision.

Le premier cas (cf. rouge et $*$ dans la Figure 4.9) correspond à une 4-`Collision` et l'ensemble des trois positions identifiées réunies avec i sont impliquées dans le calcul de $x_{2,j}$. Le second cas (cf. bleu et $\boxtimes$ dans la Figure 4.9) correspond à une 5-`Collision` et les quatre positions identifiées sont reliées à $x_{2,j}$. Dans les deux cas ces quatre positions sont égales à $\{4((c+\sigma_\ell) \bmod 4)+\ell\}_{\ell=0,\ldots,3}$ où $c = \lfloor j/4 \rfloor$ est la colonne de la collision. Elles sont toutes différentes modulo 4, il est donc aisé de déduire les paramètres σ_ℓ de celles-ci.

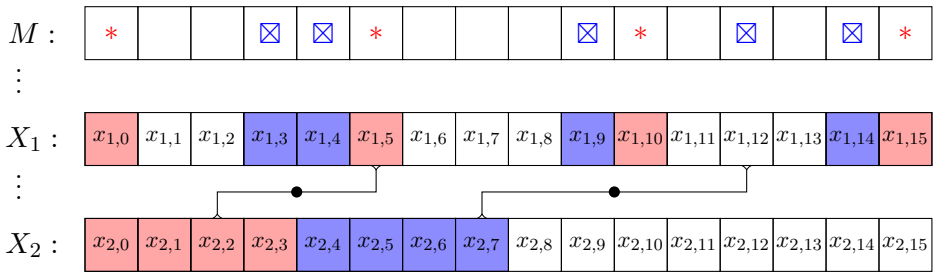

FIGURE 4.9 – Collision entre $x_{1,5}$ et $x_{2,2}$ révélant une 4-`Collision`(gris clair/rouge,$*$). Et collision entre $x_{1,12}$ et $x_{2,7}$ révélant une 5-`Collision`(gris foncé/bleu,$\boxtimes$)

4.5.1.2 Retrouver K_0 et K_{10} à un XOR avec un octet constant près

La première étape consiste à détecter des collisions entre des S-Boxs du premier tour aux indices i et j (cf. gris clair/vert dans la Figure 4.10) provenant d'une même courbe (ou de courbes différentes dans le cas d'un attaquant inter-courbes). Chacune de ces collisions implique l'égalité des deux entrées des S-Boxs entrant en collision et nous donne donc une relation linéaire entre deux octets de clef :

$$\begin{aligned} x_{1,i} = x_{1,i'} \quad &\Leftrightarrow \quad (m_i \oplus k_{0,i}) = (m_{i'} \oplus k_{0,i'}) \\ &\Leftrightarrow \quad k_{0,i} \oplus k_{0,i'} = m_i \oplus m_{i'} \end{aligned}$$

L'acquisition de telles relations en jouant des clairs aléatoires finit par nous permettre de relier entre eux tous les octets de clef. Nous avons à présent la connaissance de toutes les différentielles $\mu_{0,i,i'} = k_{0,i} \oplus k_{0,i'}$, la clef K_0 est donc retrouvée à un XOR avec une constante près. Par exemple, il suffirait de connaître la valeur de $k_{0,0}$ pour calculer les autres octets de clef par les formules $k_{0,i} = k_{0,0} \oplus \mu_{0,0,i}$.

Comme nous avons précédemment retrouvé les paramètres de `ShiftRows` nous sommes capables de relier un octet de chiffré avec un octet de sortie de S-Box du dernier tour. De la même manière que pour K_0, le chiffrement de clairs aléatoires et l'observation des collisions s'opérant entre des S-Boxs du dernier tour (cf. gris moyen/rouge dans la Figure 4.10) rendent possible l'accumulation de relations linéaires de type $\mu_{10,i,i'} = k_{10,i} \oplus k_{10,i'} = c_i \oplus c_{i'}$ qui finissent par révéler K_{10} au XOR avec un octet constant près. Il est à noter que le même lot de courbes peut être utilisé pour retrouver K_0 et K_{10} à une constante près.

4.5.1.3 Retrouver la table S-Box

Une collision entre une S-Box du premier tour à l'indice i et une S-Box du dernier tour à l'indice j (cf. gris foncé/bleu dans la Figure 4.10) implique que $x_{1,i} = x_{10,j}$ et $y_{1,i} = y_{10,j}$. Notons $x = x_{1,i}$ et $y = y_{10,j}$, la collision révèle une relation impliquant la table S-Box : $S(x) = y$ pour deux valeurs $x = x' \oplus k_{0,0}$ et $y = y' \oplus k_{10,0}$ où $x' = m_i \oplus \mu_{0,0,i}$ et $y' = c_{j'} \oplus \mu_{10,0,j'}$ [13] sont connus de l'attaquant.

La table S-Box peut ainsi être retrouvée par le chiffrement de clairs aléatoires et par l'observation de telles collisions (possiblement sur des courbes différentes) entre les S-Boxs du premier et dernier tour. Une fois que les 256 relations sur la table S-Box du type :

$$S(x' \oplus k_{0,0}) = y' \oplus k_{10,0}$$

13. À cause de `ShiftRows` l'octet de chiffré relié à la collision est à l'indice $j' = \ell + 4((c - \sigma_c) \bmod 4)$ où $\ell = j \bmod 4$ et $c = \lfloor j/4 \rfloor$.

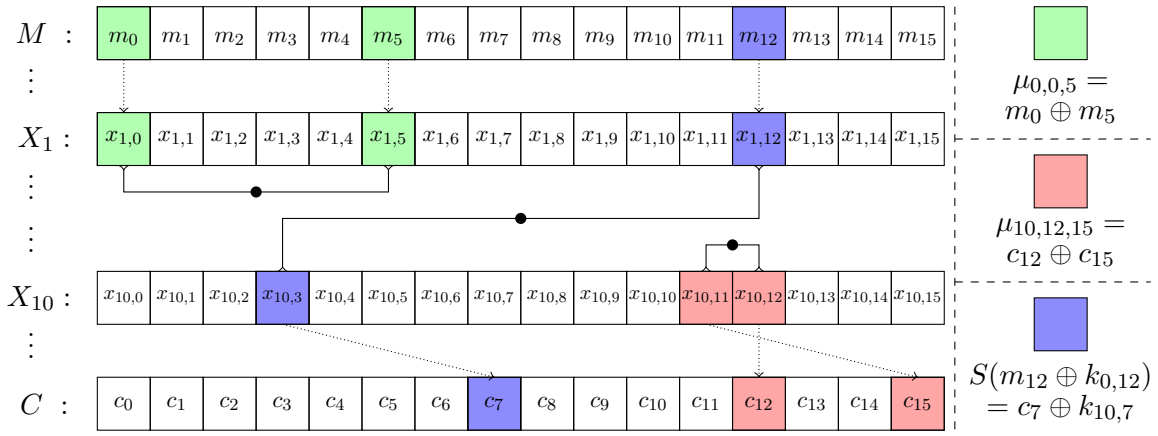

FIGURE 4.10 – Exemples de collisions utilisées durant différentes étapes de l'attaque pour retrouver K_0 (gris clair/vert), K_{10} (gris moyen/rouge) ou les S-Boxs (gris foncé/bleu)

ont été identifiées pour tous les couples (x', y') la table S est retrouvée à deux permutations près de type XOR, sur les entrées et sur les sorties respectivement.

Lorsque seules les collisions à l'intérieur d'une même courbe peuvent être exploitées, les relations sont récupérées comme dans le problème du collecteur de coupons*. Dans ce cas nous pouvons gagner un grand nombre de courbes en choisissant les clairs de manière à ce que chaque $x'_i = m_i \oplus \mu_{0,0,i}$ soient différents les uns des autres et ne fasse pas partie de relations déjà obtenues.

4.5.1.4 Retrouver K et les paramètres du *key schedule*

Cette étape consiste simplement en une recherche exhaustive parmi 2^{26} combinaisons et ne nécessite l'acquisition d'aucune nouvelle courbe. Il a pour but de retrouver la clef K simultanément avec les autres paramètres du *key schedule* constitués du nombre de rotations η de l'opération `RotWord` et de l'octet constant ρ utilisé pour le vecteur `Rcon`.

Pour chaque candidat pour $k_{0,0}$ et $k_{10,0}$ nous obtenons une valeur pour le couple K_0, K_{10} ainsi que pour la table S-Box. On peut alors faire une supposition parmi les 2^2 candidats à η et les 2^8 à la valeur de ρ de sorte à être en mesure d'exécuter l'opération *key schedule* et dériver les clefs de tour. Chacun de ces 2^{26} candidats suggère à travers son *key schedule* une valeur pour la clef du dernier tour, celle-ci est alors confrontée à la valeur présumée K_{10}. Comme c'est un test sur K_{10} de taille 128-bit, la probabilité de faux positif est très faible, ce qui a été confirmé par nos simulations.

Il est à noter qu'une extension naturelle de `Rcon` serait de définir chaque mot constant comme $\texttt{Rcon}[r] = (\rho_0^{r-1}, \rho_1^{r-1}, \rho_2^{r-1}, \rho_3^{r-1})$. La recherche exhaustive serait alors de 2^{50} exécutions ce qui peut être considéré comme engageant un coût non raisonnable. Aussi, pour faire face à ce type de cas, nous proposons en Section 4.5.3 une adaptation de notre attaque pouvant fonctionner avec ce `Rcon` augmenté à 32 bits d'entropie mais qui peut aussi fonctionner avec un `Rcon` augmenté à 320 bits d'entropie lorsque tous ces vecteurs sont indépendants.

4.5.1.5 Retrouver la matrice de `MixColumns`

En débutant cette étape nous avons retrouvé tous les paramètres secrets à l'exception des coefficients $\{\alpha_i\}_{i=0,\ldots,15}$ de la matrice de `MixColumns`. Nous sommes donc à présent à même de connaître l'entrée de l'étape `MixColumns` du premier tour pour chaque courbe déjà acquise.

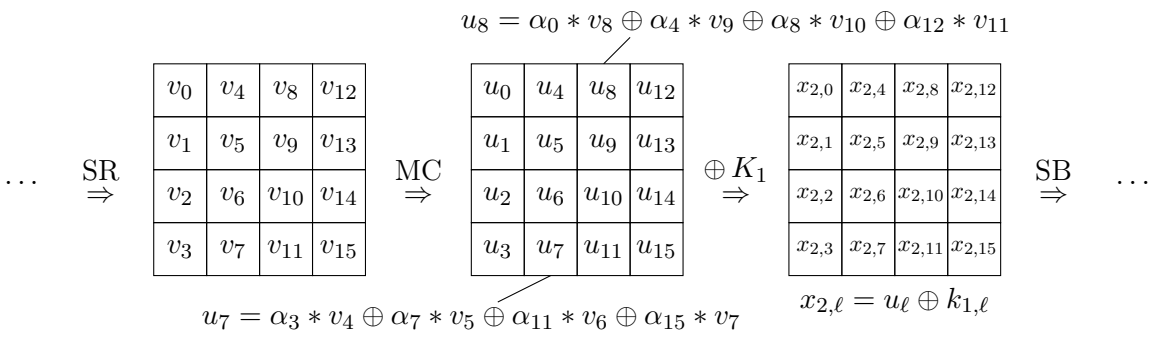

FIGURE 4.11 – Propagation des valeurs v à travers le `MixColumns` du premier tour

Comme on peut le voir sur la Figure 4.11, chaque octet u_{i+4j} en sortie de `MixColumns` dépend de 4 paramètres provenant de la même ligne : $\{\alpha_i, \alpha_{i+4}, \alpha_{i+8}, \alpha_{i+12}\}$:

$$u_{i+4j} = \alpha_i * v_{4j} \oplus \alpha_{i+4} * v_{4j+1} \oplus \alpha_{i+8} * v_{4j+2} \oplus \alpha_{i+12} * v_{4j+3}$$

Le but est d'obtenir ce type d'équation en déterminant, par collision, certaines valeurs $x_{2,\ell}$ en entrée de S-Boxs du deuxième tour, valeurs desquelles on inférera u_ℓ comme étant $u_\ell = x_{2,\ell} \oplus k_{1,\ell}$, l'entrée de l'étape `MixColumns` étant, quant à elle, simplement dérivée du clair. La capture de 4 équations indépendantes impliquant le même lot de 4 paramètres forme un système d'équations qui peut ainsi être résolu, révélant ainsi une ligne de la matrice de `MixColumns`. Établir un tel système par ligne permet de retrouver la matrice au complet.

Aucune nouvelle courbe n'est nécessaire pour cette étape, que l'attaquant soit inter-courbes ou intra-courbe, il peut exploiter les courbes déjà acquises au cours des étapes précédentes afin d'établir ses équations. Les collisions identifiant les valeurs $x_{2,\ell}$ peuvent être réalisées avec les valeurs de X_1 qui sont connues mais aussi avec les valeurs de X_{10} connues également. Nos simulations montrent que les courbes acquises précédemment à cette étape sont toujours bien plus abondantes que nécessaire pour obtenir les équations désirées.

4.5.1.6 Résultats expérimentaux

Pour vérifier la robustesse de notre attaque et faire une estimation du nombre de courbes nécessaires à l'obtention de l'ensemble des paramètres d'un AES aux spécifications secrètes nous avons réalisé des simulations sur PC. Pour chacune des étapes successives nous avons développé un programme simulant uniquement cette étape afin d'en évaluer le coût individuel. Tous ces programmes ont été exécutés sur un grand nombre de simulations comprenant les étapes suivantes :

1. Un AES modifié est généré par sélection aléatoire de ses différents paramètres conformément aux latitudes définies en Section 4.2,
2. une clef secrète K est générée aléatoirement,
3. tous les paramètres (ou connaissance sur les clefs) supposés avoir été retrouvés dans des étapes antérieures sont considérés comme connus,
4. un oracle simule une détection parfaite des collisions de valeurs : il prend en entrée tous les paramètres de l'AES, la clef, un clair et deux positions de S-Boxs (potentiellement sur deux tours/courbes différents/es) et retourne un booléen indiquant si les entrées/sorties de cette paire de S-Boxs sont entrées en collisions ou non.

TABLE 4.2 – Résultats expérimentaux sur une implémentation non protégée

Étape	# courbes		# exé-
	intra	inter	-cutions
Section 4.5.1.1 - Retrouver `ShiftRows`	11	5	10 000
Section 4.5.1.2 - Réduire K_0 et K_{10} à 8 bits d'entropie	70	8	10 000
Section 4.5.1.3 - Retrouver la S-Box	288	81	10 000
Section 4.5.1.4 - Retrouver K et le *key schedule*	0	0	–
Section 4.5.1.5 - Retrouver `MixColumns`	0	0	10 000
Total	**369**	**94**	

5. l'étape de l'attaque est réalisée telle que décrite dans la section correspondante et le nombre de courbes utilisées dans l'oracle est compté.

La Table 4.2 présente le nombre de courbes – en moyenne sur 10 000 exécutions – requises par chaque étape, en inter-courbes ainsi qu'en intra-courbe. Par souci de clarté nous avons aussi placé les deux dernières étapes malgré le fait qu'elles ne demandent aucune courbe supplémentaire.

Notre attaque sur une implémentation non protégée permet de retrouver l'ensemble des paramètres d'un AES modifié, ainsi que la clef, en moins de 400 courbes en moyenne en intra-courbe et en moins de 100 courbes en moyenne en inter-courbes.

4.5.2 Attaque SCARE par collisions en valeurs avec contre-mesures

Dans cette section nous considérons une implémentation de l'AES protégée au premier ordre contre les attaques par canaux auxiliaires. Plus précisément nous étudions une implémentation protégée par deux contre-mesures conjointes :

La première consiste en un masquage booléen de taille 8 bits sur toutes les données intermédiaires du chiffrement ainsi que du key schedule. Notre modèle d'attaquant fait que nous subissons les effets de cette contre-mesure uniquement sur les entrées et sorties des opérations `SubBytes`. Le masquage de toutes les autres opérations n'a aucun impact sur notre attaque. L'opération `SubBytes` utilise donc une version modifiée aléatoirement de la table S-Box, notée $\widetilde{S}$, au moyen de deux masques booléens de taille 8 bits, un en entrée, noté r_{in}, et un en sortie, noté r_{out} tels que $\widetilde{S}(x \oplus r_{in}) = S(x) \oplus r_{out}$ pour tout x. Les contraintes de mémoire et de temps d'exécution restreignent la plupart des implémentations à ne renouveler les aléas de la table S-Box qu'au début de chaque exécution. Nous supposons donc que chaque appel à la table S-Box utilisera la même table masquée au cours de tous les tours d'une même exécution [14]. Le principal effet négatif de cette contre-mesure pour l'attaquant est qu'il ne peut plus détecter et exploiter de collisions entre des S-Boxs provenant de différentes courbes. Il est à noter que l'analyse des collisions sur les appels à $\widetilde{S}$ dans une même courbe est toujours possible et révèle en fait des collisions sur la table non masquée. Donc, avec cette seule contre-mesure, la version intra-courbe détaillée en Section 4.5.3 s'applique parfaitement.

Nous considérons aussi une seconde contre-mesure, combinée à la précédente, modifiant aléatoirement l'ordre des 16 appels $\widetilde{y}_i = \widetilde{S}(\widetilde{x}_i)$ à chaque tour [15]. En conséquence, l'observation d'une

14. Une conversion des masques des valeurs intermédiaires sera effectuée à la fin de chaque tour afin que le masquage par r_{out} soit rétabli en r_{in} pour le début du tour suivant.

15. De même que pour le masquage, la modification de l'ordre des autres opérations telles que `ShiftRows`, `MixColumns`, `AddRoundKey`, etc. n'a aucune influence sur l'attaque dans le modèle de détection de collisions sur les

collision entre deux (ou plus) calculs de $\widetilde{S}(\widetilde{x})$ (potentiellement à des tours différents) ne donne plus d'information sur l'indice auquel la donnée d'entrée $\widetilde{x} = x \oplus r_{in}$ correspond. L'attaquant est donc limité à l'observation du nombre de valeurs différentes se trouvant en entrée des S-Boxs de chaque tour et de leurs occurrences. Afin de transcrire la vision de ce nouvel attaquant possédant une capacité d'information plus limitée nous introduisons la définition suivante :

Définition 2. *Nous appelons une n-structure (ou plus simplement une structure), de type* $n_1^{(t_1)} n_2^{(t_2)} \dots n_s^{(t_s)}$ *d'éléments de E, l'ensemble de tous les n-tuples d'éléments de E (avec* $n = \Sigma_k t_k n_k$*) tels que* $t = \Sigma_k t_k$ *éléments distincts apparaissent respectivement avec pour nombre d'occurrences* $n_1, n_2, \dots, n_s$.

Par exemple, n'importe quel X_r constitué d'octets d'entrée de S-Box tous différents appartient à une structure de type $1^{(16)}$ d'éléments de $GF(2^8)$. Comme autre exemple, le 16-tuple :

$$X_1 = (13, 47, 173, 47, 86, 119, 13, 47, 119, 223, 205, 119, 37, 88, 200, 5)$$

possède une structure $1^{(8)}2^{(1)}3^{(2)}$ car 13 apparaît deux fois et 47 et 119 apparaissent trois fois chacun.

4.5.2.1 Retrouver K_0 à un XOR avec un octet constant près

La première étape consiste en l'exécution de l'AES sur des clairs aléatoires jusqu'à en trouver un tel que l'opération `SubBytes` du premier tour présente une unique valeur collisionnante (une structure $1^{(16-n)}n^{(1)}$). Seulement quelques courbes sont suffisantes en moyenne pour obtenir une telle courbe de référence. Pour tout $i = 0, \dots, 15$, on peut alors modifier m_i et observer si la collision disparaît (ou si sa multiplicité n décroît). L'ensemble I des indices auxquels ce phénomène se produit vérifie :

$$\forall\ i, i' \in I,\ k_{0,i} \oplus k_{0,i'} = m_i \oplus m_{i'}$$

où m_i et $m_{i'}$ sont les valeurs d'octets du clair utilisé comme référence.

En comparant la structure de la courbe de référence avec celles d'au plus 16 autres modifiées, dans la plupart des cas l'attaquant se retrouvera avec $|I| = n$. Néanmoins, si $n = 2$, une non-détection peut se produire lorsque la modification d'un octet impliqué dans la collision initiale entraîne une collision avec une autre valeur qui ne collisionnait pas initialement. Dans de tels cas rares, il serait suffisant de changer à nouveau les valeurs d'octets pour révéler ceux impliqués dans la collision initiale.

Une fois qu'un ensemble de n indices collisionnants est identifié, n différents octets de clef sont alors reliés linéairement. En répétant cette procédure afin d'obtenir différents clairs de référence ayant une structure de type $1^{(16-n)}n^{(1)}$, nous finissons par obtenir des relations entre chaque octet. K_0 est alors obtenue à un XOR près avec un octet constant (e.g. $k_{0,0}$).

On peut remarquer deux astuces qui permettent de réduire le nombre de courbes requises pour établir les relations entre les octets de clef. Premièrement, lorsqu'une relation linéaire est obtenue entre des octets à des indices dans le sous-ensemble J, on est alors capable de choisir un nouveau clair de référence en faisant en sorte que les octets de X_1 appartenant à J soient tous différents, maximisant ainsi les chances de collisions apportant une information nouvelle. La seconde astuce est une interruption anticipée que l'on peut effectuer lors du processus d'établissement de l'ensemble I d'un clair de référence : dès qu'une nouvelle relation est découverte impliquant un octet de clef de J on peut alors ne pas considérer les autres indices de J car ils auront tous le même comportement vis-à-vis des éléments de I.

S-Box.

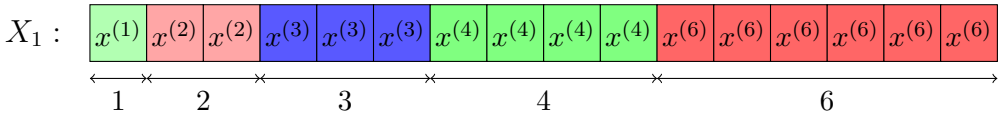

FIGURE 4.12 – Exemple d'état X_1 comportant cinq valeurs avec un nombre d'apparitions différent.

4.5.2.2 Retrouver K_{10} à un XOR avec un octet constant près

Si l'attaquant a accès à un oracle de déchiffrement à chiffré choisi, il lui est alors possible de retrouver la valeur de K_{10}, à un XOR avec une constante près, par la même méthode que celle utilisée pour K_0. On commence par chiffrer des clairs aléatoires jusqu'à en trouver un dont le `SubBytes` du dernier tour présente une structure de type $1^{(16-n)}n^{(1)}$. Soit C ce chiffré. Pour un chiffré modifié C', différant de C par un seul octet c_i, on peut chiffrer le clair $M' = \text{AES}_K^{-1}(C')$ et observer si la collision des S-Boxs au dernier tour a disparu ou non. De la même manière que pour K_0, il est alors possible d'obtenir des relations du type $k_{10,i} \oplus k_{10,i'} = c_i \oplus c_{i'}$. L'accumulation de ce genre de relations finira par révéler K_{10} à un XOR avec une constante près, avec la même complexité que pour K_0.

Lorsque l'attaquant n'a pas accès à un oracle de déchiffrement, il lui est toujours possible de retrouver K_{10}. Parmi un ensemble de clairs aléatoires, nous exploitons les courbes de ceux ne provoquant aucune collision dans les S-Boxs du dernier tour (une structure $1^{(16)}$). Dans ces cas là nous savons que pour chaque paire d'index (i, i'), $\mu_{10,i,i'}$ n'est obligatoirement pas égal à $c_i \oplus c_{i'}$. Débutant avec des listes de toutes les valeurs possibles pour tous les $\mu_{10,i,i'}$, et en accumulant de telles informations négatives, nous finissons par obtenir des listes n'ayant plus qu'une seule valeur de sorte que K_{10} est finalement retrouvée à un XOR avec une constante près (e.g. $k_{10,0}$).

Une meilleure exploitation des courbes (potentiellement a posteriori) peut être faite par l'utilisation de certaines courbes ayant des collisions au cours des S-Boxs du dernier tour. Lorsque certaines valeurs de $\mu_{10,i,i'}$ sont connues, on peut alors détecter lorsqu'une collision survient entre les S-Boxs reliées aux octets c_i et $c_{i'}$. Si la collision identifiée sur X_{10} était la seule sur cette courbe il est alors possible de récupérer des informations négatives comme précédemment mais sur les couples d'indices (j, j') non impliqués dans la collision.

4.5.2.3 Retrouver la table S-Box

À ce moment de l'attaque nous connaissons K_0 et K_{10}, chacun à un XOR avec une constante près. Nous allons à présent montrer comment retrouver la table S-Box pour chaque candidat à ces deux constantes. On recherche des couples $(x' = x \oplus k_{0,0},\ y' = y \oplus k_{10,0})$ vérifiant $S(x) = y$ comme en Section 4.5.1.3. Dans ce but nous choisissons judicieusement des clairs tels que X_1 contient exactement cinq valeurs différentes $x^{(1)}$, $x^{(2)}$, $x^{(3)}$, $x^{(4)}$ and $x^{(6)}$. Comme décrit dans la Figure 4.12, chaque valeur est répétée un nombre de fois différent, cela permettant, lorsqu'une collision apparaît, d'identifier la valeur qui collisionne en se basant sur le nombre de fois où elle collisionne.

Si une collision apparaît entre une (ou plusieurs) des cinq valeurs d'entrée $x^{(i)}$ et certains octets de X_{10} alors on identifie au moins 240 valeurs y' ne pouvant s'associer à $x' = x^{(i)} \oplus k_{0,0}$. Pour chaque autre $x^{(j)}$ non collisionnant, on peut identifier jusqu'à 16 valeurs y' ne pouvant s'associer à $x' = x^{(j)} \oplus k_{0,0}$. Après chaque exécution on peut propager l'information négative autant que possible. Par exemple si on sait que x' est nécessairement appairée à y' alors on peut invalider tous les couples (x', y'') et (x'', y') où $x'' \neq x'$ et $y'' \neq y'$. Cela pouvant mener à

l'identification d'une nouvelle paire et ainsi de suite.

Le même ensemble $x^{(i)}$ peut être au besoin utilisé à de multiples reprises en changeant simplement la position des éléments. Cependant, nous suggérons une stratégie de choix intelligente des valeurs de $x^{(i)}$, par le choix de $x' = x^{(i)} \oplus k_{0,0}$ ayant le plus petit nombre de candidats à y' invalidés. La raison de ce critère de choix est la maximisation de l'information potentiellement gagnée.

4.5.2.4 Retrouver K et les paramètres du *key schedule*

Ici nous effectuons la même recherche exhaustive parmi 2^{26} candidats qu'en Section 4.5.1.4 (non impactée par la présence des contre-mesures). Cela permet de retrouver K ainsi que les paramètres du *key schedule*.

4.5.2.5 Retrouver la matrice de `MixColumns` et les paramètres de `ShiftRows`

Connaissant K et la table S-Box, nous sommes capables de contrôler complètement les vecteurs X_1 et Y_1. Nous allons donc chiffrer le clair M_0 défini comme $m_i = k_{0,i} \oplus S^{-1}(0)$ pour tout i, induisant donc que $Y_1 = (0, \ldots, 0)$. Pour ce clair de référence la sortie de `MixColumns` est elle aussi intégralement constituée de zéros et nous avons donc $X_2 = K_1$. Par l'analyse de la courbe de cette exécution (ou simplement parce que nous connaissons K_1) nous obtenons le nombre n_0 d'entrées de S-Box du deuxième tour qui collisionnent avec la valeur $S^{-1}(0)$ contenue dans X_1 au premier tour (cf. Figure 4.13).

Pour simplifier le propos nous ne détaillons ici que la méthode permettant de retrouver la première colonne $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ de la matrice de `MixColumns`, le même procédé étant utilisé pour les autres colonnes avec un simple changement des indices. Nous rejouons le clair de référence utilisé ci-dessus où nous ne modifions qu'un seul octet m_{4c} et nous notons alors par $v = y_{1,4c}$ la valeur de la cellule active en sortie de la S-Box correspondante. La colonne numéro $((c - \sigma_0) \bmod 4)$ prend alors la valeur $(v, 0, 0, 0)$ en entrée, et $(\alpha_0 v, \alpha_1 v, \alpha_2 v, \alpha_3 v)$ en sortie de `MixColumns`, toutes les autres colonnes restant inchangées. En supposant que le quadruplet d'octets actifs de X_2 ne contenaient pas $S^{-1}(0)$ dans l'exécution de référence [16], en testant de manière exhaustive v nous pouvons identifier quatre valeurs [17] v_0, v_1, v_2 et v_3 qui induisent plus de n_0 collisions avec $S^{-1}(0)$ comme entrée de S-Box du second tour (cf. Figure 4.14). Comme ces valeurs particulières ont provoqué l'apparition de nouvelles valeurs $S^{-1}(0)$ dans X_2, nous savons que pour une permutation inconnue π de $\{0, 1, 2, 3\}$ les équations suivantes sont vérifiées :

$$\forall\, i = 0, \ldots, 3 \qquad \alpha_i\, v_{\pi(i)} \oplus k_{1,i+4((c-\sigma_0) \bmod 4)} = S^{-1}(0)$$

Pour chaque valeur possible de σ_0 ce système d'équations suggère un ensemble de 24 candidats (un par candidat pour π) pour la colonne de coefficients $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$.

Une fois cette information acquise, nous pouvons changer la valeur de c et répéter le même processus avec la cellule active au sommet d'une autre colonne. Pour chaque valeur possible de σ_0 nous obtenons un nouvel ensemble de 24 candidats pour la colonne de coefficients. Par intersection de ces deux ensembles, et avec une forte probabilité, il ne restera qu'une seule colonne de coefficients candidats pour la bonne valeur de σ_0 et aucune pour les valeurs incorrectes, ce qui nous révèle à la fois $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ et σ_0.

16. Le cas inverse est rare et est facilement détectable par l'observation d'une réduction du nombre d'apparitions de la valeur $S^{-1}(0)$. Dans ce cas, il suffit de modifier c pour déplacer la cellule active sur une autre colonne.

17. Il peut arriver qu'on en identifie moins de quatre lorsque une ou deux d'entre elles créent des collisions multiples sur les valeurs $S^{-1}(0)$. Dans ce cas, ces valeurs v spéciales doivent simplement être comptées autant de fois que le nombre de collisions qu'elles génèrent.

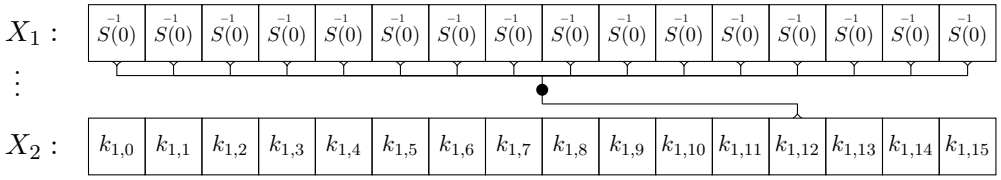

FIGURE 4.13 – Collision révélant que $n_0 = 1$

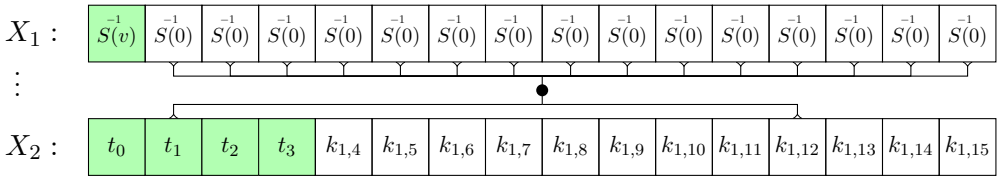

FIGURE 4.14 – Présence d'une collision supplémentaire ($n = 2$) fournissant une information exploitable

Nous pouvons accélérer le processus en ayant deux octets actifs sur la même ligne. Cela permet à l'attaquant de récupérer en même temps huit valeurs v produisant de nouvelles collisions. Le nombre de candidats à la permutations devient 8! (au lieu de $2 \times 4!$) ce qui reste raisonnable.

En répétant cette attaque avec une cellule active placée sur les autres lignes, nous pouvons retrouver successivement les trois autres colonnes de coefficients de la matrice de `MixColumns` ainsi que les trois autres paramètres de `ShiftRows`.

Comme il reste très peu de données inconnues, dans les rares cas où il reste une indécision sur une/des colonne/s de coefficients de `MixColumns` (et potentiellement de paramètres de `ShiftRows`), on peut résoudre cette indécision en testant les quelques candidats à l'AES modifié restants grâce à une paire clair/chiffré connue.

4.5.2.6 Résultats expérimentaux

Nous avons effectué des simulations des différentes étapes de l'attaque décrite dans cette section selon le même procédé que pour la version attaquant une implémentation non protégée décrite en Section 4.5.1.6.

La Table 4.3 détaille le nombre de courbes – en moyenne sur 10 000 exécutions – requises pour chaque étape de l'attaque. Par souci de clarté nous mentionnons aussi l'étape de recherche exhaustive de la Section 4.5.2.4 même si elle ne nécessite aucune courbe supplémentaire.

Au final, moins de 4 000 courbes sont nécessaires pour complètement retrouver l'ensemble des paramètres secrets d'un AES modifié dans le scénario, plus réaliste, d'une implémentation protégée au premier ordre et où on ne connaît rien à propos de la clef. Nous souhaitons insister sur le fait que l'entropie de l'information secrète à retrouver dans le cadre d'une attaque SCARE est souvent bien plus élevée que dans le cadre d'une attaque visant à retrouver une clef. Dans le cas de l'AES modifié, l'entropie totale à retrouver se monte à environ 1 830 bits [18] d'information en plus des 128 bits de clef.

18. Cela prend en compte tous les éléments secrets : S-Box ($\log_2(256!) \simeq 1684$ bits), `ShiftRows` (4 x 2 bits), `MixColumns` (16 x 8 bits), `RotWord` (2 bits), `Rcon` (8 bits).

TABLE 4.3 – Resultats expérimentaux sur une implementation masquée et dont l'ordre des opérations indépendantes a été modifié aléatoirement

Étape	# courbes	# exécutions
Section 4.5.2.1 - Réduire l'entropie de K_0 à 8 bits	246	10 000
Section 4.5.2.2 - Réduire l'entropie de K_{10} à 8 bits	1 408	10 000
Section 4.5.2.3 - Retrouver la S-Box	1 263	10 000
Section 4.5.2.4 - Retrouver K et le *key schedule*	0	–
Section 4.5.2.5 - Retrouver MC et SR	910	10 000
Total	**3 827**	

4.5.3 Extension du paramètre de `Rcon` dans le cadre d'une implémentation non protégée

Notre attaque ciblant une implémentation non protégée décrite en Section 4.5.1 suppose un vecteur de `Rcon` possédant 8 bits d'entropie et constitué de 10 mots : $\texttt{Rcon}[r] = (\rho^{r-1}, 0, 0, 0)$. Deux extensions naturelles seraient de définir un vecteur de 32 bits ou de 320 bits d'entropie donnant des mots pour `Rcon` respectivement égaux à : $(\rho_0^{r-1}, \rho_1^{r-1}, \rho_2^{r-1}, \rho_3^{r-1})$ et $(\rho_{r,0}, \rho_{r,1}, \rho_{r,2}, \rho_{r,3})$ pour $r = 1, \ldots, 10$. Dans le premier cas la recherche exhaustive de la Section 4.5.1.4 nécessite une recherche parmi 2^{50} candidats, ce qui est une tâche lourde à réaliser, et qui devient complètement inatteignable dans le deuxième cas. Dans cette section nous présentons une adaptation de cette étape de l'attaque exploitant de nouvelles collisions afin de retrouver les paramètres de `Rcon` dans ces deux cas étendus. Nous ne détaillons ici que le cas à 320 bits d'entropie, cette méthode pouvant trivialement s'adapter au cas à 32 bits d'entropie.

Nous rappelons que les paramètres de `Rcon` ne sont pas impliqués dans les premières étapes de l'attaque et que celles-ci peuvent donc être réalisées de la même manière pour retrouver K_0 à $k_{0,0}$ près, K_{10} à $k_{10,0}$ près, les paramètres de `ShiftRows` et la table S-Box à $(k_{0,0}, k_{10,0})$ près. Au contraire la recherche exhaustive parmi 2^{26} candidats au *key schedule* de la Section 4.5.1.4 devient maintenant une recherche inatteignable parmi 2^{338} candidats.

4.5.3.1 En version inter-courbes

Cette nouvelle version de cette étape de l'attaque se déroule en trois phases. La première phase utilise 2^8 courbes afin de déterminer une liste de 2^{16} candidats pour K_1. La deuxième phase exploite les contraintes inhérentes au *key schedule* pour révéler le véritable K_1 ainsi que les paramètres $\rho_{1,i}$. Enfin la troisième phase permet de retrouver tous les autres paramètres $\rho_{r,i}$.

Dans la première phase nous utilisons la connaissance des valeurs $\mu_{0,i,j}$ afin de chiffrer des clairs tels que les 16 octets en entrée des S-Boxs du premier tour soient identiques. En conséquence, un de ces 2^8 chiffrements vérifie que Y_1 ne contient que des zéros mais nous ne savons pas lequel car nous ne connaissons pas la valeur $k_{10,0}$. Pour cette exécution particulière il est à noter que la sortie de l'étape `MixColumns` est aussi composée uniquement de zéros et donc que l'état X_2 , entrée des S-Boxs du deuxième tour, est égal à K_1.

Comme nous considérons ici la version inter-courbes, nous sommes capables d'identifier les valeurs de X_2, à la valeur $k_{0,0}$ près, pour chacune des 2^8 courbes. Ceci étant réalisé par le fait de trouver une collision entre chaque $x_{2,j}$ de la courbe avec des valeurs $x_{1,i}$ sur d'autres courbes. Nous obtenons alors : $x_{2,j} = x_{1,i} = m_i \oplus k_{0,i} = (m_i \oplus \mu_{0,0,i}) \oplus k_{0,0}$. En résumé nous avons identifié

2^{16} candidats pour K_1, un par candidat pour le couple $(k_{0,0}, k_{10,0})$.

La seconde phase, ne nécessitant pas de nouvelles courbes, utilise le *key schedule* pour obtenir la clef K_1. Pour chaque candidat pour le couple $(k_{0,0}, k_{10,0})$, le paramètre η et $\rho_{1,0}$ nous sommes capables de calculer $k_{1,0}$, $k_{1,4}$, $k_{1,8}$ et $k_{1,12}$. En résumé, pour chaque candidat pour $(k_{0,0}, k_{10,0})$ nous avons obtenu : i) un candidat pour K_1 durant la première phase et ii) 2^{10} candidats pour un quadruplet d'octets de K_1 dans la deuxième phase. Ces deux données peuvent être confrontées et avec une forte probabilité il n'y aura qu'une seule configuration valide restante nous révélant les bonnes valeurs de $k_{0,0}$, $k_{10,0}$, η, $\rho_{1,0}$ et K_1. À présent l'attaquant connaît K_0, K_{10}, la table S-Box, η , K_1 et $\rho_{1,0}$. Avec cette connaissance il peut calculer $\rho_{1,1}$, $\rho_{1,2}$ et $\rho_{1,3}$ mais aussi appliquer la méthode de la Section 4.5.1.5 afin de retrouver la matrice de `MixColumns`.

La troisième phase utilise des courbes acquises précédemment afin de trouver les paramètres $\rho_{r,i}$ manquants. Nous décrivons ici comment retrouver les paramètres $\{\rho_{2,0}, \rho_{2,1}, \rho_{2,2}, \rho_{2,3}\}$ du deuxième tour, la même méthode pouvant être appliquée successivement aux autres tours. Afin de retrouver ces valeurs nous allons chercher à retrouver la valeur de la clef K_2, car lorsque celle-ci est acquise nous possédons les deux clefs consécutives K_1 et K_2 ce qui nous permettra de simplement calculer les paramètres $\{\rho_{2,0}, \rho_{2,1}, \rho_{2,2}, \rho_{2,3}\}$ utilisés dans la transition de l'une à l'autre.

Comme K_0, K_1, la table S-Box, et les paramètres de `ShiftRows` et `MixColumns` sont connus cela permet de connaître l'état T_2 pour chaque courbe déjà acquise. Nous sommes aussi capables de déterminer toutes les valeurs $x_{3,i}$ d'entrée des S-Box du troisième tour grâce à des collisions avec des entrées de S-Box connues (au premier, deuxième et dernier tours) pouvant provenir d'autres courbes, nous pouvons alors calculer $K_2 = T_2 \oplus X_3$ et par la suite calculer $\{\rho_{2,0}, \rho_{2,1}, \rho_{2,2}, \rho_{2,3}\}$.

Cette même méthode peut être appliquée pour retrouver K_3 à présent que K_2 est connue et ainsi de suite pour chaque tour sauf le dernier. Pour le dernier tour aucune collision supplémentaire n'est nécessaire car K_{10} est directement obtenu à partir de $Z_{10} \oplus C$.

4.5.3.2 En version intra-courbe

Dans le cas où seules les collisions intra-courbe sont disponibles les phases détaillées précédemment doivent être adaptées.

Dans la première phase pour chacun des 2^8 chiffrements, il nous faut à présent obtenir les valeurs des $x_{2,i}$ par des collisions avec le premier tour ce qui est incompatible avec le fait de garder constants ces octets. Nous avons fait le choix de garder constants la moitié des octets (e.g. $x_{2,0}$ à $x_{2,7}$ dans un premier temps). Par modification des octets restants au premier tour nous nous donnons l'opportunité d'observer des collisions avec les octets $x_{2,i}$ pour $i = 0, \ldots, 7$. Une fois les collisions obtenues il suffit d'inverser les rôles et garder $x_{2,8}$ à $x_{2,15}$ constants et ainsi obtenir toutes les valeurs de X_2.

La deuxième phase n'a pas besoin de nouvelles courbes et n'est donc pas impactée.

La troisième phase nécessite de trouver des collisions entre les entrées des S-Box du troisième tour et une entrée d'une S-Box connue. Ces 48 S-Box connues sont au premier, deuxième et dernier tour : X_1, X_2 et X_{10}. Pour chaque courbe et chaque $x_{3,i}$ la probabilité d'une collision est $p = 1 - (1/256)^{48} \approx 0.17$. Nous avons suffisamment de courbes pour retrouver tous les $x_{3,i}$ nous permettant d'inférer la valeur K_2. Pour les tours suivants les choses deviennent de plus en plus faciles car on a obtenu 16 opportunités supplémentaires à chaque nouvelle clef découverte.

Remarque 7. *Un seul octet de K_r par ligne i est suffisant pour obtenir la valeur $\rho_{r,i}$. Par*

exemple si l'on obtient $k_{2,8}$ cette valeur nous autorise à calculer :

$$\begin{aligned} k_{2,4} &= k_{2,8} \oplus k_{1,8} \\ k_{2,0} &= k_{2,4} \oplus k_{1,4} \\ \rho_{r,0} &= k_{2,0} \oplus k_{1,0} \oplus S(k_{1,12+\eta}) \end{aligned}$$

4.5.4 Extension à un masquage d'ordre plus élevé

Nous considérons un schéma de masquage d'ordre plus élevé utilisant 16 tables `SubBytes` masquées, une par S-Box dans un tour. Elle est toujours combinée à la contre-mesure rendant aléatoire l'ordre des opérations indépendantes. À noter que l'attaque contre un masquage d'ordre plus élevé présentée dans cette section suppose que l'attaquant possède *a priori* les valeurs relatives des octets de clef : $\mu_{0,i,j}$.

Tout comme la contre-mesure de masquage au premier ordre vue en Section 4.5.2, cette contre-mesure ne permet plus que de détecter des collisions entre deux S-Boxs dans la même courbe. En revanche elle ajoute la contrainte supplémentaire de ne pouvoir les détecter qu'entre deux S-Boxs à la même position dans des tours différents. Ces collisions sont plus rares mais donnent une information supplémentaire : les octets impliqués sont verticalement alignés. Afin de confirmer qu'une collision est bien réalisée entre deux S-Boxs à des tours différents mais verticalement alignées, nous avons besoin de rejeter les faux positifs cela étant possible en chiffrant le même clair deux fois voire plus. Comme tous les masques d'entrée et de sortie sont indépendants, la probabilité qu'une S-Box i d'un certain tour entre en collision avec une S-Box $i' \neq i$ d'un autre tour est de 2^{-16} car les deux entrées et les deux sorties doivent collisionner. Comme il y a 240 couples (i, i') la probabilité d'un faux positif entre les deux tours concernés est d'environ 2^{-8}. Donc, si une collision se conserve après plusieurs chiffrements du même clair – deux chiffrements devraient être suffisants dans la majorité des cas – la probabilité est alors forte que ce soit une véritable collision entre des valeurs alignées verticalement et donc masquées par le même couple de masques entrée/sortie.

Dans le reste de la Section 4.5.4 nous utiliserons le terme simple de *collision* pour exprimer une collision confirmée comme étant effectivement une vraie collision alignée par un rejeu multiple du même clair.

4.5.4.1 Retrouver les paramètres de `ShiftRows`

Dans cette étape nous chiffrons des clairs aléatoires jusqu'à obtenir une collision entre le premier et le deuxième tour. Nous chiffrons alors ce clair de référence où un seul octet sera modifié à la fois et nous vérifions si la collision se maintient ou disparaît. Nous apprenons ainsi si on est en présence d'une `4-Collision` ou d'une `5-Collision` en comptant le nombre de positions faisant disparaître la collision.

Dans le cas d'une `4-Collision` nous apprenons que les quatre octets actifs sont sur la même colonne après l'étape `ShiftRows`, en revanche nous ne savons pas quelle colonne. Nous pouvons donc inférer les valeurs relatives des σ_i – que nous appellerons sa silhouette – à une addition modulo 4 près.

Il faut remarquer que dans le cas d'une `4-Collision` nous avons aussi l'information supplémentaire que l'un de ces 4 σ_i doit être égal à zéro. Cela est dû au fait que les collisions sont alignées et que l'un des octets impliqués dans la collision fait aussi partie de la colonne impliquée dans le calcul de l'octet collisionnant au second tour.

Dans le cas d'une 5-`Collision` nous sommes en présence d'une ligne avec deux octets actifs et de trois lignes ne contenant qu'un seul octet actif. De ces trois dernières nous pouvons inférer les valeurs relatives des trois σ_i concernés. Un des deux éléments actifs sur la ligne restante est impliqué dans le calcul de l'octet collisionnant du second tour, et est donc aligné avec les octets actifs des trois autres lignes après `ShiftRows`. L'autre est quand à lui aligné verticalement avec l'octet collisionnant du second tour et ne fait pas partie de la colonne recherchée. Nous n'avons pas la capacité de distinguer le rôle de ces deux octets, en revanche dans chacune des deux hypothèses possibles nous connaissons la position de la collision ainsi que les quatre éléments impliquées dans le calcul de l'octet collisionnant ce qui nous donne une valeur non relative des paramètres σ_i.

La procédure permettant d'identifier précisément les paramètres de `ShiftRows` consiste en la génération d'un certain nombre de collisions premier/deuxième tour jusqu'à l'obtention de deux 5-`Collisions` ou bien jusqu'à l'obtention d'une 4-`Collision` et une 5-`Collision`.

Dans le cas où deux 5-`Collisions` sont obtenues nous identifions la bonne solution par intersection des paires de solutions données par chaque collision.

Dans le cas où une 4-`Collision` et une 5-`Collision` sont obtenues nous pouvons utiliser la silhouette obtenue par la 4-`Collision` afin de déterminer laquelle des deux solutions proposées par la 5-`Collision` est valide.

Un cas particulier peut survenir si tous les σ_i sont égaux à zéro cela aura pour effet que seules des 4-`Collisions` peuvent apparaître. Cette particularité sera identifiée dès la première collision car tous les σ_i sont égaux (information révélée par la silhouette) et comme c'est une 4-`Collision` nous avons vu que cela impliquait qu'au moins un des σ_i était nul, ils sont donc tous les quatre nuls.

4.5.4.2 Retrouver K_{10} à un XOR avec un octet constant près

Nous cherchons à présent des collisions entre le premier et le dernier tour mais la contre-mesure rendant aléatoire l'ordre d'exécution des S-Boxs nous empêche d'en déterminer la position exacte. Nous contournons ce problème par la génération de collisions triples impliquant les premier, deuxième et dernier tours à partir d'une collision double impliquant premier et deuxième tours. Afin d'obtenir de telles collisions nous obtenons dans un premier temps une collision entre premier et deuxième tour, à position connue grâce à la connaissance des paramètres de `ShiftRows`. Dans un second temps nous maintenons cette collision en gardant inchangés les 4 ou 5 octets impliqués dans son calcul et modifions tous les autres octets jusqu'à l'obtention d'une collision triple avec le dernier tour.

De cette collision triple nous pouvons conclure que les S-Boxs impliquées au premier et dernier tours ont les mêmes entrées et sorties. De plus nous obtenons l'équation $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$ où i est la position de la collision qui est connue, j est déduit de i et des paramètres de `ShiftRows`, et où seuls $k_{0,i}$ et $k_{10,j}$ sont inconnus.

Afin de retrouver K_{10} à une constante près nous utilisions deux équations provenant de triples collisions. Une première en position i nous donne un couple (m_i, c_j) vérifiant $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$. Une seconde à une position quelconque $i' \neq i$ nous apporte un deuxième couple $(m_{i'}, c_{j'})$ vérifiant une équation similaire.

La façon d'obtenir une seconde collision triple utile sera en fixant m'_i à la valeur $m_i \oplus \mu_{0,i,i'}$ (impliquant que $x'_i = x_i$), de changer les octets impliqués dans le calcul de $x_{2,i'}$ jusqu'à obtenir une collision double entre premier et deuxième tour. De la même manière que précédemment nous conservons cette collision et changeons les octets non impliqués jusqu'à obtenir la triple

collision attendue. Comme $x'_i = x_i$, nous obtenons que $c_j \oplus k_{10,j} = c_{j'} \oplus k_{10,j'}$ ce qui révèle $\mu_{10,j,j'} = c_j \oplus c_{j'}$.

4.5.4.3 Retrouver la table S-Box à deux octets près

Dans l'étape précédente nous avons détaillé une méthode pour provoquer une triple collision menant à une équation du type $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$. Chacune de ces équations révèle un couple (x, y), où $x = m_i \oplus \mu_{0,0,i}$ et $y = c_j \oplus \mu_{10,0,j}$, ce couple vérifiant $S(x \oplus k_{0,0}) = y \oplus k_{10,0}$. L'accumulation de 255 équations indépendantes de ce type révélera la table S-Box à $(k_{0,0}, k_{10,0})$ près.

4.5.4.4 Retrouver K et les paramètres du *key schedule*

Cette étape de l'attaque est effectuée sans utiliser les courbes et peut donc être réalisée telle que décrite en Section 4.5.1.4.

Après cette étape les seuls paramètres restant inconnus sont ceux de la matrice de `MixColumns`.

4.5.4.5 Retrouver la matrice de `MixColumns`

Pour retrouver les valeurs α_i nous utilisions les courbes où des collision triples avaient été identifiées au cours des étapes précédentes. Pour chacune de ces courbes nous connaissons les valeurs des 4 ou 5 octets de sortie des S-Box du premier tour et l'octet d'entrée du second tour impliqués dans la collision. Ces valeurs sont reliées par une équation faisant intervenir quatre valeurs α_i inconnues. En combinant quatre équations indépendantes nous pouvons résoudre le système et obtenir ainsi quatre paramètres α_i. En résolvant quatre systèmes nous obtenons les seize paramètres α_i.

Si cela s'avère nécessaire nous pouvons générer de nouvelles collisions triples ayant des positions choisies afin de générer les équations manquantes.

4.6 Attaque SCARE utilisant le modèle de collisions en poids de Hamming

Nous rappelons que dans le modèle en poids de Hamming l'attaquant est capable de détecter si deux appels à la table S-Box $y = S(x)$ et $y' = S(x')$ vérifient simultanément $HW(x) = HW(x')$ et $HW(y) = HW(y')$. Dans cette section nous détaillons comment retrouver les paramètres secrets d'une implémentation de l'AES ne comportant aucune contre-mesure contre les attaques par canaux auxiliaires. La méthode détaillée dessous se place dans la configuration plus contrainte de l'intra-courbe. Nous procédons étape par étape, et l'ordre de ces étapes est important car chacune d'entre elles dépend des informations retrouvées pendant les précédentes. À la fin de cette section nous étudions comment s'adapter à une implémentation protégée par un masquage au premier ordre.

4.6.1 Définitions

Définition 3. *Pour $0 \leqslant a, b \leqslant 8$, nous définissons les ensembles $\Psi_{a,b}$ et $\Omega_{a,b}$ comme suit :*

$$\Psi_{a,b} = \{x \mid \mathrm{HW}(x) = a, \mathrm{HW}(S(x)) = b\}$$
$$\Omega_{a,b} = \{y \mid \mathrm{HW}(S^{-1}(y)) = a, \mathrm{HW}(y) = b\}$$

$\lvert\Psi_{a,b}\rvert$	b 0	1	2	3	4	5	6	7	8	
0	0	0	0	0	1	0	0	0	0	1
1	0	0	2	0	1	3	2	0	0	8
2	0	2	3	8	5	4	4	2	0	28
3	1	1	4	17	16	10	5	2	0	56
a 4	0	3	9	11	21	16	9	1	0	70
5	0	1	7	10	19	14	3	2	0	56
6	0	0	3	7	5	8	4	0	1	28
7	0	1	0	2	2	1	1	1	0	8
8	0	0	0	1	0	0	0	0	0	1
	1	8	28	56	70	56	28	8	1	

$\sum_{b=0}^{8} \lvert\Psi_{a,b}\rvert$ (colonne)

$\sum_{a=0}^{8} \lvert\Psi_{a,b}\rvert$ (ligne)

$\lvert\Psi_{a,b}\rvert$	#	$\lvert\Psi_{a,b}\rvert$	#
0	34	9	2
1	12	10	2
2	8	11	1
3	5	14	1
4	4	16	2
5	3	17	1
7	2	19	1
8	2	21	1

FIGURE 4.15 – Répartition des $\lvert\Psi_{a,b}\rvert$ (à gauche) et leur nombre d'apparitions (à droite) pour la table S-Box de l'AES standard

Deux valeurs d'octets appartenant au même ensemble $\Psi_{a,b}$ (respectivement le même $\Omega_{a,b}$) sont des entrées (respectivement des sorties) de deux S-Box qui collisionneront dans le cadre du modèle de collisions en poids de Hamming défini en Section 4.3.2. On peut remarquer que $\lvert\Psi_{a,b}\rvert = \lvert\Omega_{a,b}\rvert$ car S est une permutation. À titre d'exemple la Figure 4.15 présente la répartition des cardinaux des ensembles Ψ de la table S-Box de l'AES standard ainsi que le nombre d'apparitions de chaque cardial.

4.6.2 Considérations sur les ensembles Ψ et Ω

Avant de décrire l'attaque nous commençons ici par certaines considérations à propos des ensembles Ψ et Ω expliquant comment des informations sont obtenues à leur propos et comment elles sont utilisées au cours de l'attaque.

Dans certaines étapes de l'attaque décrite ci-après nous utilisons des ensembles particuliers vérifiant $\lvert\Psi_{a,b}\rvert = 1$. Grâce à l'unicité de la valeur dans cet ensemble, une collision impliquant deux S-Boxs correspondant à un tel ensemble révèle que leurs entrées et sorties sont en fait égales au lieu de simplement partager les même poids de Hamming. Par exemple il existe 12 ensembles de ce type dans l'AES standard.

Lemme 6. *Il existe au moins deux ensembles Ψ tels que $\lvert\Psi_{a,b}\rvert = 1$*

Démonstration. Comme 0 (respectivement 255) est la seule valeur ayant un poids de Hamming à 0 (respectivement 8) nous avons $\lvert\Psi_{0,\mathrm{HW}(S(0))}\rvert = \lvert\Psi_{8,\mathrm{HW}(S(255))}\rvert = 1$ □

Dans certaines étapes de l'attaque nous aurons besoin de tester exhaustivement des entrées de S-Box jusqu'à provoquer une collision, mais nous pouvons réduire le nombre de valeurs à tester. Pour cela nous utilisons le fait que tous les éléments d'un ensemble Ψ partagent le même comportement en terme de collisions/non-collisions, cela implique que nous n'avons alors qu'à tester une seule valeur par ensemble Ψ.

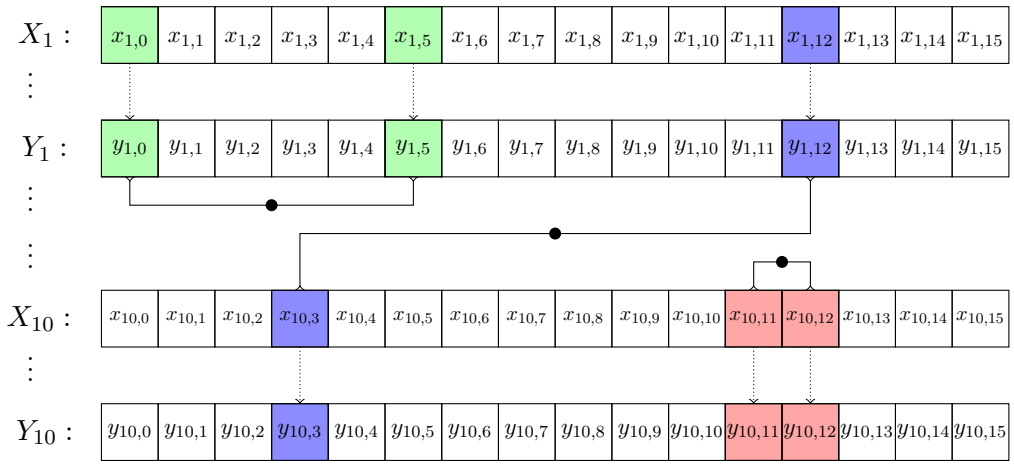

FIGURE 4.16 – Exemples de collisions révélant des informations sur les ensembles Ψ (gris clair/vert), Ω (gris moyen/rouge) ou les liens (gris foncé/bleu)

À une certaine étape de l'attaque nous aurons besoin d'avoir identifié les ensembles Ψ. Cela signifie que nous aurons besoin d'avoir partitionné les 256 entrées de S-Box dans les ensembles $\Psi_{a,b}$. À noter que pour chaque ensemble $\Psi_{a,b}$ nous connaissons le poids de Hamming a mais potentiellement pas le poids de Hamming b de leurs images à travers S.

De manière symétrique nous aurons aussi besoin d'avoir identifié les ensembles Ω signifiant avoir une partition des sorties des S-Boxs dans les ensembles $\Omega_{a,b}$. Dans ce cas nous connaissons les poids de Hamming b mais potentiellement pas le poids de Hamming a de leurs pré-images à travers S.

Il existe une bijection entre les ensembles des $\Psi_{a,b}$ et ceux des $\Omega_{a,b}$ que nous aurons aussi à identifier. Pour chaque $\Psi_{a,b}$ (dont le b est inconnu) nous avons besoin d'identifier son ensemble $\Omega_{a,b}$ compagnon (dont le a est inconnu). Lorsqu'un tel lien est identifié les valeurs a et b sont alors connues car les deux ensembles Ψ et Ω compagnons partagent les mêmes valeurs a et b. À noter qu'une fois cette bijection des ensembles compagnons identifiée nous avons l'information que les y d'un ensemble $\Omega_{a,b}$ sont des images à travers S de tous les x de son $\Psi_{a,b}$ compagnon, en revanche nous ne sommes pas forcément capables de relier les valeurs entre elles.

La Figure 4.16 présente les trois types d'informations que nous exploitons sur chaque courbe obtenue. Après que K_0 soit retrouvée, une collision au premier tour nous permet de rassembler les deux entrées de S-Box concernées dans le même ensemble $\Psi_{a,b}$. Par exemple dans le cas de la collision en gris clair/vert nous pouvons mettre les valeurs de $x_{1,0}$ et $x_{1,5}$ dans le même $\Psi_{a,b}$ où $a = \mathrm{HW}(x_{1,0})$ et où b est *a priori* inconnu. De la même manière, avec la connaissance de K_{10} et des paramètres de `ShiftRows`, nous pouvons placer $y_{10,11}$ et $y_{10,12}$ dans le même ensemble $\Omega_{a,b}$ – a inconnu et $b = \mathrm{HW}(y_{10,11})$ – grâce à l'observation de la collision gris moyen/rouge au dernier tour. Enfin si nous connaissons K_0, K_{10} et les paramètres de `ShiftRows`, il nous est alors possible d'obtenir, à partir d'une collision entre le premier et le dernier tour, telle que la gris foncée/bleue, l'information que l'ensemble Ψ contenant $x_{1,12}$ est le compagnon de l'ensemble Ω contenant $y_{10,3}$, même au cas où nous n'aurions pas identifié tous les éléments de ces ensembles.

Durant l'attaque chacune des collisions ou non-collisions sont détectées et l'information qu'elles apportent est mémorisée. En effet, chaque i) non-collision au premier tour, ii) non-collision au dernier tour ou iii) non-collision entre le premier et le dernier tour apporte aussi de l'information. Elles informent respectivement que i) les deux entrées de S-Box impliquées

ne peuvent pas faire partie du même ensemble Ψ, (ii) les deux sorties de S-Box impliquées ne peuvent pas faire partie du même ensemble Ω, (iii) que l'entrée de S-Box du premier tour et la sortie de S-Box du dernier tour font respectivement partie d'ensembles Ψ et Ω qui ne peuvent pas être compagnons.

Ces trois types d'information sont utiles durant l'attaque car on peut en déduire que :

— si une valeur particulière est dans un ensemble de cardinal 1, alors chaque collision impliquant cet élément est en réalité une collision de valeurs et pas seulement de poids de Hamming,

— si deux ensembles $\Psi_{a,b}$ et $\Omega_{a,b}$ sont identifiés comme étant compagnons, et si nous identifions une entrée (respectivement sortie) de S-Box comme faisant partie de $\Psi_{a,b}$ (respectivement $\Omega_{a,b}$), alors la sortie (respectivement entrée) de cette S-Box fait partie d'un ensemble réduit de candidats : les éléments de $\Omega_{a,b}$ (respectivement $\Psi_{a,b}$).

Le pré-requis pour commencer à construire les ensembles Ψ est la connaissance de K_0 car nous devons connaître les valeurs $x_{1,i}$ impliquées dans la collision. Nous pouvons toutefois commencer si nous connaissons K_0 à son complément près – incertitude entre K_0 et $\overline{K_0}$ – car dans ce cas nous connaissons aussi les valeurs $x_{1,i}$ à leur complément près. Dans cette configuration nous pouvons choisir arbitrairement un des deux candidats pour K_0 et commencer la création des ensembles Ψ sous cette supposition. Si ce choix est infirmé par la suite, il nous suffit alors de changer chacun des $\Psi_{a,b}$ déjà construits en leur complément $\Psi_{8-a,b}$ qui contient le complément de tous ses éléments. Par souci de clarté dans la description de l'attaque qui suit nous ne mentionnerons pas cette permutation potentielle qui ne perturbe aucunement les étapes de notre attaque.

De la même manière le pré-requis pour construire les ensembles Ω est la connaissance de K_{10} et des paramètres de `ShiftRows`, et ceux pour débuter l'identification d'ensembles compagnons nécessite K_0, K_{10} et les paramètres de `ShiftRows`. Il doit être remarqué que le raisonnement précédent – lié à l'incertitude entre une clef et son inverse – est aussi valide pour la construction des ensembles Ω (si on ne connaît K_{10} qu'à son complément près) ainsi que pour les liens entre les ensembles Ψ et Ω.

4.6.3 Description de l'attaque

4.6.3.1 Retrouver K_0 à 2^{16} candidats près

Dans la première étape nous souhaitons retrouver chaque $k_{0,i}$ à son complément près ce qui mènera à 2^{16} candidats pour K_0.

Considérons deux courbes où une collision est apparue au premier tour entre les positions i et j, pour les octets de clair m_i et m_j, sur une courbe et aux mêmes positions i et j sur la seconde mais pour les valeurs de clair $m'_i \neq m_i$ et $m'_j = m_j$. Nous avons alors $x_j = x'_j$ ce qui fait que nous pouvons inférer l'information $\mathrm{HW}(x_i) = \mathrm{HW}(x'_i)$. Les valeurs de m_i et m'_i nous permettent alors d'invalider toutes les valeurs pour $k_{0,i}$ qui ne vérifient pas $\mathrm{HW}(m_i \oplus k_{0,i}) = \mathrm{HW}(m'_i \oplus k_{0,i})$.

Nous accumulons de telles contraintes pour chaque octet de clef jusqu'à réduire les candidats à une liste de deux valeurs complémentées pour chaque $k_{0,i}$. En effet $k_{0,i}$ et $\overline{k_{0,i}}$ ne peuvent pas être différenciées car :

$$\begin{aligned} \mathrm{HW}(m_i \oplus k_{0,i}) = \mathrm{HW}(m'_i \oplus k_{0,i}) \quad &\Leftrightarrow \quad 8 - \mathrm{HW}(m_i \oplus k_{0,i}) = 8 - \mathrm{HW}(m'_i \oplus k_{0,i}) \\ &\Leftrightarrow \quad \mathrm{HW}(m_i \oplus k_{0,i} \oplus 255) = \mathrm{HW}(m'_i \oplus k_{0,i} \oplus 255) \\ &\Leftrightarrow \quad \mathrm{HW}(m_i \oplus \overline{k_{0,i}}) = \mathrm{HW}(m'_i \oplus \overline{k_{0,i}}) \end{aligned}$$

Dans le but de provoquer l'apparition de telles collisions nous fixons une moitié du clair à des valeurs arbitraires pendant que la seconde moité est choisie aléatoirement. En faisant cela nous

augmentons la probabilité que deux courbes fassent apparaître une paire de collisions valides ayant des indices i et j appartenant respectivement à la partie libre et à la partie fixe. Lorsque tous les octets de clef à des positions libres ont été retrouvés, nous procédons de la même manière après avoir inter-changé les rôles des moitiés fixées et libres.

4.6.3.2 Retrouver K_0 à deux candidats près

Dans cette étape nous allons déterminer K_0 à son complément près avec l'utilisation de seulement une nouvelle courbe. Comme chaque $k_{0,i}$ est connu à son complément près nous sommes capables de chiffrer un clair contenant $m_i \in \left\{k_{0,i}, \overline{k_{0,i}}\right\}$ impliquant que $m_i \oplus k_{0,i} = x_{1,i} \in \{0, 255\}$. Les positions des collisions permettent d'identifier les deux ensembles d'indices, un où $x_{1,i} = 0$ et l'autre où $x_{1,i} = 255$ même si nous ne savons pas lequel est lequel. En complémentant les octets de clair d'un de ces deux ensembles nous obtenons un vecteur de 16 octets qui est égal soit à K_0 soit à $\overline{K_0}$, duquel nous dérivons toutes les valeurs $\mu_{0,i,j}$.

La connaissance de K_0 à son complément près permet le début de la construction des ensembles Ψ comme détaillé en Section 4.6.2.

4.6.3.3 Retrouver les paramètres de `ShiftRows`

Dans cette étape nous cherchons une courbe particulière, parmi celles déjà acquises, comportant une collision entre le premier et le deuxième tour et dont la valeur d'entrée de S-Box au premier tour peut être identifiée comme faisant partie d'un ensemble $\Psi_{a,b}$ de cardinal 1. Cela permet de savoir que les deux valeurs impliquées dans la collision sont égales. Si nous ne trouvons pas de telles configurations dans une courbe déjà acquise nous en générons une.

À ce niveau d'information nous sommes capables de retrouver les paramètres de `ShiftRows` par la même méthode que décrite en Section 4.5.1.1 en se basant sur l'analyse des 4-`Collisions` et des 5-`Collisions`.

4.6.3.4 Retrouver K_{10} à 2^{16} candidats près

Connaissant les paramètres de `ShiftRows`, nous sommes capables de relier les positions de S-Boxs collisionnantes du dernier tour avec un octet de chiffré. Nous utilisons alors la même méthode que précédemment pour K_0 en Section 4.6.3.1 mais appliquée cette fois-ci sur les sorties des S-Boxs du dernier tour. Nous exploitons des paires de courbes possédant une collision entre $y_{10,i}$ et $y_{10,j}$ sur une courbe et entre $y'_{10,i}$ et le même $y_{10,j}$ sur la seconde. De cette manière nous retrouvons progressivement chaque octet $k_{10,i}$ à son complément près.

La différence principale avec la méthode pour K_0 est que nous ne pouvons pas fixer une partie des sorties des S-Boxs du dernier tour, mais ceci est contrebalancé par le fait que nous recherchons non seulement des collisions à l'intérieur du dernier tour mais aussi des collisions entre le premier et le dernier tour. En effet une entrée de S-Box $x_{1,j}$ peut jouer le même rôle de référence que $y_{10,j}$. L'utilisation des courbes déjà acquises fait que cette étape ne nécessite presque jamais d'acquérir de nouvelles courbes.

4.6.3.5 Retrouver K_{10} à deux candidats près

Nous connaissons les $k_{10,i}$ à leur complément près. Contrairement à K_0 nous n'avons pas besoin d'une nouvelle courbe pour retrouver K_{10} à son inverse près car les collisions entre premier et dernier tours nous donne suffisamment d'information. Considérons deux courbes (potentiellement la même) possédant toutes deux une collision entre premier et dernier tours et telles que

les entrées des deux S-Boxs du premier tour aient été identifiées comme faisant partie du même ensemble Ψ. Cela implique que les S-Boxs du dernier tour ont le même poids de Hamming et donc que $\mathrm{HW}(c_i \oplus k_{10,i}) = \mathrm{HW}(c_j \oplus k_{10,j})$ où c_i et c_j sont les octets de chiffré reliés à ces collisions. Si le poids de Hamming est différent de 4 nous pouvons alors différencier si le candidat à la paire $(k_{10,i}, k_{10,j})$ comporte des valeurs : (i) non-inversés ou inversés tous les deux, ou (ii) une inversée et l'autre non :

$$(i)\begin{cases} \mathrm{HW}(c_i \oplus k_{10,i}) = \mathrm{HW}(c_j \oplus k_{10,j}) \\ \mathrm{HW}(c_i \oplus \overline{k_{10,i}}) = \mathrm{HW}(c_j \oplus \overline{k_{10,j}}) \end{cases} \qquad (ii)\begin{cases} \mathrm{HW}(c_i \oplus k_{10,i}) = 8 - \mathrm{HW}(c_j \oplus \overline{k_{10,j}}) \\ \mathrm{HW}(c_i \oplus \overline{k_{10,i}}) = 8 - \mathrm{HW}(c_j \oplus k_{10,j}) \end{cases}$$

La connaissance de K_{10} à son complément près et des paramètres de `ShiftRows` nous permet de commencer la construction des ensembles Ω tels que définis en Section 4.6.2.

4.6.3.6 Identifier les ensembles Ψ et Ω ainsi que les liens Ψ-Ω

Les étapes suivantes nécessitent que nous ayons identifié les ensembles Ψ et Ω ainsi que les liens rassemblant un ensemble Ψ avec son ensemble Ω compagnon. Nous commençons par extraire le maximum d'information des courbes déjà acquises grâce aux trois types de collisions décrites en Section 4.6.2. Nous avons donc à générer de nouvelles courbes optimisées dans le but de réduire le plus vite possible le nombre d'informations manquantes.

En premier lieu nous finissons d'identifier des ensembles Ψ par le chiffrement de clairs ayant des octets choisis pour maximiser le nombre de couples $(x_{1,i}, x_{1,j})$ comportant des valeurs pour lesquelles il n'est pas encore déterminé si elles collisionnent ou non.

Par la suite nous choisissons des octets de clair afin de maximiser le nombre de valeurs $x_{1,i}$ faisant partie d'ensembles Ψ n'ayant pas encore été reliées à leur compagnon. Nous appliquons cela jusqu'à ce que toute l'information soit retrouvée à propos des ensembles Ω et des liens Ψ-Ω. Nous utilisons autant que possible les relations qui existent entre les ensembles afin d'accélérer le processus de récupération d'information.

4.6.3.7 Retrouver K_1

Afin de retrouver les paramètres des opérations `MixColumns` et `SubBytes` dans les deux étapes suivantes nous allons avoir besoin de connaître la valeur de K_1.

Au premier tour, la constante de `Rcon` est toujours $\rho^0 = 1$. Cela implique que K_1 ne dépend que de K_0, η, $S(k_{0,12})$, $S(k_{0,13})$, $S(k_{0,14})$ et $S(k_{0,15})$:

$$\begin{aligned} k_{1,0} &= S(k_{0,12+(0+\eta) \bmod 4}) \oplus k_{0,0} \oplus \rho^0 & \qquad k_{1,2} &= S(k_{0,12+(2+\eta) \bmod 4}) \oplus k_{0,2} \\ k_{1,1} &= S(k_{0,12+(1+\eta) \bmod 4}) \oplus k_{0,1} & \qquad k_{1,3} &= S(k_{0,12+(3+\eta) \bmod 4}) \oplus k_{0,3} \end{aligned}$$

$$k_{1,i} = k_{1,i-4} \oplus k_{0,i} \text{ pour tout } i = 4, \ldots, 15$$

Dans une première phase ne nécessitant pas de nouvelles courbes, nous établissons une liste de candidats possibles pour $(k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3})$. Si nous connaissons K_0, K_1 ne dépend alors que de ces quatre octets. Pour chacun des 16 candidats pour (K_0, K_{10}, η) nous identifions chacun des ensembles Ψ contenant respectivement $k_{0,12}$, $k_{0,13}$, $k_{0,14}$ et $k_{0,15}$. Leurs ensembles Ω compagnons sont les listes de candidats respectifs à chaque sortie de S-Box impliquée dans les équations ci-dessus. Nous pouvons alors établir une liste de candidats au quadruplet $(k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3})$ pour chaque supposition sur (K_0, K_{10}, η).

La seconde phase utilise la connaissance acquise à propos des ensembles Ψ et Ω. Pour chacun des deux candidats de K_{10} nous obtenons la connaissance de la valeur $S^{-1}(0)$, donc pour chacun

des deux candidats de K_0 nous sommes capables de chiffrer un clair produisant la valeur 0 en sortie de n'importe laquelle des S-Boxs du premier tour. Si nous choisissons de forcer à zéro une colonne entière en entrée de `MixColumns` alors la colonne correspondante en entrée de S-Box du deuxième tour aura la valeur des octets de K_1 correspondants.

Une collision entre premier et deuxième tours à la bonne position permet alors d'identifier l'ensemble Ψ contenant l'octet de K_1. Nous pouvons ainsi confronter les informations et réduire la liste de candidats au quadruplet d'octets $\{k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3}\}$. En effet la connaissance de K_0 nous permet de transformer une liste de candidats pour $k_{1,i}$ en liste de candidats pour $k_{1,i \bmod 4}$.

Ce processus de réduction mène à une seule valeur K_1 dans 96.7% des cas. Lorsque plusieurs K_1 restent candidats [19] nous scindons l'attaque en plusieurs branches : pour chaque K_1 possible nous effectuons les étapes suivantes de l'attaque où nous trouverons des incohérences, invalidant les mauvaises valeurs de K_1. Si malgré tout plusieurs valeurs de K_1 restent valides à la fin de l'attaque, il nous reste alors à tester chaque spécification d'AES obtenue grâce à un couple clair/chiffré.

En plus de K_1 cette étape permet de retrouver K_0, K_{10}, η et jusqu'à quatre paires (x, y) vérifiant $S(x) = y$.

4.6.3.8 Retrouver les valeurs α

Nous avons pour objectif de retrouver les seize coefficients α_i de la matrice de `MixColumns`. Dans cet objectif nous utilisons quatre fois une méthode nous permettant de retrouver quatre d'entre eux.

La connaissance de K_0 et K_{10} nous permet d'identifier tous les couples (x, y) vérifiant $S(x) = y$ où x et y sont dans des ensembles Ψ et Ω ne contenant qu'un seul élément, dont en particulier les valeurs $S^{-1}(0)$ et 0. Nous pouvons ainsi chiffrer des clairs tels que un vecteur appelé ci-après vecteur-V $(V, 0, 0, 0)$ soit forcé en entrée d'une des multiplications matricielles, avec V une valeur non nulle. Ce vecteur-V donne une colonne de sortie égale à $(\alpha_0 * V, \alpha_1 * V, \alpha_2 * V, \alpha_3 * V)$. Ceci implique que les entrées des S-Boxs correspondantes au deuxième tour vont être égales à :

$$x_{2,i} = (\alpha_j * V) \oplus k_{1,i} \;\Rightarrow\; \alpha_j = \frac{x_{2,i} \oplus k_{1,i}}{V}$$

où seulement les valeurs de α_j et $x_{2,i}$ sont inconnues. Une collision entre le premier et le deuxième tours peut déterminer quel ensemble $\Psi_{a,b}$ contient la valeur $x_{2,i}$, nous obtenons donc $|\Psi_{a,b}|$ candidats pour la valeur $x_{2,i}$, et par là même pour α_j. En appliquant cette méthode avec d'autres valeurs V nous pouvons croiser les ensembles de candidats pour chacun des α_j jusqu'à ce qu'il ne reste qu'une seule valeur possible.

La position de la valeur V dans le vecteur-V détermine quel sous-ensemble de valeurs α_i est impliqué dans les équations. Si V est en position c cela signifie que nous ciblons la colonne numéro c de la matrice de coefficients : $(\alpha_{4*c+0}, \alpha_{4*c+1}, \alpha_{4*c+2}, \alpha_{4*c+3})$.

Afin d'accélérer le processus de récupération des valeurs des α_i, nous avons choisi de placer des vecteurs-V en entrée de deux multiplications matricielles et nous gardons libres les huit octets non impliqués afin de provoquer les collisions nécessaires.

4.6.3.9 Retrouver la table S-Box

Avec la connaissance des coefficients α acquise nous sommes capables de compléter la connaissance sur la table S-Box. Nous cherchons dans les courbes disponibles (ou nous en créons de

19. Dans ce cas il reste seulement 2.1 candidats en moyenne.

nouvelles si besoin) celles où un seul des quatre octets d'une colonne d'entrée de multiplication matricielle est inconnu. Prenons l'exemple du cas où nous avons (W, V_1, V_2, V_3) avec W comme seule inconnue. Alors on sait que $x_{2,i} = \alpha_0 * W \oplus \alpha_4 * V_1 \oplus \alpha_8 * V_2 \oplus \alpha_{12} * V_3 \oplus k_{1,i}$ où seul W et $x_{2,i}$ sont inconnus. Nous utilisons la même méthode que dans l'étape précédente, à savoir : une collision entre $x_{2,i}$ et un octet du premier tour nous informe sur quel est l'ensemble Ψ auquel cette valeur appartient, et donc nous fournit une liste de candidats pour W.

On peut remarquer que nous sommes dans une position plus favorable que lorsque nous cherchions les valeurs α car ici pour chaque valeur x nous savons déjà quel ensemble Ω contient la valeur $y = S(x)$, nous pouvons commencer les intersections d'ensembles dès la première collision trouvée au lieu de devoir attendre d'en avoir une deuxième. Nous pouvons de plus ne considérer comme candidats pour y que les seuls éléments de l'ensemble Ω n'ayant pas déjà été reliés à une valeur x.

Une autre remarque peut être faite : lorsqu'il ne reste qu'un seul élément non associé dans un ensemble Ψ son image à travers la table S-Box est nécessairement la dernière valeur non associée dans son ensemble Ω compagnon. Nous utilisons cette propriété afin de gagner de l'information supplémentaire pendant l'analyse de courbes déjà acquises afin de choisir judicieusement les chiffrés dans le cas où il est nécessaire de générer de nouvelles courbes.

Une fois que la pré-image est trouvée pour une valeur de W nous pouvons vérifier à nouveau toutes les courbes où une colonne possédant W comportait une seule autre valeur inconnue W' : (W', W, V_1, V_2). Cette configuration n'était pas exploitable à cause des deux inconnues mais vu que nous venons de déterminer la valeur de W, cette colonne devient exploitable afin de retrouver W'.

Cette étape ne nécessite de nouvelles courbes que dans de très rares cas.

4.6.3.10 Retrouver ρ

Le seul paramètre restant inconnu est la constante ρ. Afin de trouver cette valeur nous exécutons (sans nécessité d'obtenir de nouvelles courbes) une version du *key schedule* pour chaque candidat à la valeur ρ. Seul le bon candidat à la valeur ρ va produire le vrai K_{10}, que l'on connaît, en sortie du *key schedule*.

4.6.4 Résultats expérimentaux

Nous avons suivi la même procédure de simulation que pour les autres attaques. La seule différence réside dans l'oracle qui permet de déterminer si une collision en poids de Hamming a eu lieu au lieu de collisions en valeurs.

La Table 4.4 présente le nombre de courbes – en moyenne sur 10 000 exécutions – requis par chaque étape. Par souci de clarté nous détaillons aussi les étapes n'ayant pas nécessité de courbes supplémentaires.

Notre attaque visant une implémentation non protégée retrouve l'ensemble des paramètres secrets d'un AES modifié ainsi que la clef en moins de 250 courbes en moyenne dans la configuration d'un attaquant intra-courbe. A fortiori, il faudrait significativement moins de courbes si l'attaquant était en configuration inter-courbes.

4.6.5 Extention à une implémentation comportant une contre-mesure de type masquage

Dans la Section 4.6.3 nous avons décrit une attaque SCARE en mode intra-courbe s'appliquant sur une implémentation non protégée dans le modèle de collisions en poids de Hamming.

TABLE 4.4 – Resultats expérimentaux sur une implémentation non protégée dans le modèle de collisions en poids de Hamming

Étape	# courbes	# exécutions
Section 4.6.3.1 - Réduire l'entropie de K_0 à 16 bits	151.9	10 000
Section 4.6.3.2 - Réduire l'entropie de K_0 à 1 bit	1.	10 000
Section 4.6.3.3 - Retrouver `ShiftRows`	11.3	10 000
Section 4.6.3.4 - Réduire l'entropie de K_{10} à 16 bits	0.007	10 000
Section 4.6.3.5 - Réduire l'entropie de K_{10} à 1 bit	0.	10 000
Section 4.6.3.6 - Identifier Ψ, Ω et les liens Ψ-Ω	53.8	10 000
Section 4.6.3.7 - Retrouver K_1	12.3	10 000
Section 4.6.3.8 - Retrouver `MixColumns`	26.2	10 000
Section 4.6.3.9 - Retrouver `SubBytes`	0.2	10 000
Section 4.6.3.10 - Retrouver `Rcon`	0.	10 000
Total	**256.6**	

Dans le cas d'une implémentation protégée par la même contre-mesure de masquage au premier ordre considérée en Section 4.5.2 l'attaque n'est plus applicable à cause de l'effet des masques d'entrée et de sortie qui viennent perturber la détection des collisions en poids de Hamming. En effet de fausses collisions peuvent apparaître lorsque $\mathrm{HW}(x \oplus r_{in}) = \mathrm{HW}(x' \oplus r_{in})$ et $\mathrm{HW}(y \oplus r_{out}) = \mathrm{HW}(y' \oplus r_{out})$ alors que $\mathrm{HW}(x) \neq \mathrm{HW}(x')$ et $\mathrm{HW}(y) \neq \mathrm{HW}(y')$. De la même manière, de vraies collisions peuvent ne pas être détectées lorsqu'on a $\mathrm{HW}(x \oplus r_{in}) \neq \mathrm{HW}(x' \oplus r_{in})$ ou $\mathrm{HW}(y \oplus r_{out}) \neq \mathrm{HW}(y' \oplus r_{out})$ alors que $\mathrm{HW}(x) = \mathrm{HW}(x')$ et $\mathrm{HW}(y) = \mathrm{HW}(y')$.

Un moyen de contourner ces cas de mauvaises détections est de chiffrer à plusieurs reprises le même clair et ainsi détecter les collisions qui persistent sur toutes les tentatives. Cette procédure permet effectivement de ne pas considérer les cas de faux positifs, mais l'effet secondaire de celle-ci est qu'elle ne considère pas non plus les collisions qui ne sont pas des collisions en valeur. En fait, seules les collisions de valeurs ($x = x'$ and $y = y'$) restent détectables après de multiples chiffrements avec des masques différents.

L'oracle de collision en poids de Hamming devient donc un oracle de collisions en valeur lorsqu'on l'applique sur de multiples exécutions masquées. Nous pouvons tirer avantage de cela et appliquer l'attaque de la Section 4.5.2. Dans ce cas là, le nombre de courbes requises pour cette attaque est multipliée par le nombre de fois que chaque clair est chiffré. Répéter chaque chiffrement 5 fois devrait être une borne suffisante pour exclure tous les faux positifs et mène à une attaque requérant environ 20 milliers de courbes [20].

Nous n'avons pas supposé ci-dessus que l'ordre des opérations étaient changé de manière aléatoire à chaque exécution. Ce scénario rend plus difficile la transformation de l'oracle de collision en poids de Hamming en oracle de collisions en valeur. Nous laissons l'étude de ce cas en problème ouvert.

On peut aussi remarquer que le fait de ne pas considérer la modification aléatoire de l'ordre des étapes va certainement faciliter la méthode d'attaque de la Section 4.5.2 que l'on se propose d'appliquer ici. Une dernière remarque peut être faite sur le fait que la méthode de la Section 4.5.4 – ou une version simplifiée – pourrait certainement s'appliquer dans le cas d'un masquage de

20. En fait ce chiffre est certainement surestimé au vu du fait que dans bien des cas il peut n'y avoir aucune collision entre les tours que nous scrutons. Ces cas là devraient être identifiés rapidement en seulement un ou deux chiffrements.

taille 128 bits. À nouveau nous laissons l'étude de ces idées en tant que problèmes ouverts.

4.7 Recommandations de sécurité en lien avec nos attaques

Toutes les attaques décrites dans ce chapitre supposent des implémentations logicielles d'AES modifiés, exécutées sur un microprocesseur de taille 8 bits.

Une première remarque consiste à dire que l'attaque FIRE, de la Section 4.4, basée sur des IFA peut être mise en défaut simplement par l'emploi d'un masquage booléen du premier ordre. Dans ce cas, l'IFA révèle que la valeur $y' = y \oplus r_{out}$ lue en mémoire était zéro ce qui n'apprend rien à l'attaquant sur la valeur que pourrait prendre y vu que le masque est changeant et choisi aléatoirement.

Considérons à présent les attaques de type SCARE. Dans le modèle plus réaliste de la collision en poids de Hamming nous montrons en Section 4.6 qu'une attaque SCARE est possible malgré la présence de masquage de premier ordre. En Section 4.5.2 nous décrivons aussi une attaque SCARE qui compromet des implémentations où une combinaison de masquage au premier ordre et d'exécution en ordre aléatoire est utilisée dans le modèle de collisions en valeurs. Une extension (probablement coûteuse) de cette attaque en cas de masquage d'ordre plus élevé est présenté en Section 4.5.4 dans le cas où l'attaquant possède, a priori, une connaissance partielle sur la clef. Nos découvertes tendent à limiter la confiance dans la résistance de ce type de contre-mesures à l'encontre des attaques de type SCARE pour les deux modèles de collisions. C'est la raison pour laquelle nous pensons qu'un moyen efficace de rendre inopérantes nos attaques serait de limiter la fuite d'information par les canaux auxiliaires, réduisant, voire supprimant, par là même la possibilité de l'attaquant à détecter des collisions. Cela peut être réalisé par l'insertion d'aléas temporels ou, si le composant le propose, en activant une génération de bruit sur les canaux. De manière évidente une meilleure sécurité serait de combiner ce type de perturbations à la lecture avec des contre-mesures de masquage et d'exécutions en ordre aléatoire.

On peut remarquer que toutes nos attaques seraient difficiles à appliquer sur des implémentations matérielles d'AES modifiés, particulièrement en cas de contre-mesure appelée *dual-rail logic*" qui limite fortement la qualité de lecture des canaux auxiliaires et qui permet de détecter des injections de fautes.

Enfin, comme toutes nos attaques requièrent la capacité de l'attaquant de choisir les clairs, n'importe quelle implémentation ne laissant pas le choix sur les entrées de l'AES serait invulnérable à nos attaques.

4.8 Conclusion

Dans ce chapitre nous avons investigué le problème de retrouver les paramètres secrets d'un AES modifié dérivé de l'AES standard par la modification d'une partie ou de tous les paramètres constants (table S-Box, rotations de `ShiftRows`, matrice de coefficient de `MixColumns` ainsi que la rotation de `RotWord` et la constante de `Rcon`).

Notre étude de la rétro-conception a envisagé plusieurs types d'attaques physiques sur implémentation logicielle 8 bits d'une telle fonction secrète : une de type IFA basée sur la non-détection de fautes de type remise à zéro d'octet de sortie de S-Box, et plusieurs attaques basées sur la détection de collisions par les canaux auxiliaires, avec différents modèles de collisions détectées entre les paires d'entrée/sortie de deux S-Boxs.

Nous avons décrit en détail quatre attaques différentes et fourni des résultats de simulations précises réalisées sur PC pour chacune d'entre elles :

(a) Une attaque FIRE basée sur des IFA, s'appliquant sur une implémentation non protégée. Cette attaque requiert environ 25000 injections de fautes.

(b) Une attaque SCARE dans le modèle de collisions en valeur sur une implémentation non protégée. Cette attaque requiert entre respectivement 100 et 400 courbes environ, en fonction de la capacité de l'attaquant à détecter des collisions entre deux courbes ou seulement à l'intérieur d'une même courbe.

(c) Une attaque SCARE dans le modèle de collisions en valeur sur une implémentation protégée par deux contre-mesures combinées : un masquage booléen au premier ordre et une modification aléatoire de l'ordre des étapes indépendantes. Nous avons montré qu'une attaque était possible en environ 4000 courbes.

(d) Une attaque SCARE dans le modèle plus réaliste de collisions en poids de Hamming sur une implémentation non protégée. Cette attaque est particulièrement efficace et ne requiert l'analyse que de 250 courbes.

Nous avons aussi décrit – sans les avoir simulé en revanche – plusieurs extensions de nos attaques : trois extensions des Attaques (a) et (b) avec notamment des définitions alternatives de la constante de `Rcon` avec plus d'entropie, et une extension de l'Attaque (c) pour le cas d'un masquage booléen d'ordre supérieur lorsque l'attaquant possède une information *a priori* sur la clef. Nous avons aussi exprimé le fait que de multiples chiffrements du même clair rendaient l'Attaque (c) applicable au cas moins contraint des collisions en poids de Hamming.

En plus de proposer différentes recommandations de sécurité en lien avec nos attaques, nous avons identifié plusieurs problèmes qui pourront être étudiés lors de futurs travaux de recherche. L'optimalité des attaques proposées ici n'étant pas démontrée, la réduction du coût de ces attaques pourra par exemple être étudiée, de meilleurs chemins d'attaques pouvant être découverts, réduisant ainsi leurs coûts.

Nos résultats démontrent la nécessité de protéger les implémentations de chiffrement symétrique face aux attaques physiques même dans le cas où les spécifications ne sont pas publiques.

Chapitre 5

Camouflage logiciel

Sommaire

Les travaux présentés dans ce chapitre ont été publiés en 2013 [GMN+13]. Ils consistent en la description d'une méthode de camouflage logiciel permettant de protéger les instructions d'une rétro-conception en se basant sur l'utilisation d'un lot réduit d'instructions difficiles à différencier pour l'attaquant.

5.1 Introduction

Les attaques du Chapitre 4 de type SCARE et FIRE sont des exemples qui montrent que les composants de sécurité peuvent être sensibles même lorsque l'implémentation n'est pas divulguée. En revanche elles demandent de connaître la structure de l'algorithme utilisé, or il est parfois possible que cette information soit elle aussi récupérée grâce aux canaux auxiliaires. Cela peut être fait par des méthodes de type SPA en analysant la forme de la courbe de consommation électrique et en y repérant des motifs se répétant, révélateurs des étapes de l'algorithme. Certaines méthodes plus évoluées de rétro-conception vont utiliser les méthodes d'analyse des canaux auxiliaires en ne les appliquant pas aux données traitées, mais aux opérations bas niveau exécutées sur un processeur. Grâce à la reconnaissance de celles-ci l'attaquant va alors tenter de comprendre ce qui est effectué comme opérations sur le composant cible.

Les protections contre ce type d'attaques sont souvent des contre-mesures classiques conçues pour faire face aux canaux auxiliaires (telles que celles décrites dans [MOP07]) et qui peuvent être coûteuses à mettre en place. Par exemple le masquage au premier ordre de l'AES décrit

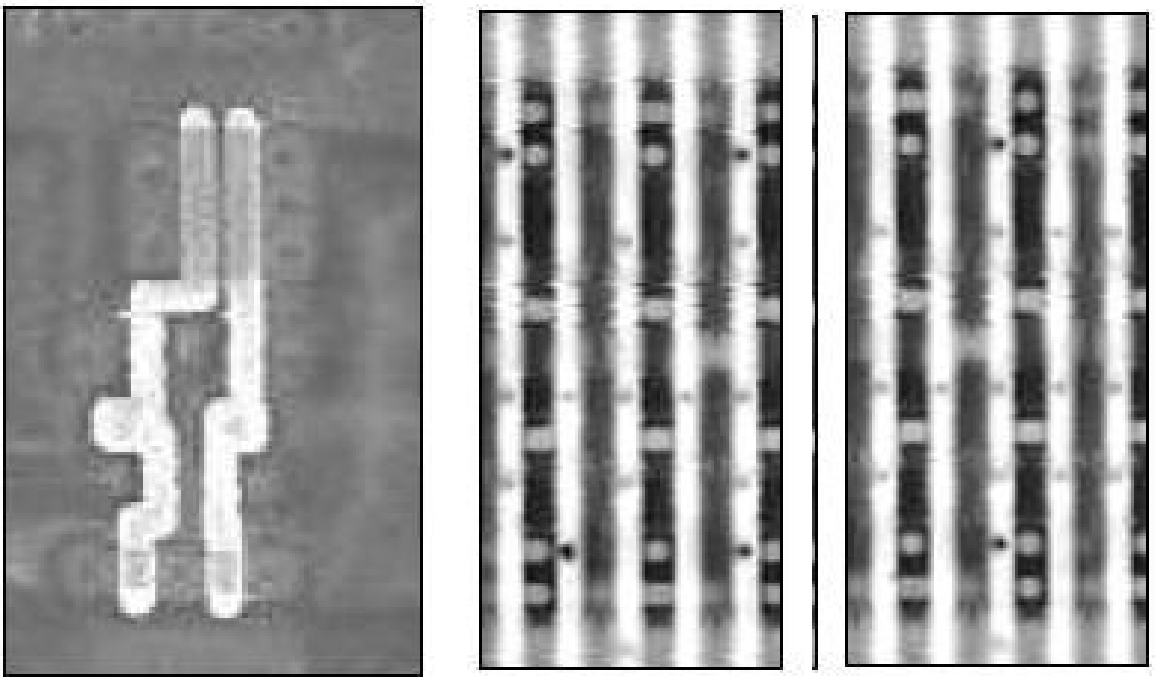

FIGURE 5.1 – Camouflage bas niveau de portes logiques. Gauche : porte logique non protégée dont la fonction peut être identifiée visuellement. Milieu, droite : Portes logiques ET et OU pratiquement indistinguables visuellement. *[avec l'aimable autorisation de SMI / SypherMedia Library]*

dans [RP10] peut augmenter son coût en temps de environ 3 000 cycles à environ 129 000 cycles .

Des protections à bas niveau ont été conçues – une protection illustrée en Figure 5.1 – consistant en une mise en forme des portes logiques volontairement proches (milieu et droite) afin de les rendre moins facilement identifiables, contrairement à leur forme fonctionnelle originelle (gauche). Ces méthodes sont appelées des camouflages matériels.

Dans ce chapitre nous nous plaçons dans la situation où nous devons protéger, des attaques de type SCARE, un code exécuté sur un processeur. Nous développons une solution basée sur des instructions camouflées, empêchant l'attaquant de reconnaître lesquelles sont exécutées. Elle se base sur l'utilisation d'un lot d'instructions difficiles à différencier amenant l'attaquant à une complexité d'analyse rapidement trop élevée.

Dans la Section 5.2 nous décrivons notre apport principal, une analyse de la différentiabilité des instructions sur plusieurs dispositifs. La méthode de camouflage est décrite et analysée de manière plus succincte en Section 5.3. Nous concluons en Section 5.4.

5.2 Analyse de la différentiabilité des instructions

Dans cette section, nous allons montrer qu'il est possible de choisir un ensemble d'instructions qui soient difficiles à différencier les unes des autres par une analyse de canaux auxiliaires.

Nous avons effectué trois études distinctes sur deux dispositifs différents. La première sur un

	OR (AOR)	**XOR** (EOR)	**AND** (AND)	**ADD** (ADC)	**SUB** (SBC)
IMM	ORR	ORR	ORR	ORR	ORR
Z-PAGE	ORR	ORR	ORR	ORR	ORR
Z-PAGE, X	ORR	ORR	ORR	ORR	ORR
ABS	RORRR	RORRR	RORRR	RORRR	RORRR
ABS, X	RORRR	RORRR	RORRR	RORRR	RORRR
ABS, Y	RORRR	RORRR	RORRR	RORRR	RORRR
(IND, X)	ORRORRR	ORRORRR	ORRORRR	ORRORRR	ORRORRR
(IND), Y	ORRORRR	ORRORRR	ORRORRR	ORRORRR	ORRORRR

TABLE 5.1 – Mise en évidence de l'égalité de signatures (Rd,We) pour un lot d'instructions dans tous les modes d'adressage

ASIC contenant un microcontrôleur 6502* (instructions 8 bits) ; la deuxième est un microcontrôleur AT91* (instructions 32 bits) avec un ARM7-TDMI* exécutant une machine virtuelle Java ; la troisième est sur la même plate-forme mais exécutant cette fois-ci du code natif.

5.2.1 Analyse du microcontrôleur 6502

Le microcontrôleur CISC 6502 que nous avons utilisé pour cette analyse était synthétisé dans une technologie CMOS STMicroelectronics 130 nm à partir d'un code VHDL* du 6502 [Kes]. Les courbes de courant obtenues concernent une exécution d'une étape `SubBytes` – 16 appels à la table S-Box – de l'AES. Grâce à l'outil de synthèse MODELSIM*, nous étions capables de connaître les valeurs de tous les états internes (registres*, drapeaux*, signaux, ...) à chaque cycle d'horloge de l'exécution. En utilisant la méthode des moindres carrés – aussi appelée méthode stochastique [LP07] – la modélisation linéaire de la consommation par une paire de signaux révèle que plus de 98% de la consommation électrique est expliquée par les valeurs prises par les signaux binaires Rd et We. Ces signaux correspondent respectivement aux signaux de *lecture donnée* (***R**ead **d**ata*) et *écriture autorisée* (***W**rite **e**nable*), ils sont utilisés par le 6502 afin de contrôler la mémoire RAM. La paire de signaux (Rd,We) ne peut prendre que trois valeurs sur les quatre possibles :

— (Rd,We)=(0,0), notée par la suite O, pas d'accès mémoire,
— (Rd,We)=(1,0), notée par la suite R, accès mémoire en lecture,
— (Rd,We)=(0,1), notée par la suite W, accès mémoire en écriture,
— (Rd,We)=(1,1), impossible en pratique d'accéder simultanément en lecture et en écriture.

L'outil *MODELSIM* nous a permis de constater que la séquence de valeurs du couple (Rd,We) était toujours identique pour tous les appels d'une instruction donnée. On peut ainsi parler de *signature* pour désigner la séquence de valeurs du couple (Rd,We) pour chaque instruction. Nous avons pu identifier la signature de chaque instruction et il a alors été intéressant de constater que certaines instructions partageaient la même signature dans tous les modes d'adressage* comme illustré en Table 5.1 par les instructions **OR**, **XOR**, **AND**, **ADD** et **SUB** (respectivement *ou simple*, *ou exclusif* et *et logique* binaires puis *addition* et *soustraction* arithmétiques).

Au vu du fait que 98% de la consommation est expliquée par les valeurs du couple (Rd,We) et que ces cinq instructions possèdent la même signature, elles sont donc virtuellement indistinguables par analyse des canaux auxiliaires. Cette remarque mène à penser que ce sont les signaux

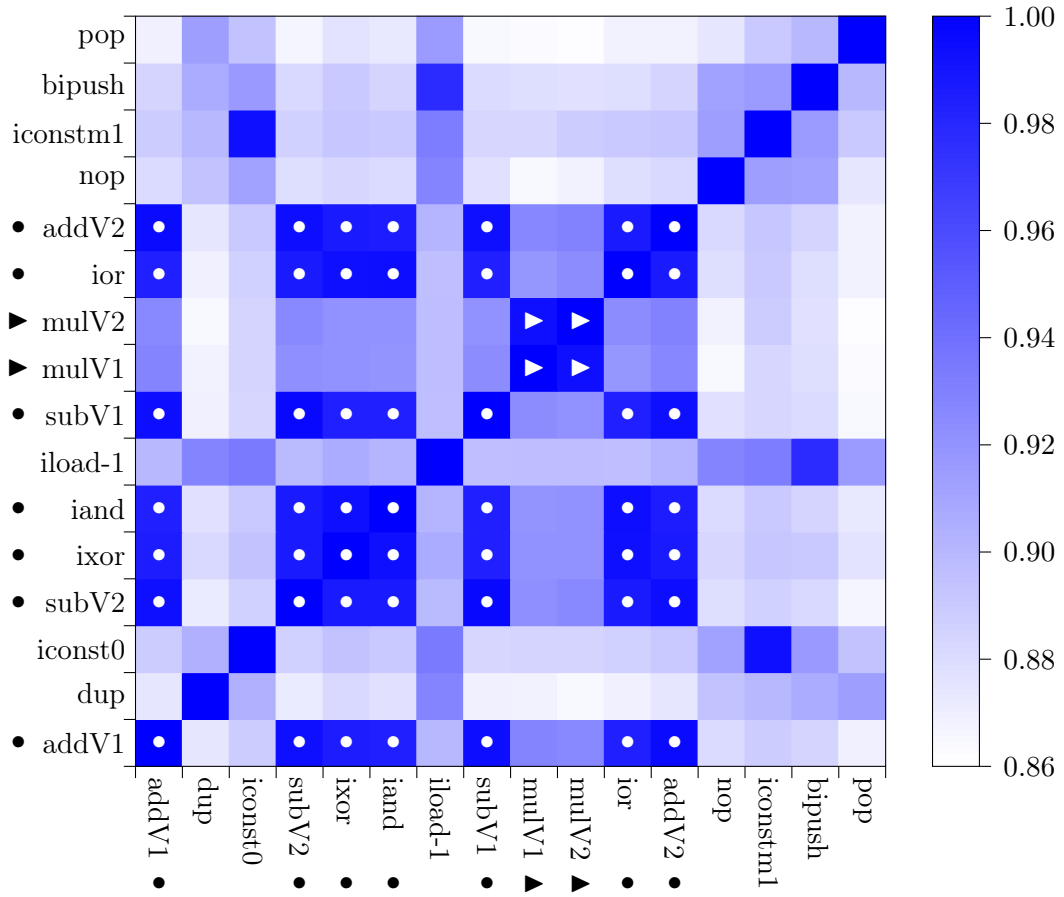

FIGURE 5.2 – Détail des corrélations des instructions par paires. Mise en avant de 2 instructions pour le lot ▶ et 7 instructions pour le lot •.

de communication avec la mémoire RAM qui fuient fortement et qui prennent le dessus sur les signaux du processeur lorsqu'il réalise les opérations.

5.2.2 Analyse sur ARM7 avec machine virtuelle Java

Le but de notre travail sur ARM7 avec la machine virtuelle Java est le même que pour le 6502 en Section 5.2.1 : trouver un ensemble d'instructions qui aient la propriété d'être difficile – voire impossible – à différencier par analyse de canaux auxiliaires. Dans ce but, nous avons choisi un ensemble d'instructions que nous avons exécutées sur notre plate-forme et dont nous avons enregistré l'émission électromagnétique (en enregistrant la moyenne de 50 exécutions pour chaque instruction). Ensuite, pour chaque paire possible d'instructions nous avons calculé la corrélation entre les deux mesures. La Figure 5.2 montre les résultats de toutes les corrélations que nous avons obtenues. Le fait que nous ayons un fort taux de corrélation (toujours supérieur à 0,86) est dû au fait que chaque mesure comporte des parties communes à cause des pré-charges d'instructions qu'effectue la machine virtuelle Java. Nous pouvons alors regrouper les instructions par similarité de fuite électromagnétique : e.g. les groupes (**MULV1**, **MULV2**) ou bien (**IXOR**, **IAND**, **IOR**, **SUBV1**, **SUBV2**, **ADDV1**, **ADDV2**) respectivement identifiés par les symboles ▶ et • sur la Figure 5.2.

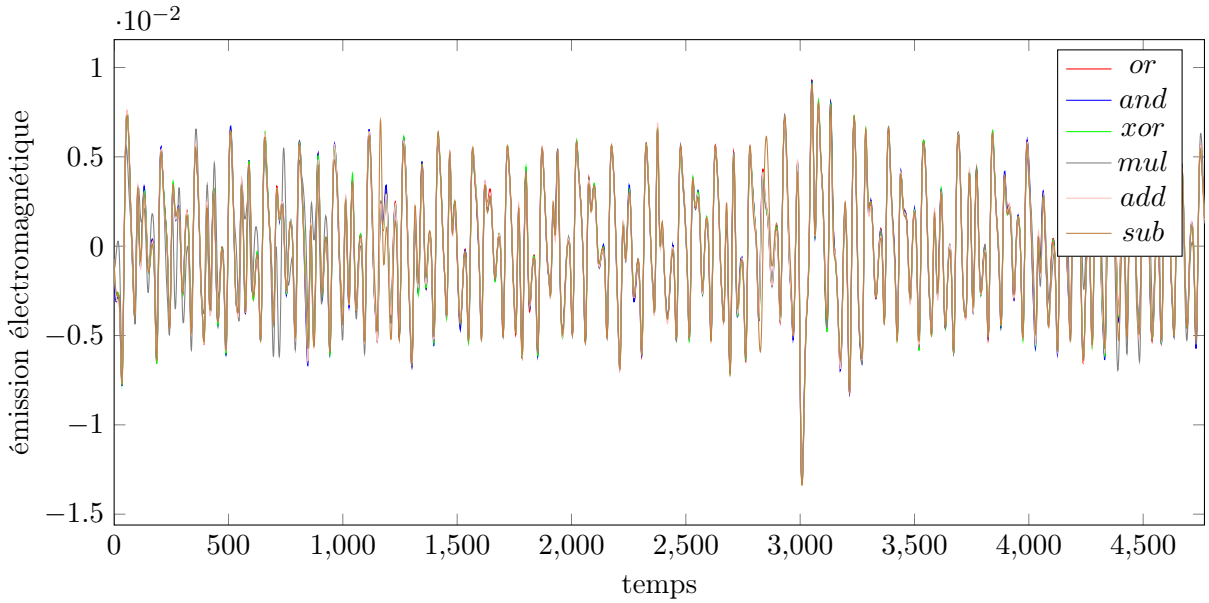

FIGURE 5.3 – Mise en évidence de la similitude globale des courbes d'émissions électromagnétiques pour les instructions étudiées

5.2.3 Analyse sur ARM7 natif

Nous avons effectué une autre expérience sur la plate-forme ARM7 consistant cette fois-ci en l'exécution de code natif au niveau assembleur. Le protocole est identique à celui appliqué Section 5.2.2, les mesures de fuites électromagnétiques obtenues pendant l'exécution des instructions sont récupérées et analysées. Le résultat de ce travail est détaillé par les trois courbes en Figures 5.3, 5.4 et 5.5.

La première (Figure 5.3) présente les mesures d'émissions électromagnétiques pour une liste d'instructions (**OR**, **AND**, **XOR**, **SUB**, **ADD** et **MUL**) où on peut constater que leur forme générale est similaire. La deuxième (Figure 5.4) compare les mesures uniquement pour les instructions **ADD** et **SUB** afin de mettre en avant des différences suffisantes pour les distinguer. La dernière (Figure 5.5) compare les mesures pour les instructions **OR**, **AND** et **XOR** en mettant en avant leur grande proximité et donc la difficulté de les différencier par une analyse des canaux auxiliaires.

5.3 Méthode de camouflage logiciel

Cette section détaille la méthode de camouflage logiciel [21], expliquant comment on peut tirer avantage de l'indistinguabilité des instructions **AND**, **OR** et **XOR** – illustrée en Section 5.2 – afin de dissimuler la fonctionnalité d'une table S-Box à un attaquant utilisant les canaux cachés. On considérera ici le contexte d'un algorithme de chiffrement symétrique, tel l'AES, qui est modifié avant d'être implanté dans un dispositif. La modification de l'algorithme peut être réalisée comme l'*AES modifié* introduit pour les attaques SCARE et FIRE en Chapitre 4 (Section 4.2). Cette pratique de variations sur des algorithmes connus pour être résistants est

21. La méthode est ici décrite de manière plus succincte que dans le papier originel [GMN+13] car elle a été majoritairement menée par une autre partie des contributeurs.

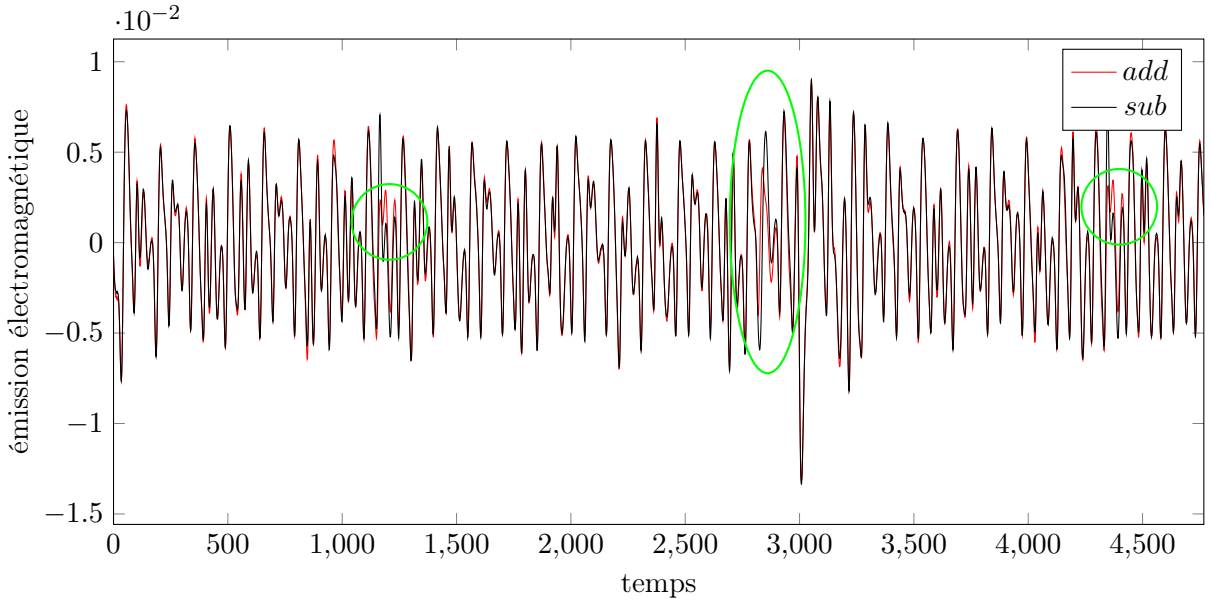

FIGURE 5.4 – Mise en évidence des différences entre les courbes d'émissions électromagnétiques pour les instructions **ADD** et **SUB**

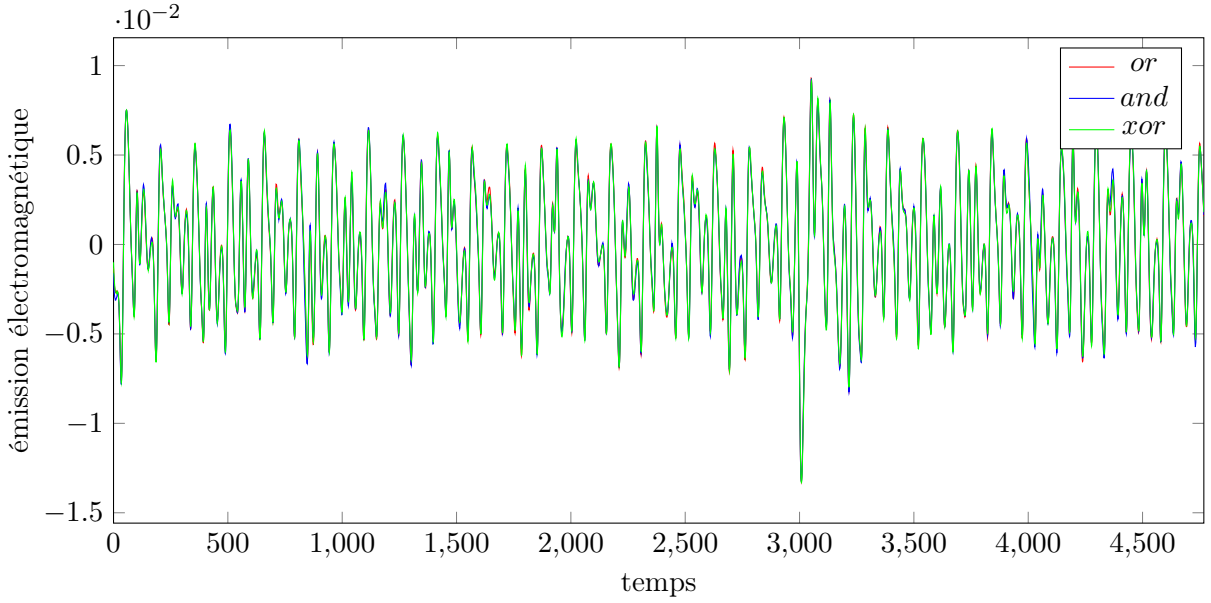

FIGURE 5.5 – Mise en évidence de l'indistinguabilité des courbes d'émissions électromagnétiques pour les instructions **OR**, **AND** et **XOR**

courante dans certains domaines de gestion d'accès à un service (cartes à puce de *Pay-TV**) ou de télécommunications (algorithmes de chiffrement entre le terminal et la station relais). Nous nous plaçons donc dans la situation où le but du concepteur du composant est de créer une S-Box dont les instructions ne peuvent être reconnues même en présence de fuites d'information à travers les canaux auxiliaires.

5.3.1 Description des méthodes

Nous appelons ici $f : \mathbb{F}_2^n \to F_2^m$ une fonction S-Box générique avec x_i (pour $i = 0, \ldots, n-1$) les n bits d'un mot en entrée de S-Box et $f_j(x)$ (pour $j = 0, \ldots, m-1$) les m bits de l'image du mot x à travers f. Nous présentons ici deux façons d'implémenter f en n'utilisant que des **AND** et des **XOR**, et qui garantissent – si ces deux opérations sont indistinguables à travers les canaux auxiliaires – un camouflage de la table utilisée.

Methode CNF

Pour la méthode CNF*, pour *Conjunctive Normal Form*, le calcul de $f(x)$ est scindé au niveau bit, les $f_j(x)$ sont calculés de la manière suivante :

$$f_j(x) = \bigvee_{y \in \mathbb{F}_2^n} (f_j(y) \wedge \bigwedge_{i=0}^{n-1} (1 \oplus x_i \oplus y_i))$$

Cette équation est valide car le terme $\bigwedge_{i=0}^{n-1} (1 \oplus x_i \oplus y_i)$ ne sera égal à 1 que lorsque $x = y$. Ceci permettant une autre écriture remplaçant les **OR** par des **XOR** :

$$f_j(x) = \bigoplus_{y \in \mathbb{F}_2^n} (f_j(y) \wedge \bigwedge_{i=0}^{n-1} (1 \oplus x_i \oplus y_i))$$

Le coût de cette méthode pour calculer $f(x)$ est de $m \times (2^n - 1) \times (2n + 1)$ opérations sur des bits. Ce coût peut être réduit en traitant plusieurs bits en parallèle, ramenant le nombre d'opérations à $(2^n - 1) \times (2n + 1)$ sur des mots de m bits. Par exemple la S-Box de l'AES qui, sur un processeur de taille de bus au moins 8 bits, peut être calculée octet par octet.

Methode ANF

La méthode ANF*, pour *Algebraic Normal Form*, se base sur le même découpage que pour la méthode précédente mais applique une formule différente :

$$f_j(x) = \bigoplus_{y \in \mathbb{F}_2^n} (a_y \wedge \bigwedge_{i=0}^{n-1} (1 \oplus x_i^{y_i}))$$

où $a_y = \bigoplus_{z \in \mathbb{F}_2^n / z \wedge y = z} f_j(z)$ et où $x_i^{y_i}$ vaut 1 si $y_i = 0$ et vaut x_i sinon. Le coût de cette implémentation est de $m \times (2^n - 1) \times (n + 1)$ opérations sur les bits, ce qui est moins élevé que le choix de la CNF. Cela est dû au fait qu'en moyenne le calcul de x^y ne nécessite que $n/2$ **AND**.

Une implémentation de la méthode classique de S-Box lue dans une table et une implémentation de la méthode ANF décrite ici ont été réalisées, les mesures de leurs émissions électromagnétiques sont présentées en Figure 5.6.

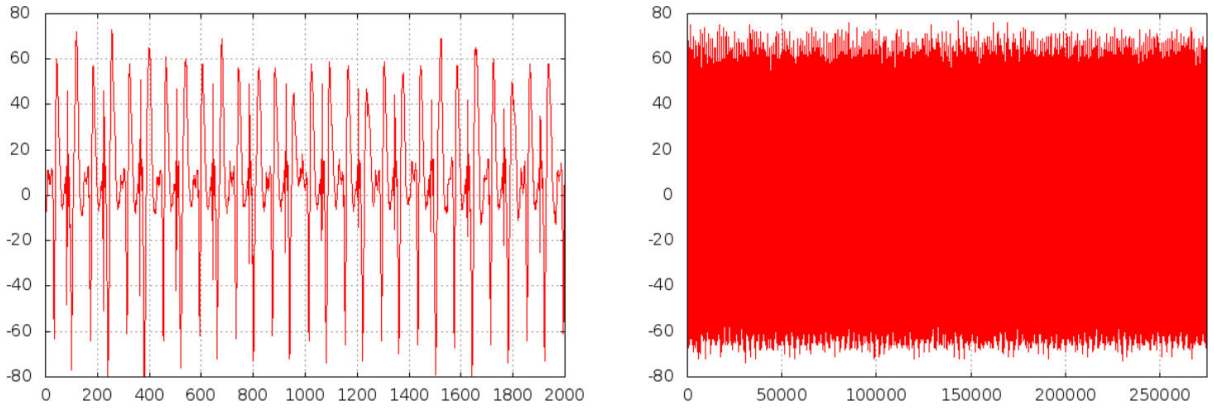

FIGURE 5.6 – Abscisse : Temps, Ordonnée : Émission électromagnétique. Mesure des émissions électromagnétiques des implémentations de S-Box : à gauche par la recherche dans une table en mémoire et à droite par la méthode ANF

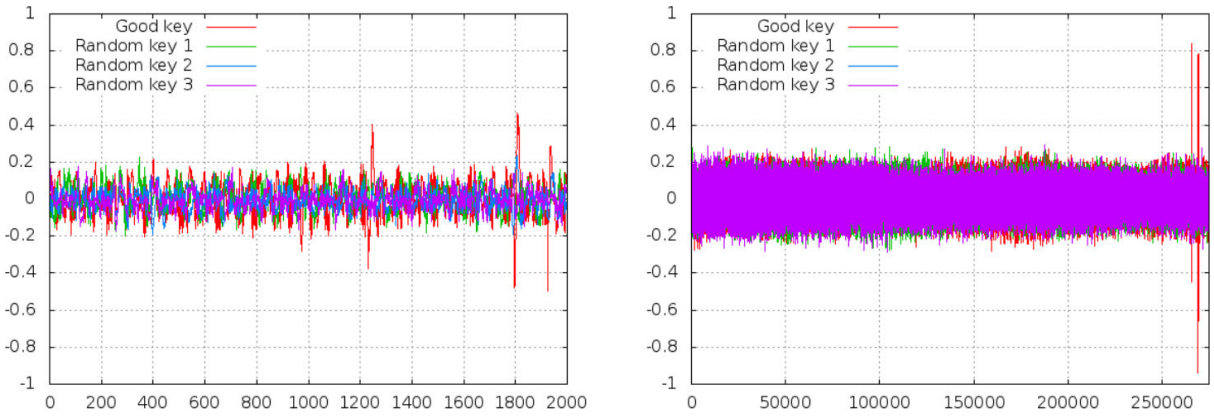

FIGURE 5.7 – Abscisse : Temps, Ordonnée : Coefficient de corrélation. Corrélation entre les émissions électromagnétiques et la valeur de sortie des S-Box : à gauche appliqué à la recherche dans une table en mémoire et à droite appliqué à la méthode ANF

Sécurités additionnelles

Un dispositif exécutant une S-Box d'une des deux façons décrites en Section 5.3.1 reste sensible à une attaque par corrélation car chaque exécution réalise la même séquence d'opérations dans le même ordre. Afin de se protéger de ce type d'attaque il est aisé d'appliquer une permutation aléatoire de l'ordre de calcul des différentes sous-opérations. Ce type de contre-mesure s'applique à l'intérieur du calcul d'une S-Box mais peut potentiellement être étendu au mélange d'opérations provenant du calcul de différentes S-Boxs, e.g. les 16 calculs de S-Box d'un tour d'AES.

Une autre source d'insécurité provient de la fin du calcul car les données obtenues vont être stockées en mémoire et risquent alors d'être détectées comme le montrent les résultats de l'analyse CPA sur 200 courbes en Figure 5.7. Afin d'éviter cette fuite d'information un masquage à bas coût peut être mis en place et protéger des attaques au premier ordre, en effet la formule ANF peut utiliser $\overline{a_0}$, l'opposé de la valeur a_0, et ainsi calculer l'opposé de la table S-Box.

5.3.2 Résultats

En prenant la table S-Box de l'AES comme référence, une estimation d'une méthode classique de masquage au premier ordre coûte plus de 7 000 cycles, soit trois fois plus que la méthode ANF décrite dans ce chapitre. Il est de plus pressentit que la méthode ANF proposée est plus efficace car elle utilise l'indistinguabilité des opérations **AND** et **XOR**.

L'exécution en ordre aléatoire rend les attaques de type FIRE difficiles à réaliser car il est ardu de reproduire une même faute. De plus des méthodes de détections d'injections peuvent être mises en place. La difficulté de délimiter clairement la consommation des S-Boxs complexifie aussi grandement les attaques de type SCARE.

À titre d'exemple les méthodes SCARE et FIRE présentées en Chapitre 4 sont impossibles à réaliser dans le cadre de cette contre-mesure.

5.4 Conclusion

En utilisant l'observation que certaines opérations élémentaires des processeurs pouvaient être indistinguables aux méthodes d'analyse de canaux auxiliaires nous avons conçu une méthode permettant de sécuriser l'utilisation d'une table S-Box tenue secrète. La méthode de *camouflage logiciel* présentée ici permet de protéger une table S-Box quelconque pour un surcoût trois fois moindre que d'autres options de la littérature et utilise l'indistinguabilité de certaines opérations élémentaires afin d'offrir à une table une protection contre les méthodes de rétro-conception par canaux auxiliaires. La méthode permet aussi de se protéger des attaques différentielles classiques visant à retrouver les données traitées.

Conclusion de la thèse

Dans cette thèse plusieurs aspects de l'analyse de canaux auxiliaires ont été abordés, de nouvelles attaques physiques classiques visant à retrouver une clef et l'application des méthodes d'analyse des canaux auxiliaires à la rétro-conception.

En Chapitre 2 nous avons montré qu'une attaque peut être étendue même en cas de contre-mesures d'ordre élevé et donc que des combinaisons de contre-mesures, telles que masquages et exécutions en ordre aléatoire peuvent être nécessaires pour se protéger.

En Chapitre 3 nous avons mis en avant que l'analyse des fuites doit être réalisée à différents niveaux de granularité. En effet notre analyse pour chaque bit sur un composant s'est révélée payante, car un bit fuyant plus que les autres nous a permis d'accélérer le processus d'attaque. Ce phénomène n'étant pas détectable au niveau de l'octet.

On a vu en Chapitre 4 plusieurs méthodes de rétro-conception efficaces qui peuvent s'appliquer même lorsque certaines protections classiques sont mises en place. L'existence de ce type d'attaque justifie d'autant plus que le principe de Kerckhoffs doit être respecté et donc que le fait de garder une implémentation secrète ne doit pas être utilisé pour justifier une réduction des coûts et de la sécurité. Les concepteurs de dispositifs de sécurité doivent envisager qu'une implémentation secrète doit être protégée des attaques au même niveau que celles dont les spécifications sont publiques.

Le Chapitre 5 met en avant la possibilité de construire une contre-mesure grâce à un phénomène observé sur les canaux auxiliaires pour s'adapter au mieux à la réalité des fuites d'information.

Un effet revenant régulièrement dans nos analyses est que les contre-mesures doivent être étudiées avec soin sur tous les effets qu'elles produisent. Certes, elles protègent de certaines attaques augmentant leur complexité, mais elles fournissent fréquemment de nouvelles informations permettant en même temps de réduire la complexité de ces attaques ou bien d'autres types d'attaques. La balance s'équilibrant on perd – ou au moins diminue – alors la protection qu'elles apportent.

La suite des travaux de cette thèse peut être envisagée de nombreuses façons, notamment avec le fait de s'adapter à des mesures moins précises pour la plupart des attaques présentées. On peut envisager l'application des attaques SPA à d'autres modèles de processeurs. Nous avons montré que des attaques FIRE étaient possibles et que c'est donc un domaine à explorer. On peut aussi viser l'amélioration de la méthode FIRE proposée ici en tentant de réduire son coût mais aussi par l'évaluation d'autres modèles de fautes qui pourraient être appliqués. Les méthodes SCARE peuvent également être envisagées face à d'autres modèles de contre-mesures et avec d'autres modes de détection de collisions.

Publications

[CDD+14] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M. Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, Daisuke Nakatsu, Kazuo Ohta, Kazuo Sakiyama, Laurent Sauvage, Werner Schindler, Marc Stöttinger, Nicolas Veyrat-Charvillon, Matthieu Walle, and Antoine Wurcker. Practical Improvements of Side-Channel Attacks on AES Feedback From the 2nd DPA Contest. *J. Cryptographic Engineering*, 4(4) :259–274, 2014.

[CFW15] Christophe Clavier, Julien Francq, and Antoine Wurcker. Study of a Parity Check Based Fault-Detection Countermeasure for the AES Key Schedule. *IACR Cryptology ePrint Archive*, 2015 :877, 2015.

[CIMW15] Christophe Clavier, Quentin Isorez, Damien Marion, and Antoine Wurcker. Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks. *Cryptography and Communications*, 7(1) :121–162, 2015.

[CIW13] Christophe Clavier, Quentin Isorez, and Antoine Wurcker. Complete SCARE of AES-like Block Ciphers by Chosen Plaintext Collision Power Analysis. In Goutam Paul and Serge Vaudenay, editors, *International Conference on Cryptology in India – INDOCRYPT '13*, Lecture Notes in Computer Science, pages 116–135. Springer, 2013.

[CMW14] Christophe Clavier, Damien Marion, and Antoine Wurcker. Simple Power Analysis on AES Key Expansion Revisited. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES '14*, volume 8731 of *Lecture Notes in Computer Science*, pages 279–297. Springer, 2014.

[CW13] Christophe Clavier and Antoine Wurcker. Reverse Engineering of a Secret AES-like Cipher by Ineffective Fault Analysis. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC '13*, pages 119–128. IEEE Computer Society Press, 2013.

[GMN+13] Sylvain Guilley, Damien Marion, Zakaria Najm, Youssef Souissi, and Antoine Wurcker. Software Camouflage. In Jean Luc Danger, Mourad Debbabi, Jean-Yves Marion, Joaquín García-Alfaro, and A. Nur Zincir-Heywood, editors, *Foundations and Practice of Security – FPS '13*, volume 8352 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2013.

Table des figures

Liste des tableaux

Bibliographie

[BBKK07] Alex Biryukov, Andrey Bogdanov, Dmitry Khovratovich, and Timo Kasper. Collision Attacks on AES-Based MAC : Alpha-MAC. In Paillier and Verbauwhede [PV07], pages 166–180.

[BCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Joye and Quisquater [JQ04], pages 16–29.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[Bog07] Andrey Bogdanov. Improved Side-Channel Collision Attacks on AES. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography – SAC '07*, volume 4876 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2007.

[Bog08] Andrey Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In Oswald and Rohatgi [OR08], pages 30–44.

[BR14] Lejla Batina and Matthew Robshaw, editors. *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*. Springer, 2014.

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski, Jr, editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.

[BS99] Eli Biham and Adi Shamir. Power Analysis of the Key Scheduling of the AES Candidates. In *Second AES Candidate Conference – AES2*. Rome, Italy, 1999.

[CDD+14] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M. Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, Daisuke Nakatsu, Kazuo Ohta, Kazuo Sakiyama, Laurent Sauvage, Werner Schindler, Marc Stöttinger, Nicolas Veyrat-Charvillon, Matthieu Walle, and Antoine Wurcker. Practical Improvements of Side-Channel Attacks on AES Feedback From the 2nd DPA Contest. *J. Cryptographic Engineering*, 4(4) :259–274, 2014.

[CFG+11] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Improved Collision-Correlation Power Analysis on First Order Protected AES. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES '11*, volume 6917 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2011.

[CIMW15] Christophe Clavier, Quentin Isorez, Damien Marion, and Antoine Wurcker. Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks. *Cryptography and Communications*, 7(1) :121–162, 2015.

[CIW13] Christophe Clavier, Quentin Isorez, and Antoine Wurcker. Complete SCARE of AES-like Block Ciphers by Chosen Plaintext Collision Power Analysis. In Goutam Paul and Serge Vaudenay, editors, *International Conference on Cryptology in India – INDOCRYPT '13*, Lecture Notes in Computer Science, pages 116–135. Springer, 2013.

[Cla07a] Christophe Clavier. An Improved SCARE Cryptanalysis Against a Secret A3/A8 GSM Algorithm. In Patrick Drew McDaniel and Shyam K. Gupta, editors, *International Conference on Information Systems Security – ICISS '07*, volume 4812 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2007.

[Cla07b] Christophe Clavier. Secret External Encodings Do not Prevent Transient Fault Analysis. In Paillier and Verbauwhede [PV07], pages 181–194.

[CMW14] Christophe Clavier, Damien Marion, and Antoine Wurcker. Simple Power Analysis on AES Key Expansion Revisited. In Batina and Robshaw [BR14], pages 279–297.

[CW13] Christophe Clavier and Antoine Wurcker. Reverse Engineering of a Secret AES-like Cipher by Ineffective Fault Analysis. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC '13*, pages 119–128. IEEE Computer Society Press, 2013.

[DLMV03] Rémy Daudigny, Hervé Ledig, Frédéric Muller, and Frédéric Valette. SCARE of the DES. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security – ACNS '05*, volume 3531 of *Lecture Notes in Computer Science*, pages 393–406. Springer-Verlag, 2003.

[GMN+13] Sylvain Guilley, Damien Marion, Zakaria Najm, Youssef Souissi, and Antoine Wurcker. Software Camouflage. In Jean Luc Danger, Mourad Debbabi, Jean-Yves Marion, Joaquín García-Alfaro, and A. Nur Zincir-Heywood, editors, *Foundations and Practice of Security – FPS '13*, volume 8352 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2013.

[GPT14] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get Your Hands Off My Laptop : Physical Side-Channel Key-Extraction Attacks on PCs. In Batina and Robshaw [BR14], pages 242–260.

[GSM+10] Sylvain Guilley, Laurent Sauvage, Julien Micolod, Denis Réal, and Frédéric Valette. Defeating Any Secret Cryptography with SCARE Attacks. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *Progress in Cryptology – LATINCRYPT '10*, volume 6212 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2010.

[JQ04] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems – CHES '04 : 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[JQYY02] Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen, and Moti Yung. Observability Analysis – Detecting When Improved Cryptosystems Fail. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA '02*, volume 2271 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2002.

[Kes] D. Kesner. Free VHDL 6502 core. `http://web.archive.org/web/20040603222048/http://www.free-ip.com/6502/index.html`.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[KP00] Çetin Kaya Koç and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems – CHES '00, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[LP07] Kerstin Lemke-Rust and Christof Paar. Analyzing side channel leakage of masked implementations with stochastic methods. In Joachim Biskup and Javier Lopez, editors, *European Symposium on Research in Computer Security – ESORICS '07*, volume 4734 of *Lecture Notes in Computer Science*, pages 454–468. Springer, 2007.

[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 '02*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, 2003.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *WOST '99 : Proceedings of the USENIX Workshop on Smartcard Technology*, pages 151–162, Berkeley, CA, USA, 1999. USENIX Association.

[Mes00] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Koç and Paar [KP00], pages 238–251.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MS00] Rita Mayer-Sommer. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards. In Koç and Paar [KP00], pages 78–92.

[Nat01] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standard #197, 2001.

[Nov03a] Roman Novak. Side-Channel Attack on Substitution Blocks. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *Applied Cryptography and Network Security – ACNS '03*, volume 2846 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2003.

[Nov03b] Roman Novak. Sign-Based Differential Power Analysis. In Kijoon Chae and Moti Yung, editors, *Workshop on Information Security Applications – WISA '03*, volume 2908 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2003.

[OR08] Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*. Springer, 2008.

[PSG11] Manuel San Pedro, Mate Soos, and Sylvain Guilley. FIRE : Fault Injection for Reverse Engineering. In Claudio Agostino Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, volume 6633 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2011.

[PV07] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems – CHES '07, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[RDG+08] Denis Réal, Vivien Dubois, Anne-Marie Guilloux, Frédéric Valette, and M'hamed Drissi. SCARE of an Unknown Hardware Feistel Implementation. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Application – CARDIS '08*, volume 5189 of *Lecture Notes in Computer Science*, pages 218–227. Springer, 2008.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES '10*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[RR13] Matthieu Rivain and Thomas Roche. SCARE of Secret Ciphers with SPN structures. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT '13*, Lecture Notes in Computer Science. Springer-Verlag, 2013. (To appear).

[SLFP04] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A Collision-Attack on AES : Combining Side Channel- and Differential-Attack. In Joye and Quisquater [JQ04], pages 163–175.

[SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Thomas Johansson, editor, *Fast Software Encryption – FSE '03*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer-Verlag, 2003.

[VBC05] Joel VanLaven, Mark Brehob, and Kevin J. Compton. Side Channel Analysis, Fault Injection and Applications - A Computationally Feasible SPA Attack on AES via Optimized Search. In Ryôichi Sasaki, Sihan Qing, Eiji Okamoto, and Hiroshi Yoshiura, editors, *Security and Privacy in the Age of Ubiquitous Computing – SEC '05*, pages 577–588. Springer, 2005.

[YJ00] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9) :967–970, 2000.

Résumé

La cryptographie prend une place de plus en plus importante dans la vie des sociétés depuis que ses utilisateurs se rendent compte de son importance pour sécuriser divers aspects de la vie, depuis les moyens de paiement, de communication et de sauvegarde des éléments de la vie privée des citoyens, jusqu'à la sécurité nationale des pays et de leurs armées. Depuis une vingtaine d'années on sait que les algorithmes de cryptographie ne doivent pas seulement être sûrs mathématiquement parlant, mais que leurs implémentations dans un dispositif les rendent vulnérables à d'autres menaces par des voies d'informations alternatives : les canaux auxiliaires. Que ce soit la consommation électrique, le temps ou les émissions électromagnétiques, ... ces biais ont été évalués et depuis leur découverte les recherches de nouvelles attaques et protections se succèdent afin de garantir la sécurité des algorithmes.

La présente thèse s'inscrit dans ce processus et présente plusieurs travaux de recherche traitant d'attaques et de contre-mesures dans le domaine de l'exploitation de canaux auxiliaires et d'injections de fautes. Une première partie présente des contributions *classiques* où l'on cherche à retrouver une clef cryptographique lorsque la seconde s'attelle à un domaine moins étudié pour l'instant consistant à retrouver les spécifications d'un algorithme tenu secret.

Abstract

Cryptography is taking an ever more important part in the life of societies since the users are realising the importance to secure the different aspects of life from citizens means of payment, communication and records of private life to the national securities and armies. During the last twenty years we learned that to mathematically secure cryptography algorithms is not enough because of the vulnerabilities brought by their implementations in a device through an alternative means to get information : side channels. Whether it is from power consumption, time or electromagnetic emissions ... those biases have been evaluated and, since their discovery, the researches of new attacks follow new countermeasures in order to guarantee security of algorithms.

This thesis is part of this process and shows several research works about attacks and countermeasures in the fields of side channel and fault injections analysis. The first part is about *classic* contributions where an attacker wants to recover a secret key when the second part deals with the less studied field of secret specifications recovery.