Analyse de fonctions de hachage cryptographiques

Christina Boura

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Christina Boura

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Analyse de Fonctions de Hachage Cryptographiques

soutenue le 7 décembre 2012

devant le jury composé de :

Anne Canteaut	INRIA Paris-Rocquencourt	Directrice de thèse
Pascal Paillier	CryptoExperts	Co-directeur de thèse
Pierre-Alain Fouque	Université de Rennes 1	Rapporteur
Henri Gilbert	ANSSI	Rapporteur
Jean-Claude Bajard	Université Pierre et Marie Curie	Examinateur
Joan Daemen	STMicroelectronics	Examinateur
Louis Goubin	Université de Versailles-Saint Quentin	Examinateur
Lars Knudsen	Technical University of Denmark	Examinateur
Aline Gouget	Gemalto	Examinatrice

Christina Boura

Analyse de Fonctions de Hachage Cryptographiques

INRIA-équipe-projet SECRET
Domaine de Voluceau
78153 Le Chesnay

Remerciements

Tout d'abord je veux remercier Anne Canteaut d'avoir été la meilleure directrice de thèse que je pouvais imaginer ainsi qu'une personne extraordinaire. Je la remercie de m'avoir fait travailler sur un sujet que j'ai adoré, d'avoir partagé avec moi tout son savoir-faire, d'avoir toujours été disponible et de s'être toujours occupée de tout de manière très incroyable. Je tiens à la remercier pour toutes ces heures passées à s'occuper d'un article, à corriger un dossier de candidature ou à relire cette thèse, qui n'aurait jamais été ce qu'elle est aujourd'hui sans elle. Merci de m'avoir donner l'occasion de participer à autant de conférences et de m'avoir accueillie à Copenhague au cours de cette dernière année. Grâce à ces courts séjours, je suis maintenant mieux préparée à consommer du cochon et du hareng sucré pendant les deux années qui suivront.

Je suis également très reconnaissante à Pierre-Alain Fouque et Henri Gilbert d'avoir accepté de rapporter cette thèse, bien que leur emploi du temps à tous les deux soit très chargé. Je les remercie pour leur lecture très attentive du manuscrit et leurs corrections et remarques qui ont permis d'améliorer le document final.

Je tiens à remercier Jean-Claude Bajard, Joan Daemen, Louis Goubin, Aline Gouget, Lars Knudsen et Pascal Paillier d'avoir accepté de participer à ce jury de thèse. Merci à Pascal Paillier de m'avoir permis de découvrir les fonctions de hachage lors de mon stage chez Gemalto. Je le remercie, bien qu'il soit à ma connaissance la personne la plus occupée du monde, d'avoir toujours trouvé du temps pour travailler ensemble quand rien semblait ne plus marcher. Je lui suis infiniment reconnaissante d'avoir tout arrangé pour que cette thèse aie lieu sous les meilleures conditions. Merci à Aline Gouget de m'avoir beaucoup guidée lors de mes expérimentations avec les attaques physiques, d'avoir toujours eu la bonne réponse à mes questions et d'être une personne avec qui j'ai eu beaucoup de plaisir à discuter lors de mes pérégrinations à Meudon. Merci à Louis Goubin pour l'occasion d'organiser avec lui (et Maria) le pot de notre départ commun de chez Gemalto. Merci à Lars Knudsen d'avoir accepté de venir jusqu'à Paris pour assister à une soutenance en français, malgré son emploi du temps très chargé. J'en profite également pour le remercier d'avoir accepté de m'accueillir à partir de janvier 2013 dans l'équipe de cryptographie à la DTU (bien que je ne sois pas allemande). Merci à Joan Daemen de me faire l'immense honneur d'être à mon jury de thèse. Je remercie par la même occasion toute la "Keccak team" d'avoir conçu une fonction qui a aidée à inspirer tant de résultats de cette thèse. Merci pour tous les échanges très éducatifs ainsi que pour les 24 bières belges qu'ils ont eu la gentillesse de faire parvenir jusqu'à Paris. Merci à Jean-Claude Bajard de me faire l'honneur d'assister à ma soutenance (même sans son chapeau jaune).

Rien n'aurait été pareil si je n'avais pas été accueillie au projet SECRET pendant ces trois magnifiques années. Merci à tous ses membres, permanents ou non, avec qui j'ai eu l'occasion de partager tant de bons moments et qui m'ont fait sentir là-bas comme dans ma deuxième

maison. Merci pour tous ces jours où je me suis réveillée avec l'envie de me retrouver le plus vite possible au projet, malgré les 70 minutes du trajet. Merci aux permanents du projet, Anne Canteaut, Pascale Charpin, Nicolas Sendrier, Jean-Pierre Tillich, et récemment Maria Naya-Plasencia pour toutes les discussions, scientifiques ou non, pour être toujours si accessibles et pour avoir créé cette ambiance familiale au projet. Merci à Jean-Pierre d'avoir été le meilleur beta-testeur de gâteaux que j'amenais au projet.

Merci au bureau 1 et à tous ses occupants (par ordre d'occupation), Céline, Gregory, Ayoub, Marion et Valentin d'avoir été les meilleurs voisins du monde et d'avoir fait tant d'efforts pour ma culture française (blagues carambar, Monsieur et Madame, contrepèteries, Les Inconnus...). Merci à Céline de m'avoir initié à la course à pied et pour ses instructions de musculation que je suis toujours. Je la remercie également d'avoir partagé avec moi le roadtrip au Texas avant la mémorable conférence ISIT 2010. Merci à Gregory de mettre toujours la bonne humeur au bureau et pour tous ces nombreux et très précieux conseils d'informatique. C'est peut être dur à croire pour lui, mais j'ai quand même beaucoup progressé grâce à son aide ! Je le remercie également d'avoir retardé son départ à Indocrypt afin d'assister à ma soutenance. Merci à Ayoub pour sa voiture si confortable. Sinon, merci à Ayoub pour tous les moments très agréables, les heures passées à faire un footing ou à la salle de sport, à jouer au tennis et à bavarder de tout et n'importe quoi, au cours des deux premières années. Je tiens à le remercier également pour son accueil chaleureux aux séminaires de Caen et de Rouen (merci également à Magalie) et de m'avoir fait visiter ces villes de la Normandie. C'était un grand plaisir pour moi ! Merci à Valentin (après avoir pris le bureau d'Ayoub) de toute son aide avec la rédaction de cette thèse. Merci de m'avoir appris tant de règles de français dont je ne soupçonnais même pas l'existence et d'avoir relu aussi attentivement ce manuscrit. Maintenant je sais que dire "en outre" ça fait classe, mais quatre fois dans la même phrase, ça fait beaucoup. Merci également d'avoir choisi (et transporté) le champagne pour le pot de la soutenance. Merci pour le temps passé à rigoler dans la première navette (et pas seulement). Merci enfin à Marion pour la solidarité féminine du bureau et pour avoir été une voisine de chambre parfaite lors des Journées C2 à Dinard.

Merci également à toutes les autres personnes que j'ai eu la chance de croiser lors de mon séjour : Anthony, Audrey, Ayca, Baudoin, Benoît, Bhaskar, Chloé, Chrysanthi, Denise, Dimitris, Joëlle, Mamdouh, Mathieu, Maxime A., Maxime C., Rafael, Richi, Stéphane J., Stéphane M., Sumanta, Sunandan, Valérie, Vincent et Yann. En particulier, merci à tous ceux qui étaient toujours prêts pour aller boire un verre, déguster un plat d'une cuisine exotique, voir une exposition ou un film. Je me souviendrai toujours de toutes ces sorties du groupe très animées. Un grand merci à Christelle pour son efficacité à régler toutes les tâches administratives et son sourire constant. Merci à Matthieu F. pour ses passages au projet qui créaient toujours la bonne ambiance et pour être toujours prêt à aider. Merci à Maria pour sa gentillesse infinie, de m'avoir tant de fois remonter le moral et d'avoir fait le train de conférences de l'août 2010 (SAC, CRYPTO, CHES, SHA-3) si agréable.

Merci au groupe de Los Pumas (Anne, Andrea, Céline, Fred, Gaëtan, Joana, Maria, Stéphane, Thomas, Yann) de m'avoir permis une intégration si rapide et agréable au monde de la crypto, aux soirées très amusantes de mon début thèse et de m'avoir fait découvrir que je suis définitivement nulle au blind test.

Merci à Maria G. d'avoir fait mon tout premier séjour cryptographique (Journées C2 à Fréjus) si agréable et de m'avoir accueillie chez elle à Caen après le séminaire du groupe.

Cependant, ma thèse s'est déroulée en deux endroits différents. Merci à Gemalto d'avoir financé ma thèse et de m'avoir offert une vision très différente du monde de la crypto. Merci à

toutes les personnes de l'équipe de SL à Meudon que j'ai eu l'occasion de croiser pendant les années de mon stage et de ma thèse : Alain, Anis, Blandine, Bouteina, Christian, Christine, David, Dris, Frédéric, Irene, Jean-Henri, Julien, Gilles, Huy, Laurent, Li Xun, Louis, Mathieu, Maria, Mélanie, les deux Nicolas, Pascal, Pierre, Stéphanie, Steven, Sylvain et Tino. Merci à la bande de stagiaires de 2009 (Dris, Matthieu, Maria, Mélanie et Irene) pour les dîners à thème cuisine du monde. Merci à Stéphanie de m'avoir intégrée à l'équipe et d'avoir tout fait (avec Pascal) pour que cette thèse se réalise. Merci à Christian pour sa gentillesse et pour m'avoir laissé toute la liberté à poursuivre mes recherches. Merci à Sylvain d'avoir passé tant de temps à m'expliquer comment marchaient les "vraies" attaques sur les cartes à puce et pour toute sa patience face à mes questions. Merci à David et Sylvain pour la collaboration lors de l'écriture (et re-écriture) de notre article. Merci à Mathieu pour le voyage à Copenhague pour la conférence FSE 2011 et pour le piano. Merci enfin à Maria d'avoir partagé le chemin avec moi jusqu'au bout, d'avoir été la personne la plus serviable au monde, pour toutes les heures passés à discuter des galères de la vie d'un thésard et pour tous les vols Athènes-Paris (et ses turbulences). Merci également pour toute son aide pour ce pot de thèse et pour plein d'autres choses dont je n'aurai pas la place de tout décrire.

Je tiens à remercier mes deux autres copines grecques à Paris, Irini et Mairi pour tous les bons moments passés ensemble et pour leur aide inestimable avec le pot. Merci d'avoir raté le premier jour de votre conférence de maths pures pour assister à une soutenance de maths appliquées. Je sais que ce n'est pas facile.

Je voudrais remercier ma famille, mes parents et ma soeur, pour leur amour et leur soutien permanent, sans lesquels je ne serai jamais arrivée là. Je voudrais les remercier de m'avoir servi de modèle, de m'avoir laissée libre de faire mes choix, de m'avoir réconfortée quand les choses allaient mal et de toujours transformer mes séjours à Athènes en un vrai paradis. Merci de m'avoir offert un séjour de haute qualité à Kos pendant la rédaction de cette thèse et d'avoir créé toutes les bonnes conditions afin que celle-ci se termine à temps. Merci à tous les trois d'avoir fait ce voyage pour voir un exposé sur les choses qu'ils ne comprennent pas, dans une langue qu'ils ne parlent pas.

Ευχαριστώ τους γόνεις μου γιά όλα όσα έχουν κάνει για μένα, για την ελευθερία που μου έδωσαν στο να ακολουθήσω τον δρόμο που διέλεξα, για την συνεχή τους συμπαράσταση και αγάπη. Η παρουσία σας και η στήριξη σας όλα αυτά τα χρόνια με οδήγησαν έως εδώ σήμερα και ξέρω ότι χώρις εσάς τίποτα δεν θα ήταν το ίδιο. Ευχαριστώ την αδερφή μου που ταξιδέψε μέχρι εδώ για αυτή τη μέρα. Η παρουσίαση σου, Τάνια, μου δίνει μεγάλη χαρά!

Finalement, un grand merci à Tony, qui a tout suivi depuis le début, pour son support quotidien, pour savoir me remonter le moral aux moments (un soir sur deux) où je pleurnichais que ma thèse ne marche pas, pour croire en moi, et pour devoir subir plein de détails sur Alice, Bob et leurs échanges. Je le remercie de me pardonner pour tous les week-ends où je devais (encore) travailler et pour notre voyage en Bretagne qui n'a pas eu lieu. Merci d'avoir accepté d'abandonner la Butte aux Cailles pour Copenhague. Sans toi, ce départ n'aurait jamais été le même.

Overview

This document covers the work that I have carried out during the period 2010-2012, as a PhD student of the SECRET Project-Team at INRIA Paris-Rocquencourt. This work is located in the area of symmetric key cryptography.

The modern cryptography is splitted into two major branches, the symmetric (or secret-key) cryptography and the public-key cryptography. The symmetric cryptography ensures the secrecy of the communication by means of a unique key that is shared and kept secret by both communicating parties. The symmetric algorithms were the first cryptographic algorithms to be used many hundreds years ago in order to protect confidential communications. During the last century, one of the major problems related to the secret-key approach was revealed to be the key distribution, that means the difficulty of the two communication parties to share safely the common secret key. In response to this problem, a new type of cryptography, called the public-key cryptography was born in the beginning of 1970's. In this concept, every person possesses a pair of keys, a public one that is freely distributed and a private one, which is exclusively known by its holder. Despite of these new advances, the area of the symmetric cryptography was not abandoned, but in the contrary, always concentrates the attention of many cryptographers. The reason for this is that the public-key cryptosystems, even if providing a very high level of security, are relatively slow and have a big implementation cost, that is most of the time unacceptable for applications where a big amount of data has to be encrypted or for very constrained environments.

Symmetric cryptography includes three different types of algorithms : block ciphers, stream ciphers and hash functions. Hash functions are public algorithms that associate to a message of arbitrary length, a fixed-length string, called the hash. Hash functions that are used for cryptographic reasons must possess a certain number of security requirements. Even if no secret key is involved in the hash procedure, hash functions are qualified as symmetric algorithms because of their design principles that are very similar to those of block ciphers. During my thesis, I have mostly worked on the security analysis of hash functions. However, most of the results can be equally applied to the security analysis of the components of some families of block ciphers because of the duality in their design.

As a consequence of the publication of some devastating attacks against many hash functions of the MD-SHA family, the American National Institute of Standards and Technology (NIST) initiated in 2008 a public competition, called the SHA-3 contest in order to determine a new standard for hash functions. This competition has reached recently to an end with the announcement of the winning algorithm, the Keccak hash function. As my thesis has taken place during the last three years of the competition, most of my results are naturally related to it. This document is divided in two parts.

The first part focuses on the analysis of mathematical security of hash functions. We start by analysing and formalising some recently introduced distinguishers, called the *zero-sum dis-*

tinguishers. We determine the minimal size of a zero-sum for any vectorial function and prove that this notion is related to linear codes and to APN (Almost Perfect Nonlinear) functions. We show how to construct some distinguishers by exploiting the nonlinear as also the linear part of an iterated permutation. We finally apply these results to the new hash function standard, the KECCAK hash function, as also to the Hamsi hash function, one of the algorithms selected for the second round of the SHA-3 competition. These results have been presented at the *International Symposium on Information Theory-ISIT 2010* [BC10] as also at the international conference *Selected Areas in Cryptography-SAC 2010* [BC11b]. We then present a general study on the evolution of the algebraic degree of iterated permutations. We establish two major results. The first is a new upper bound for the degree of a family of iterated permutations following the SPN (Substitution Permutation Network) design. This result and some applications for the hash functions KECCAK and *Luffa* have been presented at the international conference *Fast Software Encryption-FSE 2011* [BCD11]. The second result establishes a link between the degree of an iterated permutation and the degree of the inverse round permutation. This relation leads to another, more general bound, on the degree of iterated permutations. This work was published in the journal *IEEE Transactions on Information Theory* [BC12b]. Some applications of this new bound to block ciphers have been equally presented at the *10th International Conference on Finite Fields and their Applications-Fq10 2011* [BC11a], while some applications for hash functions have been exposed at the *NIST Third SHA-3 Candidate Conference* [BC12a]. Finally, another type of algebraic properties of Sboxes are investigated at the end of the first part. These properties rely on the existence of linear relations between some output bits and some input bits of the Sbox, when the other input bits are fixed to a well-chosen value. This analysis has permitted the improvement of a second-preimage attack against the Hamsi hash function.

The second part concerns the analysis of the physical security of some candidates to the SHA-3 competition. In particular, the resistance against side-channel attacks of the functions Skein and Grøstl is investigated and some countermeasures are proposed. This work has been presented at the *International Workshop on Trustworthy Embedded Devices-TrustED 2012* [BLV12].

Part I

In this part, some issues concerning the mathematical security of symmetric algorithms are investigated. In particular, different algebraic properties are considered.

Chapter 1 to 2

The first two chapters of this document are dedicated to the introduction of the notions that will be useful for the understanding of the results of this thesis. In the first chapter, the basic constructions of hash functions and block ciphers are presented and some design and security issues are discussed. As a symmetric primitive can be described through the language of vectorial Boolean functions, some basic notions concerning Boolean functions are exposed in Chapter 2. At the end of the chapter some fundamental attacks, such as the algebraic, the higher-order differential, the saturation or the cube attacks, deriving from weaknesses of the confusion part or of the diffusion part of a symmetric algorithm, are presented.

Chapter 3

This chapter deals with the notion of a new type of distinguisher for symmetric primitives, the *zero-sum distinguisher*. This concept was for the first time developed for hash functions by Jean-Philippe Aumasson and Willi Meier [AM09] and it was applied to three candidates of the SHA-3 competition, namely KECCAK, Hamsi and *Luffa*. A zero-sum for a vectorial function F from $\mathbf{F}_2^n$ into $\mathbf{F}_2^m$ of size k is a subset $\{x_1 \ldots, x_k\}$ such that

$$\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} F(x_i) = 0.$$

We prove in this chapter that zero-sums can be associated to linear codes and to APN functions. From this relation we deduce that all vectorial functions possess at least one zero-sum of size at least 5. When F is a permutation, we introduce a notion that is much stronger, that of a *zero-sum partition*. This concept expresses the fact that when F is a permutation, every coset of such a zero-sum is also a zero-sum. In other words, the input space of F is partitioned into zero-sums. We then show a general method for constructing zero-sum partitions for iterated permutations from higher-order differentials. We first show how to exploit the nonlinear part of the permutation and we present in the sequel a method for exploiting the linear part. We mention in particular the importance of correctly estimating the algebraic degree of an iterated permutation in order to construct efficient zero-sum partitions for a big number of rounds. We then apply this method to KECCAK, the winner of the SHA-3 competition and we present zero-sum partitions for up to 20 rounds of its inner permutation. We also present zero-sum partitions of very low complexity for the inner permutation of the Hamsi hash function.

Chapter 4

This chapter presents a new bound on the algebraic degree of a certain class of iterated permutations. The ability of correctly estimating the algebraic degree of an iterated construction provides us with a tool for measuring its resistance against several classes of attacks, such as the algebraic or the higher-order differential attacks. For implementation reasons, the nonlinear part of the round function of many symmetric primitives is composed of the parallel application of some nonlinear permutations of small size. These components are called Sboxes (Substitution Boxes). We show here that this particular construction has an influence on the evolution of the algebraic degree.

For any Sbox S defined over $\mathbf{F}_2^n$ we define by $\delta_i(S)$, $1 \leq i \leq n$ the maximum degree of any product of i coordinates of S. The main result of this chapter is then the following.

Let F be a function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^n$ that corresponds to the concatenation of m smaller balanced Sboxes, $S_1, \ldots, S_m$, defined over $\mathbf{F}_2^{n_0}$. Let $\delta_i = \max_{1 \leq j \leq m} \delta_i(S_j)$. Then, for any function G from $\mathbf{F}_2^n$ into $\mathbf{F}_2^\ell$ we have

$$\deg(G \circ F) \leq n - \frac{n - \deg G}{\gamma},$$

where

$$\gamma = \max_{1 \leq i \leq n_0 - 1} \frac{n_0 - i}{n_0 - \delta_i}.$$

We used this bound to estimate the evolution of the algebraic degree of the inner permutation of the KECCAK hash function. This permitted us to construct zero-sum partitions for the entire permutation. These partitions, even is they are of very high complexity and do not threat the security of the hash function reveal a conflict with the assumptions made by the hermetic sponge strategy. We study next the evolution of the algebraic degree of the Luffa hash function. The previous bound, combined with the fact that the Sboxes of *Luffa* have several quadratic components, permitted us to construct several higher-order differential distinguishers for the full hash function *Luffa* v1 and for the compression function of *Luffa* v2.

Chapter 5

In the previous chapter we have presented a bound on the algebraic degree of iterated functions based on a nonlinear layer is composed of smaller functions. Here, we study the same notion but in a more general setting. Our main result shows that the degree after several iterations depends on the degree of the inverse of the permutation that is iterated. We show that the degree of the inverse permutation influences the degree of the function that is iterated even in situations where the inverse function is never used in practice, as it is the case of Feistel constructions or hash functions. One of the central results of this chapter is the following bound, that involves the degree of the inverse function.

Let F be a permutation of $\mathbf{F}_2^n$ and G a function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^m$. Then,

$$\deg(G \circ F) \leq n - \left\lfloor \frac{n - 1 - \deg G}{\deg F^{-1}} \right\rfloor.$$

We deduce from this main observation some interesting corollaries that explain some behaviours that were remarked before but that remained unexplained. We present some applications to the block cipher $\mathcal{KN}$ and also to some hash functions. Finally, we generalize this result to the case where F is a balanced vectorial function that is not a permutation.

Chapter 6

In this chapter we develop a new notion concerning some algebraic properties of nonlinear vectorial functions. This notion expresses the fact that several components of an Sbox are of algebraic degree at most 1 on a vectorial space and on all of its cosets. In particular, it formalizes a property used by Thomas Fuhr in a second-preimage attack of Hamsi [Fuh10]. More precisely, we introduce the concept of (v, w)-linear functions. We say that a vectorial function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^m$ is (v, w)-linear if there exist a subspace $V \subset \mathbf{F}_2^n$ with $\dim V = v$ and a subspace $W \subset \mathbf{F}_2^m$ with $\dim W = w$, such that any component of S determined by W, *i.e.* $x \mapsto \lambda \cdot S(x)$, $\lambda \in W$, is of degree at most 1 on V. It follows that if a function S is (v, w)-linear then it has w components that are v-weakly normal and that the linearity of S is such that

$$\mathcal{L}(S) \geq 2^v.$$

We see next that this notion is related to a well-known generalisation of the Maiorana-McFarland construction for bent Boolean functions. A notable property of Maiorana-McFarland functions is that they can be characterized by means of their second-order derivatives. We use this remark in order to establish an algorithm for determining whether a given vectorial function is (v, w)-linear for some v and w. We examine next a special class of (v, w)-linear

functions, the one of $(n-1,1)$-linear functions. We prove that this class of functions can be completely characterized. We show in particular that a Boolean function of n variables is $(n-1,1)$-linear if and only if

$$\deg(f) = 2 \text{ and } \mathcal{L}(f) \geq 2^{n-1}.$$

We show equally that for $n \geq 5$ there is no permutation of $\mathbf{F}_2^n$ that is both $(n-1,1)$-linear and has optimal nonlinearity. Afterwards, we analyze the notion of $(n-1,1)$ linearity for permutations of $\mathbf{F}_2^4$. As we have seen, $(n-1,1)$-linearity is related to the number of quadratic components of a given permutation. For this reason, we study this number for all classes of permutations of 4 bits.

We show next how this notion is related to a second-preimage attack on the hash function Hamsi presented by Thomas Fuhr in 2010 [Fuh10]. Analyzing the properties of the Hamsi Sbox through the framework of (v,w)-linearity permitted us to slightly improve this attack.

Part II

This part deals with the physical security of hash functions.

Chapter 7

We present here an analysis of the security against side-channel attacks of two finalists of the SHA-3 competition, namely Grøstl and Skein. The goal of this work was to propose some concrete optimal countermeasures on software level against first-order side-channel attacks. We have mounted our attacks on the reference implementation of the two algorithms that we had previously embedded in an ARM-32 chip and we have checked that no secret information is leaked after the application of the proposed countermeasures.

Introduction générale

Les travaux de recherche exposés dans ce document s'inscrivent dans le cadre de la cryptographie symétrique. Ces travaux concernent principalement l'analyse de la sécurité des fonctions de hachage cryptographiques.

En 2008, l'Institut National des Standards et de la Technologie américain (NIST) a initié une compétition publique, nommée SHA-3, afin de sélectionner une nouvelle norme pour les fonctions de hachage. Dans le cadre de ce concours, plusieurs fonctions de hachage dont la sécurité nécessitait une étude approfondie ont été conçues. Ma thèse s'est déroulée pendant les trois dernières années de cette compétition et c'est la raison pour laquelle mes travaux sont fortement liés à celle-ci. En particulier, certains de ces travaux concernent l'analyse des propriétés algébriques de la fonction Keccak, qui a été recemment choisie par le NIST comme le nouveau standard des fonctions de hachage.

En 2009, Jean-Philippe Aumasson et Willi Meier ont introduit un nouveau type de distingueur [AM09], appelé *distingueur à somme nulle*. Cette notion, qui est fortement liée au degré algébrique d'une fonction, a été utilisée afin de construire des distingueurs sur les permutations internes de trois fonctions de hachage participant au deuxième tour du concours SHA-3, à savoir Keccak, Hamsi et *Luffa*. Nous avons étudié et formalisé la notion du distingueur à somme nulle et nous avons introduit des techniques afin d'améliorer les distingueurs de [AM09]. Ces travaux ont été présentés à la conférence *IEEE International Symposium on Information Theory-ISIT 2010* [BC10], ainsi qu'à la conférence *Selected Areas of Cryptography-SAC 2010* [BC11b].

Pendant l'étude des distingueurs à somme nulle, une question qui s'est naturellement posée est celle de l'évaluation du degré algébrique d'une permutation itérée. Estimer le degré algébrique d'une primitive cryptographique après un certain nombre d'itérations est très important afin de déterminer la résistance de la primitive symétrique contre plusieures classes d'attaques, comme par exemple les attaques algébriques ou les attaques différentielles d'ordre supérieur. Pendant cette thèse, nous avons développé une nouvelle borne sur le degré algébrique d'une famille de permutations itérées, à savoir les permutations dont la partie non-linéaire est composée de plusieurs petites boîtes-S. Ce résultat, ainsi que ses applications aux fonctions candidates au concours SHA-3, Keccak et *Luffa*, ont été présentés à la conférence *Fast Software Encryption-FSE 2011* [BCD11]. Nous avons ensuite prouvé que le degré algébrique d'une permutation itérée est lié au degré de la permutation inverse. Ce résultat a ainsi conduit à une borne plus générique sur le degré. Ce travail a été publié dans le journal *IEEE Transactions on Information Theory* [BC12b]. Les applications de cette borne aux chiffrements par blocs ont été également présentées à la conférence *The 10th International Conference on Finite Fields and their Applications Fq10 2011* [BC11a], tandis que les applications pour les fonctions de hachage ont été exposées à la conférence organisée par le NIST *The Third SHA-3 Candidate Conference* [BC12a]. Ces nouvelles bornes ont en parallèle

permis d'évaluer le degré algébrique de certains chiffrements par bloc à bas coût et de faire le lien entre un degré élevé et la facilité de retrouver la clé secrète du chiffrement à l'aide d'un solveur SAT. Cette dernière étude a été présentée au *33rd WIC Symposium on Information Theory in the Benelux* [GBGS12].

Un autre type de cryptanalyse exploitant les propriétés algébriques des boîtes-S est l'attaque en deuxièmes préimages contre la fonction de hachage Hamsi, presentée en 2010 par Thomas Fuhr [Fuh10]. Cette attaque repose sur l'existence de relations linéaires entre certaines entrées et certaines sorties de la fonction de compression, quand les autres sont fixées. Nous analysons au chapitre 6 les permutations non-linéaires qui sont vulnérables à ce type d'attaque. Cette analyse a conduit à l'amélioration de la méthode de [Fuh10].

La dernière partie de cette thèse concerne l'analyse de la sécurité physique de certains candidats du concours SHA-3. Plus précisément, nous avons analysé la résistance contre les attaques par canaux cachés de deux fonctions finalistes du concours. Cette étude a été présentée au colloque international *International Workshop on Trustworthy Embedded Devices-TrustED 2012* [BLV12].

Références

[BC10] Christina Boura et Anne Canteaut : A zero-sum property for the Keccak-f permutation with 18 rounds. *Dans International Symposium in Information Theory-ISIT'*10, pages 2488-2492. IEEE, 2010.

[BC11a] Christina Boura et Anne Canteaut : On the Algebraic Degree of Iterated Permutations. Finite Fields and Applications - Fq10, Gent, Belgium, July 2011.

[BC11b] Christina Boura et Anne Canteaut : Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256. *Dans* Alex Biryukov, Guang Gong et Douglas R. Stinson, éditeurs : *Selected Areas in Cryptography-SAC'10*, volume 6544 de *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011.

[BC12a] Christina Boura et Anne Canteaut : On the Algebraic Degree of some SHA-3 Candidates. *Dans* Proceedings of the Third SHA-3 Candidate Conference, Washington D.C, March 22-23, 2012.

[BC12b] Christina Boura et Anne Canteaut : On the influence of the algebraic degree of F^{-1} on the algebraic degree of $G \circ F$. *IEEE Transactions on Information Theory*, 2012. À paraître.

[BCD11] Christina Boura, Anne Canteaut et Christophe De Cannière : Higher-Order Differential Properties of Keccak and *Luffa*. *Dans* Antoine Joux, éditeur : *FSE'11*, volume 6733 de *Lecture Notes in Computer Science*, pages 252–269, 2011.

[BLV12] Christina Boura, Sylvain Lévêque et David Vigilant : Side-channel Analysis of Grøstl and Skein. In the Proceedings of the 2nd International Workshop on Trustworthy Embedded Devices, TrustED, IEEE Symposium on Security and Privacy, pages 16-26, May 2012.

[GBGS12] Vincent Grosso, Christina Boura, Benoît Gérard et François-Xavier Standaert : A Note on the Empirical Evaluation of Security Margins against Algebraic Attacks (with Application to Low Cost Ciphers LED and Piccolo). *Dans* Proceedings of the 33rd WIC Symposium on Information Theory in the Benelux, pages 52-59, Boekelo, The Netherlands, May 2012.

Première partie

Analyse de primitives symétriques

Chapitre 1

Introduction

1.1 Introduction à la cryptologie

La cryptologie est la science de la protection de données secrètes. Son nom provient du mot grec « κρυπτός », qui signifie « secret ». Cette science comprend deux branches majeures, celle de la cryptographie et celle de la cryptanalyse. La cryptographie consiste en la conception de toutes les méthodes et techniques visant à protéger la confidentialité, l'intégrité et l'authenticité des données échangées à travers un canal considéré comme non fiable. La cryptanalyse de l'autre côté correspond, dans le cas d'une méthode de chiffrement, au développement de techniques visant à décrypter des messages qui ont été chiffrés à l'aide d'une clé secrète inconnue.

On retrouve déjà des mécanismes cryptographiques dans l'Antiquité. Les Spartiates utilisaient au V^e siècle avant J.-C. un bâton de bois, appelé la « scytale », pour lire et écrire des messages secrets. Néanmoins, le plus fameux exemple d'utilisation des codes secrets à cette époque appartient à Jules César. Le « chiffre de César » était une méthode de chiffrement simple, basée sur le décalage des lettres de l'alphabet, qui a été utilisée par Jules César pour ses communications confidentielles, dans le but de protéger des secrets militaires.

Le développement de la cryptographie a été assez lent pendant les siècles suivants et la protection de communications militaires est restée le principal motif de l'utilisation des codes secrets. Divers codes ont été utilisés tout au long de l'histoire jusqu'au XIX^e siècle. La sécurité de tous ces systèmes était basée sur le secret de l'algorithme utilisé, une sécurité tout à fait illusoire. Si l'adversaire arrivait à trouver le mécanisme, souvent très complexe, derrière ce code, il était ensuite capable de décrypter toute communication ayant été chiffrée par ce code. August Kerckhoffs détecte alors le besoin de créer des systèmes dont la sécurité doit dépendre exclusivement d'un paramètre facilement modifiable : la clé. En 1883, il énonce alors le principe qui allait poser les bases de la cryptographie moderne : « La sécurité d'un cryptosystème doit résider dans le secret de la clé. Les algorithmes utilisés doivent pouvoir être rendus publics ».

Au cours du XX^e siècle, la cryptographie a joué un rôle majeur dans le déroulement des deux Guerres mondiales. En particulier pendant la Deuxième Guerre l'utilisation de la machine de chiffrement et de déchiffrement Enigma par l'Allemagne nazie et son cassage par les cryptanalystes polonais et anglais ont certainement influencé le résultat de la guerre.

Toutefois, l'invention de l'ordinateur et le développement des télécommunications et du réseau Internet ont montré le besoin de protection de toutes sortes de communications et

transactions, concernant la vie économique, publique et privée. Cette nouvelle ère de l'informatique a parallèlement créé de nouveaux besoins, auxquels la cryptographie moderne a dû répondre. Désormais, la cryptographie n'est plus seulement utilisée pour assurer la confidentialité des communications, le rôle historique qu'elle a dû tenir pendant des siècles. En effet, de nouvelles fonctionnalités ont dû être développées, afin de prouver l'identité d'un émetteur d'un message ou de tester l'intégrité d'un message envoyé.

La cryptographie moderne englobe aujourd'hui deux grandes familles. La première, qui comprend tous les algorithmes historiques, s'appelle la cryptographie *à clé secrète*. Dans ce type de cryptographie, si Alice et Bob souhaitent communiquer, ils doivent posséder la même clé qui va servir à chiffrer et à déchiffrer les messages échangés. C'est à cause de ce rôle *symétrique* de la clé que cette cryptographie est aussi connue sous le nom de *cryptographie symétrique*.

À cause de l'explosion du nombre des communications, le principal problème de la cryptographie est devenu l'échange des clés. Pour cette raison, un nouveau type de cryptographie, la cryptographie *à clé publique* ou cryptographie asymétrique est apparu. Dans ce modèle, chaque personne possède sa propre paire de clés, une clé publique, potentiellement connue par toutes les personnes et une clé privée, connue uniquement de son propriétaire. Si Alice souhaite envoyer un message à Bob, elle utilise la clé publique de Bob pour chiffrer le message. Seul Bob peut déchiffrer ce message en utilisant sa clé privée. Avec ce mécanisme, Alice et Bob peuvent communiquer entre eux sans avoir eu besoin d'échanger la clé secrète au préalable.

1.2 La cryptographie symétrique

La cryptographie symétrique, ou cryptographie à clé secrète, est la plus ancienne forme de cryptographie. Elle se base sur l'utilisation d'une clé différente pour chaque communication. Cette clé, qui sert à la fois à chiffrer et à déchiffrer les messages échangés, est connue par les deux personnes impliquées dans la communication.

La cryptographie symétrique comprend plusieurs types de constructions qui sont employés pour assurer la confidentialité, l'intégrité ou l'authenticité de la communication. Les principales constructions sont :

Les systèmes de chiffrement par bloc : Un chiffrement par bloc est une transformation inversible d'un bloc de taille fixe n paramétré par une clé secrète k. Pour chiffrer un message de taille quelconque, celui-ci est découpé en blocs de taille n et ces blocs sont ensuite chiffrés les uns après les autres. Pour chaîner les blocs chiffrés, un mode opératoire est utilisé. Le chiffrement par bloc le plus utilisé aujourd'hui est l'algorithme AES. Il a été conçu par Joan Daemen et Vincent Rijmen [DR00b, FIP01] et sélectionné comme norme internationale, lors d'une compétition publique organisée par le NIST. Il a ainsi remplacé le standard précédent, DES [FIP99], qui a dû être abandonné à cause de la taille trop courte de sa clé.

Les systèmes de chiffrement à flot : Le principe du chiffrement à flot, également appelé *chiffrement à la volée*, consiste en la combinaison du message clair avec une suite chiffrante de taille égale à la taille du message à chiffrer, par une opération de OU EXCLUSIF (XOR). Cette suite est générée à l'aide d'un générateur pseudo-aléatoire, qui a été initialisé avec une valeur secrète relativement courte, dérivée de la clé secrète du chiffrement. La sécurité du système est alors basée sur la qualité du générateur. Une suite chiffrante doit ressembler le plus possible à

une suite aléatoire, aucune propriété structurelle ne doit pouvoir être exploitée. Les systèmes de chiffrement à flot sont utilisés dans les contextes où il est primordial de pouvoir chiffrer et déchiffrer très rapidement et où les ressources matérielles, comme par exemple la taille du circuit ou la capacité de stockage, sont très restreintes. Ceci est la raison pour laquelle les chiffrements à flot sont implantés dans les téléphones mobiles et dans d'autres dispositifs embarqués.

Les fonctions de hachage : Les fonctions de hachage servent à calculer à partir d'une donnée de taille arbitraire fournie en entrée une empreinte de taille fixe. Cette taille varie en général entre 128 et 512 bits. Cette empreinte, appelée aussi *condensé* ou simplement *haché* doit dépendre de tous les bits du message et est utilisée pour représenter le message de façon compacte. Une fonction de hachage est un algorithme entièrement public et aucune valeur secrète n'intervient à aucun moment du calcul. Néanmoins, les fonctions de hachage appartiennent à la famille des algorithmes symétriques, car leur construction ressemble beaucoup à la construction d'un chiffrement par bloc. Une fonction de hachage doit se comporter idéalement comme une fonction aléatoire. En parallèle, de nombreux propriétés doivent être respectées. En particulier, il doit être difficile de trouver des collisions ou d'inverser la fonction. Les standards actuels sont les fonctions SHA-1 et SHA-2, mais certaines faiblesses sont connues pour cette famille de fonctions. Pour cette raison, un concours public a été lancé en 2008 afin de déterminer un nouveau standard. La fonction KECCAK a récemment remporté cette compétition et est ainsi devenue la nouvelle norme SHA-3.

Les codes d'authentification de message ou MAC : Ces mécanismes cryptographiques permettent à la fois d'assurer l'intégrité du message envoyé et d'authentifier son expéditeur. Pour réussir cela, lors de l'envoi du message, le code d'authentification est ajouté au message. Un MAC peut être construit à partir d'une fonction de hachage, comme par exemple la construction HMAC ou envelope MAC, à partir d'une fonction de hachage universelle (UMAC) ou encore d'un chiffrement par bloc, comme dans OMAC ou CBC-MAC.

1.3 Les fonctions de hachage

Une fonction de hachage H est une fonction qui prend en entrée une donnée de taille aléatoire m, et donne en sortie un condensé de taille fixe, n.

$$\begin{array}{rcl} H : \{0,1\}^* & \to & \{0,1\}^n \\ m & \mapsto & H(m) \end{array}$$

L'empreinte d'une donnée, c'est-à-dire son image par la fonction de hachage, sert à la représenter et permet facilement son identification.

Les fonctions de hachage sont des outils indispensables dans beaucoup de processus informatiques. Une de leurs premières utilisations a été la construction de structures des données, appelées *tables de hachage*. Une table de hachage est un tableau qui permet de stocker des données de natures diverses. Chaque donnée possède un identifiant et l'accès à un élément du tableau se passe par l'empreinte de cet identifiant, calculée à l'aide d'une fonction de hachage. Ceci permet la recherche en temps constant d'un élément dans une grande base de données.

Pour cette utilisation, une fonction de hachage ne doit avoir aucune propriété particulière en dehors d'une distribution proche de la distribution uniforme et une fonction assez simple peut être utilisée. Au contraire, il existe des applications, où des propriétés de sécurité supplémentaires sont exigées. Pour cela, les fonctions de hachage cryptographiques sont utilisées.

1.3.1 Fonctions de hachage cryptographiques

Les fonctions de hachage sont utilisées pour diverses applications, comme nous pourrons le voir dans la section suivante. Pour la plupart de ces applications, les données sont échangées à travers un canal non-fiable comme cela peut être le cas pour une ligne téléphonique ou une application web. Des personnes malveillantes sont alors susceptibles d'intercepter ou même de modifier cette communication. Pour cette raison, les fonctions de hachage utilisées doivent vérifier des propriétés de sécurité supplémentaires.

Une des utilisations principales des fonctions de hachage est la protection des mots de passe. Pour cette application il est crucial d'utiliser une fonction de hachage pour laquelle un adversaire ne soit pas capable de trouver un antécédent ayant une empreinte donnée. Pour d'autres utilisations, comme par exemple dans le cas des signatures numériques, un utilisateur ne doit pas être capable de produire à partir d'un message m un deuxième message m' ayant le même haché que m, ou de produire deux messages avec le même haché. Nous pourrons alors constater que le niveau de sécurité exigé pour une fonction de hachage dépend de son utilisation. Néanmoins, pour ne pas construire une fonction de hachage différente pour chaque application, il est courant de construire des fonctions de hachage qui soient sûres dans toutes les situations possibles.

De façon générale, l'empreinte d'un message produit avec une fonction de hachage cryptographique doit dépendre de tous les bits de message. En parallèle, une fonction de hachage cryptographique doit être un procédé assez complexe de façon que si un bit du message est modifié, le haché ne doit plus avoir aucune liaison avec le haché du message précédent.

Mis à part ces propriétés de conception génériques, une fonction de hachage est dite avoir des bonnes propriétés cryptographiques si elle est résistante aux préimages, aux secondes-préimages et aux collisions. Les trois problèmes suivants doivent donc être difficiles.

- *préimage :* étant donné un haché h choisi aléatoirement, trouver un message m tel que $H(m) = h$.
- *seconde préimage :* étant donné un message m choisi aléatoirement, trouver un message m' tel que $H(m) = H(m')$.
- *collision :* trouver deux messages m, m', tels que $m \neq m'$ et $H(m) = H(m')$.

Selon sa définition, une fonction de hachage est une fonction dont l'ensemble de départ est plus grand que l'ensemble d'arrivée. Théoriquement, l'ensemble de départ peut être infini, en pratique, l'ensemble de départ comprend généralement tous les messages d'une taille inférieure à un certain seuil. Par exemple, la fonction de hachage SHA-1 est capable de traiter des messages de taille inférieure à $2^{64} - 1$ bits.

L'existence de collisions est alors inévitable pour une fonction de hachage, H, donnant des empreintes de taille n ; si on choisit $2^n + 1$ messages distincts, il existe forcement une paire de messages aboutissant au même haché. De la même manière, si on restreint H à un domaine de taille 2^t et on considère que les sorties de H sont uniformément distribuées, alors un condensé aléatoire h possède environ 2^{t-n} préimages.

On définit alors la résistance d'une fonction aux collisions, aux préimages et aux deuxièmes

préimages par rapport à la difficulté de résoudre ces problèmes en pratique. Cette difficulté est évaluée par rapport au nombre d'opérations nécessaires pour que la meilleure attaque générique contre une fonction de hachage idéale réussisse.

Ainsi, un attaquant ne doit pas être en mesure de trouver une préimage en moins de $\mathcal{O}(2^n)$ opérations, puisque la meilleure attaque générique consiste à essayer 2^n messages distincts pour avoir une bonne probabilité de réussite. En suivant le même raisonnement, il ne doit pas être possible de trouver une deuxième préimage en moins de $\mathcal{O}(2^n)$ opérations.

Pour la recherche de collisions, la probabilité de réussite de la meilleure attaque générique repose sur le *paradoxe des anniversaires*.

Paradoxe des anniversaires :

Ce paradoxe désigne un phénomène contre-intuitif : dans un ensemble de 23 personnes choisies aléatoirement, la probabilité que deux personnes fêtent leur anniversaire le même jour de l'année est supérieure à 1/2. Ce comportement inattendu peut néanmoins être expliqué en suivant le raisonnement suivant.

Soit k éléments $x_1, x_2, \ldots, x_k$ tirés uniformément et indépendamment dans un ensemble E de taille n. La probabilité que tous les x_i soient distincts est

$$\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\cdots\left(1 - \frac{k-1}{n}\right) = \frac{n!}{(n-k)!} \cdot \frac{1}{n^k}.$$

Par conséquent, la probabilité qu'au moins deux éléments soient identiques est

$$p = 1 - \frac{n!}{(n-k)!} \cdot \frac{1}{n^k} \approx 1 - e^{-\frac{k(k-1)}{2n}}.$$

De ce fait, pour le cas des anniversaires, on peut constater que pour $n = 365$ et $k = 23$, cette probabilité devient proche de 1/2.

En appliquant ce principe dans le cas des fonctions de hachage, pour trouver une collision dans un ensemble de taille 2^n, il faudra essayer $2^{n/2}$ valeurs distinctes pour produire une collision avec une probabilité supérieure à 1/2. Cette attaque générique qui est due à G. Yuval [Yuv97] nécessite $2^{n/2}$ calculs dans le pire cas, mais aussi une mémoire de $2^{n/2}$, ce qui peut être assez contraignant pour certaines applications. Pour contourner ce problème, l'algorithme ρ de Pollard [Pol75] peut être utilisé pour rechercher des collisions. Des versions parallélisables existent également dans la littérature [QD90, vOW99]. Enfin, une généralisation du problème à $k > 2$ éléments a été publiée par David Wagner [Wag02].

Les complexités des attaques génériques pour le cas des collisions, des préimages et des deuxièmes préimages sont rassemblées à la table 1.1

Attaque générique	**Complexité**
Recherche de préimages	2^n
Recherche de deuxièmes préimages	2^n
Recherche de collisions	$2^{\frac{n}{2}}$

TABLE 1.1 – Complexité des meilleures attaques génériques.

La taille des empreintes construites par une fonction de hachage est choisie de façon que le nombre d'opérations nécessaire pour une attaque générique réussie soit inatteignable avec la

puissance calculatoire actuellement accessible. Aujourd'hui, on peut considérer qu'un nombre d'opérations supérieur ou égal à 2^{80} est trop grand pour être réalisé en pratique. De cette façon, une fonction de hachage donnant en sortie des hachés de taille 160 bits est aujourd'hui considérée comme sûre contre toutes les attaques génériques. Néanmoins, pour garantir une certaine marge de sécurité, il est demandé que la taille de sortie des fonctions de hachage modernes varie entre 256 et 512 bits.

Une fonction de hachage est considérée comme « cassée » si une attaque plus rapide que la meilleure attaque générique existe pour cette fonction, même si sa complexité ne lui permet pas d'être réalisée en pratique. En effet, cela signifie que la fonction a des faiblesses, qui n'étaient pas prévues par le concepteur.

Relation entre les différentes notions de sécurité

Des relations entre les différentes propriétés existent. Si une fonction de hachage est résistante aux collisions, alors elle est aussi résistante aux deuxièmes préimages. Ceci s'explique par le fait que si on connaît une méthode pour construire des deuxièmes préimages pour une fonction on peut l'utiliser aussi pour construire des collisions. Cependant, si on connaît une attaque en deuxième préimage, celle-ci s'appliquera aussi aux collisions seulement dans le cas où sa complexité est inférieure à $2^{\frac{n}{2}}$, à cause de la différence dans les complexités des attaques correspondant.

De la même façon, si une fonction est résistante aux préimages, alors elle l'est aussi pour les secondes préimages, car on peut toujours utiliser une attaque en secondes préimages pour construire une préimage en ignorant simplement le premier message. La réciproque n'est pas vraie et des contre-exemples existent [MvOV97].

Une définition formelle de ces notions de sécurité et leurs relations sont données dans [RS04].

1.3.2 Utilisations

Les fonctions de hachage cryptographiques sont utilisées pour assurer des besoins informatiques divers. Cette primitive, qui possède beaucoup de fonctionnalités, est considérée comme le « couteau suisse » de la cryptographie.

Intégrité des données

La vérification de l'intégrité d'une donnée est parmi les principales utilisations d'une fonction de hachage. Un utilisateur doit être capable de vérifier si une donnée n'a pas été modifiée depuis sa création ou pendant sa transmission à travers un canal de communication.

Beaucoup de sites de téléchargement de logiciels affichent sur leur page principale les empreintes des fichiers proposés au téléchargement, calculées au moyen d'une fonction de hachage. Il suffit pour l'utilisateur de télécharger un fichier, calculer son haché et comparer la valeur calculée à celle affichée sur le site. Si la fonction de hachage utilisée est connue pour être résistante aux deuxièmes préimages, alors l'utilisateur peut être sûr avec une grande probabilité qu'il détient le bon fichier.

Protection de mots de passe

Dans beaucoup de systèmes informatiques, l'authentification d'un utilisateur se fait à partir d'un mot de passe. Les mots de passe de tous les utilisateurs doivent être stockés

dans le système et à chaque requête une vérification est faite pour permettre l'accès aux utilisateurs légitimes. Toutefois, le stockage des mots de passe en clair peut engendrer de sérieux problèmes. Un attaquant qui arrive à récupérer le contrôle du système sera ensuite capable de s'authentifier sous l'identité de n'importe quel utilisateur.

Pour éviter une telle situation, le haché de chaque mot de passe est calculé et conservé dans le système au lieu du mot de passe lui-même. De cette façon, pour chaque tentative d'authentification, le mot de passe que l'utilisateur insère est haché et le résultat est comparé avec la valeur sauvegardée dans le système. Si la comparaison est réussie alors l'utilisateur a le droit de profiter des services du système, autrement l'accès est interdit. Même si un attaquant arrive à gagner le contrôle du système, il arrivera seulement à récupérer les hachés des mots de passe. Pour retrouver les valeurs cachées derrière les empreintes il devra être capable de retrouver des préimages de ces valeurs. Pour cela, une fonction de hachage résistante aux préimages doit être utilisée.

Il n'est pas rare qu'une grande partie des utilisateurs choisissent comme mots de passe des mots courants, ne contenant donc pas suffisamment d'entropie. Pour éviter alors les attaques de type *dictionnaire*, où tous les mots du dictionnaire sont testés les uns après les autres, dans plusieurs systèmes la fonction de hachage est itérée plusieurs fois sur le mot de passe. Ce nombre est par exemple fixé à 1000 pour beaucoup d'applications. Des instructions et des recommandations sur la manière de protéger des mots de passe pour les applications cryptographiques peuvent par exemple être trouvées dans PKCS # 5 (RFC 2898) [Kal00].

Signatures électroniques

Une des applications les plus importantes des fonctions de hachage est leur utilisation dans les signatures électroniques. La signature électronique est un mécanisme analogue de la signature manuscrite permettant de garantir l'intégrité d'un document électronique et prouvant au lecteur du document l'identité de son auteur. De tels mécanismes utilisent les procédés de la cryptographie asymétrique.

Si Alice veut garantir l'authenticité d'un message à Bob, elle le signe avec sa clé privée d_A. Ensuite, Bob peut utiliser la clé publique d'Alice, e_A, pour vérifier la signature. Cette opération consiste à vérifier que la signature a été bien produite par d_A et le message original.

Un algorithme asymétrique comme RSA, ElGamal, DSA ou encore le ECDSA basé sur les courbes elliptiques, peut être utilisé pour atteindre cet objectif. Un problème courant intervient quand la taille des données à signer est grande. Dans ce cas, la procédure de chiffrement avec un système asymétrique peut devenir très longue. De plus, la taille de la signature elle-même peut devenir déraisonnablement grande. Pour cette raison, les fonctions de hachage sont utilisées.

Lorsque Alice veut envoyer un message signé m à Bob elle suit la procédure suivante. Elle calcule l'empreinte de m à l'aide d'une fonction de hachage H, $H(m)$. Ensuite, elle applique la signature σ sur le haché. Enfin, le résultat $\sigma(H(m))$ est envoyé à Bob. Bob peut maintenant vérifier qu'Alice a bien produit m en utilisant sa clé publique.

Pour ce type d'utilisation on demande qu'il ne soit pas possible de produire des collisions ou de secondes préimages pour H. Si Charlie est capable de générer deux messages, m, m' tels que $m \neq m'$ et $H(m) = H(m')$, il peut donner à signer à Alice m' au lieu de m et produire ainsi une signature légitime à partir d'un message contrefait, sans que Alice puisse s'en apercevoir. De même, si Alice est capable de produire des collisions ou de secondes préimages, elle peut utiliser une collision pour nier plus tard qu'elle a produit un message

(problème de la *répudiation*).

Protocoles d'engagement

Un protocole d'engagement permet à une personne de s'engager sur une valeur ou sur un message m sans dévoiler son contenu dans l'immédiat. Pour ceci il suffit de calculer le haché du message et ensuite diffuser $H(m)$. Le haché du message ne révèle aucune information sur le message même. Par contre, quand le message sera révélé il suffira de calculer son haché et de le comparer avec la valeur préalablement diffusée.

On peut supposer que le secret que nous souhaitons temporairement protéger est un message simple, par exemple une valeur numérique, réponse à un problème de mathématiques difficile. Pour éviter qu'une personne connaissant le haché du secret puisse essayer quelques valeurs susceptibles d'être la bonne réponse, il est souvent courant de hacher le message concaténé avec une valeur secrète aléatoire k, c'est-à-dire calculer $H(m||k)$. A la révélation du secret, la transmission de m et k suffiront pour vérifier la vérité.

Si la fonction de hachage qui a été utilisée pour cette application est résistante aux collisions et aux deuxièmes préimages, alors on peut être sûr de la valeur révélée.

Une autre application du même type est l'*horodatage* ou *timestamping* en anglais. Il s'agit d'un mécanisme qui associe une heure et une date à un document numérique. Parfois, nous pouvons avoir besoin de prouver qu'un document a déjà été créé à un certain moment dans le temps. Ceci peut par exemple arriver pour prouver que nous possédions la description d'une certaine invention avant la dévoiler au public. Dans ce cas également, une fonction de hachage peut être utilisée pour protéger le message.

Générateurs pseudo-aléatoires

Un générateur pseudo-aléatoire de nombres est un algorithme capable de produire une suite de bits qui a les propriétés d'une suite générée aléatoirement, comme l'indépendance et la distribution uniforme des bits. La production de telles suites est nécessaire pour diverses applications cryptographiques, comme par exemple la signature électronique ou la génération des clés pour un chiffrement asymétrique.

Les bonnes propriétés statistiques des fonctions de hachage sont souvent exploitées pour cela. Par exemple, si H est une fonction de hachage, initialisée avec une graine c, il est possible de construire une suite pseudo-aléatoire à partir de la suite récurrente $x_i = H(x_{i-1}||0)$, où $x_0 = c$, de laquelle on extrait des bits pseudo-aléatoires.

Dérivation de clé

Dans le cadre de la cryptographie symétrique, plusieurs clés secrètes sont souvent utilisées pour assurer une communication entre deux entités. Au lieu de transmettre toutes ces valeurs à travers un canal de communication, qui parfois peut être peu fiable, une méthode consiste à dériver toutes les clés secrètes nécessaires à partir d'une seule valeur secrète, d'une clef maître ou d'un mot de passe. L'utilisation d'une fonction de hachage pour cette fonctionnalité assure que même si un adversaire détient une des clés dérivées, il n'est pas capable d'obtenir des informations sur la clé maître, puisque les fonctions de hachage sont des fonctions à sens unique.

1.3.3 Les modes opératoires

Une fonction de hachage doit être capable de traiter des messages de taille arbitraire. Construire directement une fonction qui compresse des données de taille quelconque peut être une procédure assez complexe. Une façon beaucoup plus simple pour mettre cela en pratique est de découper le message en blocs de taille fixe et de travailler avec les blocs de façon itérative. Cette idée a été présentée pour la première fois par Rabin en 1978 [Rab78]. Ainsi, une fonction de hachage peut être constituée de deux mécanismes distincts. Le premier mécanisme est une transformation qui permet de traiter des données de taille fixe. Cette transformation est très souvent une *fonction de compression* ou une *permutation*. Le deuxième mécanisme, appelé *mode opératoire* ou *algorithme d'extension de domaine* est une construction qui permet de lier entre eux les résultats de la transformation interne. Un bon algorithme d'extension de domaine doit idéalement préserver les propriétés cryptographiques de la fonction interne.

Nous allons voir maintenant les constructions les plus connues.

Construction de Merkle-Damgård

En 1989, le premier algorithme d'extension de domaine a été indépendamment développé par Ralph Merkle [Mer90] et Ivan Damgård [Dam90] et est aujourd'hui connu sous le nom de « construction de Merkle-Damgård ». Pour cette méthode, une fonction de compression est utilisée de façon itérative. Le message m, après avoir été préparé, est découpé en blocs de taille fixe $m_1, \ldots, m_k$ et la fonction de compression h traite les blocs les uns après les autres, comme nous pouvons le voir sur la figure 1.1.

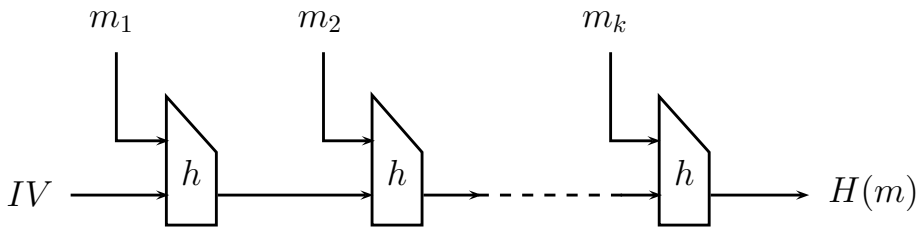

FIGURE 1.1 – Construction de Merkle-Damgård

Dans une première étape, un algorithme injectif `pad`, appelé *rembourrage* ou *padding* en anglais, est appliqué au message. Ceci consiste en la concaténation du message avec une suite de bits de façon que la taille finale soit multiple de la taille t d'un bloc. Le caractère injectif de cet algorithme est nécessaire pour éviter les collisions.

Il existe plusieurs façons de rembourrer un message m, mais la plupart des fonctions connues, comme les fonctions de la famille MD ou SHA, utilisent la procédure suivante. Le bit '1' est d'abord ajouté au message, suivi d'un nombre de bits valant '0'. Le nombre de '0' doit être le plus petit nombre tel que la longueur ℓ de la donnée obtenue après cette opération vérifie $\ell \equiv t - u \mod t$, où u est une valeur constante, souvent égale à 64. Enfin, la longueur de m est codée en u bits et est concaténée au résultat :

$$\texttt{pad}(m) = m||10\ldots00||\ell.$$

De façon générale, le rembourrage avec ajout de la longueur du message est connu sous le nom « MD-strengthening ».

Après la préparation du message, $\texttt{pad}(m)$ est divisé en k blocs $m_1, m_2, \ldots, m_k$ de longueur t. A chaque instant i, le bloc de message m_i entre dans la fonction de compression h, accompagné de la valeur de chaînage h_{i-1}, c.à.d. de la sortie de la fonction de compression précédente. La fonction h mettra alors à jour la valeur de chaînage :

$$h_i = h(h_{i-1}, m_i),$$

où h_0 est une valeur publique initiale, notée IV (pour *initial value* en anglais). Une fois tous les blocs de message traités, la dernière valeur de chaînage h_k désignera le haché du message m. Il est aussi possible d'appliquer à h_k une *fonction de finalisation* afin d'obtenir la valeur du haché, mais cette étape est souvent omise en pratique.

Cet algorithme d'extension de domaine a été largement utilisé par les fonctions de hachage de la première génération (MD4, MD5, SHA-0 et SHA-1). Son succès vient de sa preuve de sécurité très simple et utile. En particulier, il est possible de prouver que si la fonction de compression est résistante aux collisions, alors c'est le cas aussi pour la fonction de hachage elle-même [Mer90, Dam90].

L'attaque par extension de longueur Cette attaque, qui est une des faiblesses principales de la construction Merkle-Damgård, permet de construire à partir de l'empreinte $H(m)$ d'un message m, le haché d'un message m' qui est suffixe de m, sans connaître le message lui-même.

Soient m et m' deux messages, tels que $\texttt{pad}(m)$ soit un préfixe de $\texttt{pad}(m')$. Il existe alors des indices k, ℓ tels que :

$$\begin{aligned} \texttt{pad}(m) &= m_1||m_2||\ldots||m_k \\ \texttt{pad}(m') &= \texttt{pad}(m)||m_{k+1}||\ldots||m_\ell. \end{aligned}$$

On peut maintenant construire le haché de m' en fonction de $h_k = H(m)$ en calculant la suite :

$$h_i = h(h_{i-1}, m_i) \text{ avec } i \geq k+1.$$

Cette attaque qui ne menace pas la sécurité en préimages ou en collisions des fonctions utilisant la construction Merkle-Damgård, pose pourtant un vrai problème pour les codes d'authentification (MAC). Plus précisément, à cause de cette faiblesse la construction MAC la plus naturelle, n'est pas sûre, obligeant l'utilisation de constructions plus complexes.

Ce problème peut être contourné en ajoutant une fonction de finalisation non-réversible à la fin du calcul [CDMP05], ou un rembourrage dit "sans préfixe", pour lequel il n'existe aucun couple (m, m') tel que $\texttt{pad}(m)$ est un préfixe de $\texttt{pad}(m')$.

Multi-collisions Une k-multi-collision est un ensemble de k messages ayant tous le même haché. La complexité de la construction des k-multi-collisions pour une fonction de hachage générique qui produit des hachés de taille n, est $2^{\frac{n\cdot(k-1)}{k}}$. C'est une conséquence directe d'un résultat de Girault et Stern [GS94]. Cette complexité devient très proche de 2^n quand k est suffisamment grand. Néanmoins, Antoine Joux a prouvé en 2004 [Jou04] que pour une fonction itérée, la complexité de la recherche de multi-collisions devient étonnement faible.

Pour construire une 8-multi-collision pour une fonction de hachage basée sur une fonction de compression h, nous pouvons procéder comme suit. Nous trouvons d'abord deux messages m_0, m_0', de la taille d'un bloc, tels que $h(IV, m_0) = h(IV, m_0') = z_0$. Nous cherchons ensuite deux autres blocs de message m_1, m_1', tels que $h(z_0, m_1) = h(z_0, m_1')$. Nous appelons cette valeur commune z_1 et nous cherchons deux messages m_2, m_2', tels que $h(z_1, m_2) = h(z_1, m_2')$. De cette façon nous avons produit la 8-multi-collision suivante :

$$\begin{aligned} h(h(h(IV, m_0), m_1), m_2) &= h(h(h(IV, m_0'), m_1), m_2) \\ = h(h(h(IV, m_0), m_1'), m_2) &= h(h(h(IV, m_0'), m_1'), m_2) \\ = h(h(h(IV, m_0), m_1), m_2') &= h(h(h(IV, m_0'), m_1), m_2') \\ = h(h(h(IV, m_0), m_1'), m_2') &= h(h(h(IV, m_0'), m_1'), m_2') \end{aligned}$$

Cette 8-multi-collision est représentée à la figure 1.2 La complexité de cette 8-multi-

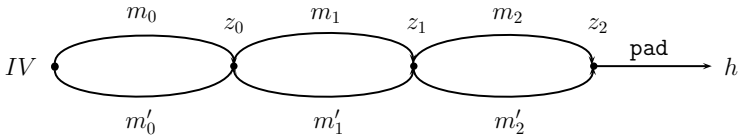

FIGURE 1.2 – Une 8-multi-collision

collision est $3 \cdot 2^{\frac{n}{2}}$ calculs de la fonction de compression, où n est la taille de l'état interne. Pour une fonction suivant le principe de Merkle-Damgård, la taille de l'état interne est égale à la taille du haché, donc la complexité de la recherche des multi-collisions est relativement faible. Pour cette raison les fonctions construites selon le mode Merkle-Damgård sont très sensibles à ce type d'attaque.

Dans le cas général, l'algorithme 1 permet de produire des 2^k-multi-collisions pour une fonction itérée f, avec une complexité moyenne de $k \cdot 2^{\frac{n}{2}}$, où n est la taille de la fonction interne.

Algorithme 1: Recherche des k-multi-collisions

Données : La valeur initiale de la fonction de hachage IV
Résultat : 2^k messages différents $m_1, m_2, \ldots, m_{2^k}$ dont
$H(m_1) = H(m_2) = \cdots = H(m_{2^k})$
$h_0 = IV$;
pour *i de 1 à k* **faire**
  Trouver m_i, m_i' tels que $h(h_{i-1}, m_i) = h(h_{i-1}, m_i')$;
  Affecter $h_i = h(h_{i-1}, m_i)$;

La construction *wide pipe*

Afin d'éviter les attaques par extension de longueur, les multi-collisions et d'autres vulnérabilités du mode Merkle-Damgård, Stephan Lucks a proposé dans [Luc05] un nouveau mode opératoire, appelé *wide pipe*. La complexité des attaques mentionnées dépend de la taille de l'état interne. Dans la construction Merke-Damgård, la taille de l'état est égale à la taille de l'empreinte, conduisant à des attaques plus efficaces que les attaques génériques sur une fonction idéale. La construction *wide pipe*, qui est assez similaire au mode Merkle-Damgård,

permet de calculer des hachés de taille n en utilisant une fonction de compression f dont la taille de l'état interne, b, est plus grande que n, $b \gg n$:

$$\begin{array}{rcl} h : \mathbf{F}_2^b \times \mathbf{F}_2^t & \rightarrow & \mathbf{F}_2^b \\ (h_{i-1}, m_i) & \mapsto & h_i \end{array}$$

Une deuxième fonction de compression G, $G : \mathbf{F}_2^b \rightarrow \mathbf{F}_2^n$ est ensuite appliquée au résultat pour produire un haché de taille n (figure 1.3).

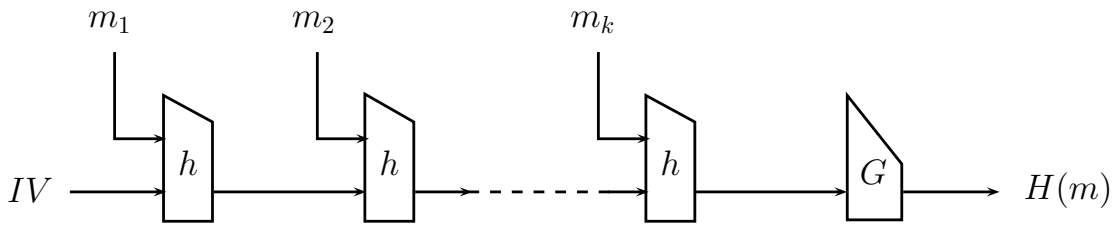

FIGURE 1.3 – Construction *wide pipe*

Il existe plusieurs instances de la construction *wide pipe*. Parmi les plus connues, il y a le mode *double pipe*, définie par $b = 2n$, et le mode *Chop-MD*, où la fonction G est une simple troncation. Les fonctions avec $b = n$ sont aussi connues sous le nom *narrow pipe*.

La construction éponge

Une autre construction introduite récemment, appelée construction *éponge* est due à Bertoni *et al.* [BDPV08]. Il s'agit d'une construction itérée capable de produire des sorties de taille arbitraire. Contrairement à la construction Merkle-Damgård qui est basée sur une fonction de compression, la construction éponge repose sur une transformation de taille fixe, c'est-à-dire sur une permutation f, opérant sur les mots de b bits. Dans la pratique, toutes les instances connues de cette construction, sont basées sur une permutation.

Le hachage d'un message m se déroule de la manière suivante. On commence par rembourrer le message avec un padding injectif et on découpe ensuite le résultat de cette opération en blocs $m_1, \ldots, m_k$ de taille t. Ensuite, les b bits de l'état interne sont initialisés avec la valeur '0'. La procédure se déroule en deux étapes successives.

- *L'étape d'absorption :* Dans cette étape, les blocs du message rembourré sont « absorbés » de façon itérative. Le premier bloc m_1 est combiné à l'aide d'un OU exclusif (XOR) avec l'état. La transformation f est ensuite appliquée au résultat de cette opération. Puis, le deuxième bloc de message m_2 est ajouté à l'état et la transformation f est de nouveau appelée. La même procédure est répétée jusqu'à ce que tous les blocs de message soient absorbés.
- *L'étape de pressage :* Pendant cette étape, des blocs de l'état interne sont extraits à des endroits séparés par des applications de la transformation f. La taille des blocs extraits, peut être choisie par l'utilisateur.

Ces deux procédures sont résumées dans la figure 1.4.

La sécurité des fonctions éponges est associée aux paramètres de la construction et plus précisément à la *capacité* c, définie par la relation $c = b - t$. Il est possible par exemple de prouver que pour une instance de la construction éponge qui produit des empreintes de taille

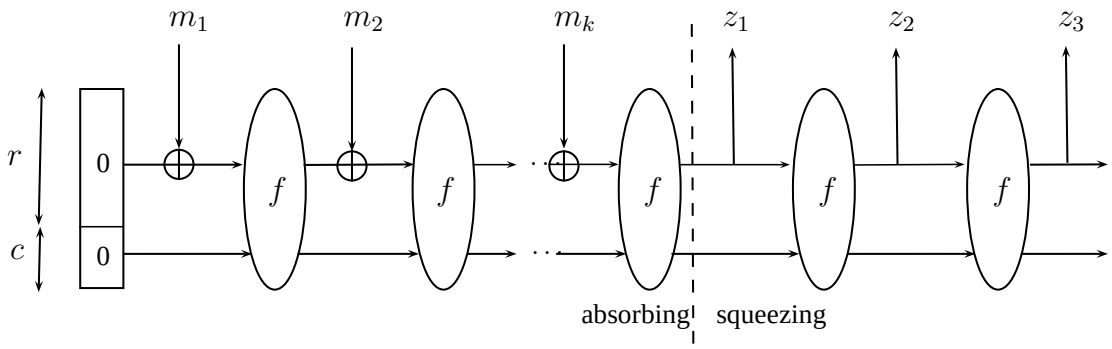

FIGURE 1.4 – La construction éponge

n, la résistance de la fonction aux collisions est donnée par $\min(2^{n/2}, 2^{c/2})$ évaluations de la fonction interne [BDPV08]. De cette façon, l'utilisateur peut sélectionner la valeur de c en fonction du niveau de sécurité souhaité ; plus la valeur de c est grande, plus le niveau de sécurité atteint est élevé, mais plus la performance de la fonction est pauvre. Pour une même taille de permutation b, plusieurs compromis sécurité/performance existent.

La construction éponge a été utilisée comme algorithme d'extension de domaine pour plusieurs fonctions de hachage récemment introduites. Parmi elles, on trouve le nouveau standard SHA-3 et les fonctions de hachage à bas coût, `Quark, Photon` et `Spongent`. D'autres fonctions de hachage, comme les fonctions `Luffa` ou `CubeHash` sont construites avec des algorithmes dérivés.

Certaines fonctions construites avec la construction éponge sont basées sur une stratégie appelée *hermetic sponge strategy* en anglais. Elle consiste dans l'instanciation d'une éponge avec une permutation qui ne doit avoir aucune propriété exploitable.

Autres constructions

Plusieurs autres constructions ont été proposées pour corriger les faiblesses du mode Merkle-Damgård. Parmi elles se trouvent des généralisations de Merkle-Damgård et des modes novateurs. Nous pouvons par exemple citer la construction HAIFA, présentée par Eli Biham et Orr Dunkelman [BD06], ainsi que la construction EMD [BR06].

1.3.4 Fonctions de compression fondées sur un algorithme de chiffrement par blocs

Les fonctions de compression sont au cœur des fonctions de hachage itérées. Leur construction est un problème difficile et il existe plusieurs manières pour y parvenir. Nous n'allons pas détailler dans ce mémoire les différentes types de construction. Néanmoins, nous allons brièvement introduire les constructions les plus importantes, puisque plusieurs fonctions de hachage du concours SHA-3, que nous allons examiner par la suite, les utilisent.

Construire une fonction de compression basée sur un chiffrement par bloc est un concept assez intéressant, car les propriétés standard d'une fonction de hachage (résistance en préimages, en deuxièmes préimages et en collisions) peuvent être déduites en supposant la primitive interne idéale.

Nous commençons néanmoins par montrer que la construction la plus intuitive présente des problèmes de sécurité évidents. Une fonction de compression h prend en entrée une valeur de chaînage h_{i-1} et un message m_i et donne en sortie une nouvelle valeur de chaînage, h_i :

$$h(h_{i-1}, m_i) = h_i. \tag{1.1}$$

Nous souhaitons toutefois que la fonction de compression ne soit pas inversible, c'est-à-dire la connaissance de h_{i-1} et h_i ne permet pas de trouver un message m_i tel que l'équation (1.1) soit vérifiée. Si E_K est un chiffrement par bloc paramétré par une clé K, la construction la plus intuitive pour parvenir à cela consiste à utiliser le message m_i comme clé et à calculer

$$h(h_{i-1}, m_i) = E_{m_i}(h_{i-1}) = h_i.$$

Toutefois, cette construction présente une faiblesse fondamentale, qui nécessite d'utiliser une variable de chaînage 2 fois plus large que le haché, autrement dit un chiffrement par bloc ayant une taille de bloc $b = 2n$. En connaissant h_i et en utilisant la fonction de déchiffrement, soit D_K, avec un message m_i quelconque comme clé, nous pouvons trouver facilement une valeur h_{i-1} qui vérifie l'équation (1.1). Cette faiblesse peut alors conduire à une attaque en préimage à IV choisi sur la fonction de hachage et peut également, en combinant avec d'autres techniques, être étendue à une attaque en deuxième préimage. Il suffit pour cela de considérer $2^{b/2}$ blocs de messages aléatoires, m, (nous supposons ici que la taille des blocs de message t vérifie $t \geq b/2$, sinon il faut considérer des messages de plusieurs blocs), et nous calculons pour chacun d'eux $x = E_m(h_0)$. De même, nous choisissons $2^{b/2}$ autres blocs de message m' aléatoires et on calcule $y = D_{m'}(h_k)$. Avec probabilité $1/2$ nous trouvons alors un couple (m, m') tel que $E_m(h_0) = D_{m'}(h_k)$, ce qui signifie que $(m||m')$ a le même haché que le message connu qui produisait h_k.

Pour pallier ce problème, des solutions plus complexes ont dû être utilisées. Les premières constructions sont dû à Davies et Meyer et Matyas, Meyer et Oseas. Toutes les constructions de ce type ont été analysées par Preneel *et al.* [PGV94]. Dans cette approche générale, les auteurs étudient toutes les constructions simples de la forme,

$$h_i = h(h_{i-1}, m_i) = E_{x_1}(x_2) \oplus x_3,$$

où x_1, x_2 et x_3 sont des combinaisons linéaires de deux entrées h_{i-1} et m_i. Il existe alors, $(2^2)^3 = 64$ schémas possibles. Parmi ces constructions, seulement 49 font intervenir à la fois la variable de chaînage et le bloc de message. Pour les schémas restants, des faiblesses ont été démontrées pour 37 modes et la sécurité des 12 autres candidats a été conjecturée. Cette conjecture a été finalement validée par Black *et al.* [BRS02] quelques années plus tard, dans le modèle de la *boîte noire*. Dans ce modèle de sécurité, les primitives internes sont considérées comme des permutations aléatoires. L'attaquant n'a pas le droit d'utiliser des propriétés spéciales de leur structure.

Ces douze constructions, sont usuellement appelées PGVi, pour $1 \leq i \leq 12$. Parmi ces constructions, les trois plus utilisées, sont les schémas Davies-Meyer (PGV5), Matyas-Meyer-Oseas (PGV1) et Miyaguchi-Preneel (PGV7), qui sont présentés à la figure 1.5.

Dans la construction Davies-Meyer, chaque bloc de message m_i joue le rôle de la clé du chiffrement tandis que chaque valeur de chaînage h_{i-1} prend la place du bloc de message à chiffrer. Cette valeur est également additionnée au résultat du chiffrement :

$$h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}.$$

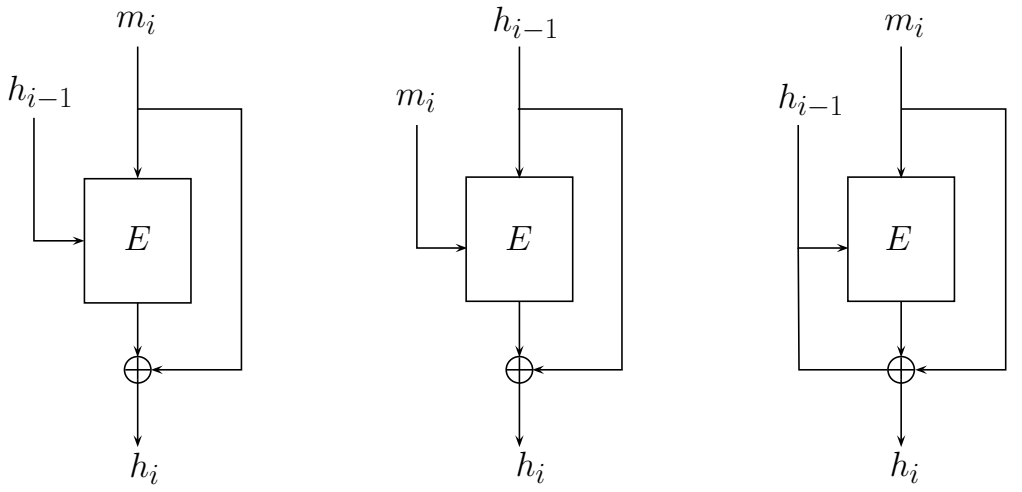

FIGURE 1.5 – Les schémas les plus connus pour la construction d'une fonction de compression à partir d'un chiffrement par blocs

Une vulnérabilité de ce schéma est que même si le chiffrement par blocs sous-jacent se comporte de façon idéale, il est possible de trouver des *points fixes*, c'est-à-dire des valeurs de chaînage h, tels que $h = E_{m_i}(h) \oplus h$. Pour ceci il suffit de considérer $h = D_{m_i}(0)$. La possibilité de construire facilement des points fixes est une propriété que les fonctions aléatoires ne possèdent pas. Certaines attaques en seconde préimage de Kelsey et Schneier [KS05] exploitent ces faiblesses.

La construction Davies-Meyer est utilisée pour construire plusieurs fonctions de hachage connues, comme les fonctions MD4, MD5, SHA-0, SHA-1 ainsi que la famille SHA-2.

Le schéma Matyas-Meyer-Oseas [MMO85] peut être vu comme le dual de la construction Davies-Meyer, car le rôle du message et de la valeur de chaînage y sont inversés :

$$h_i = E_{h_{i-1}}(m_i) \oplus m_i.$$

Finalement, le schéma Miyaguchi-Preneel, proposé de façon indépendante par Shoji Miyaguchi *et al.* [MOI90] et Bart Preneel [Pre93], est une variante généralisée de la construction Matyas-Meyer-Oseas :

$$h_i = E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1}.$$

1.3.5 Conception des primitives internes

La construction des primitives internes des fonctions de hachage que nous allons étudier pendant cette thèse, suit de façon générale les modèles de construction des chiffrements par blocs. Ces deux familles sont très proches et c'est pour cette raison que les fonctions de hachage sont considérées comme faisant partie de la cryptographie symétrique, malgré l'absence de la clé.

Les principes de conception des primitives internes que nous allons analyser sont donc décrits dans la section introductive aux chiffrements par blocs (Section 1.4).

1.3.6 Code d'authentification ou MAC

Pour les applications où les informations circulent à travers un canal de communication non-fiable, le récepteur d'un message doit être capable de vérifier un certain nombre d'informations. Dans un premier temps, afin d'éviter les utilisateurs malveillants, l'identité du récepteur doit être prouvée. Ensuite, dans un deuxième temps, l'intégrité du message reçu doit être examinée. Ces deux fonctionnalités peuvent être regroupées dans un seul mécanisme, appelé *code d'authentification de message* ou MAC. Un MAC peut être vu comme l'équivalent symétrique d'une signature électronique à la différence près qu'il ne permet pas la non-répudiation.

Pour authentifier un message m, Alice et Bob doivent partager une clé secrète, K. Alice calcule $\text{MAC}_K(m)$ et envoie m accompagné du résultat à Bob. Un MAC doit être difficile à contrefaire, c'est-à-dire il doit être difficile de construire un code légitime sans la connaissance de la clé. Bob reçoit le message m et calcule $\text{MAC}_K(m)$ à son tour. Si le résultat du calcul correspond au code transmis, il peut être à la fois sûr de l'identité d'Alice et de l'intégrité du message.

Une méthode intuitive pour construire un code d'authentification pour un message m en utilisant une fonction de hachage H est de concaténer m à K et calculer $H(K||m)$. Cette construction très simple, appelée *secret prefix* MAC ne peut cependant pas être utilisée avec des fonctions dont l'extension de domaine est la construction Merkle-Damgård, à cause de l'*attaque par extension de longueur* (Section 1.3.3) connue pour ce mode. Plus précisément, à cause de cette faiblesse, il est possible de construire à partir de deux messages m et m', tels que m est préfixe de m' et de la valeur de $\text{MAC}_K(m)$, un code légitime $\text{MAC}_K(m')$ sans connaître la clé K.

Pour cette raison, les constructions de codes d'authentification doivent avoir une forme plus complexe.

Construction HMAC

La construction de MAC la plus utilisée est la construction HMAC. Elle a été introduite par Bellare *et al.* [BCK96]. Elle a été standardisée par plusieurs organismes [KBC97, FIP02] et utilisée dans des nombreux protocoles.

Le mode HMAC basé sur une fonction de hachage H est défini de la façon suivante :

$$\text{HMAC-}H(K, m) = H((\overline{K} \oplus opad)||H((\overline{K} \oplus ipad)||m)).$$

Dans cette définition, *ipad* et *opad* sont deux valeurs constantes de la taille d'un bloc de message alors que $\overline{K}$ est la clé K complétée de '0' jusqu'à atteindre la taille d'un bloc. Si la taille d'une clé est supérieure à la taille d'un bloc alors celle-ci est d'abord hachée par la fonction H.

La construction HMAC pour une fonction H construite selon le principe de Merkle-Damgård est résumée à la figure 1.6.

Construction *envelope* MAC

Une autre construction, antérieure à celle de HMAC est la construction *envelope* MAC. Cette méthode, connue aussi sous le nom de « sandwich MAC », a été proposée par Tsudik en 1992 [Tsu92].

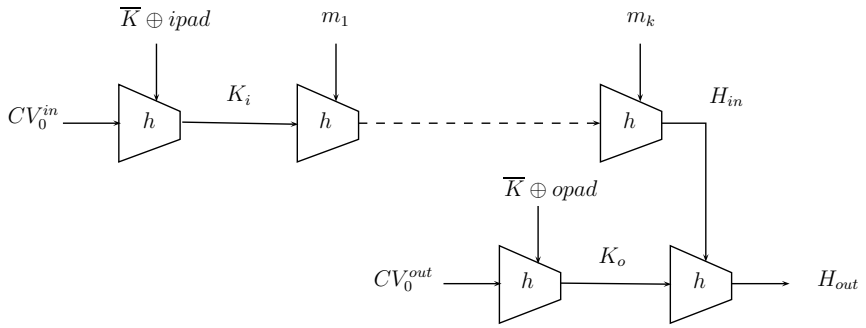

FIGURE 1.6 – La construction HMAC pour une fonction de hachage H construite selon l'algorithme de Merkle-Damgård avec une fonction de compression h.

Il s'agit d'une construction beaucoup moins complexe que HMAC, qui combine le *secret prefix* MAC et le *secret suffix* MAC à une seule construction, en utilisant une clé secrète unique K. La construction originale peut être alors décrite comme :

$$\text{MAC}_K(m) = H(\overline{K}||m||K),$$

où $\overline{K} = K||00\ldots0$ est la clé complétée de '0' afin d'atteindre la taille d'un bloc. En 1995, cette construction s'est montrée vulnérable contre des attaques susceptibles de récupérer la clé secrète [PvO96]. La faiblesse exploitée dans ces attaques était basée sur le manque de rembourrage du message m. Yasuda a montré en 2007 [Yas07] qu'en appliquant un padding cohérent sur m, il était possible de réparer ce mode. La construction *envelope* MAC réparée,

$$\text{MAC}_K(m) = H(\overline{K}||\overline{m}||K),$$

où $\overline{m}$ est le message m rembourré, pour une fonction suivant la construction de Merkle-Damgård est illustrée à la figure 1.7.

FIGURE 1.7 – La construction *envelope* MAC pour une fonction de hachage H construite selon l'algorithme de Merkle-Damgård avec une fonction de compression h.

Ce mode propose une alternative très intéressante à la construction HMAC, en particulier en termes de performances. Contrairement à HMAC qui nécessite deux appels à la fonction de hachage, un seul appel est fait dans le cas d'*envelope* MAC. En plus, il nécessite un appel de moins à la fonction de compression. Pour ces raisons, l'utilisation de *envelope* MAC est particulièrement avantageuse quand des messages de petite taille doivent être traités.

Par exemple, les concepteurs de la fonction de hachage `Grøstl`, finaliste du concours SHA-3, ont proposé d'utiliser la construction *envelope* MAC pour transformer `Grøstl` en code d'authentification.

1.3.7 Le concours SHA-3

En 2004 et 2005, Wang *et al.* ont publié des attaques en collision très dévastatrices pour plusieurs fonctions de hachage de la famille MD et SHA [WY05, WLF+05, WYY05a, WYY05b]. Parmi ces résultats figuraient des collisions pour les fonctions SHA-0, MD4 et MD5 et une attaque en complexité 2^{69} pour la fonction SHA-1.

Le standard actuel SHA-2 n'a pas été affaibli par ces attaques, cependant à cause de sa conception très proche des autres fonctions attaquées, de l'inquiétude sur son avenir a commencé à régner. Pour cette raison, l'Institut américain des normes et de la technologie (NIST) a décidé en 2007 d'organiser un concours public, appelé SHA-3 [Nat], pour définir une nouvelle norme qui pourrait remplacer SHA-2 en cas de défaillance éventuelle.

La durée totale du concours a été fixée à quatre ans ; la compétition a débuté en octobre 2008 et s'est terminé le 2 octobre 2012 avec l'annonce du vainqueur. Parmi les spécifications annoncées, il a été demandé que chaque fonction candidate puisse hacher des messages de taille au moins 2^{64} bits et puisse calculer des empreintes de longueur 224, 256, 384 et 512 bits, pour pouvoir ainsi garder une comparaison possible avec les instances de l'algorithme SHA-2. En parallèle, au lancement de la compétition, une multitude de critères susceptibles d'influencer le choix final ont été annoncés. Parmi ces critères on pouvait trouver la sécurité théorique, l'existence de preuves de sécurité pour le mode opératoire, les bonnes performances sur plusieurs plateformes, le besoin contraint en ressources et la résistance aux attaques physiques.

Le déroulement du concours a été similaire au déroulement du concours AES. Trois tours différents ont été prévus. À la fin de chaque tour, certains candidats ont été éliminés par le NIST. Les décisions du NIST ont été influencées dans une très grande mesure par la recherche publique et les efforts fournis par un grand nombre de cryptanalystes. Pour une diffusion facile des résultats de recherche, plusieurs colloques spécifiques se sont organisés tout au long de la compétition, et une page internet, appelée « SHA-3 zoo » [ECR] a été consacrée au sujet pour pouvoir recueillir les avancées récentes.

Déroulement du concours

Le 31 octobre 2008, le NIST a reçu 64 soumissions. Parmi elles, 51 algorithmes jugés conformes aux contraintes imposées, ont été choisis pour participer au premier tour de la compétition, qui a débuté en décembre 2008. Pour cette première phase, presque une année d'analyse des différentes propositions par la communauté académique a été prévue. Suite à cela, le 24 juillet 2009, les 14 candidats choisis pour continuer dans la deuxième phase du concours ont été annoncés. Peu d'un an après, en décembre 2010, le choix des 5 finalistes a été dévoilé au public. Enfin la décision finale a été présentée par le NIST le 2 octobre 2012.

Fonctions retenues pour le deuxième et le troisième tour

Dans cette thèse nous allons présenter des résultats d'analyse sur un certain nombre de candidats de la compétition SHA-3. Tous les candidats analysés sont des fonctions qui ont été sélectionnées pour le deuxième tour du concours. Pour cette raison, nous allons brièvement présenter les principes de conception des 14 candidats de la deuxième phase.

Parmi les 14 propositions, nous pouvons observer des constructions très diverses. Tout d'abord, des modes opératoires différents ont été utilisés : la construction éponge, ou des constructions lui ressemblant, la construction Merkle-Damgård et ses améliorations (HAIFA),

Candidat	Mode opératoire	Primitive interne
BLAKE	HAIFA avec Davies-Meyer	ARX
`Blue Midnight Wish`	Double pipe avec PGV3	ARX
`CubeHash`	≈ éponge	ARX
ECHO	Wide-pipe	AES
`Fugue`	≈ éponge	AES
`Grøstl`	Wide-pipe	AES
`Hamsi`	Narrow pipe avec Davies-Meyer	petites Boîtes-S/op. booléennes
JH	nouvelle construction	petites Boîtes-S/op. booléennes
KECCAK	éponge	petites Boîtes-S/op. booléennes
Luffa	≈ éponge	petites Boîtes-S/op. booléennes
`Shabal`	nouvelle construction	ARX
SHA*vite-3*	HAIFA avec PGV5	AES
SIMD	Wide pipe avec Davies-Meyer	ARX
`Skein`	Matyas-Meyer-Oseas avec tweak	ARX

TABLE 1.2 – Les 14 candidats du deuxième tour de la compétition SHA-3

wide-pipe ou narrow pipe. Les approches utilisées pour la construction de la primitive interne sont également très diverses. Certains concepteurs ont choisi d'utiliser des primitives proches du chiffrement par blocs AES, en utilisant en particulier des boîtes-S de 8 bits pour assurer la confusion. Ceci est par exemple le cas des fonctions ECHO, `Fugue`, `Grøstl` et SHAVITE-3. D'autres fonctions internes sont basées sur des opérations mélangeant des additions modulaires, des rotations et des XOR. On dit que ces fonctions sont de type ARX (pour *Add-Rotate-XOR* en anglais). Enfin, quelques constructions utilisent comme primitives non-linéaires des boîtes-S de petite taille (4 ou 5 bits) et des opérations booléennes simples (XOR, rotations) pour assurer la diffusion.

Les différents principes utilisés pour chaque fonction du deuxième tour sont résumés à la table 1.2

Les candidats qui ont été sélectionnés pour le troisième et dernier tour de la compétition sont : BLAKE, JH, `Grøstl`, KECCAK et `Skein`. La fonction KECCAK a été finalement choisie pour être le nouveau standard SHA-3.

1.4 Les chiffrements par blocs

Les chiffrements par blocs font, avec les chiffrements à flot et les fonctions de hachage, partie des trois grandes familles de la cryptographie symétrique.

Un chiffrement est dit *par blocs* s'il s'applique à des messages de taille fixe (souvent 64 ou 128 bits).

Définition 1.1. *Un chiffrement par blocs est une permutation E qui prend en entrée deux arguments : un bloc de n bits et une clé secrète de k bits et qui retourne en sortie un bloc de n bits.*

$$\begin{aligned} E : \{0,1\}^k \times \{0,1\}^n &\to \{0,1\}^n \\ (K, m) &\mapsto E_K(m) \stackrel{déf}{=} E(K, m) \end{aligned}$$

1.4.1 Les chiffrements par blocs itératifs

Aujourd'hui, la façon la plus courante pour construire un chiffrement par blocs est d'utiliser une méthode *itérative*. Cela consiste en l'utilisation d'une même fonction interne qui est itérée plusieurs fois avec une clé différente à chaque itération. Cette fonction interne est en général cryptographiquement faible en elle-même ; cependant en l'itérant plusieurs fois on arrive à une fonction solide et résistante aux diverses attaques.

De manière formelle, un chiffrement itératif par blocs consiste à itérer une fonction interne F, r fois, comme on peut le voir sur la figure 1.8. Chacune de ces r itérations est paramétrée par un paramètre secret k_i, appelé *clé de tour*. Les clés de tours sont dérivées de la clé secrète du chiffrement, K, dite aussi *clé maître*, par un algorithme de *cadencement des clés* :

$$E_k = F_{k^{(r)}} \circ F_{k^{(r-1)}} \circ \cdots \circ F_{k^{(1)}}.$$

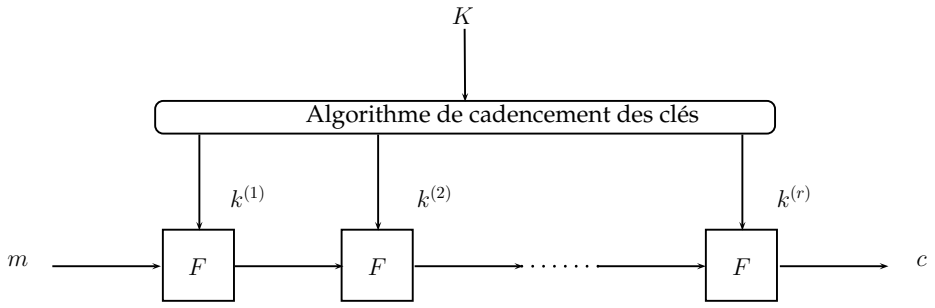

FIGURE 1.8 – Le chiffrement itératif par blocs

Cette construction présente plusieurs avantages : premièrement, l'usage d'une seule « brique » primitive permet la conception de fonctions qui sont bien comprises et ainsi faciles à analyser. Ensuite, ces constructions peuvent être implémentées d'une façon rapide et compacte et offrent des performances très intéressantes. Les chiffrements par blocs sont présents aujourd'hui dans plusieurs plateformes différentes (carte à puce, RFID...), ainsi ce dernier point est très important.

Les itérations successives de la fonction F diffèrent souvent d'une constante, en particulier pour éviter des attaques de type "slide attacks". De telles constantes n'ont généralement pas d'influence sur les propriétés que nous allons étudier ici.

La structure de la fonction interne varie selon les algorithmes. Cependant, elle suit généralement les principes de conception définis par Claude Shannon [Sha49]. Selon ces principes, un algorithme de chiffrement symétrique doit répondre aux deux besoins suivants, la *confusion* et la *diffusion* :

- **Confusion :** La confusion correspond à la volonté de rendre la relation entre la clé de chiffrement et le texte chiffré la plus complexe possible.
- **Diffusion :** La redondance statistique dans un texte en clair est dissipée dans les statistiques du texte chiffré.

Aujourd'hui il existe deux grandes familles de chiffrement itératifs par blocs : les *réseaux de substitution-permutation* et les *réseaux de Feistel*.

1.4.2 Réseaux de substitution-permutation

Un algorithme de substitution-permutation ou SPN (Substitution Permutation Network en anglais) est constitué d'une succession de tours de la forme $L \circ S \circ C$, où C est une opération de mélange de clé (souvent on ajoute simplement la clé avec un OU exclusif), S une fonction non-linéaire assurant la confusion et L une fonction linéaire assurant la diffusion (figure 1.9).

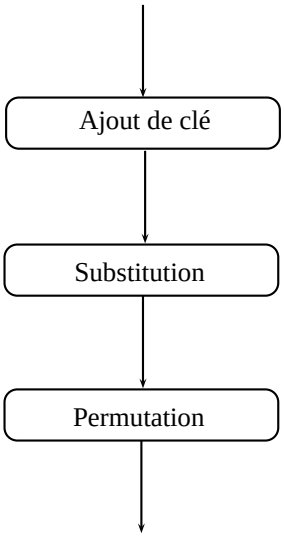

FIGURE 1.9 – Fonction de tour d'un réseau substitution-permutation

Cette construction ne garantit pas en elle-même l'inversibilité du chiffrement. Pour cette raison il est nécessaire que chacune des composantes soit inversible.

Le standard actuel de chiffrement par blocs, AES, suit ce principe de conception. Les réseaux SPN sont aussi utilisés à grande échelle pour la construction des fonctions de hachage. Pour cet usage, l'étape de mélange de clé est évidemment ignorée.

Boîtes-S La construction d'une fonction non-linéaire ayant de bonnes propriétés cryptographiques est un problème très important. Cependant, la mise en œuvre d'une telle fonction de grande taille est très coûteuse tant en termes de temps qu'en termes de mémoire. Pour cette raison il est courant de découper l'état interne en petits morceaux et d'appliquer une petite fonction non-linéaire, appelée boîte-S, ou Sbox en anglais, sur chacun d'entre eux. Parmi les primitives symétriques connues, certaines utilisent des boîtes-S de 8 bits (AES) ou des boîtes-S de 4 bits, comme c'est le cas du chiffrement à bas coût PRESENT et des fonctions du concours SHA-3 Hamsi et Luffa. Plus rarement, des boîtes-S de taille 3 (chiffrement à bas coût PRINTCIPHER) ou 5 (KECCAK) sont utilisées. Il est aussi possible dans d'autres constructions d'utiliser des boîtes-S non-inversibles, comme c'est le cas de DES. Les propriétés de cette primitive seront discutées et analysées dans les chapitres suivants.

Description de Rijndael

Rijndael est l'exemple le plus connu des chiffrements par blocs de type SPN. Il a été conçu par Joan Daemen et Vincent Rijmen [DR00b, DR00a] pour la compétition internationale AES du NIST qui a débuté en 1997. En octobre 2002, Rijndael a remporté le concours et a été renommé en AES (acronyme de *Advanced Encryption Standard*). Il a été ensuite normalisé (NIST FIPS 197 [FIP01]) et est devenu le nouveau standard de chiffrement par blocs en remplaçant définitivement son prédécesseur DES.

Il existe plusieurs instances de l'algorithme Rijndael, Rijndael-N_b, avec $N_b \in \{128, 160, 192, 224, 256\}$, qui dépendent de la taille de l'état interne et de la taille de la clé. De façon générale, nous pouvons visualiser l'état comme une matrice d'octets $4 \times N_b/32$ $A = (a_{i,j})$. Les états respectifs de Rijndael-128 et Rijndael-256 sont représentés à la figure 1.10.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

FIGURE 1.10 – Les états de Rijndael-256 et Rijndael-128

La taille de la clé secrète N_k peut varier entre 128, 192 et 256 bits.

Le standard AES a été fixé à Rijndael-128. La taille de la clé secrète N_k peut varier entre 128, 192 et 256 bits. Le nombre de tours r varie lui aussi en fonction des tailles de l'état et de la clé. Nous pouvons observer le nombre de tours pour chaque version dans la table 1.3.

r	$N_b = 128$	$N_b = 160$	$N_b = 192$	$N_b = 224$	$N_b = 256$
$N_k = 128$	10	11	12	13	14
$N_k = 192$	12	12	12	13	14
$N_k = 256$	14	14	14	14	14

TABLE 1.3 – Nombre de tours r pour Rijndael en fonction de la taille de l'état N_b et de la taille de la clé N_k.

La permutation de tour R, qui s'applique sur l'état $A = (a_{i,j})$ est composée de quatre fonctions :

- `SubBytes` : La seule transformation non-linéaire du chiffrement. Chaque octet de l'état est mis à jour par une boîte-S, de degré algébrique 7.
- `ShiftRows` : Transformation linéaire qui applique une rotation à chaque ligne de l'état, avec un décalage différent par ligne.
- `MixColumns` : Transformation linéaire qui s'applique en parallèle sur les colonnes de l'état.
- `AddRoundKey` : Addition de la sous-clé à l'état.

La première itération est précédée d'une addition de la sous-clé d'indice 0. Ensuite, toutes les itérations sont identiques, sauf la dernière, où l'opération `MixColumn` est omise.

La transformation `SubBytes` La transformation `SubBytes` est la seule transformation non-linéaire de la fonction de tour. Elle consiste à appliquer en parallèle une boîte-S, notée S, à chaque octet de l'état. S est une permutation sur le corps $\mathbf{F}_{2^8}$, dont toutes les coordonnées sont de degré 7. La permutation inverse est également de degré 7. Cette opération peut être visualisée à la figure 1.11. S est construite à partir de l'opération inverse dans le corps fini à 256 éléments :

$$\begin{array}{rcl} S : \mathbf{F}_{2^8} & \leftarrow & \mathbf{F}_{2^8} \\ x & \mapsto & x^{254} \end{array}$$

c'est à dire, $x \mapsto x^{-1}$, où $0^{-1} = 0$, suivie ensuite d'une transformation affine sur $\mathbf{F}_2^8$.

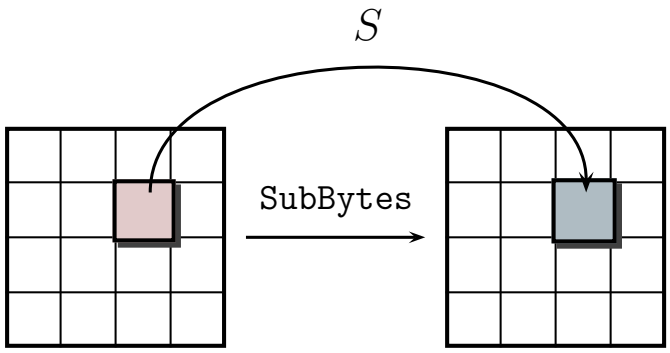

FIGURE 1.11 – La transformation `SubBytes` pour Rijndael-128

La transformation `ShiftRows` Cette transformation consiste à effectuer une rotation vers la gauche sur chaque ligne de l'état. Le décalage est différent pour chaque ligne de l'état et diffère selon l'instance de Rijndael.

Pour Rijndael-128 la ligne i ($0 \leq i \leq 3$) est décalée de i octets vers la gauche, comme nous pouvons le voir à la figure 1.12. Le décalage de chaque ligne pour chaque version de l'algorithme est défini à la table 1.4.

	Rijndael-128	Rijndael-160	Rijndael-192	Rijndael-224	Rijndael-256
ligne 0	0	0	0	0	0
ligne 1	1	1	1	1	1
ligne 2	2	2	2	2	3
ligne 3	3	3	3	4	4

TABLE 1.4 – Longueur de décalage pour chaque ligne en fonction de la taille de l'état interne.

La transformation `MixColumns` Cette fonction est une transformation linéaire, appliquée en parallèle sur les colonnes de l'état. Plus précisément, chaque colonne de l'état est multipliée par une matrice M de la façon suivante :

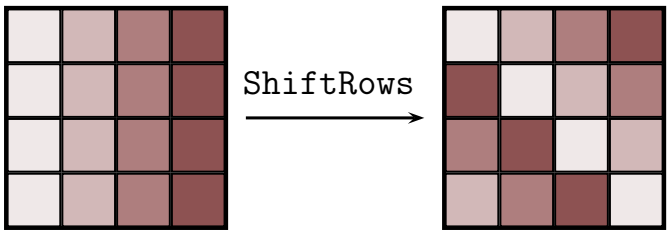

FIGURE 1.12 – La transformation `ShiftRows` pour Rijndael-128

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

où α est racine du polynôme $x^8 + x^4 + x^3 + x + 1$. Cette transformation peut être visualisée à la figure 1.13.

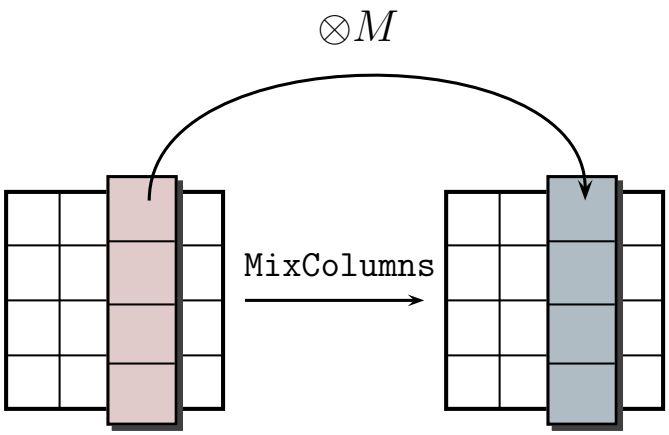

FIGURE 1.13 – La transformation `MixColumns` pour Rijndael-128

La matrice M a été choisie pour ses bonnes propriétés de diffusion. Elle fait partie des matrices appelées *MDS*, de l'acronyme anglais "Maximum Distance Separable", qui assurent une diffusion *parfaite*. Pour une fonction de diffusion L agissant sur des octets, nous souhaitons de façon générale que même si très peu d'octets sont modifiés en entrée, le nombre d'octets affectés en sortie soit grand. Autrement dit, si $wt(v)$, avec v un vecteur d'octets, symbolise le nombre d'octets non-nuls, nous voulons que la quantité suivante :

$$\min_{x \in \mathbf{F}_2^8} (wt(x) + wt(L(x))),$$

appelée en anglais *"branch number"*, soit la plus élevée possible. Les matrices représentant une fonction L pour laquelle la borne supérieure est atteinte sont appelées *matrices MDS*. En effet, le branch number de L correspond à la distance minimale du code correcteur linéaire composé des mots $(x||L(x))$. Si L est une fonction de $\mathbf{F}_q^n$ vers $\mathbf{F}_q^n$, cette distance atteint son maximum, $n+1$, pour les codes dits MDS. Pour une fonction s'appliquant sur une colonne

de 4 octets à la fois, comme c'est le cas de `MixColumns`, ce nombre peut être au plus 5, car si un seul octet en entrée est modifié, il peut influencer au plus les 4 octets de sa colonne. La matrice M de *MixColumns* est choisie de façon que cette borne maximale soit atteinte.

La transformation `AddRoundKey` Il s'agit de l'addition de la sous-clé de tour, bit à bit.

1.4.3 Réseaux de Feistel

Un réseau de Feistel (figure 1.14), qui port le nom du cryptologue d'IBM Hors Feistel, est une structure développée dans les années 1970. Dans la version équilibrée de cette construction, les données d'entrée sont divisées en deux parties de taille identique. Les deux parties sont ensuite traitées de façon différente :

$$\begin{array}{rcl} F_K : \mathbf{F}_2^n \times \mathbf{F}_2^n & \to & \mathbf{F}_2^n \times \mathbf{F}_2^n \\ (L, R) & \mapsto & (R, L \oplus f(K, R)) \end{array}$$

La fonction interne f, de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ est une fonction composée de façon générale d'une fonction linéaire et d'une fonction de substitution. La structure générale d'un réseau de Feistel garantit l'inversibilité du chiffrement quelle que soit la fonction f employée, puisque $\pi \circ F_K$ est une involution, où π désigne la permutation des deux entrées, $\pi(L, R) = (R, L)$. Un des avantages principaux de ce système est que la fonction de déchiffrement est identique à celle de chiffrement ; il suffit juste d'inverser l'ordre des sous-clés.

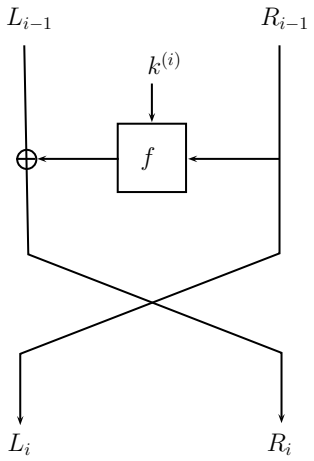

FIGURE 1.14 – Un tour d'un chiffrement d'un réseau de type Feistel

La fonction de chiffrement DES, standardisée en 1977, utilise cette méthode de construction. Aujourd'hui, il existe des généralisations de cette construction, notamment des versions *non-équilibrées* où les données sont découpées en deux parties de tailles différentes [SK96] ou encore des versions où les données sont découpées en plus de deux parties [ZMI90].

Chapitre 2

Propriétés algébriques des cryptosystèmes symétriques

Sommaire

2.1 Fonctions booléennes 29
2.1.1 Spectre de Walsh et non-linéarité d'une fonction booléenne 32
2.1.2 Restriction d'une fonction booléenne et fonctions booléennes normales 35
2.2 Quelques attaques exploitant un degré algébrique faible 36
2.2.1 Attaques algébriques 37
2.2.2 Attaques différentielles d'ordre supérieur 38
2.2.3 Attaque cube 45
2.2.4 Attaque intégrale 47

2.1 Fonctions booléennes

Les fonctions qui interviennent dans la conception des primitives symétriques peuvent être vues de façon générale comme des applications de l'ensemble des mots binaires de taille n, $\{0,1\}^n$, dans l'ensemble des mots binaires de taille m, $\{0,1\}^m$. Ces fonctions sont connues sous le nom de *fonctions vectorielles*. Les m coordonnées d'une fonction vectorielle correspondent à des fonctions à n variables ayant $\{0,1\}$ comme ensemble d'arrivée et s'appellent *fonctions booléennes*. L'étude des propriétés de ces dernières permet d'évaluer la sécurité des cryptosystèmes dont elles sont parties intégrantes.

Nous commençons par introduire quelques notations dont nous aurons besoin par la suite. À partir de maintenant, l'ensemble $\{0,1\}$ sera identifié au corps fini à deux éléments, $\mathbf{F}_2$. De même, l'ensemble des éléments $\{0,1\}^n$ sera représenté par l'espace vectoriel $\mathbf{F}_2^n$. Nous noterons les deux opérations du corps $\mathbf{F}_2^n$ de la façon suivante :

- L'addition de deux vecteurs $x, y \in \mathbf{F}_2^n$ sera noté par $x \oplus y$, ou par $x + y$ quand il n'y a pas d'ambiguité. Cette opération bit à bit désigne le OU-EXCLUSIF ou XOR.
- Le produit scalaire de deux vecteurs $x, y \in \mathbf{F}_2^n$, sera noté par $x \cdot y$:

$$x \cdot y = \bigoplus_{i=1}^{n} x_i y_i.$$

Le *poids de Hamming* d'un vecteur $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbf{F}_2^n$ est défini comme le nombre des coordonnées non-nulles de x :

$$wt(x) = \#\{x_i : x_i \neq 0\}.$$

Nous introduisons ici les définitions et les propriétés des fonctions booléennes dont nous aurons besoin plus tard.

Définition 2.1. *Une* fonction booléenne *à n variables est une application de $\mathbf{F}_2^n$ dans $\mathbf{F}_2$.*

Définition 2.2. *Le* support *d'une fonction booléenne f à n variables est l'ensemble des vecteurs dont l'image par f est différente de 0 :*

$$supp(f) = \{x \in \mathbf{F}_2^n : f(x) \neq 0\}.$$

Définition 2.3. *On appelle* poids de Hamming *$wt(f)$ d'une fonction booléenne f à n variables le cardinal de son support :*

$$wt(f) = \#supp(f).$$

Pour diverses applications cryptographiques, comme par exemple la génération de suites pseudo-aléatoires, nous aurons besoin de fonctions qui possèdent un comportement proche de celui des fonctions aléatoires. Plus précisément, nous souhaitons utiliser des fonctions produisant une suite (par exemple dans le cas d'un chiffrement à flot) pour laquelle aucun biais ne puisse être observé. Pour cette raison, nous utilisons des fonctions booléennes *équilibrées.*

Définition 2.4. *Une fonction booléenne est dite* équilibrée *lorsque elle prend autant de fois la valeur 0 que la valeur 1. Autrement dit, une fonction f à n variables est équilibrée si et seulement si*

$$wt(f) = 2^{n-1}.$$

La manière la plus simple de représenter une fonction booléenne à n variables est à travers sa *table de vérité.* Cela consiste à associer à chaque valeur de $\mathbf{F}_2^n$ son image par f et à représenter ceci sous la forme d'un tableau, comme nous pouvons le voir dans l'exemple 2.1.

Exemple 2.1. *Soit $n = 3$. Nous définissons une fonction booléenne f par sa table de vérité, qui coïncide avec la table 2.1.*

$x = (x_0, x_1, x_2)$	$f(x)$
$(0, 0, 0)$	1
$(1, 0, 0)$	0
$(0, 1, 0)$	1
$(1, 1, 0)$	0
$(0, 0, 1)$	1
$(1, 0, 1)$	1
$(0, 1, 1)$	1
$(1, 1, 1)$	0

TABLE 2.1 – La table de vérité d'une fonction booléenne à 3 variables.

Dans cet exemple, le support de f correspond à

$$supp(f) = \{(0,0,0),(0,1,0),(0,0,1),(1,0,1),(0,1,1)\}.$$

Par conséquent, selon la définition 2.3, le poids de Hamming de f, $wt(f)$ est égal à 5.

La description d'une fonction booléenne par sa table de vérité est peu utilisée dans la pratique. Nous pouvons employer à la place une autre représentation, beaucoup plus compacte. Il peut être démontré à l'aide de l'interpolation de Lagrange que chaque fonction booléenne peut s'écrire de façon unique sous la forme d'un polynôme multivarié à coefficients dans $\mathbf{F}_2$. Cette représentation est appelée *forme algébrique normale* ou ANF (pour "algebraic normal form" en anglais).

Définition 2.5. *Soit f une fonction booléenne à n variables. La* forme algébrique normale *(ANF) de f est l'unique polynôme de $\mathbf{F}_2[x_0,\ldots,x_{n-1}]/(x_0^2-x_0,\ldots,x_{n-1}^2-x_{n-1})$ représentant f :*

$$f(x) = \bigoplus_{u\in\mathbf{F}_2^n} c_f(u)x^u, \qquad avec\ \ x^u = x_0^{u_0}x_1^{u_1}\cdots x_{n-1}^{u_{n-1}},$$

où $c_f(u) = \bigoplus_{v\preceq u} f(v) \in \mathbf{F}_2$. Si $v=(v_0,v_1,\ldots,v_{n-1})$ et $u=(u_0,u_1,\ldots,u_{n-1})$, $v\preceq u$ si et seulement si $v_i \leq u_i$ pour chaque $i\in\{0,1,\ldots,n-1\}$.

À partir de la forme algébrique normale d'une fonction booléenne, nous pouvons définir son *degré algébrique*.

Définition 2.6. *Le* degré algébrique *d'une fonction booléenne f correspond au poids maximum du vecteur u tel que le coefficient $c_f(u)$ de l'ANF de f est non-nul :*

$$deg(f) = \max_{u\in\mathbf{F}_2^n}\{wt(u) : c_f(u)\neq 0\}.$$

Exemple 2.2. *La forme algébrique normale de la fonction booléenne f de l'exemple 2.1 est :*

$$f(x_0,x_1,x_2) = x_0x_1x_2 + x_0x_2 + x_0 + 1.$$

Son degré algébrique $deg(f)$ est égal à 3.

Pour les applications cryptographiques, afin qu'un cryptosystème résiste bien aux attaques dites *algébriques*, les fonctions booléennes utilisées doivent posséder un degré élevé.

Une fonction vectorielle F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ peut être décrite en moyen de ses m coordonnées, $f_0,f_1,\ldots,f_{m-1}$:

$$\forall x\in\mathbf{F}_2^n, \qquad F(x) = (f_0(x),f_1(x),\ldots,f_{m-1}(x)).$$

Définition 2.7. *Soit $F=(f_0,\ldots,f_{m-1})$ une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Nous appellerons* composantes *de F, les 2^m-1 combinaisons linéaires non-triviales de ses fonctions coordonnées :*

$$\forall\lambda=(\lambda_0,\ldots,\lambda_{m-1})\in\mathbf{F}_2^m\setminus\{0\}: \quad \lambda\cdot F(x) = \lambda_0 f_0(x)\oplus\cdots\oplus\lambda_{m-1}f_{m-1}(x).$$

Une composante $\lambda\cdot F$, $\lambda\neq 0$ sera également notée par F_λ.

Le degré algébrique pour une fonction vectorielle sera défini à l'aide de ses fonctions coordonnées.

Définition 2.8. *Soit $F = (f_0, f_1, \ldots, f_{m-1})$ une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Le degré algébrique de F, $\deg(F)$, est défini de la manière suivante :*

$$\deg(F) = \max_{i=0,\ldots,m-1} \deg(f_i).$$

Exemple 2.3. *Soit F la fonction vectorielle de $\mathbf{F}_2^3$ dans $\mathbf{F}_2^2$, avec*

$$F(x_0, x_1, x_2) = (x_0x_1x_2 + x_0x_2 + x_0 + 1, x_0x_1 + x_1x_2).$$

Selon la définition 2.8, $deg(F) = 3$.

Grâce à l'isomorphisme entre l'espace vectoriel $\mathbf{F}_2^n$ et le corps fini à 2^n éléments $\mathbf{F}_{2^n}$, une fonction vectorielle F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ peut être vue comme un polynôme univarié sur $\mathbf{F}_{2^n}$. En particulier, il peut être démontré ([LN83, chapitre 7] par exemple) que pour une telle fonction F il existe une unique représentation univariée de degré au plus $2^n - 1$,

$$F(x) = \sum_{i=0}^{2^n-1} b_i x^i, \quad b_i \in \mathbf{F}_{2^n}. \tag{2.1}$$

La définition du degré algébrique de F avec cette représentation est la suivante.

Définition 2.9. *Soit F une fonction vectorielle, représentée sous la forme (2.1). Le degré algébrique de F est donné par*

$$\deg(F) = \max\{wt(i) : 0 \leq i < 2^n \text{ et } b_i \neq 0\}.$$

Comme pour les fonctions booléennes, la notion d'« équilibre » joue également un rôle très important pour les applications cryptographiques.

Définition 2.10. *Une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ est dite* équilibrée *si elle prend toutes les valeurs dans $\mathbf{F}_2^m$ exactement 2^{n-m} fois.*

Si $n = m$, une fonction vectorielle équilibrée est une permutation.

Proposition 2.1 ([LN83, CP05])**.** *Une fonction vectorielle F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ est équilibrée si et seulement si toutes ses composantes $\lambda \cdot F$, $\forall \lambda \in \mathbf{F}_2^m \setminus \{0\}$ sont des fonctions booléennes équilibrées.*

2.1.1 Spectre de Walsh et non-linéarité d'une fonction booléenne

Dans de nombreuses situations, nous avons besoin d'exploiter la corrélation d'une fonction booléenne f avec une fonction linéaire. Pour cela il semble naturel de mesurer le biais de la fonction $f + \varphi$, où φ représente une fonction linéaire. Afin d'effectuer ceci, nous introduisons la quantité suivante

$$\mathcal{F}(f) = \sum_{x \in \mathbf{F}_2^n} (-1)^{f(x)} = 2^n - 2\mathrm{wt}(f), \tag{2.2}$$

appelée souvent *poids d'une fonction* à cause de sa dépendance au poids véritable, $\mathrm{wt}(f)$, de la fonction.

D'après cette définition, nous pouvons voir qu'une fonction booléenne f est équilibrée si et seulement si $\mathcal{F}(f) = 0$, donc la quantité (2.2), mesure le *biais* ou le *déséquilibre* de f.

Par la suite, on utilisera souvent la notation suivante pour représenter les fonctions booléennes linéaires.

Notation 2.1. *Pour tout $a \in \mathbf{F}_2^n$ nous noterons par φ_a la fonction linéaire à n variables $x \mapsto a \cdot x$. La fonction φ_0 correspond à la fonction nulle.*

La quantité (2.2) est maintenant utilisée pour définir ce qu'on appellera le *spectre de Walsh* d'une fonction booléenne. Cette nouvelle notion mesurera la corrélation entre la fonction f et toutes les fonctions linéaires et pourra être utilisée pour mesurer la distance d'une fonction booléenne à l'ensemble des fonctions affines.

Définition 2.11. *Soit f une fonction booléenne à n variables. Le* coefficient de Walsh *de f au point $a \in \mathbf{F}_2^n$ correspond à la quantité*

$$\mathcal{F}(f + \varphi_a) = \sum_{x \in \mathbf{F}_2^n} (-1)^{f(x)+a\cdot x}.$$

On appelle spectre de Walsh *de la fonction f l'ensemble*

$$f^{\mathcal{W}} = \{\mathcal{F}(f + \varphi_a),\ a \in \mathbf{F}_2^n\}.$$

Nous définissons, pour une fonction booléenne f, sa corrélation maximale avec l'ensemble des fonctions affines, c'est-à-dire des fonctions dont le degré est inférieur ou égal à 1 :

$$\mathcal{L}(f) = \max_{a \in \mathbf{F}_2^n} |\mathcal{F}(f + \varphi_a)|.$$

Cette quantité, appelée souvent *linéarité* de f, est utilisée entre autres pour définir la *non-linéarité* d'une fonction.

Définition 2.12. *Soit f une fonction booléenne à n variables. La* non-linéarité *de f, notée $\mathcal{NL}(f)$, est définie par*

$$\mathcal{NL}(f) = 2^{n-1} - \frac{1}{2}\mathcal{L}(f).$$

La non-linéarité d'une fonction booléenne mesure sa distance à l'ensemble des fonctions affines.

Les fonctions possédant la plus haute non-linéarité possible n'existent que pour n pair et sont appelées fonctions *courbes*.

Définition 2.13. *Soit f une fonction booléenne à n variables. Alors,*

$$\mathcal{L}(f) \geq 2^{\frac{n}{2}}.$$

Les fonctions atteignant cette borne inférieure sont dites fonctions courbes *("bent" en anglais). Elles n'existent que pour n pair.*

Les fonctions courbes malgré l'avantage d'avoir une non-linéarité maximale présentent également un grand désavantage pour les applications cryptographiques car ce sont des fonctions non-équilibrées.

La notion du spectre de Walsh est également définie dans le cas d'une fonction vectorielle.

Définition 2.14. *Soit F une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Le spectre de Walsh de F correspond à l'ensemble des spectres de Walsh de ses composantes, $\varphi_b \circ F + \varphi_a$:*

$$F^{\mathcal{W}} = \left\{ \sum_{x \in \mathbf{F}_2^n} (-1)^{b \cdot F(x) + a \cdot x}, b \in \mathbf{F}_2^n \setminus \{0\}, a \in \mathbf{F}_2^n \right\}.$$

Nous démontrons ici que le spectre de Walsh d'une permutation, est égal au spectre de Walsh de la permutation inverse.

Proposition 2.2. *Soit F une permutation de $\mathbf{F}_2^n$ et notons par F^{-1} la permutation inverse. Alors,*

$$F^{\mathcal{W}} = (F^{-1})^{\mathcal{W}}.$$

Démonstration. Nous allons démontrer la première inclusion. Soit $b \in \mathbf{F}_2^n \setminus \{0\}$ et $a \in \mathbf{F}_2^n$. Comme F est une permutation, nous avons que :

$$\begin{aligned} \mathcal{F}(\varphi_b \circ F + \varphi_a) &= \sum_{x \in \mathbf{F}_2^n} (-1)^{b \cdot F(x) + a \cdot x} \\ &= \sum_{y = F(x),\, x \in \mathbf{F}_2^n} (-1)^{b \cdot y + a \cdot F^{-1}(y)} \\ &= \sum_{y \in \mathbf{F}_2^n} (-1)^{b \cdot y + a \cdot F^{-1}(y)} \\ &= \mathcal{F}(\varphi_a \circ F^{-1} + \varphi_b). \end{aligned}$$

□

Nous définissons de façon analogue que pour les fonctions booléennes, la *linéarité* $\mathcal{L}(F)$ et la *non-linéarité* $\mathcal{NL}(F)$ d'une fonction vectorielle F.

Définition 2.15. *Soit F une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Nous appelons par* linéarité *de F la quantité suivante :*

$$\mathcal{L}(F) = \max_{b \in \mathbf{F}_2^m \setminus \{0\}, a \in \mathbf{F}_2^n} |\mathcal{F}(\varphi_b \circ F + \varphi_a)|.$$

La non-linéarité *de F est définie par :*

$$\mathcal{NL}(F) = 2^{n-1} - \frac{1}{2}\mathcal{L}(F).$$

Nous pouvons définir de façon équivalente la non-linéarité d'une fonction vectorielle F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ en fonction de ses composantes F_λ. La non-linéarité de F est alors définie comme la non-linéarité minimale de celles-ci.

Les fonctions possédant une haute non-linéarité présentent une bonne résistance contre les attaques linéaires. En composant une fonction vectorielle avec une transformation affine, la non-linéarité reste invariante. Pour cette raison, il est important pour la construction d'une primitive symétrique de choisir une fonction de substitution avec une non-linéarité élevée.

Définition 2.16. *Soit F une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors,*

$$\mathcal{L}(F) \geq 2^{\frac{n}{2}}.$$

Les fonctions pour lesquelles cette borne est atteinte, sont dites fonctions courbes.

Il est connu que des fonctions courbes n'existent pas pour $m > n/2$, en particulier pour $m = n$. Pour cette raison, une notion plus faible est utilisée pour caractériser des fonctions de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ qui atteignent la plus haute non-linéarité possible.

Définition 2.17. *Pour toute fonction F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$,*

$$\mathcal{L}(F) \geq 2^{\frac{n+1}{2}}.$$

Les fonctions qui atteignent cette borne minimale, sont dites presque courbes *("almost bent" en anglais). Elles n'existent que pour n impair.*

2.1.2 Restriction d'une fonction booléenne et fonctions booléennes normales

Une dernière notion que nous allons introduire dans ce chapitre est la restriction d'une fonction booléenne à un sous-espace. Ce concept est central pour définir les fonctions k-normales que nous allons voir par la suite.

Définition 2.18. *Soit f une fonction booléenne à n variables, V un sous-espace vectoriel de $\mathbf{F}_2^n$ et $a \in \mathbf{F}_2^n$. On appelle* restriction *de f à un translaté $a + V$ de V, la fonction*

$$\begin{aligned} f|_{a+V} : V &\to \mathbf{F}_2 \\ x &\mapsto f(a+x). \end{aligned}$$

Nous pouvons voir la restriction de f comme une fonction à $\dim V$ variables. Si par W on note un sous-espace supplémentaire de V, alors l'ensemble des restrictions $\{f(a+V), a \in W\}$ forme la décomposition de f relativement à V.

Il existe une relation générale entre le biais de la restriction d'une fonction et son spectre de Walsh. Plus précisément, la proposition suivante associe le coefficient de Walsh de la restriction d'une fonction, aux coefficients de Walsh de celle-ci.

Proposition 2.3 ([CC03])**.** *Soit f une fonction booléenne à n variables, V un sous-espace de $\mathbf{F}_2^n$ de dimension k et $a \in W$, où W un sous-espace supplémentaire de V. Alors,*

$$\sum_{v \in V^\perp} (-1)^{a \cdot v} \mathcal{F}(f + \varphi_v) = 2^{n-k} \mathcal{F}(f|_{a+V}),$$

où $V^\perp$ est le sous-espace dual de V relativement au produit scalaire :

$$V^\perp = \{x \in \mathbf{F}_2^n \mid \forall y \in V, x \cdot y = 0\}.$$

Trouver des fonctions booléennes qui sont équilibrées et qui possèdent une non-linéarité maximale est un problème très important en cryptographie, puisque ces fonctions sont des candidats parfaits pour les applications cryptographiques. Comme nous l'avons déjà vu, des fonctions atteignant une non-linéarité maximale existent et s'appellent fonctions courbes.

Malheureusement, ces fonctions ne sont pas équilibrées, par conséquence elles sont rarement utilisées dans la construction des primitives symétriques. Néanmoins, il a été prouvé qu'une fonction courbe à n variables qui est constante sur un sous-espace affine de $\mathbf{F}_2^n$ de dimension $n/2$ est équilibrée sur tous les autres sous-espaces affines de $\mathbf{F}_2^n$. Ainsi, Hans Dobbertin [Dob95] qui cherchait à construire des fonctions équilibrées hautement non-linéaires à partir de fonctions courbes, a commencé par étudier des fonctions courbes avec un nombre pair de variables qui sont constantes sur un sous-espace de dimension $n/2$. Il a ensuite étudié des fonctions qui à la place d'être constantes, sont équilibrées sur ces mêmes sous-espaces. Il a nommé les fonctions courbes de départ *normales*. Plus tard, ces fonctions ont été généralisées pour inclure des fonctions avec un nombre impair de variables.

Définition 2.19. *Une fonction booléenne f à n variables est dite* normale, *si elle est constante sur un sous-espace affine U de $\mathbf{F}_2^n$ de dimension $\lceil \frac{n}{2} \rceil$. Dans ce cas on dit que f est normale par rapport à U.*

Pascale Charpin a étendu la notion de la normalité [Cha04] pour inclure les fonctions qui sont affines sur un sous-espace affine de dimension $\lceil \frac{n}{2} \rceil$.

Définition 2.20. *Une fonction booléenne f à n variables est dite* faiblement normale *si elle est affine sur un sous-espace affine U de $\mathbf{F}_2^n$ de dimension $\lceil \frac{n}{2} \rceil$.*

Un problème intéressant dans le contexte de la *normalité* est de déterminer la plus grande dimension possible d'un sous-espace affine U de façon qu'une fonction f soit constante sur U.

Définition 2.21. *Une fonction booléenne f à n variables est appelée k-normale, $k \leq n$, s'il existe un sous-espace affine de dimension k sur lequel f est constante. Elle est appelée k-faiblement normale si elle est affine sur un sous-espace affine de dimension k.*

La notion de la faible normalité conduit à une borne supérieure sur la non-linéarité de la fonction.

Proposition 2.4 (Corollaire V.3, [CCCF01])**.** *Soit f une fonction booléenne qui est k-faiblement normale. Alors,*

$$\mathcal{L}(f) \geq 2^k.$$

2.2 Quelques attaques exploitant un degré algébrique faible

La construction des chiffrements par blocs, ainsi que celle des fonctions de hachage modernes, suit les principes annoncés par Claude Shannon en 1949. Plus précisément, chaque primitive symétrique est constituée de deux parties différentes ; une partie non-linéaire assurant la confusion et une partie linéaire garantissant la diffusion. Certaines attaques développées contre les cryptosystèmes symétriques exploitent des faiblesses de ces deux parties distinctes. Certaines attaques s'appliquent aux primitives ayant un degré algébrique faible ou une faible non-linéarité tandis que d'autres exploitent une diffusion lente.

La plupart des attaques et des distingueurs que nous avons appliqués contre diverses primitives symétriques tirent profit de ces deux faiblesses distinctes. Ces attaques seront présentées dans les quatre chapitres qui suivent. Dans cette partie du mémoire nous allons montrer l'importance d'un degré algébrique élevé ainsi que d'une bonne diffusion pour la

sécurité d'une primitive cryptographique. Nous allons faire cela en présentant brièvement quelques attaques *classiques* et leurs variantes.

Parmi ces attaques nous allons énumérer les plus importantes. Nous allons commencer par des attaques qui exploitent des faiblesses dans la partie non-linéaire (cryptanalyse algébrique, cryptanalyse différentielle d'ordre supérieur, attaques cube) et nous verrons ensuite le principe de la cryptanalyse intégrale qui exploite à la fois une mauvaise diffusion et un degré faible. Il faudra noter qu'il n'est pas facile de distinguer clairement ces différentes attaques à cause des multiples intersections qui existent entre les méthodes.

2.2.1 Attaques algébriques

Le premier type d'attaques contre les cryptosystèmes symétriques pouvant exploiter un degré algébrique faible est l'attaque algébrique.

Les attaques algébriques sur les chiffrements par blocs font partie de la famille des attaques à clair connu. Le principe consiste à exploiter des relations algébriques entre les bits du message clair, les bits du message chiffré et les bits de la clé secrète. Un attaquant peut écrire des équations dont les inconnues sont les bits de la clé et essayer de retrouver ceux-ci en résolvant le système. La résolution des systèmes algébriques de ce type est en général un problème très difficile. Sa réussite dépend de plusieurs paramètres, comme le degré algébrique des équations, la taille du système ou la présence d'une structure particulière.

Dans son article novateur de 1949 [Sha49], Claude Shannon mentionne que « *casser un "bon" chiffrement, doit demander autant de travail que la résolution d'un système complexe d'équations en un grand nombre de variables.* »[1] Pour valider cela il propose une attaque simple, la première de ce genre, qui consiste à exprimer tous les bits du texte chiffré en fonction de bits du texte clair et les bits de la clé. Afin de résoudre un système de ce type, une technique connue sous le nom de *linéarisation*, peut être utilisée.

Définition 2.22. *La technique de la* linéarisation *consiste à remplacer tout monôme de degré strictement supérieur à* 1 *par une nouvelle variable indépendante de degré* 1.

Après la phase de linéarisation, le système peut être inversé, en utilisant par exemple une élimination de Gauss. Malgré la simplicité de sa description cette attaque élémentaire ne peut pas être appliquée en pratique sur les chiffrements actuels à cause de la grande taille et de la complexité des systèmes générés.

Il existe aujourd'hui des algorithmes plus efficaces pour traiter ce type de problèmes. Un exemple de telles méthodes sont les algorithmes de calcul de bases de Gröbner. Les algorithmes $F4$ et $F5$ de Jean-Charles Faugère [Fau99, Fau02] sont beaucoup utilisés en cryptographie.

En 2002, Nicolas Courtois et Joseph Pieprzyk ont proposé l'idée d'une cryptanalyse algébrique d'AES [CP02] qui exploiterait le fait que l'inversion dans le corps $\mathbf{F}_2^{256}$ employé dans l'AES conduit à des relations quadratiques entre les entrées et les sorties de chaque tour de l'algorithme. L'article d'origine prétendait conduire à une cryptanalyse de l'AES plus rapide que la recherche exhaustive. Mais, plusieurs chercheurs ont montré que cette méthode n'aboutissait pas [Cop02, CL05].

1. *Breaking a "good" cipher should require as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type.*

Attaques algébriques sur les fonctions de hachage Pour l'analyse des fonctions de hachage, l'approche algébrique en termes de résolution d'un système d'équations a été relativement peu utilisée. On doit pourtant noter, le travail M. Sugita, M. Kawazoe, L. Perret et H. Imai [SKPI07] qui proposent une modélisation algébrique de l'attaque en collisions par Wang *et. al* [WY05] sur la fonction de hachage SHA-1. Cette attaque améliore la *technique de modification de message* utilisée dans [WY05] en utilisant des moyens algébriques et conduit à une meilleure complexité de l'attaque sur la fonction SHA-1 réduite à 58 tours. Plus récemment, des travaux algébriques basés sur des implémentations des algorithmes de SAT ("SAT solvers" en anglais) ont ouvert la piste de la recherche d'antécédents sur les fonctions de hachage en utilisant des moyens algébriques. De telles attaques ont été appliquées à la fonction de hachage KECCAK par Morawiecki et Srebrny [MS10] et par Morawiecki [Mor11]. Plus précisement, une préimage pour trois tours de la fonction donnant des empreintes de 1024 bits et absorbant de blocs de messages de 40 bits est présentée dans [MS10]. Cette préimage a été trouvée en 1852 secondes. De plus, des préimages pour un ou deux tours sont données dans [Mor11] pour des variantes ayant une capacité de $c = 160$ bits.

2.2.2 Attaques différentielles d'ordre supérieur

Le principe de la cryptanalyse différentielle

Une attaque très puissante contre les chiffrements par blocs, connue sous le nom de *cryptanalyse différentielle* a été introduite par Shamir et Biham en 1991 [BS91]. Cette méthode a été utilisée pour cryptanalyser un grand nombre de chiffrements par blocs et de fonctions de hachage. Elle a été appliquée entre autres à l'ancien standard de chiffrement par blocs, DES, et a été la première attaque sur cet algorithme plus rapide que la recherche exhaustive [BS93].

Pour une fonction vectorielle F nous introduisons la notion de la *dérivée* de F.

Définition 2.23. *Soit F une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Pour tout $a \in \mathbf{F}_2^n$, la dérivée de F relativement au vecteur a est la fonction définie pour tout $x \in \mathbf{F}_2^n$ par*

$$D_a F(x) = F(x + a) + F(x).$$

Dans le concept classique de la cryptanalyse différentielle, l'attaquant cherche une dérivée $D_a F$ du chiffrement, dont la distribution est éloignée de la distribution uniforme. Plus précisement, l'attaquant essaie de trouver un vecteur $a \in \mathbf{F}_2^n$, que nous appelons *différence en entrée*, tel que $D_a F$ prend une valeur $b \in \mathbf{F}_2^n$, appelée *différence en sortie*, avec une probabilité plus élevée que celle attendue avec une permutation aléatoire.

Définition 2.24. *Une* différentielle *au tour i d'un chiffrement itératif de la fonction de tour F paramétrée par la clé de tour k_i, est un couple (a, b), $a \neq 0$, tel qu'il existe un $x \in \mathbf{F}_2^n$ pour lequel*

$$D_a(F_{k_i} \circ \cdots \circ F_{k_1})(x) = b.$$

La probabilité de la différentielle (a, b) au tour i est définie par

$$\mathrm{Dp}^{(i)}(a, b) = \Pr_X[D_a(F_{k_i} \circ \cdots \circ F_{k_1})(X) = b].$$

Suivant les contextes, cette probabilité est calculée à clé fixée, en moyenne sur les clés ...

Une différentielle pour une fonction vectorielle F peut être visualisée à la Figure 2.1.

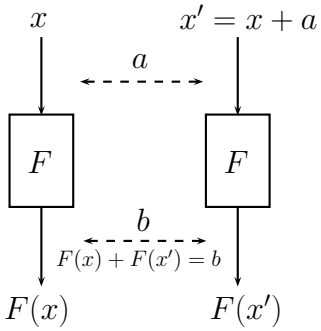

FIGURE 2.1 – Une différentielle pour une fonction F

Une "bonne différentielle" (a, b) pour une famille de fonctions $(F_k)_K$ est une différentielle pour laquelle la quantité

$$\delta_{F_K}(a, b) = \big|\{x \in \mathbf{F}_2^n, D_a F_K(x) = b\}\big|,$$

est élevée pour toute valeur de K. Les différentielles pour lesquelles $\delta_{F_k}(a, b)$ est élevée pour certaines clefs sont également très utiles car elles permettent d'identifier les clefs faibles. La résistance de F_K contre la cryptanalyse différentielle est alors clairement déterminée par les valeurs suivantes :

$$\delta(F_K) = \max_{a \neq 0, b} \delta_{F_K}(a, b).$$

Cette quantité est invariante par la composition de F avec une fonction linéaire. Pour cette raison, la résistance d'un système symétrique contre la cryptanalyse différentielle dépend essentiellement des propriétés de sa partie non-linéaire. Si cette partie est constituée d'une couche de boîtes-S, S, alors la résistance du système contre la cryptanalyse différentielle dépendra de la valeur de $\delta(S)$. Il est alors naturel de rechercher des fonctions S dont la valeur $\delta(S)$ soit la moins élevée possible.

Il est facile de voir que le nombre $\delta(S)$ est toujours pair. Ceci provient du fait que pour tout $x \in \mathbf{F}_2^n$, $D_a S(x) = D_a S(x+a)$. Kaisa Nyberg a prouvé dans [Nyb94] le théorème suivant :

Théorème 2.1 ([Nyb94])**.** *Soit S une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors,*

$$\delta(S) \geq 2^{n-m}.$$

Et, si $n = m$, alors

$$\delta(S) \geq 2. \tag{2.3}$$

Définition 2.25. *Les fonctions atteignant la borne (2.3) sont dites* presque parfaitement non-linéaires *ou APN (abréviation de l'anglais* almost perfect nonlinear*).*

Les différences en entrée et en sortie, sont le plus souvent considérées selon l'opération XOR, c'est-à-dire l'addition bit à bit des deux vecteurs de la même longueur, car c'est l'opération la plus souvent utilisée pour l'insertion des sous-clés dans un chiffrement par blocs. Dans ce cas, pour deux vecteurs $x, x' \in \mathbf{F}_2^n$, leur différence $x + x'$ exprime les positions des bits différents entre x et x'. Néanmoins, selon les opérations utilisées par l'algorithme, il est possible d'utiliser d'autres opérations internes additives. Dans le cas par exemple des fonctions de

hachage de la famille MD-SHA, plusieurs opérations sont considérées modulo 2^{32}. Dans cette situation, la différence entre deux messages est souvent exprimée par la soustraction modulo 2^{32}.

Dans une attaque différentielle appliquée sur un chiffrement par blocs, l'attaquant doit être en mesure de choisir les textes clairs qu'il va ensuite chiffrer. C'est pour cette raison que cette méthode fait partie des attaques dites *à clair choisi*. Dans le cas des fonctions de hachage, un attaquant peut toujours avoir le contrôle des messages à hacher.

Cryptanalyse différentielle d'ordre supérieur

Dans l'attaque différentielle classique, l'étude porte sur l'analyse de la différence entre deux vecteurs à l'entrée et à la sortie de l'algorithme. Il est possible d'étendre de façon naturelle la notion d'une dérivée d'une fonction vectorielle à un ordre supérieur. L'introduction des *dérivées d'ordre supérieur* d'une fonction en cryptographie est généralement attribué à Xuejia Lai en 1994 [Lai94], mais cette notion est déjà présente dans le thèse de John Dillon en 1974 [Dil74].

Définition 2.26. *Soit F une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ et V un sous-espace vectoriel de $\mathbf{F}_2^n$ de dimension k. Nous appelons* dérivée d'ordre k *de F relativement à V la fonction notée par D_V qui est définie pour chaque $x \in \mathbf{F}_2^n$ par :*

$$D_V(x) = D_{a_1} D_{a_2} \dots D_{a_k}(x) = \sum_{v \in V} F(x+v), \tag{2.4}$$

où $(a_1, a_2, \dots, a_k)$ est une base de V.

Nous pouvons voir que la valeur de D_V en x est indépendante du choix de la base pour V.

Selon cette définition, la dérivée d'une fonction donnée par la définition 2.23 correspond à la dérivée d'ordre 1 d'une fonction F.

Exemple 2.4. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ et $V = \langle a, b \rangle$ un sous-espace de $\mathbf{F}_2^n$ de dimension 2. La dérivée d'ordre 2 de F par rapport à V évaluée au vecteur x vaut :*

$$\begin{aligned} D_V(x) &= D_{a_1} D_{a_2}(x) = D_{a_1}(F(x) + F(x+a_2)) \\ &= F(x) + F(x+a_1) + F(x+a_2) + F(x+a_1+a_2). \end{aligned}$$

En 1994, Lars Knudsen a utilisé des dérivées d'ordre supérieur pour attaquer des chiffrements par blocs [Knu95] et a introduit ainsi un nouveau type d'attaque, connu depuis sous le nom de cryptanalyse différentielle d'ordre supérieur. L'idée de cette attaque est basée sur l'observation suivante.

Proposition 2.5. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ de degré d. Alors pour tout $a \in \mathbf{F}_2^n$, nous avons*

$$D_a F \leq d - 1.$$

L'idée derrière cette proposition est très naturelle. Puisque toute fonction dans un corps fini peut être vue comme un polynôme multivarié, dériver diminue d'au moins 1 le degré. De cette propriété, et en dérivant la fonction F "suffisamment de fois", nous déduisons facilement la proposition suivante.

Proposition 2.6 ([Lai94]). *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ de degré d. Alors pour tout sous-espace $V \subset \mathbf{F}_2^n$ de dimension strictement supérieure à d, nous avons*

$$D_V(x) = 0, \text{ pour tout } x \in \mathbf{F}_2^n.$$

Nous allons voir comment la proposition 2.6 peut être appliquée pour conduire à une attaque contre un chiffrement par blocs. Pour ceci, nous allons décrire une des premières attaques de ce genre à avoir été appliquée. Il s'agit de l'attaque de Thomas Jakobsen et Lars Knudsen [JK97] contre le chiffrement par blocs $\mathcal{KN}$, présenté par Lars Knudsen et Kaisa Nyberg en 1995 [NK95].

Le chiffrement par blocs $\mathcal{KN}$

Le chiffrement par blocs $\mathcal{KN}$ (pour les initiales de ces auteurs Knudsen et Nyberg) est un chiffrement de type Feistel, proposé en 1995 [NK95]. Il s'agit d'une construction chiffrant des blocs de messages de 64 bit en utilisant une clé secrète de 198 bits. En particulier, aucun cadencement de clé n'est utilisé, donc chaque sous-clé est simplement constitué de 33 bits consécutifs de la clé maître. Le nombre d'itérations est fixé à 6. La permutation de tour est définie de la manière suivante :

$$\begin{array}{rcl} \mathbf{F}_2^{32} \times \mathbf{F}_2^{32} & \to & \mathbf{F}_2^{32} \times \mathbf{F}_2^{32} \\ (x, y) & \mapsto & (y, F_{k_i}(y) + x). \end{array}$$

La fonction de tour paramétrée par la sous-clé k_i, $F_{k_i}(x)$, est :

$$\begin{array}{rcl} \mathbf{F}_2^{32} & \to & \mathbf{F}_2^{32} \\ x & \mapsto & F_{k_i}(x) = x + \mathcal{T} \circ S(\mathcal{E}(x) + k_i), \end{array}$$

où $\mathcal{E}$ est une expansion linéaire de $\mathbf{F}_2^{32}$ dans $\mathbf{F}_2^{33}$ qui concatène au vecteur d'entrée une combinaison affine de ses coordonnées, $\mathcal{T}$ une troncation linéaire de $\mathbf{F}_2^{33}$ dans $\mathbf{F}_2^{32}$ qui écarte un bit du vecteur d'entrée et S la fonction puissance x^3 sur le corps $\mathbf{F}_{2^{33}}$.

Nous pouvons voir un tour du chiffrement à la figure 2.2.

La conception de ce chiffrement avait comme but principal de présenter une méthode générale de construction d'un chiffrement de type Feistel résistant aux attaques différentielles et linéaires. Ce chiffrement est d'ailleurs aussi connu sous le nom CRADIC (abréviation de "Cipher Resistant Against Differential Cryptanalysis" en anglais [Nyb12]). Le choix de la fonction S, qui est le seul élément non-linéaire de cette construction, joue un rôle crucial dans cette direction. Pour cette raison, la fonction x^3 sur le corps $\mathbf{F}_{2^{33}}$, qui est une fonction presque courbe, a été choisie. Ces fonctions sont connues pour avoir une bonne résistance contre les attaques linéaires et différentielles.

Par ailleurs, les auteurs ont présenté des bornes inférieures sur la probabilité des meilleurs chemins différentiels et linéaires en prouvant que ce chiffrement est résistant contre ces deux types d'attaques. Plus précisement, ils ont prouvé que toutes les différentielles pour 4 et 5 tours sont de probabilité au plus 2^{-61}.

Une caractéristique très important de la fonction x^3 sur $\mathbf{F}_{2^{33}}$ est que son degré algébrique est assez bas, seulement égal à 2. Cet élément a été utilisé par Jakobsen et Knudsen pour monter une attaque différentielle d'ordre supérieur contre le chiffrement $\mathcal{KN}$.

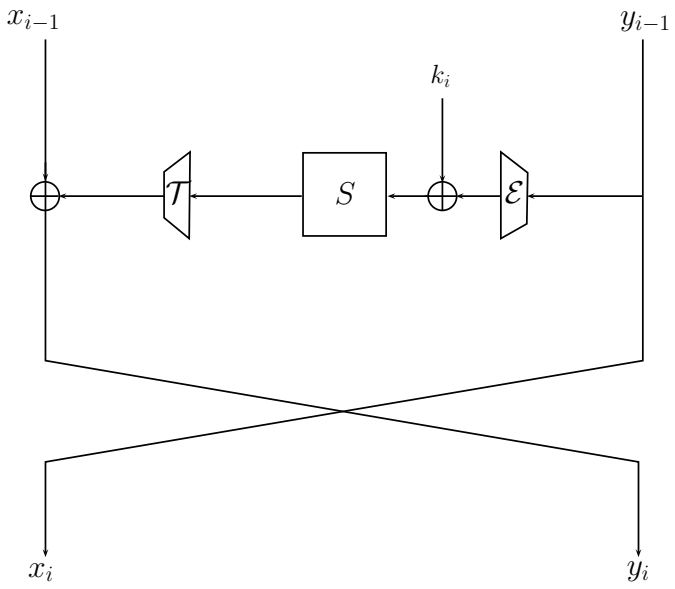

FIGURE 2.2 – Le tour i du chiffrement par blocs $\mathcal{KN}$

Attaque contre le chiffrement $\mathcal{KN}$

Nous allons décrire l'attaque contre le chiffrement $\mathcal{KN}$, présentée dans [JK97]. Nous faisons cela premièrement pour montrer le principe général d'une attaque différentielle d'ordre supérieur et ensuite pour montrer l'importance de l'utilisation d'une fonction de substitution de degré élevé afin que le cryptosystème résiste à cette attaque.

Le but de cette attaque est de récupérer la clé secrète k du chiffrement. Elle consiste à retrouver d'abord la sous-clé de 33 bits, k_6, utilisée pour le dernier tour et à répéter ensuite la même méthode pour récupérer les autres sous-clés. Si la taille de la clé n'est pas grande, alors les autres sous-clés peuvent également être retrouvées avec une recherche exhaustive. Très souvent, les deux méthodes sont combinées.

Nous considérons des clairs choisis, c'est-à-dire des vecteurs d'entrée $(x_0, y_0) \in \mathbf{F}_2^{32} \times \mathbf{F}_2^{32}$ du chiffrement $\mathcal{KN}$ dont la partie droite est égale à une constante de 32 bits, $y_0 = c$. Alors, nous pouvons décrire les bits de la partie droite du chiffrement après r tours, x_r, comme des fonctions des bits de x_0.

Nous allons suivre l'évolution de la partie droite du chiffrement, y_i.

$$\begin{aligned}
y_0(x) &= c \\
y_1(x) &= x + F_{k_1}(c) := x + c' \\
y_2(x) &= F_{k_2}(x + c') + c \\
y_3(x) &= F_{k_3}(F_{k_2}(x + c') + c) + x + c' \\
y_4(x) &= F_{k_4}(F_{k_3}(F_{k_2}(x + c') + c) + x + c') + F_{k_2}(x + c') + c
\end{aligned}$$

Nous allons maintenant évaluer le degré algébrique de y_4. Il est facile de voir que son degré est inférieur ou égal au degré de la fonction

$$G = F_{k_4} \circ F_{k_3} \circ F_{k_2}.$$

La seule composante non-linéaire de la fonction de tour F_{k_i} est la fonction puissance x^3 sur $\mathbf{F}_{2^{33}}$, qui est une fonction quadratique. Par conséquent, le degré de F_{k_i} est 2. Ceci implique que le degré de G et donc le degré de y_4 est au plus 2^3.

Selon la proposition 2.6, si V est un sous-espace de $\mathbf{F}_2^{32}$ de dimension $\dim(V) = 9$, nous avons que :

$$D_V y_4(x) = 0, \qquad \forall x \in \mathbf{F}_2^{32}.$$

À cause de l'équation (2.4) cette relation s'écrit

$$\sum_{v \in V} y_4(v + w) = 0, \qquad \forall w \in \mathbf{F}_2^{32}. \tag{2.5}$$

Cela est vrai pour toute clé secrète k. A partir de la Figure 2.3 nous pouvons maintenant établir l'équation suivante :

$$x_6(x) = F_{k_6}(y_5(x)) + y_4(x),$$

et en inversant les termes,

$$y_4(x) = x_6(x) + F_{k_6}(y_5(x)). \tag{2.6}$$

Finalement, en combinant les équations (2.5) et (2.6) et en fixant une constante $w \in \mathbf{F}_2^{32}$, nous obtenons l'équation suivante, que nous allons appeler *équation de l'attaque*.

$$\sum_{v \in V} F_{k_6}(y_5(v + w)) + \sum_{v \in V} x_6(v + w) = 0. \tag{2.7}$$

L'équation (2.7) est utilisée pour retrouver la sous-clé k_6 du chiffrement en faisant une recherche exhaustive parmi toutes les clés de $\mathbf{F}_2^{33}$. La clé pour laquelle l'équation (2.7) est vérifiée, est la vraie clé. Cependant, puisque l'équation porte sur 32 bits, pour une mauvaise clé (il y en a $2^{33} - 1$ au total), l'équation (2.7) est vérifiée avec probabilité $1/2^{32}$. Donc il y a $\frac{2^{33}-1}{2^{32}} \approx 2$ "fausses alertes". Pour cette raison, afin d'éliminer la mauvaise clé, il faut refaire l'attaque avec un autre couple d'entrées.

Cela peut s'expliquer de la façon suivante. On considère des chiffrés (x_6, y_6) correspondant aux 2^9 clairs de la forme $(v + w, c)$, où v décrit un sous-espace V de dimension 9 et w et c sont des constantes quelconques. Pour chaque valeur de la sous-clé k_6 on déchiffre partiellement x_6. Si la clé est la bonne, alors le déchiffrement annule les effets du chiffrement, et donc la partie gauche de l'équation (2.7) s'annule. Dans le cas contraire, le résultat est équivalent à l'ajout d'un tour supplémentaire au chiffrement et donc comme le degré total augmente, l'équation ne sera plus vraie.

Cette attaque nécessite le choix de 2^9 textes clairs et a une complexité moyenne en temps de 2^{33+8}. L'élément crucial pour sa réussite est le degré algébrique faible de la fonction de tour, F_{k_i}.

Cette attaque peut s'appliquer à tous les réseaux de Feistel dont la fonction de tour possède un degré algébrique faible. Si le degré du chiffrement est au plus d après $r - 2$ tours, et que le nombre de bits de chaque sous-clé est b, alors il existe une attaque sur r tours avec complexité en données 2^{d+1} et une complexité moyenne en temps 2^{b+d}.

Cette méthode peut également être utilisée pour distinguer un chiffrement par blocs de type Feistel réduit à r tours d'une permutation aléatoire. Si F_{k_i} est de degré 2, ce distingueur a une complexité en temps et en données égale à $2^{2^{r-2}+1}$. En particulier, il est possible de distinguer 4 et 5 tours de chiffrement d'une permutation aléatoire avec complexité 2^5 et 2^9 respectivement. Ce résultat doit être comparé aux complexités des meilleures attaques génériques sur 4 et 5 tours d'un chiffrement de Feistel [Pat04].

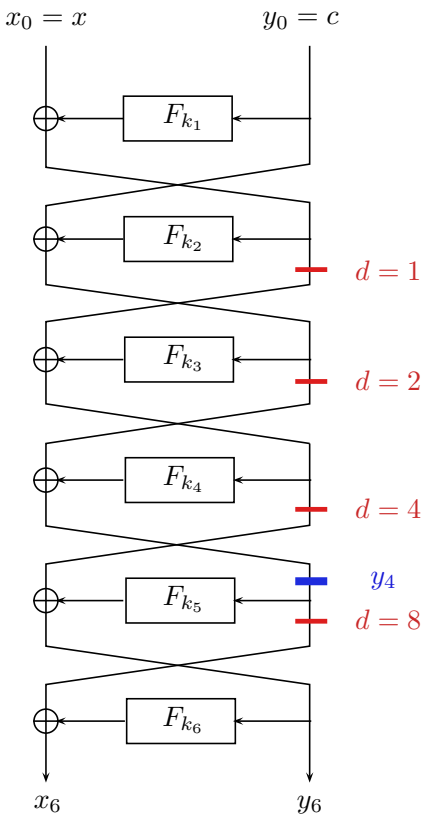

FIGURE 2.3 – L'attaque différentielle d'ordre supérieur sur le chiffrement $\mathcal{KN}$

Attaques génériques sur 4 et 5 tours d'un chiffrement de Feistel

Pour attaquer quatre tours d'un chemin de Feistel dont les sous-blocs droits et gauches sont de taille n, un attaquant peut procéder de la façon suivante. Il commence par choisir plusieurs entrées $(x_0, y_0) = (x^i, y)$ où tous les blocs droits sont égaux à une valeur constante, soit $y = c$. Nous allons ici noter par F_1, F_2, F_3, F_4 les quatre premières permutations internes du schéma de Feistel, par y_4 la partie droite du chiffrement après 4 tours, et par x_4 la partie gauche.

Rappelons l'évolution de la partie droite du chiffrement , $y_r(x^i)$ jusqu'à 4 tours.

$$\begin{aligned} y_1(x^i) &= x^i + F_1(c) \\ y_2(x^i) &= c + F_2(x^i + F_1(c)) \\ y_3(x^i) &= x^i + F_1(c) + F_3(c + F_2(x^i + F_1(c))) \end{aligned}$$

De plus $x_4(x^i) = y_3(x^i)$. L'attaquant teste ensuite l'égalité suivante :

$$x^i + x^j = x_4(x^i) + x_4(x^j).$$

Pour une permutation aléatoire, cette égalité arrive au bout de $2^{\frac{n}{2}}$ évaluations avec probabilité supérieure à $1/2$ à cause du paradoxe des anniversaires. Nous allons montrer, que pour un schéma de Feistel, cette égalité ne peut jamais être réalisée.

Comme $x^i \neq x^j$ alors $y_1(x^i) \neq y_1(x^j)$. Ensuite, comme F_2 est une permutation, alors $F_2(x^i + F_1(c)) \neq F_2(x^j + F_1(c))$ et donc $F_3(x^i) \neq F_3(x^j)$. La même chose est vraie pour $y_5(x) + x$, puisque F_3 est également une permutation. Nous déduisons alors que $x_4(x^i) + x^i \neq x_4(x^j) + x^j$.

Ceci conduit à une attaque pour 4 tours d'un schéma de Feistel de complexité en mémoire et en temps égale à $2^{\frac{n}{2}}$. Pour 5 tours, l'attaque ressemble à celle pour 4 tours, donc nous allons omettre les détails.

Nous allons continuer en décrivant une autre attaque qui est liée à la cryptanalyse différentielle d'ordre supérieure, et donc au degré algébrique d'une primitive cryptographique. Cette attaque est appelée *attaque cube*.

2.2.3 Attaque cube

Un nouveau type d'attaque, appelé *attaque cube* a été introduit par Itai Dinur et Adi Shamir en 2008 [DS09]. Il s'agit d'une variante des attaques différentielles d'ordre supérieur. Elle exploite aussi des équations de degré algébrique faible afin de récupérer la clé secrète.

De façon générale, les attaques cube peuvent être appliquées à tous les cryptosystèmes symétriques possédant une clé secrète (chiffrements à flot, chiffrements par blocs, MAC), quelle que soit la taille du système, à condition que certaines fonctions constituantes aient un degré algébrique faible. La connaissance des détails du système n'est pas nécessaire, celui-ci est traité comme une boîte noire. Chaque bit de sortie du système est représenté par un polynôme p en $n + m$ variables, dont la représentation est a priori inconnue. Ces variables sont de deux types ; les variables publiques du système, $u_1, \ldots, u_m$, qui correspondent à l'IV dans un chiffrement à flot, ou au message dans un chiffrement par blocs, et les variables secrètes, $x_1, \ldots, x_n$. Le but de l'attaquant est de retrouver les variables secrètes en résolvant un système linéaire. Pour ceci, il utilise des accès à une boîte noire qui évalue p sur l'entrée $(x_1, \ldots, x_n, u_1, \ldots, u_m)$.

Les définitions suivantes sont introduites par les auteurs.

Définition 2.27. *Soit p un polynôme à n variables $x_1, \ldots, x_n$, et I un sous-ensemble de $\{1, \ldots, n\}$ de cardinalité $|I| = k$. Si $t_I = x_{i_1} \cdots x_{i_k}$, où $\{i_1, \ldots, i_k\} = I$, nous notons la factorisation de p par rapport à t_I de la façon suivante :*

$$p(x_1, \ldots, x_n) = t_I \cdot p_{S(I)} + q(x_1, \ldots, x_n).$$

Le polynôme $p_{S(I)}$ est appelé le superpoly *de I dans p.*

Cette notion est illustrée dans l'exemple suivant.

Exemple 2.5. *Soit*

$$p(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3 + x_1x_3x_4 + x_2x_3 + x_1 + x_4 + x_5,$$

un polynôme à 5 variables de degré 3. Si $I = \{2, 3\}$, p peut être représenté comme suit :

$$p(x_1, x_2, x_3, x_4, x_5) = x_2x_3(x_1 + 1) + x_1x_3x_4 + x_1 + x_4 + x_5,$$

avec,

$$\begin{aligned} t_I &= x_2x_3 \\ p_{S(I)} &= x_1 + 1 \\ q(x_1, x_2, x_3, x_4, x_5) &= x_1x_3x_4 + x_1 + x_4 + x_5. \end{aligned}$$

Définition 2.28. *Un monôme t_I est appelé* maxterm *si* $\deg(p_{S(I)}) = 1$*, c'est-à-dire si le superpoly de I dans p est affine.*

Tout ensemble I de cardinalité k peut définir un sous-espace vectoriel V de $\mathbf{F}_2^n$, de dimension k. Ce sous-espace peut être représenté comme un cube booléen C_I de dimension k, dont chaque sommet correspond à un vecteur de V. C'est la raison pour laquelle cette classe d'attaques est connue sous le nom cube. Pour un vecteur $v \in C_I$, nous notons par $p(v)$ l'évaluation de p en v.

Le théorème suivant est l'observation centrale dans [DS09].

Théorème 2.2 (Théorème 1 [DS09])**.**

$$p_{S(I)} = D_{C_I} p.$$

Démonstration. Nous montrons ce résultat par récurrence sur $|I| = k$.

Pour $k = 1$: Soit $I = \{e_i\}$. Alors, pour tout $x \in \mathbf{F}_2^n$ nous avons

$$\begin{aligned} D_{C_I} p(x) &= D_{e_i} p(x) = p(x) + p(x + e_i) \\ &= p(x) + (q(x) + x_i p_{S(I)}(x) + p_{S(I)}(x)) \\ &= p_{S(I)}(x). \end{aligned}$$

Supposons maintenant que le résultat est vrai pour chaque I de dimension k et soit I' un ensemble de dimension $k+1$. Soit $i \in I' \setminus I$. Alors pour tout $x \in \mathbf{F}_2^n$ nous avons

$$\begin{aligned} D_{C_{I'}} p(x) &= D_{e_i}(D_{C_I} p(x)) \\ &= p_{S(I)}(x) + p_{S(I)}(x + e_i) \\ &= p_{S(I')}, \end{aligned}$$

où la dernière égalité est obtenue en utilisant le fait que la dérivée de $p_{S(I)}$ par rapport à e_i est égale au superpoly de $\{i\}$ dans $p_{S(I)}$. □

En utilisant ce théorème il est possible de retrouver la valeur d'un superpoly en faisant la somme sur tous les vecteurs du cube. Nous pouvons voir cela avec un exemple.

Exemple 2.6. *Soit p un polynôme à 5 variables, avec v_1, v_2 étant des valeurs publiques, et x_1, x_2, x_3 des variables secrètes :*

$$p(v_1, v_2, x_1, x_2, x_3) = v_1 v_2 x_1 + v_1 v_2 x_2 + v_1 x_1 x_2 + v_2 x_1 + v_1 + x_2 + x_3.$$

Nous considérons l'ensemble I qui correspond aux variables publiques, $I = \{1, 2\}$ et $t_I = x_1 x_2$. Alors,

$$p(v_1, v_2, x_1, x_2, x_3) = v_1 v_2 (x_1 + x_2) + v_1 x_1 x_2 + v_2 x_1 + v_1 + x_2 + x_3,$$

avec $p_{S(I)} = x_1 + x_2$ et $q(v_1, v_2, x_1, x_2, x_3) = v_1 x_1 x_2 + v_2 x_1 + v_1 + x_2 + x_3$.

En sommant p sur le cube I on obtient :

$$\begin{aligned} p_I &= 0 \cdot 0(x_1 + x_2) + 0 \cdot x_1 \cdot x_2 + 0 \cdot x_1 + 0 + x_2 + x_3 \\ &+ 0 \cdot 1(x_1 + x_2) + 0 \cdot x_1 \cdot x_2 + 1 \cdot x_1 + 0 + x_2 + x_3 \\ &+ 1 \cdot 0(x_1 + x_2) + 1 \cdot x_1 \cdot x_2 + 0 \cdot x_1 + 1 + x_2 + x_3 \\ &+ 1 \cdot 1(x_1 + x_2) + 1 \cdot x_1 \cdot x_2 + 1 \cdot x_1 + 1 + x_2 + x_3 \\ &= x_1 + x_2. \end{aligned}$$

Grâce à la façon dont l'ensemble I a été choisi, le superpoly de I n'est constitué que d'une combinaison linéaire des variables secrètes. Ici, t_I est un maxterm de degré 2. De façon générale, si p est de degré d, le degré d'un maxterm est au plus $d-1$. Par conséquent, pour retrouver un superpoly, au plus $\mathcal{O}(2^{d-1})$ évaluations de p sont nécessaires.

L'attaque se divise en deux parties ; une phase de *précalcul* et une phase *en direct*. Pendant la phase de précalcul, l'attaquant essaie de trouver le plus possible de maxterms, n'impliquant que de variables publiques ayant des superpolys linéairement indépendants en les bits secrets. Afin de déterminer si un polynôme t_I est un maxterm, un test de linéarité probabiliste [BLR90] peut être utilisé sur $p_{S(I)}$.

Après la phase de précalcul, l'attaquant possède une liste de maxterms et leurs superpolys (linéaires) correspondant. Une fois les variables secrètes fixées, l'attaquant peut former un système linéaire d'équations qu'il peut ensuite résoudre pour récupérer la clé secrète. Si le nombre d'équations n'est pas suffisant, certains bits de la clé peuvent être recherchés de façon exhaustive.

Nous avons omis les détails de cette attaque, qui peuvent être retrouvés dans [DS09]. L'objectif de cette section était de mentionner les différents types d'attaques qui peuvent être menés contre un chiffrement symétrique si son degré algébrique est suffisamment faible.

Dans l'article d'origine, les auteurs utilisent les attaques cubes pour cryptanalyser des versions réduites du chiffrement à flot Trivium [CP05]. Par exemple, les auteurs ont réussi à récupérer les 80 bits de la clé secrète de Trivium, réduit à 767 tours d'initialisation, en 2^{36} opérations. Joel Lathrop [Lat05] a appliqué ces attaques aux configurations MAC des versions réduites de certaines fonctions candidates au concours SHA-3.

Distingueurs basés sur les cubes Le principe des attaques cube peut être utilisé pour détecter des comportements non-aléatoires et monter des distingueurs sur les primitives cryptographiques. Cette technique, présentée par Aumasson *et al.* [ADMS09], est connue sous le nom *testeurs de cube* ou *"cube testers"* en anglais. Le degré algébrique faible ou l'équilibre d'une fonction font partie des propriétés qui peuvent être testées. Dans [ADMS09], les auteurs ont présenté des applications contre la fonction de hachage MD6 [Riv08] et les chiffrements à flot Trivium et Grain [HJM07].

2.2.4 Attaque intégrale

Le dernier type d'attaque que nous allons présenter dans cette section est l'attaque intégrale. Cette attaque exploite à la fois des faiblesses de la partie non-linéaire et de la partie linéaire du système.

En 1997, Joan Daemen, Lars Knudsen et Vincent Rijmen ont présenté un nouveau chiffrement par blocs, appelé SQUARE [DKR97]. Lors de la conception de cet algorithme les auteurs ont découvert une nouvelle attaque de type clair-choisi qui pouvait casser six tours de SQUARE. Suite à cette attaque, les auteurs ont renforcé la fonction en ajoutant deux tours supplémentaires et ont publié la description de l'algorithme avec les détails de cette nouvelle attaque, qui a ensuite été référencée sous le nom de *square attaque*.

La première forme de cette attaque pouvait s'appliquer relativement bien aux chiffrements de type SPN, et plusieurs fonctions ont été cryptanalysées ainsi [DBRP99, BRJ+02, YPK02, HLL+02]. Stephan Lucks a ensuite généralisé l'attaque aux chiffrements non SPN [Luc02] en la nommant *attaque par saturation* et l'a utilisée pour attaquer Twofish [SKW+98], un chiffrement par blocs de type Feistel. Cette attaque est aujourd'hui plus connue sous le nom

de *cryptanalyse intégrale*, grâce à l'article de Lars Knudsen et David Wagner [KW02] qui ont unifié les diverses techniques et les aspects de ce type de cryptanalyse dans un cadre commun.

Nous allons ici présenter les éléments principaux de cette attaque.

Un élément central de cette méthode est la notion de *multi-ensemble*, qui généralise la notion de l'ensemble.

Définition 2.29. *Un* multi-ensemble *est un couple* (S, m)*, où* S *est un ensemble appelée support et* $m : S \to \mathbb{N}$ *une fonction appelée multiplicité.*

Nous pouvons voir un tel objet comme un ensemble d'éléments de S, où chaque élément peut apparaître plusieurs fois.

Exemple 2.7. *L'objet* $\{1, 2, 2, 2, 3, 3, 4\}$ *peut être vu comme un multi-ensemble, où* $S = \{1, 2, 3, 4\}$*,* $m(1) = 1, m(2) = 3, m(3) = 2$ *et* $m(4) = 1$*.*

Nous considérons ici l'attaque classique qui s'applique sur les chiffrements *orientés mots*, dont la plupart des chiffrements de type SPN font partie. Nous allons noter par w le nombre de mots dans un bloc de texte clair ou un bloc de texte chiffré. Par exemple, dans l'AES, les données sont représentées sous la forme d'une matrice d'octets 4×4. Pour cette raison, un mot dans le cadre d'une attaque intégrale sur l'AES est un octet, et donc comme la taille de l'état est de 128 bits, $w = 16$. Nous notons dans la suite par M le nombre de clairs/chiffrés utilisés simultanément dans l'attaque. Ces textes clairs sont choisis de façon que le multi-ensemble de tous leur i-èmes mots ($1 \leq i \leq w$) vérifie une propriété spécifique pour au moins une valeur de i. Dans la plupart des cas, l'attaquant utilise un nombre de clairs/chiffrés égal au nombre de mots possibles.

Définition 2.30. *Si* S *est un multi-ensemble d'éléments d'un groupe* G*, on appelle* intégrale *sur* S *la somme de tous les éléments dans* S*,*

$$\sum_{v \in S} v.$$

Pour la plupart des attaques de ce type, les vecteurs sont des chaînes des bits. Pour cette raison, l'addition bit-à-bit des vecteurs est considérée. Dans une attaque intégrale, l'attaquant essaie de prédire les valeurs des intégrales après un certain nombre de tours du chiffrement. Nous pouvons distinguer trois cas différents, selon que tous les i-ièmes mots sont égaux, distincts ou que leur somme est une valeur spécifique :

1. $\mathcal{C}$: Si un mot i ($1 \leq i \leq w$) est noté par le symbole $\mathcal{C}$, cela signifie que tous les clairs dans le multi-ensemble sont égaux sur le mot i.
2. $\mathcal{A}$: Si le mot i est noté par le symbole $\mathcal{A}$, alors tous les clairs considérés pour l'attaque, sont distincts sur le mot i. Si M est égal au nombre de mots possibles, alors la somme des tous les i-èmes mots est égale à zéro.
3. $\mathcal{S}$: Si le mot i est noté par le symbole $\mathcal{S}$, alors la somme de tous les i-èmes mots peut être prévue à l'avance.
4. ? : Aucune information n'est connue pour le multi-ensemble formé par les i-èmes mots.

Une attaque intégrale pour 3 tours de l'AES

Nous allons ici présenter un exemple classique de distingueur intégral sur trois tours de l'AES, qui conduit à une attaque sur 4, 5 et 6 tours du chiffrement, en retrouvant une partie de la clé secrète. Ce distingueur peut être observé à la figure 2.4.

Soient 256 textes clairs qui prennent toutes les $2^8 = 256$ valeurs possibles sur le premier octet et qui sont égaux à une valeur constante sur les 15 octets restants. Nous allons voir comment cet état est transformé après l'application de trois tours de l'AES.

La première fonction à être appliquée sur l'état est la transformation `SubBytes`. Cette transformation est composée de 16 applications parallèles d'une boîte-S, S. Chaque boîte-S S s'applique sur un octet x_i, $S(x_i) = y_i$. Si tous les x_i sont égaux, ceci est également le cas pour les y_i. Dans le cas contraire, si tous les x_i sont différents entre eux, comme S est une permutation, tous les y_i seront eux aussi tous différents. En suivant la notation précédemment introduite, `SubBytes` transforme un mot $\mathcal{C}$ en un mot $\mathcal{C}$ et un mot $\mathcal{A}$ en un mot $\mathcal{A}$. La transformation `ShiftRows` est une simple rotation des octets de chaque ligne et son application ne modifie alors pas la situation. Ceci explique la première partie du distingueur de la figure 2.4.

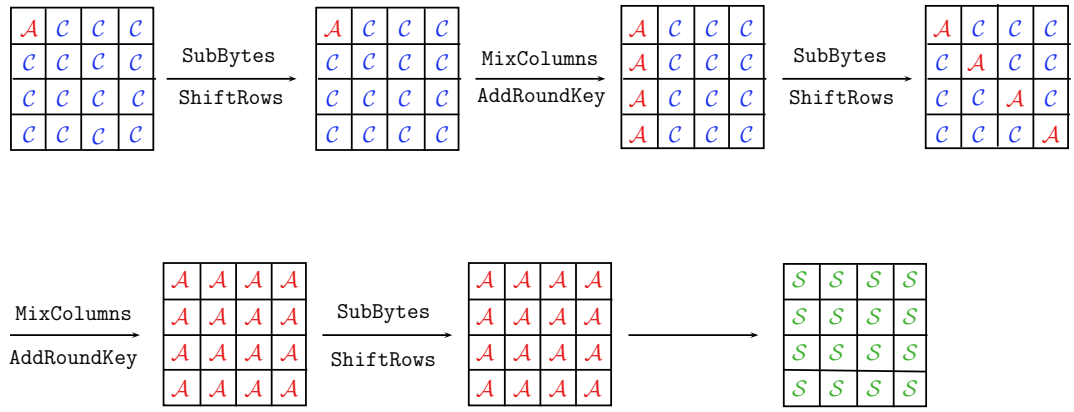

FIGURE 2.4 – Distingueur pour 3 tours de l'AES, avec $\mathcal{S} = 0$

La transformation `MixColumns` est une transformation linéaire qui s'applique en parallèle sur les colonnes de l'état. Elle peut être vue comme une combinaison linéaire des quatre octets de chaque colonne. Pour expliquer comment l'intégrale se transforme sous `MixColumns` nous allons examiner un exemple trivial. Supposons que nous avons une transformation linéaire qui calcule le résultat de l'addition de deux octets de l'état, x_i, y_i, bit à bit : $w_i = x_i + y_i$. Alors, la somme de tous les w_i est égale à $\sum_i w_i = \sum_i x_i + \sum_i y_i$. Si les sommes $\sum_i x_i$ et $\sum_i y_i$ sont connues alors nous pouvons déterminer la somme $\sum_i w_i$. Si par exemple tous les x_i et les y_i sont constants ceci sera aussi le cas pour les w_i. Autrement, si les x_i sont égaux à une valeur constante et les y_i sont tous différents, alors les w_i seront eux aussi tous différents. Grâce à cette propriété, chaque octet de la première colonne de l'état prend toutes les valeurs possibles, après l'application de `MixColumns` au premier tour. Ceci explique également pourquoi tous les octets de l'état prennent toutes les 256 valeurs après deux tours de chiffrement. L'application `AddRoundKey`, qui consiste en l'ajout d'une constante (secrète) octet par octet conserve les propriétés $\mathcal{C}$ et $\mathcal{A}$, exactement comme la fonction `SubBytes`.

Après 3 tours de chiffrement, la somme de toutes les 256 valeurs de chaque octet est égale

à zéro. Ceci vient du fait que la somme de tous les éléments de $\mathbf{F}_2^8$ est nulle.

Ce distingueur sur trois tours de l'AES peut conduire à une attaque sur quatre tours du chiffrement en retrouvant la clé du dernier tour octet par octet. Pour cela il faut noter que le dernier tour de l'AES ne contient pas `MixColumns`. Ensuite pour récupérer un octet de la clé, il suffit pour chaque candidat d'inverser partiellement le calcul et tester si la propriété prévue par l'intégrale est vérifiée. La clé pour laquelle la somme s'annule est avec une grande probabilité la bonne clé. Le même processus doit ensuite être répété pour récupérer les octets restant de la sous-clé.

Cette attaque peut être étendue à cinq et à six tours du chiffrement [DR99]. Elle a été ensuite améliorée par Ferguson *et al.* [FKL+00] et par Gilbert et Minier [GM00].

L'attaque intégrale a été également étendue à l'ordre supérieur [KW02].

Lien avec la cryptanalyse différentielle d'ordre supérieur

Les intégrales sont liées aux différentielles d'ordre supérieur. Plus précisément, une dérivée d'ordre d, est comme nous l'avons vu à la section 2.2.2, la somme des images des 2^k vecteurs d'un sous-espace de dimension k, et peut par conséquence être vue comme une intégrale.

Chapitre 3

Distingueurs à somme nulle

Sommaire

3.1 Une application sur l'AES 51
3.2 Structures à somme nulle et leurs propriétés 53
3.2.1 Taille minimale d'une somme nulle 53
3.2.2 Algorithmes génériques 54
3.3 Les partitions en sommes nulles 56
3.3.1 Algorithme générique pour trouver des partitions en sommes nulles . 56
3.4 Exploiter le degré algébrique de la partie non-linéaire 57
3.4.1 Des partitions en sommes nulles à partir des différentielles d'ordre supérieur 58
3.4.2 Trouver une borne supérieure pour le degré 59
3.5 Exploiter la structure de la partie linéaire 61
3.5.1 Propriété pour un tour 61
3.5.2 Propriété pour plusieurs tours 62
3.6 Application à la permutation interne de Keccak 67
3.6.1 Description de la permutation Keccak-f 68
3.6.2 Les partitions en sommes nulles dans [AM09] 69
3.6.3 Partitions en sommes nulles pour 18 tours de Keccak-f 70
3.6.4 Partitions en sommes nulles pour 19 tours de Keccak-f 71
3.6.5 Partitions en sommes nulles pour 20 tours de Keccak-f 71
3.7 Application à la permutation de finalisation de Hamsi-256 72
3.7.1 Description de Hamsi-256 72

Dans les chapitres qui suivent, nous allons présenter et analyser les travaux effectués au cours de cette thèse. Le premier travail concerne une analyse des structures à somme nulle.

Lors de leur étude de distingueurs sur les chiffrements par blocs dans le modèle à clé connue, Lars Knudsen et Vincent Rijmen mentionnent pour la première fois ce type de structure [KR07].

3.1 Une application sur l'AES

Dans un travail apparu en 2007 [KR07], Knudsen et Rijmen étudient la sécurité des chiffrements par blocs dans un modèle où la clé est supposée connue. Étudier la sécurité d'un

chiffrement quand la clé est donnée peut paraître un concept dénué de sens. Cependant, cette approche peut présenter des intérêts pour un attaquant. Plus précisément, si la clé secrète est donnée, l'attaquant peut essayer de voir si la permutation qui a été fixée par la clé, présente des biais importants ou d'autres comportements qu'une permutation aléatoire ne posséderait pas. Des faiblesses de ce type ne sont probablement pas désirées pour un chiffrement par blocs. De plus, comme plusieurs fonctions de hachage sont construites à partir des chiffrements par blocs, une faiblesse de ce type pour le chiffrement sous-jacent peut conduire à un véritable problème de sécurité pour la fonction de hachage, dans laquelle le message, connu, joue le rôle de clé.

Les auteurs analysent donc AES sous ce modèle de sécurité. Ils établissent un distingueur basé sur l'attaque intégrale, pour 7 tours du chiffrement. Une caractéristique des attaques à clé connue est que l'attaquant est libre de calculer dans les deux sens du chiffrement et peut même commencer le calcul par le milieu. Nous n'allons pas décrire ce distingueur en détails, car cela dépasse l'intérêt de ce chapitre. Cependant, sa forme nous intéresse car il s'agit du premier distingueur basé sur une structure à somme nulle, appliquée à une primitive symétrique.

Nous avons déjà vu dans la section 2.2.4 comment construire une intégrale sur 3 tours de l'AES. Dans le cadre du travail décrit dans [KR07], les auteurs ont établi deux intégrales différentes en utilisant 2^{32} textes clairs. La première intégrale s'étale sur 4 tours et chaque octet des textes chiffrés est équilibré. Ceci signifie en particulier que la somme des chiffrés est nulle. La deuxième intégrale s'étale sur 3 tours mais sur la fonction inverse. L'équilibre des octets est garanti ici aussi et conduit à une deuxième somme nulle. Les auteurs choisissent ensuite une structure de 2^{56} textes après 3 tours de chiffrement, qui sont différents sur 7 octets et égaux sur les 9 octets restant. Ils calculent ensuite trois tours en arrière suivant l'intégrale pour 3 tours et 4 tours en avant suivant l'autre intégrale. Ils obtiennent finalement une structure de 2^{56} textes, dont la somme XOR est nulle et dont la somme des images par l'AES réduit à 7 tours, est également nulle. Cette attaque peut être visualisée à l'aide de la figure 3.1. La notation A_0^7 signifie que la concatenation de tous les octets d'indice 0 prennent toutes les $2^{7\cdot 8}$ valeurs de 56 bits exactement une fois. Si un octet est noté par $\mathcal{A}^7$, cela signifie qu'en suivant l'integrale cet octet est équilibré, c'est-à-dire qu'il prend toutes les valeurs $2^{8\cdot(7-1)}$ fois exactement.

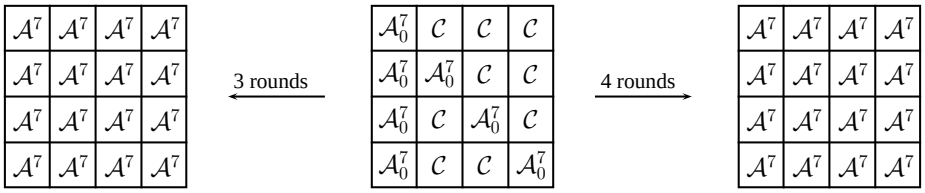

FIGURE 3.1 – Une integrale pour 7 tours de l'AES utilisant 2^{56} textes clairs.

Nous appelons une structure de ce type, *somme nulle*. La terminologie de *distingueur à somme nulle* a été introduite pour la première fois par Jean-Philippe Aumasson et Willi Meier [AM09]. Dans cet article, ils ont appliqué le distingueur aux permutations de certaines des fonctions candidates au concours SHA-3, à savoir KECCAK, *Luffa* et Hamsi.

3.2 Structures à somme nulle et leurs propriétés

Au cours de cette thèse, nous avons étudié et formalisé la notion de distingueur à somme nulle. Ce travail a été présenté à la conférence SAC 2010 [BC11b], avec Anne Canteaut. Nous avons également montré plusieurs applications de ces distingueurs [BC10, BC11b, BCD11].

Définition 3.1. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Une* somme nulle *de taille k pour F est un sous-ensemble $\{x_1, \ldots, x_k\} \subset \mathbf{F}_2^n$ d'éléments dont la somme est nulle et dont la somme des images correspondantes par F est également nulle :*

$$\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} F(x_i) = 0.$$

3.2.1 Taille minimale d'une somme nulle

La première question que nous pouvons nous poser est, quelle est la plus petite taille des sommes nulles existant pour une fonction vectorielle F ? Pour répondre à ceci, nous avons eu recours à la théorie des codes correcteurs d'erreurs. Plus précisément, nous avons montré qu'il est possible d'associer les sommes nulles aux codes correcteurs d'erreurs ainsi qu'aux fonctions APN. Grâce à ces relations, nous avons pu montrer que chaque fonction vectorielle possède au moins une somme nulle de taille au moins 5. De plus, si la fonction F est APN, alors des sommes nulles de taille strictement inférieure à 5 ne peuvent pas exister pour F. Tout cela est résumé dans la proposition 3.2.

Pour présenter ces résultats, nous allons utiliser la notation courante de la théorie algébrique des codes (*cf.* [MS78]). Un code linéaire binaire $\mathcal{C}$ de longueur n et de dimension k, noté par $[n, k]$, est un sous-espace de $\mathbf{F}_2^n$ de dimension k. Un tel code peut être représenté par une matrice binaire G, de taille $k \times n$, appelée *matrice génératrice* de $\mathcal{C}$:

$$\mathcal{C} = \{xG,\ x \in \mathbf{F}_2^k\}.$$

Tout code $[n, k]$-linéaire, $\mathcal{C}$, peut être associé à son code dual, noté par $\mathcal{C}^\perp$ et ayant comme paramètres $[n, n-k]$. Le code dual $\mathcal{C}^\perp$ est défini de la façon suivante :

$$\mathcal{C}^\perp = \{x \in \mathbf{F}_2^n :\ x \cdot c = 0,\ \text{pour tout } c \in \mathcal{C}\} = \{y \in \mathbf{F}_2^n : Gy = 0\}.$$

Il est possible d'associer à chaque fonction vectorielle F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ un code linéaire $\mathcal{C}_F$. Pour cela, nous notons par $(x_i,\ 0 \leq i < 2^n)$ l'ensemble de tous les éléments de $\mathbf{F}_2^n$. Nous associons F au code linéaire $\mathcal{C}_F$ de longueur 2^n et de dimension $n + m$, défini par la matrice génératrice suivante :

$$G_F = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 & \cdots & x_{2^n-1} \\ F(x_0) & F(x_1) & F(x_2) & F(x_3) & \cdots & F(x_{2^n-1}) \end{pmatrix},$$

où chaque entrée de la matrice est vue comme un vecteur colonne binaire.

Proposition 3.1. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. L'ensemble $\{x_{i_1}, \ldots, x_{i_k}\} \subset \mathbf{F}_2^n$ est une somme nulle pour F si et seulement si le vecteur de poids de Hamming k de support $\{i_1, \ldots, i_k\}$ appartient au code dual $\mathcal{C}_F^\perp$. En particulier, si $m = n$, nous déduisons que :*

- *il existe au moins une somme nulle de taille 5 pour F*

– *aucune somme nulle de taille inférieure ou égale à 4 n'existe pour F si et seulement si F est une fonction APN.*

Démonstration. Par définition du code dual, un vecteur binaire $(c_0, \ldots, c_{2^n-1})$ appartient à $\mathcal{C}_F^\perp$ si et seulement si

$$\sum_{i=0}^{2^n-1} c_i x_i = 0 \text{ et } \sum_{i=0}^{2^n-1} c_i F(x_i) = 0.$$

Ceci est équivalent à dire que le support de c, c'est-à-dire l'ensemble $\{i : c_i = 1\}$, définit une somme nulle pour F. La taille de cette somme nulle correspond au poids de Hamming du mot du code. Si $m = n$, le code $\mathcal{C}_F^\perp$ est un code linéaire de longueur 2^n et de dimension $2^n - 2n$. Il est connu que la distance minimale d'un code linéaire ayant ces paramètres ne peut pas dépasser 5 [DZ84, BT93]. La correspondance entre la propriété APN et le fait que $\mathcal{C}_F^\perp$ ait une distance minimale 5, a été établie dans [CCZ98]. Par ailleurs, comme la taille minimale possible pour une somme nulle est 3, F a des sommes nulles de taille 3 ou 4 si et seulement si F n'est pas une fonction APN. □

3.2.2 Algorithmes génériques

Il est facile de voir que si F est une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ choisie aléatoirement, alors un ensemble d'éléments de $\mathbf{F}_2^n$ de taille k est une somme nulle pour F avec probabilité $2^{-(n+m)}$. Il existe des algorithmes génériques qui trouvent des sommes nulles pour une fonction F.

La plus connue des méthodes est l'algorithme de David Wagner pour le problème des anniversaires généralisé [Wag02]. Dans l'article d'origine, ce problème est aussi mentionné comme le *problème de k-sommes* et est formulé de la manière suivante.

Définition 3.2. *Étant données k listes $L_1, \ldots, L_k$ d'éléments uniformément et indépendamment distribués dans $\{0,1\}^n$, trouver $x_1 \in L_1, \ldots, x_k \in L_k$ tels que*

$$x_1 \oplus \cdots \oplus x_k = 0.$$

L'algorithme que Wagner propose est assez efficace pour des tailles qui sont des puissances de 2. Si $k = 2^\kappa$, alors l'algorithme trouve une somme nulle avec complexité

$$\mathcal{O}\left(2^{\frac{n+m}{\kappa+1}+\kappa}\right).$$

Cette complexité correspond aux $2^{\frac{n+m}{\kappa+1}+\kappa}$ évaluations de la fonction F afin de construire les 2^κ listes initiales de taille $2^{\frac{n+m}{\kappa+1}}$.

Si la taille de la somme nulle, k, est supérieure à $n+m$, l'algorithme de Wagner peut être amélioré par l'attaque XHASH, due à Mihir Bellare et Daniele Micciancio [BM97], comme cela a été remarqué dans [AKK+10, BDPV10]. La complexité de cet algorithme est de l'ordre k évaluations de F. L'algorithme de Wagner a donc le même comportement seulement pour $k \geq 2^{\sqrt{n+m}}$.

Un algorithme générique, inspiré par l'attaque XHASH, a été décrit par Bertoni *et al.* dans [BDPV10]. Il s'agit de l'algorithme 2. Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Notons par

$$X_i = \begin{pmatrix} x_i \\ F(x_i) \end{pmatrix},$$

le vecteur colonne comportant $n+m$ lignes, composé des coordonnées de x_i suivi par les coordonnées de $F(x_i)$. L'algorithme se déroule de la manière suivante.

Algorithme 2: Méthode générique pour la recherche des sommes nulles

Résultat : Un ensemble $\mathcal{Z} = \{z_1, \ldots, z_k\} \subset \mathbf{F}_2^n$ tel que $\sum_{i=1}^{k} z_i = 0$ et $\sum_{i=1}^{k} F(z_i) = 0$
Choisir aléatoirement k valeurs $x_1, \ldots, x_k \in \mathbf{F}_2^n$;
pour *i de 1 à k* **faire**
  Calculer $F(x_i)$ et former X_i;
Calculer

$$S = \sum_{i=1}^{k} X_i \tag{3.1}$$

Fixer $t = n + m + \varepsilon$, tel que $t \leq k$ où ε est un petit entier;
Choisir aléatoirement t valeurs $y_i \in \mathbf{F}_2^n$;
pour *i de 1 à t* **faire**
  Calculer $F(y_i)$ et former $Y_i = \begin{pmatrix} y_i \\ F(y_i) \end{pmatrix}$;
Résoudre le système linéaire de $n+m$ équations en les $n+m+\varepsilon$ variables binaires a_i ($(X_i \oplus Y_i)$ sont vus comme des coefficients constants) :

$$\sum_{i=1}^{t} (X_i \oplus Y_i) a_i = S \tag{3.2}$$

Pour chaque solution (a_i), former l'ensemble $\mathcal{Z} = \{z_i : 1 \leq i \leq k\}$ tel que :

$$z_i = \begin{cases} y_i & \text{si } i \leq t \text{ et } a_i = 1 \\ x_i & \text{sinon.} \end{cases}$$

Renvoyer $\mathcal{Z}$;

L'ensemble $\mathcal{Z}$ forme une somme nulle. En effet, en ajoutant les équations (3.1) et (3.2) on obtient :

$$\sum_{i=1}^{k} X_i \oplus \sum_{i=1}^{t} (X_i \oplus Y_i) a_i = \sum_{i=1}^{t} (Y_i a_i \oplus X_i \bar{a_i}) \oplus \sum_{i=t+1}^{k} X_i = \sum_{i=1}^{k} Z_i = 0.$$

La valeur de ε est liée à la probabilité que le système (3.2) ait une solution. Plus la valeur de ε est grande, plus la probabilité augmente.

La complexité de l'algorithme correspond principalement aux $k+n+m+\varepsilon$ évaluations de F, puis à la résolution du système linéaire qui peut être faite en $\mathcal{O}((n+m)^3)$.

Algorithme de décodage par ensembles d'informations Des algorithmes génériques basés sur les codes correcteurs d'erreurs peuvent également être utilisés pour résoudre ce problème. En particulier, l'algorithme de décodage par ensembles d'informations et ses variantes [CC98] ainsi que les techniques proposées récemment [MMT11, BJMM12] peuvent améliorer la complexité des algorithmes précédents pour des sommes nulles de petite taille [FS09]. En revanche, toutes ces méthodes prennent en entrée une matrice génératrice et nécessitent donc une évaluation complète de la fonction F.

Relation avec les codes de Reed-Muller

Les structures à somme nulle sont également liées aux codes correcteurs de Reed-Muller. Le code de Reed Muller $R(n,r)$, $0 \leq r \leq n$, est un code linéaire de longueur 2^n et d'ordre r. Un tel code peut être décrit à l'aide des fonctions booléennes. Plus précisément, le code $R(n,r)$ correspond à l'ensemble des fonctions booléennes en n variables dont le degré algébrique est au plus r. Ainsi, le code $\mathcal{C}_F$ associé à une fonction F à n variables d'entrée est inclus dans le code de Reed-Muller $R(n,r)$. Or, le code dual de $R(n,r)$ est le code $R(n,n-r-1)$. On en déduit donc que $R(n,n-\deg F-1)$ est inclus dans $\mathcal{C}_F^{\perp}$, ce qui implique que les mots de $R(n,n-\deg F-1)$ définissent des sommes nulles pour F.

Le théorème suivant donne les mots de poids minimal des codes de Reed-Muller.

Théorème 3.1. *([MS78], Chapitre 13) Les mots de $R(n,r)$ de poids minimal sont exactement les sous-espaces affines de codimension r.*

Ainsi, les sous-espaces affines de dimension $(\deg F+1)$, qui sont les mots de poids minimal de $R(n,n-\deg F-1)$ sont des sommes nulles de F. Autrement dit, si V un sous-espace affine de dimension $\deg(F)+1$, nous avons que :

$$\sum_{v \in V} F(v+x) = 0, \text{ pour chaque } x \in \mathbf{F}_2^n.$$

Ceci correspond également au résultat de la proposition 2.6.

3.3 Les partitions en sommes nulles

Dans le cas où la fonction F est une permutation sur $\mathbf{F}_2^n$, les mots de poids minimal de $R(n,n-\deg(F)-1)$ correspondent aux sommes nulles avec une propriété en plus : chaque translaté d'une telle somme nulle est lui-même une somme nulle. Cela conduit à une propriété beaucoup plus forte, que nous appellerons *partition en sommes nulles*.

Définition 3.3. *Soit P une permutation de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$. Une* partition en sommes nulles *pour P de taille 2^k est une collection de 2^{n-k} sommes nulles disjointes $X_i = \{x_{i,1}, \ldots, x_{i,2^k}\} \subset \mathbf{F}_2^n$:*

$$\bigcup_{i=1}^{2^{n-k}} X_i = \mathbf{F}_2^n \quad \text{et} \quad \sum_{j=1}^{2^k} x_{i,j} = \sum_{j=1}^{2^k} P(x_{i,j}) = 0, \quad \forall 1 \leq i \leq 2^{n-k}.$$

3.3.1 Algorithme générique pour trouver des partitions en sommes nulles

Nous allons présenter ici un algorithme générique pour trouver une partition en sommes nulles de taille 2^k, avec $2^k \geq 2n$, pour une permutation P. Cet algorithme consiste simplement

à appliquer l'attaque XHASH de façon récursive, comme décrit par l'algorithme 3.

Algorithme 3: Méthode générique pour la recherche des partitions en sommes nulles

Résultat : Une partition de $\mathbf{F}_2^n$ en 2^{n-k} sommes nulles de taille 2^k
$Y \leftarrow \mathbf{F}_2^n$
appeler `partition`$(Y, 1)$;
Fonction `partition` (X, i);
si $i < n - k$ **alors**
  Trouver dans X une somme nulle S de taille 2^{n-i} avec l'algorithme XHASH;
  Utiliser cette somme nulle pour partitionner $\mathbf{F}_2^n$, en translatant S, en 2^i sommes nulles $X_1, \ldots, X_{2^i}$ de taille 2^{n-i} ;
  pour *j de 1 à* 2^i **faire**
    appeler `partition`$(X_j, i-1)$;

Avec cet algorithme, puisque P est une permutation, nous avons besoin d'évaluer P en tous les points, sauf aux 2^k derniers environ. À part ces évaluations, la complexité de l'algorithme peut être approchée par $((2n)^3(2^{n-k}-1))$. En effet, la complexité de trouver une somme nulle de taille K avec l'algorithme XHASH est de l'ordre de $(2n)^3K$, où $(2n)^3$ est la complexité de résoudre le système linéaire (en utilisant par exemple le pivot de Gauss). En conséquence, si on suppose que nous pouvons stocker les évaluations de P déjà effectuées et de les réutiliser, la complexité à part les évaluations ne dépend que du nombre de résolutions des systèmes linéaires. Ce nombre est au début 1 (pour trouver une partition de l'espace total en deux sommes nulles) et double à chaque étape. Par conséquence la complexité de cette étape peut être approchée par :

$$(2n)^3(1 + 2^2 + \ldots 2^{n-k-1}) = (2n)^3 \sum_{i=0}^{n-k-1} 2^i = (2n)^3(2^{n-k} - 1).$$

La complexité totale est alors de l'ordre de

$$2^n - 2^k + (2n)^3(2^{n-k} - 1).$$

Nous voyons que pour trouver des partitions en sommes nulles pour une permutation aléatoire avec une méthode générique, nous avons besoin d'évaluer la permutation en presque tous les points, puisque la technique de recherche n'est pas déterministe. Cela fait une énorme différence par rapport aux partitions en sommes nulles qui proviennent d'une propriété structurelle de la permutation, comme par exemple les sommes nulles pour l'AES que nous avons vues à la section 3.1. Par ailleurs, les partitions provenant des propriétés structurelles peuvent être utilisées pour prouver qu'une permutation ne satisfait pas la propriété désirée. Pour cela, seulement quelques évaluations de la permutation sur un petit nombre d'ensembles X_i peuvent être demandées.

Par la suite, nous allons voir comment construire des partitions en sommes nulles en exploitant des propriétés structurelles d'une permutation P. Pour cela, nous allons montrer comment exploiter à la fois des faiblesses venant de la partie non-linéaire de la primitive et des faiblesses de la partie linéaire.

3.4 Exploiter le degré algébrique de la partie non-linéaire

Nous avons vu que plusieurs primitives symétriques présentent une structure itérative. Dans cette partie nous allons chercher des partitions en sommes nulles pour des permutations

itérées de la forme

$$P = R_r \circ \cdots \circ R_1,$$

où $R_1 \ldots, R_r$ sont des permutations simples sur $\mathbf{F}_2^n$, appelées *permutations de tour*. De façon générale, les R_i sont dérivées d'une permutation paramétrée unique, en choisissant r paramètres différents.

Nous allons d'abord montrer comment construire des partitions en sommes nulles en exploitant le degré algébrique de la permutation de tour et celui de son inverse. C'était ce type de propriétés que Aumasson et Meier ont utilisé pour construire des structures à somme nulle pour les fonctions Keccak, *Luffa* et Hamsi [AM09].

3.4.1 Des partitions en sommes nulles à partir des différentielles d'ordre supérieur

Comme précédemment mentionné, si F est une permutation, tout sous-espace affine V de $\mathbf{F}_2^n$ de dimension $(\deg F + 1)$ conduit à une partition en sommes nulles. Ce résultat provient des propriétés des différentielles d'ordre supérieur car

$$D_V F(x) = \sum_{v \in V} F(x+v) = 0, \qquad \text{pour tout } x \in \mathbf{F}_2^n.$$

Dans le cas des sommes nulles pour l'AES, Knudsen et Rijmen [KR07] ont exploité la connaissance de la clé, due au modèle à la clé connue, pour partir d'un état intermédiaire. Grâce à l'absence de clé pour les fonctions de hachage, cette méthode a également été utilisée par Aumasson et Meier [AM09] pour trouver des partitions en sommes nulles sur trois candidats au concours SHA-3. La seule information dont nous avons besoin pour utiliser cette première approche est une borne supérieure sur le degré de la transformation de tour et de son inverse.

Soit $P = R_r \circ \cdots \circ R_1$ et t un entier $1 \leq t \leq r$. Nous définissons les fonctions F_{r-t} et G_t, impliquées dans la décomposition de P :

- F_{r-t} : fonction qui consiste en les $(r-t)$ dernières transformations de tour, c'est-à-dire $F_{r-t} = R_r \circ \cdots \circ R_{t+1}$
- G_t : fonction inverse des t premières transformations de tour, c'est-à-dire $G_t = R_1^{-1} \circ \cdots \circ R_t^{-1}$.

La technique introduite dans [AM09] est alors décrite dans la proposition 3.2 et peut être visualisée avec la figure 3.2.

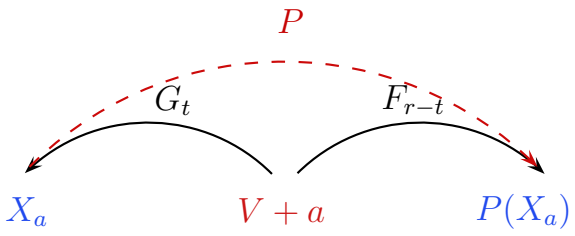

Figure 3.2 – Méthode pour construire une partition en sommes nulles pour une permutation P, composée de r tours

Proposition 3.2. *Soient d_1 et d_2 deux entiers tels que $\deg(F_{r-t}) \leq d_1$ et $\deg(G_t) \leq d_2$. Soit V un sous-espace de $\mathbf{F}_2^n$ de dimension $d+1$, où $d = \max(d_1, d_2)$ et soit W un supplémentaire de V, c'est-à-dire $V \oplus W = \mathbf{F}_2^n$. Alors, les ensembles*

$$X_a = \{G_t(a+z), z \in V\}, \quad a \in W$$

forment une partition en sommes nulles de $\mathbf{F}_2^n$ de taille 2^{d+1}, pour la permutation P.

Démonstration. Soit a un élément quelconque dans W. Nous montrons d'abord que la somme de tous les états $x \in X_a$ est nulle :

$$\sum_{x \in X_a} x = \sum_{z \in V} G_t(a+z) = D_V G_t(a).$$

Cette somme correspond à la valeur d'une dérivée d'ordre $(d+1)$ de la fonction G_t dont le degré est $d_2 \leq d$. En conséquence, cette somme est nulle. Les images de ces états par P correspondent aux images des états intermédiaires z par F_{r-t}. De la même façon,

$$\sum_{x \in X_a} P(x) = \sum_{z \in V} F_{r-t}(a+z) = D_V F_{r-t}(a),$$

qui est la valeur d'une dérivée d'ordre $(d+1)$ d'une fonction de degré plus petit que d. Ainsi, cette somme vaut également zéro, impliquant que chaque X_a est une somme nulle. Comme tous les ensembles X_a sont des images par la permutation G_t des ensembles disjoints, alors les X_a sont également disjoints et forment donc une partition de $\mathbf{F}_2^n$. □

Pour pouvoir créer des partitions en sommes nulles de cette façon, nous devons être capable d'estimer le degré de la permutation après un certain nombre de tours et de pouvoir faire la même chose avec la transformation inverse. Si le nombre de tours r est fixé, nous souhaitons construire pour les r tours de la permutation une partition dont la taille est la plus petite possible. Pour cela, nous devons bien choisir l'instant t à partir duquel nous allons calculer en avant et en arrière. En choisissant t comme étant la valeur de l'indice i, $1 \leq i \leq t$ pour laquelle la quantité

$$\max(\deg(G_i), \deg(F_{r-i})), \tag{3.3}$$

soit la plus petite possible. De cette façon nous somme sûres de créer une partition de $\mathbf{F}_2^n$ dont les sommes nulles pour P sont de la plus petite taille possible. Il suffit juste de choisir un sous-espace V de dimension $(d+1)$ engendré par $(d+1)$ éléments de la base canonique, où $d = \max(\deg(G_t), \deg(F_{r-t}))$. C'est la méthode qui a été utilisée dans [AM09].

3.4.2 Trouver une borne supérieure pour le degré

La description de la méthode précédente laisse apparaître clairement l'importance d'une estimation correcte du degré d'une permutation itérée. Les bonnes estimations conduisent à l'amélioration de la complexité des distingueurs basés sur les structures à somme nulle. Dans [AM09], les auteurs ont utilisé une borne supérieure pour le degré, que nous appellerons désormais *borne triviale*. Cette borne est décrite par la Proposition 3.3.

Proposition 3.3. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ et G une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Le degré de leur composition, $G \circ F$, est majoré par :*

$$\deg(G \circ F) \leq \deg(G) \deg(F).$$

Quand le nombre de tours est petit, cette borne présente une estimation assez correcte du degré. Au contraire, pour un nombre de tours élevé, les résultats de la borne triviale s'éloignent considérablement de la situation réelle.

Dans certains cas, de meilleurs bornes existent. La recherche de bornes supérieures pour les fonctions itérées est l'objet d'une grande partie de cette thèse. Ces résultats seront analysés plus tard. Dans ce chapitre, nous nous contentons de présenter un résultat ancien de Anne Canteaut et Marion Videau [CV02], que nous avons utilisé dans [BC11b] pour estimer le degré de la fonction Keccak. Ce résultat est basé sur l'analyse de certaines des propriétés spectrales des permutations, afin d'établir une meilleure borne. Plus précisément, il a été montré dans [CV02] que la borne triviale peut être améliorée dans le cas où les valeurs du spectre de Walsh de F sont divisibles par une grande puissance de 2.

Théorème 3.2 ([CV02])**.** *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ telle que toutes les valeurs dans son spectre de Walsh sont divisibles par 2^ℓ, pour un entier ℓ. Alors, pour une fonction G de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ nous avons*

$$\deg(G \circ F) \leq n - \ell + \deg(G).$$

Par la suite, nous allons nous concentrer sur une construction très commune dans le cas des primitives symétriques. Nous avons vu que la partie non-linéaire d'un très grand nombre de chiffrements par blocs et de fonctions de hachage, que nous noterons par χ, est constituée de n/n_0 applications parallèles d'une petite permutation (boîte-S) χ_0 de $\mathbf{F}_2^{n_0}$. Nous appellerons *mot*, une sous-partie de l'état sur laquelle s'applique χ_0. Par exemple, dans le cas de l'AES, un mot est un octet et l'état entier de l'AES est divisé en 16 mots. De façon générale, chaque vecteur de n bits peut être vu comme une collection de $n_r = n/n_0$ mots, où chaque mot est un élément de $\mathbf{F}_2^{n_0}$.

Examinons maintenant le spectre de Walsh de la permutation de tour R. Puisque le spectre de Walsh est invariant par composition avec une transformation linéaire, pour chaque $\alpha \in \mathbf{F}_2^n$ il existe un β tel que

$$\begin{aligned} \mathcal{F}(R + \varphi_\alpha) &= \mathcal{F}(\chi + \varphi_\beta) = \sum_{x=(x_1,\ldots,x_{n_r})} (-1)^{\chi(x)+\beta\cdot x} \\ &= \prod_{i=1}^{n_r} \sum_{x_i} (-1)^{\chi_0(x_i)+\beta_i \cdot x_i} = \prod_{i=1}^{n_r} \mathcal{F}(\chi_0 + \varphi_{\beta_i}). \end{aligned} \tag{3.4}$$

Par conséquent, si tous les éléments dans le spectre de Walsh de χ_0 sont divisibles par 2^{ℓ_0}, nous pouvons déduire que le spectre de Walsh de la transformation de tour est divisible par $2^{n_r \ell_0}$.

Au cours de cette thèse, nous avons trouvé de nouveaux résultats génériques et de nouvelles bornes sur le degré algébrique. Ces résultats seront toutefois analysés dans les chapitres qui suivent, tandis qu'ici nous allons nous concentrer sur les méthodes liées aux structures à somme nulle.

3.5 Exploiter la structure de la partie linéaire

Hormis le degré de la transformation de tour, un deuxième élément peut être exploité afin de construire des partitions en sommes nulles. En particulier, le fait que χ soit constituée de plusieurs applications parallèles d'une fonction plus petite peut être utilisé pour étendre les partitions en sommes nulles décrites auparavant, à un tour supplémentaire. De plus, on peut exploiter le fait qu'un petit nombre d'itérations de la permutation de tour R n'est pas suffisant pour fournir une diffusion complète.

Nous notons par B_i, $0 \leq i < n_r$ les sous-espaces de dimension n_0 correspondant aux mots de l'état :

$$B_i = \langle e_{n_0 i}, \ldots, e_{n_0 i + n_0 - 1} \rangle,$$

où $e_0, \ldots, e_{n-1}$ désignent les éléments de la base canonique de $\mathbf{F}_2^n$. Nous supposons que les bits de l'état sont énumérés de façon que les n_0 bits d'un mot correspondent à n_0 bits consécutifs.

3.5.1 Propriété pour un tour

Nous allons maintenant montrer comment étendre une partition en sommes nulles de r tours, à $r+1$ tours. Pour cela, nous devons choisir le sous-espace V d'une façon particulière. Les partitions en sommes nulles décrites à la proposition 3.2 sont obtenues à partir d'un ensemble d'états intermédiaires après t tours. Cet ensemble est un translaté d'un sous-espace V de dimension $(d+1)$, obtenu pour un choix de V quelconque. Toutefois, nous allons nous concentrer sur des sous-espaces V ayant une description particulière.

Plus précisément, nous allons considérer des sous-espaces V qui correspondent à une collection de $\lceil (d+1)/n_0 \rceil$ mots :

$$V = \bigoplus_{i \in \mathcal{I}} B_i,$$

où $\mathcal{I}$ est un sous-ensemble $\mathcal{I} \subset \{0, \ldots, n_r - 1\}$ de taille $\lceil (d+1)/n_0 \rceil$. Comme χ s'applique de façon indépendante sur chaque mot, les variables des mots différents ne seront pas mélangées entre elles après l'application de χ. Ceci implique qu'il existe un b tel que $\chi(a+V) = b+V$.

Dans toute cette partie, nous supposons que les permutations que nous examinons suivent le principe de construction annoncé par Shannon, c'est-à-dire que la permutation de tour est composée d'une partie non-linéaire, χ, et de deux parties affines A_1, A_2, de façon à ce que

$$R_t = A_2 \circ \chi \circ A_1.$$

Selon la construction, la partie A_1 ou la partie A_2 peuvent simplement être la fonction identité. La proposition 3.4 décrit une méthode générale pour trouver des partitions en sommes nulles de taille 2^{d+1} pour une permutation P composée de r tours.

Proposition 3.4. *Soient d_1 et d_2 deux entiers tels que $\deg(F_{r-t-1}) \leq d_1$ et $\deg(G_t) \leq d_2$. Nous décomposons la transformation de tour du tour $(t+1)$ comme ceci : $R_{t+1} = A_2 \circ \chi \circ A_1$, où A_1, A_2 sont des transformations affines. Soit $\mathcal{I}$ un sous-ensemble de $\{0, \ldots, n_r - 1\}$ de taille $\lceil (d+1)/n_0 \rceil$,*

$$V = \bigoplus_{i \in \mathcal{I}} B_i,$$

et W un supplémentaire de V. Alors, les ensembles

$$X_a = \{(G_t \circ A_1^{-1})(a+z), z \in V\}, a \in W,$$

forment une partition en sommes nulles de $\mathbf{F}_2^n$ *de taille* 2^k*, où* $k = n_0\lceil\frac{d+1}{n_0}\rceil$*, pour la permutation* P *composée de* r *tours.*

Démonstration. Pour un $a \in W$, la somme de tous les états dans X_a est donnée par

$$\sum_{x\in X_a} x = \sum_{z\in V} G_t \circ A_1^{-1}(a+z) = D_V(G_t \circ A_1^{-1})(a).$$

Comme $\deg(G_t \circ A_1^{-1}) = \deg(G_t) \leq d$, cette somme vaut zéro. En utilisant l'égalité $\chi(a+V) = b+V$, nous obtenons que la somme des sorties correspondantes satisfait

$$\begin{aligned}\sum_{x\in X_a} P(x) &= \sum_{z\in V} F_{r-t-1} \circ A_2 \circ \chi(a+z) = \sum_{z\in V} F_{r-t-1} \circ A_2(b+z)\\ &= D_V(F_{r-t-1} \circ A_2)(b) = 0,\end{aligned}$$

puisque $\deg(F_{r-t-1} \circ A_2) = \deg(F_{r-t-1}) \leq d$. □

Nous pouvons mieux nous rendre compte de cette méthode, grâce à la figure 3.3.

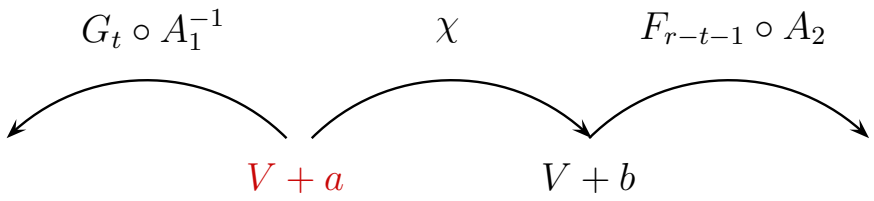

FIGURE 3.3 – Méthode pour étendre une partition en sommes nulles à un tour supplémentaire

3.5.2 Propriété pour plusieurs tours

Nous allons maintenant présenter une méthode pour étendre les partitions en sommes nulles déjà connues à plusieurs tours. Pour cela nous allons exploiter des propriétés qui proviennent à la fois de la structure particulière de la transformation de tour et de la partie linéaire.

Pour simplifier la description, nous n'allons présenter en détail que la méthode pour deux tours supplémentaires. La généralisation sur plusieurs tours est immédiate (Théorème 3.4). Supposons que nous connaissions une partition pour t tours d'une permutation. Nous allons alors étendre cette partition aux tours $(t+1)$ et $(t+2)$ que nous décomposerons comme suit :

$$R_{t+2} \circ R_{t+1} = A_2 \circ \chi \circ A \circ \chi \circ A_1,$$

où A_1, A_2 et A sont de degré 1.

Théorème 3.3. *Soient* d_1 *et* d_2 *deux entiers tels que* $\deg(F_{r-t-2}) \leq d_1$ *et* $\deg(G_t) \leq d_2$*. Notons par* L *la partie linéaire de la permutation affine* A*. Soit* V *un sous-espace de* $\mathbf{F}_2^n$ *de dimension* k*, qui satisfait les deux conditions suivantes :*

(i) *il existe un sous-ensemble* $\mathcal{I} \subset \{0, \ldots, n_r - 1\}$ *tel que*

$$B_b := \bigoplus_{i\in\mathcal{I}} B_i \subset V \text{ et } |\mathcal{I}| \geq \left\lceil \frac{d_2+1}{n_0} \right\rceil.$$

(ii) *il existe un sous-ensemble* $\mathcal{J} \subset \{0, \ldots, n_r - 1\}$ *tel que*

$$B_f := \bigoplus_{j \in \mathcal{J}} B_j \subset L(V) \text{ et } |\mathcal{J}| \geq \left\lceil \frac{d_1 + 1}{n_0} \right\rceil.$$

Soit W *un espace supplémentaire de* V*. Alors, les ensembles*

$$X_a = \{G_t \circ A_1^{-1} \circ \chi^{-1}(a + z), z \in V\}, a \in W,$$

forment une partition en sommes nulles de $\mathbf{F}_2^n$ *de taille* 2^k *pour la permutation* P *constituée de* r *tours.*

Démonstration. D'après la définition des ensembles X_a, nous choisissons les états intermédiaires z dans un translaté de V après la partie non-linéaire du tour R_{r+1}. De la deuxième relation nous déduisons que $A(V)$ peut être vu comme une union de translatés de B_f :

$$A(V) = \bigcup_{b \in \mathcal{E}} (b + B_f),$$

où $\mathcal{E}$ est un sous-ensemble de $\mathbf{F}_2^n$. De plus, la même propriété est valable pour l'image par A de n'importe quel translaté de V. Comme χ s'applique aux mots séparément, les variables des différents mots ne sont pas mélangées entre elles. En conséquence,

$$\chi(A(V)) = \bigcup_{b \in \mathcal{E}'} (b + B_f),$$

où $\mathcal{E}'$ est un autre sous-ensemble de $\mathbf{F}_2^n$. Par définition, les images par P des éléments dans X_a correspondent aux images de $a + z$, $z \in V$, par $F_{r-t-2} \circ A_2 \circ \chi \circ A$. Il s'ensuit que leur somme est donnée par

$$\begin{aligned} \sum_{z \in V} F_{r-t-2} \circ A_2 \circ \chi \circ A(a + z) &= \sum_{b \in \mathcal{E}'} \sum_{x \in B_f} (F_{r-t-2} \circ A_2)(b + x) \\ &= \sum_{b \in \mathcal{E}'} D_{B_f}(F_{r-t-2} \circ A_2)(b). \end{aligned}$$

Puisque

$$\dim B_f \geq n_0 \left\lceil \frac{d_1 + 1}{n_0} \right\rceil > d_1,$$

cette dérivée s'annule. Nous calculons ensuite en arrière les images de $a + V$ par $G_t \circ A_1^{-1} \circ \chi^{-1}$. Étant donné que $B_b \subset V$, V peut être exprimé comme une union de translatés de B_b. Comme χ^{-1} ne mélange pas les mots entre eux, nous déduisons que

$$\chi^{-1}(a + V) = \bigcup_{c \in \mathcal{E}''} (c + B_b),$$

pour un ensemble $\mathcal{E}'' \subset \mathbf{F}_2^n$. Alors, la somme des états d'entrée correspondant à $x \in X_a$ est donnée par

$$\begin{aligned} \sum_{x \in X_a} x = \sum_{z \in V} (G_t \circ A_1^{-1} \circ \chi^{-1})(a + z) &= \sum_{c \in \mathcal{E}''} \sum_{x \in B_b} (G_t \circ A_1^{-1})(c + x) \\ &= \sum_{c \in \mathcal{E}''} D_{B_b}(G_t \circ A_1^{-1})(b). \end{aligned}$$

Cette dérivée s'annule également, puisque

$$\dim B_b \geq n_0 \left\lceil \frac{d_2+1}{n_0} \right\rceil > d_2.$$

□

La figure 3.4 nous permet de mieux comprendre cette méthode.

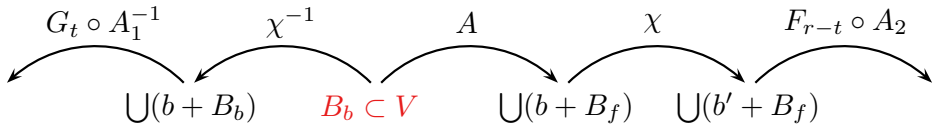

FIGURE 3.4 – Méthode pour étendre une partition en sommes nulles à deux tours supplémentaires

Cependant, le théorème ci-dessus est souvent difficile à utiliser dans la pratique puisque les sous-espaces V et $L(V)$ peuvent être de très grande taille. Pour cette raison, nous utiliserons le corollaire suivant pour les applications qui suivront.

Corollaire 3.1. *Soient d_1 et d_2 deux entiers tels que $\deg(F_{r-t-2}) \leq d_1$ et $\deg(G_t) \leq d_2$. Notons par L la partie linéaire de la permutation affine A. Soit W un sous-espace de $\mathbf{F}_2^n$ de dimension k, qui satisfait les deux conditions suivantes :*

(i) *il existe un sous-ensemble $\mathcal{I} \subset \{0, \ldots, n_r - 1\}$ tel que*

$$W \subset \bigoplus_{i \in \mathcal{I}} B_i \text{ et } |\mathcal{I}| \leq n_r - \left\lceil \frac{d_2+1}{n_0} \right\rceil.$$

(ii) *il existe un sous-ensemble $\mathcal{J} \subset \{0, \ldots, n_r - 1\}$ tel que*

$$L^T(W) \subset \bigoplus_{j \in \mathcal{J}} B_j \text{ et } |\mathcal{J}| \leq n_r - \left\lceil \frac{d_1+1}{n_0} \right\rceil,$$

où L^T est la fonction linéaire dont la matrice est la transposée de la matrice qui définit L. Soit V un sous-espace tel que $V = W^\perp$. Alors, les ensembles

$$X_a = \{G_t \circ A_1^{-1} \circ \chi^{-1}(a + z), z \in V\}, a \in W,$$

forment une partition en sommes nulles de $\mathbf{F}_2^n$ de taille 2^{n-k} pour la permutation P constituée de r tours.

Démonstration. Les propriétés (i) et (ii) exigées pour W impliquent qu'il existe deux sous-espaces B_b et B_f tels que :

$$B_b = \bigoplus_{i \in \overline{\mathcal{I}}} B_i \subset V \text{ et } B_f = \bigoplus_{j \in \overline{\mathcal{J}}} B_j \subset L(V)$$

avec $\overline{\mathcal{I}} = \{0, \ldots, n_r - 1\} \backslash \mathcal{I}$ et $\overline{\mathcal{J}} = \{0, \ldots, n_r - 1\} \backslash \mathcal{J}$. Le résultat découle alors immédiatement du théorème 3.3. □

Remarque 3.1. *Il existe une condition nécessaire simple pour l'existence des sous-espaces W de la forme décrite par le théorème 3.3. Nous définissons ici le poids d'un $x \in \mathbf{F}_2^n$ par rapport à sa décomposition en mots de l'état. Plus précisément, la quantité $H_w(x)$ correspondra au nombre de mots sur lesquels x ne s'annule pas. Dans ce cas, le sous-espace W défini dans le corollaire précèdent satisfait :*

$$\forall x \in W, \quad H_w(x) \leq n_r - \left\lceil \frac{d_2+1}{n_0} \right\rceil \text{ et } H_w(L^T(x)) \leq n_r - \left\lceil \frac{d_1+1}{n_0} \right\rceil. \tag{3.5}$$

Si $\dim W = 1$, cette condition est également suffisante. À partir de la propriété (3.5), nous pouvons déduire une condition nécessaire, qu'une transformation de tour doit satisfaire, afin d'être immunisée contre ce type d'attaque. Cette condition consiste à choisir la partie linéaire L de la transformation de tour, de façon que

$$\min_{x \neq 0}(H_w(x) + H_w(L^T(x))) > 2n_r - \left\lceil \frac{d_1+1}{n_0} \right\rceil - \left\lceil \frac{d_2+1}{n_0} \right\rceil.$$

Nous allons maintenant généraliser le théorème 3.3 à plusieurs tours. Plus précisément, nous allons fournir des conditions qui doivent être satisfaites afin de pouvoir étendre à plus de 3 tours une partition en sommes nulles déjà connue.

Théorème 3.4. *Soient d_1 et d_2 deux entiers tels que $\deg(F_{r-t-2}) \leq d_1$ et $\deg(G_t) \leq d_2$. Notons par L la partie linéaire de la permutation affine A. Soit V un sous-espace de $\mathbf{F}_2^n$ de dimension k, qui satisfait les conditions suivantes, pour deux entiers s_b et s_f non-nuls :*

(i) *il existe un sous-ensemble $\mathcal{I}_1 \subset \{0, \ldots, n_r - 1\}$ tel que*

$$B_{b_1} := \bigoplus_{i \in \mathcal{I}_1} B_i \subset V \text{ et } |\mathcal{I}_1| \geq \left\lceil \frac{d_2+1}{n_0} \right\rceil.$$

(ii) *il existe un sous-ensemble $\mathcal{J}_1 \subset \{0, \ldots, n_r - 1\}$ tel que*

$$B_{f_1} := \bigoplus_{j \in \mathcal{J}_1} B_j \subset L(V) \text{ et } |\mathcal{J}_1| \geq \left\lceil \frac{d_1+1}{n_0} \right\rceil.$$

(iii) *Pour tous s, $1 \leq s < s_f$, il existe un ensemble $\mathcal{J}_{s+1} \subset \{0, \ldots, n_r - 1\}$ tel que*

$$B_{f_{s+1}} := \bigoplus_{j \in \mathcal{J}_{s+1}} B_j \subset L\left(\bigoplus_{j \in \mathcal{J}_s} B_j \right) \text{ et } |\mathcal{J}_{s_f}| \geq \left\lceil \frac{d_1+1}{n_0} \right\rceil.$$

(iv) *Pour tous s, $1 \leq s < s_b$, il existe un ensemble $\mathcal{I}_{s+1} \subset \{0, \ldots, n_r - 1\}$ tel que*

$$B_{b_{s+1}} = \bigoplus_{j \in \mathcal{I}_{s+1}} B_j \subset L^{-1}\left(\bigoplus_{j \in \mathcal{I}_s} B_j \right) \text{ et } |\mathcal{I}_{s_b}| \geq \left\lceil \frac{d_2+1}{n_0} \right\rceil.$$

Soit V le sous-espace complémentaire de W. Alors, les ensembles

$$X_a = \{(G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_b - 1} \circ \chi^{-1})(a + z), z \in V\}, a \in W$$

forment une partition en sommes nulles de $\mathbf{F}_2^n$ de taille 2^k pour la permutation P, constituée de $(r + s_b + s_f - 2)$ tours.

Démonstration. D'après la définition des ensembles X_a nous choisissons des états intermédiaires après la partie non-linéaire du tour $t + s_b$. De la deuxième relation nous déduisons que $A(V)$ peut être vu comme une union de translatés de B_{f_1} :

$$A(V) = \bigcup_{b \in \mathcal{E}_1} (b + B_{f_1}),$$

pour un sous-ensemble $\mathcal{E}_1$ de $\mathbf{F}_2^n$. Comme χ s'applique aux mots séparément, les variables des différents mots ne sont pas mélangées entre elles. En conséquence,

$$\chi(A(V)) = \bigcup_{b' \in \mathcal{E}'_1} (b' + B_{f_1}),$$

où $\mathcal{E}'_1$ est un autre sous-ensemble de $\mathbf{F}_2^n$. De la même façon, d'après la condition (iii), il existe pour chaque $1 \leq s < s_f$ des sous-ensembles $\mathcal{E}_{s+1}, \mathcal{E}'_{s+1} \subset \mathbf{F}_2^n$ tels que

$$\begin{aligned} A(B_{f_s}) &= \bigcup_{b \in \mathcal{E}_{s+1}} (b + B_{f_{s+1}}), \\ \chi(A(B_{f_s})) &= \bigcup_{b' \in \mathcal{E}'_{s+1}} (b' + B_{f_{s+1}}). \end{aligned}$$

Par définition, les images par P des éléments dans X_a correspondent aux images de $a + z$, $z \in V$, par $F_{r-t-2} \circ A_2 \circ (\chi \circ A)^{s_f}$. Il s'ensuit que leur somme est donnée par

$$\begin{aligned} \sum_{z \in V} F_{r-t-2} \circ A_2 \circ (\chi \circ A)^{s_f}(a+z) &= \sum_{z \in V} F_{r-t-2} \circ A_2 \circ (\chi \circ A)^{s_f - 1} \circ (\chi \circ A)(a+z) \\ &= \sum_{b_1 \in \mathcal{E}'_1} \sum_{x \in B_{f_1}} (F_{r-t-2} \circ A_2) \circ (\chi \circ A)^{s_f - 1}(b_1 + x) \\ &= \sum_{b_1 \in \mathcal{E}'_1} \sum_{x \in B_{f_1}} (F_{r-t-2} \circ A_2) \circ (\chi \circ A)^{s_f - 2} \circ (\chi \circ A)(b_1 + x) \\ &= \sum_{b_2 \in \mathcal{E}'} \sum_{x \in B_{f_2}} (F_{r-t-2} \circ A_2) \circ (\chi \circ A)^{s_f - 2}(b_2 + x) \\ &\vdots \\ &= \sum_{b_{s_f} \in \mathcal{E}''} \sum_{x \in B_{f_s}} (F_{r-t-2} \circ A_2)(b_{s_f} + x) \\ &= \sum_{b_{s_f} \in \mathcal{E}''} D_{B_{f_s}} (F_{r-t-2} \circ A_2)(b_{s_f}). \end{aligned}$$

Puisque

$$\dim B_{f_s} \geq n_0 \left\lceil \frac{d_1 + 1}{n_0} \right\rceil > d_1,$$

cette dérivée s'annule.

Nous calculons ensuite en arrière les images de $a + V$ par $G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_b - 1} \circ \chi^{-1}$. Comme $B_{b_1} \subset V$, V peut être exprimé comme une union de translatés de B_{b_1}. Comme χ^{-1} ne mélange pas les mots entre eux, nous avons que

$$\chi^{-1}(V) = \bigcup_{c \in \mathcal{Z}'_1} (c + B_{b_1}),$$

pour un ensemble $\mathcal{Z}'_1$.

D'après la condition (iv) il existe pour chaque $1 \leq s < s_b$ des sous-ensembles $\mathcal{Z}_{s+1}, \mathcal{Z}'_{s+1} \subset \mathbf{F}_2^n$ tels que

$$\begin{aligned} A^{-1}(B_{b_s}) &= \bigcup_{c \in \mathcal{Z}_{s+1}} (c + B_{b_{s+1}}), \\ \chi^{-1}(A^{-1}(B_{b_s})) &= \bigcup_{c' \in \mathcal{Z}'_{s+1}} (c' + B_{b_{s+1}}). \end{aligned}$$

Alors, la somme des états d'entrée correspondant à $x \in X_a$ est donnée par

$$\begin{aligned} \sum_{x \in X_a} x &= \sum_{z \in V} G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_b} \circ \chi^{-1}(a + z) \\ &= \sum_{c_1 \in \mathcal{Z}'_1} \sum_{x \in B_{b_1}} G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_b}(c_1 + x) \\ &= \sum_{c_1 \in \mathcal{Z}'_1} \sum_{x \in B_{b_1}} G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_{b-1}} \circ (\chi^{-1} \circ A^{-1})(c_1 + x) \\ &= \sum_{c_2 \in \mathcal{Z}'} \sum_{x \in B_{b_2}} G_t \circ A_1^{-1} \circ (\chi^{-1} \circ A^{-1})^{s_{b-1}}(c_2 + x) \\ &\vdots \\ &= \sum_{c_{s_b} \in \mathcal{Z}''} \sum_{x \in B_{b_s}} (G_t \circ A_1^{-1})(c_{s_b} + x) \\ &= \sum_{c_{s_b} \in \mathcal{Z}''} D_{B_{s_b}} (G_t \circ A_1^{-1})(c_{s_b}). \end{aligned}$$

Puisque

$$\dim B_{b_s} \geq n_0 \left\lceil \frac{d_2 + 1}{n_0} \right\rceil > d_2,$$

cette dérivée s'annule. □

Il est intéressant de constater qu'il n'existe aucune exigence pour la taille des ensembles intermédiaires $\mathcal{I}_2, \ldots, \mathcal{I}_{s_b - 1}$ et $\mathcal{J}_2, \ldots, \mathcal{J}_{f_b - 1}$, à part évidemment que la taille des ensembles $\mathcal{I}_i$ doit être inférieure à la taille de $\mathcal{I}_{s_b}$ et celle des ensembles $\mathcal{J}_i$ doit être inférieure à la taille de $\mathcal{J}_{s_f}$.

Nous allons ensuite montrer des applications de ces théorèmes à deux candidats du concours SHA-3, à savoir KECCAK et Hamsi.

3.6 Application à la permutation interne de KECCAK

KECCAK est une famille de fonctions de hachage conçue par Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche [BDPV09]. Ses instances sont des fonctions de type éponge, basées sur une permutation interne, appelée KECCAK-f. Chaque instance de la famille est caractérisée par la taille de la permutation, b. Il existe 7 permutations KECCAK-f différentes, notées KECCAK-$f[b]$, avec $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. La fonction KECCAK-$f[1600]$ a été soumise au concours SHA-3 et a été recemment sélectionnée par le

NIST comme le nouveau standard. Dans le reste de ce mémoire, KECCAK-f fera référence à KECCAK-f[1600].

3.6.1 Description de la permutation KECCAK-f

La permutation KECCAK-f s'applique sur un état de 1600 bits, qui est représenté par une matrice binaire en trois dimensions, $a[x][y][z]$, avec $0 \leq x, y < 5$ et $0 \leq z < 64$. En conséquence, l'état peut être visualisé comme 64 *tranches* dont chacune contient 5 lignes et 5 colonnes. Le nombre initial de tours pour la permutation KECCAK-f a été fixé à 18 tours, mais a augmenté à 24 pour le deuxième tour de la compétition SHA-3. Ce changement est du à la publication des distingueurs par somme nulle par Aumasson et Meier [AM09]. Ces distingueurs n'affectent pas directement la sécurité de la fonction, mais violent la stratégie *hermetic sponge* qui garantit la sécurité de la fonction de hachage sous la condition que la permutation interne ne présente aucune propriété structurelle exploitable.

Chaque tour R est composé d'une séquence de 5 permutations, $\theta, \rho, \pi, \chi, \iota$, qui modifient l'état, comme suit :

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta.$$

Chaque transformation est chargée d'une tâche différente, afin d'assurer à la fois la confusion et une bonne diffusion dans toutes les directions de l'état en trois dimensions.

La transformation θ : Cette fonction est une transformation linéaire qui ajoute au bit $a[x][y][z]$ la somme modulo 2 des parités de deux colonnes $a[x-1][\cdot][z]$ et $a[x][\cdot][z+1]$:

$$a[x][y][z] = a[x][y][z] + \sum_{i=0}^{4} a[x-1][i][z] + \sum_{i=0}^{4} a[x][i][z+1].$$

Elle assure une diffusion de très bonne qualité, la plus importante parmi les 5 transformations de tour.

La transformation ρ : C'est une transformation linéaire qui fournit la diffusion dans chaque tranche de l'état. La permutation ρ translate les bits d'une tranche i d'un certain nombre de positions en avant, comme défini à la table 3.1. Il existe 25 translations différentes, une pour chaque tranche de l'état.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	25	39	3	10	43
$y=1$	55	20	36	44	6
$y=0$	28	27	0	1	62
$y=4$	56	14	18	2	61
$y=3$	21	8	41	45	15

TABLE 3.1 – Valeurs de translations pour la transformation ρ de KECCAK

La transformation π : La fonction π permute les bits dans une tranche.

La transformation χ : Cette permutation est la seule fonction non-linéaire de R. Elle s'applique sur les 320 lignes de l'état. Plus précisément, une boîte-S 5×5, notée χ_0, est appliquée en parallèle sur chaque ligne de l'état. Il s'agit d'une fonction quadratique dont l'ANF de chaque coordonnée est donnée par :

$$\begin{aligned} \chi_0 : \mathbf{F}_2^5 &\to \mathbf{F}_2^5 \\ (x_0, x_1, x_2, x_3, x_4) &\mapsto (x_0x_2 + x_1 + x_2, \\ &\quad x_1x_3 + x_2 + x_3, \\ &\quad x_2x_4 + x_3 + x_4, \\ &\quad x_3x_0 + x_4 + x_0, \\ &\quad x_4x_1 + x_0 + x_1). \end{aligned}$$

La permutation inverse de χ_0, notée χ_0^{-1} est une fonction de degré 3 :

$$\begin{aligned} \chi_0^{-1} : \mathbf{F}_2^5 &\to \mathbf{F}_2^5 \\ (x_0, x_1, x_2, x_3, x_4) &\mapsto (x_0 + x_2 + x_4 + x_1x_2 + x_1x_4 + x_3x_4 + x_1x_3x_4, \\ &\quad x_0 + x_1 + x_3 + x_0x_2 + x_0x_4 + x_2x_3 + x_0x_2x_4, \\ &\quad x_1 + x_2 + x_4 + x_0x_1 + x_1x_3 + x_3x_4 + x_0x_1x_3, \\ &\quad x_0 + x_2 + x_3 + x_0x_4 + x_1x_2 + x_2x_4 + x_1x_2x_4, \\ &\quad x_1 + x_3 + x_4 + x_0x_1 + x_0x_3 + x_2x_3 + x_0x_2x_3). \end{aligned}$$

La transformation ι : Cette fonction ajoute des constantes à certains bits de l'état.

Afin d'être cohérents avec la notation introduite dans les sections 3.4 et 3.5, nous allons noter par $A_1 = \pi \circ \rho \circ \theta$ et par $A_2 = \iota$. En conséquence, la partie linéaire de $A = A_1 \circ A_2$ correspond à $L = \pi \circ \rho \circ \theta$.

En parallèle, nous avons besoin de définir une numérotation des $n = 1600$ bits de l'état interne de Keccak-f. Pour cela, nous associons au bit qui se trouve à l'intersection de la ligne x et de la colonne y de la tranche z, c'est-à-dire à l'élément (x, y, z), $0 \leq x, y < 5$ et $0 \leq z < 64$, le numéro $25z + 5y + x$. Selon les spécifications de la fonction, les éléments de la forme $(0, 0, z)$ se trouvent au centre de chaque tranche.

Comme la fonction non-linéaire χ s'applique aux lignes de l'état de Keccak-f, un mot selon la notation que nous avons introduit auparavant, correspondra à une ligne de l'état. Au total, il y en a alors 320 mots. Par conséquent, le sous-espace généré par la ligne y de la tranche z est défini par :

$$B_{5z+y} = \langle e_{25z+5y}, e_{25z+5y+1}, e_{25z+5y+2}, e_{25z+5y+3}, e_{25z+5y+4} \rangle.$$

3.6.2 Les partitions en sommes nulles dans [AM09]

Les premières partitions en sommes nulles pour Keccak-f sont présentées dans [AM09]. Le meilleur résultat consiste en plusieurs partitions pour la permutation réduite à 16 tours. Pour y parvenir, les auteurs n'ont utilisé que des techniques exploitant le degré algébrique de la fonction. Plus précisément, ils ont appliqué la borne triviale pour trouver une estimation pour le degré en avant et en arrière.

Selon cette borne, 10 tours de la permutation ont degré au plus $d_1 = 2^{10} = 1024$. De façon équivalente, 6 tours en arrière sont de degré au plus $d_2 = 3^6 = 729$. Ainsi, ils choisissent des

états intermédiaires après $t = 6$ tours de la permutation, dans un translaté d'un sous-espace V de dimension $d + 1$ avec

$$d = \max(1024, 729) = 1024.$$

Cette méthode conduit à plusieurs partitions de taille 2^{1025}.

Nous présentons ensuite des améliorations de cette méthode, pour trouver des partitions pour un nombre de tours supérieur à 16. Pour cela, nous utilisons des techniques présentées dans les sections 3.4 et 3.5.

3.6.3 Partitions en sommes nulles pour 18 tours de Keccak-f

La première amélioration que nous avons apportée par rapport à [AM09] est une estimation plus précise du degré de la permutation inverse. Plus précisément, nous avons montré que le degré de 7 tours de la permutation inverse de Keccak-f est au plus 1369, qui est une valeur beaucoup plus basse que l'estimation par la borne triviale, $\min(3^7, 1599$[1]$)$. Pour cela nous avons utilisé le résultat de Anne Canteaut et Marion Videau (proposition 3.2).

Nous avons remarqué que tous les éléments du spectre de Walsh de la permutation χ_0 sont divisibles par 2^3. D'après la proposition 2.2, les spectres de Walsh d'une permutation et de son inverse, sont égaux. Par conséquent, le spectre de Walsh de χ_0^{-1} est également divisible par 2^3. Nous pouvons ici remarquer que pour une permutation quadratique de $\mathbf{F}_2^n$ pour n impair, $2^{\frac{n+1}{2}}$ est la divisibilité la moins élevée possible. Pour cette raison, le fait que le spectre de Walsh de χ_0^{-1} soit divisible par 2^3 aurait été vrai pour tout choix de permutation quadratique χ_0 dans $\mathbf{F}_2^5$.

Dans le cas de Keccak, $n_r = 320$. En conséquence, nous déduisons d'après l'équation (3.4), que le spectre de la permutation de tour R, ainsi que le spectre de la permutation inverse R^{-1}, sont divisibles par $2^{3\times 320} = 2^{960}$. En utilisant que le degré de six tours de la permutation inverse est au plus $3^6 = 729$, nous obtenons d'après la proposition 3.2 que

$$\deg(R^{-7}) = \deg(R^{-6} \circ R^{-1}) \leq 1600 - 960 + \deg(R^{-6}) \leq 1369.$$

Cette observation nous permet d'améliorer d'un tour le résultat dans [AM09], et de trouver des partitions en sommes nulles pour 17 tours de Keccak-f. Pour cela, il suffit de choisir des états intermédiaires après $t = 7$ tours de la permutation dans des translatés d'un sous-espace V de dimension 1370 et de calculer 7 tours en arrière.

Ensuite, nous pouvons étendre ces partitions à 18 tours, en utilisant la proposition 3.4. Pour cela, il suffit de choisir un sous-espace $V = \bigoplus_{i\in\mathcal{I}} B_i$, où l'ensemble $\mathcal{I}$ est une collection quelconque de 274 lignes de l'état. Comme la dimension de V est $5\times 274 = 1370$, nous pouvons désormais trouver des partitions en sommes nulles de taille 2^{1370} pour 18 tours de Keccak-f.

Remarque 3.2. *Afin de déterminer une borne pour le degré de 7 tours de la permutation inverse, nous avons décomposé R^{-7} en R^{-6} et R^{-1}. Nous avons fait cela afin d'exploiter la divisibilité du spectre de Walsh de R^{-1} par une grande puissance de 2. Cependant, au cours de cette thèse, nous avons testé une autre décomposition de R^{-7}. Dans le détail, nous avons décomposé R^{-7} en R^{-5} et R^{-2}, et nous avons essayé d'extraire des informations sur le spectre de Walsh de R^{-2}. Notre but était de comparer les bornes données par les deux décompositions et de diminuer éventuellement la taille des partitions déjà connues. Nous avons alors examiné, pour une permutation F de $\mathbf{F}_2^n$, la divisibilité du spectre de Walsh de*

1. Le degré d'une permutation F de $\mathbf{F}_2^n$ vers $\mathbf{F}_2^n$, ne peut pas dépasser $n - 1$.

F^2 en utilisant les informations sur le spectre de Walsh de F. Néanmoins, nous n'avons pas pu trouver une relation qui relie les deux quantités.

3.6.4 Partitions en sommes nulles pour 19 tours de KECCAK-f

Nous allons ensuite utiliser la méthode décrite dans le théorème 3.3, pour étendre les partitions en sommes nulles déjà trouvées, à 19 tours. Dans ce cas, $r = 19$ et nous choisissons, comme avant, $t = 7$. En suivant les notations introduites dans les sections précédentes, $F_{r-t-2} = F_{10}$, avec $\deg(F_{10}) \leq 1024$ et $G_t = G_7$, avec $\deg(G_7) \leq 1369$. Nous cherchons ensuite un sous-espace W tel qu'il existe deux sous-ensembles $\mathcal{I}, \mathcal{J} \subset \{0, \ldots, 319\}$ de façon que :

$$W \subset \bigoplus_{i \in \mathcal{I}} B_i \text{ avec } |\mathcal{I}| \leq 46 \text{ et } L^T(W) \subset \bigoplus_{j \in \mathcal{J}} B_j \text{ avec } |\mathcal{J}| \leq 115,$$

où les bornes sur $|\mathcal{I}|$ et $|\mathcal{J}|$ sont imposées par les degrés de F_{10} et G_7. Nous avons choisi pour W l'espace généré par les lignes $0, 5, 105, 110, 209$ et 214 de l'état, c'est-à-dire

$$W = B_0 \oplus B_5 \oplus B_{105} \oplus B_{110} \oplus B_{209} \oplus B_{214}.$$

Nous avons vérifié qu'il existe un sous-ensemble $\mathcal{J}$ de taille 110, tel que $L^T(W) \subset \bigoplus_{j \in \mathcal{J}} B_j$. Comme $\dim W = 5 \times 6 = 30$, nous déduisons du théorème 3.3 qu'il existe une partition en sommes nulles de taille 2^{1570} pour 19 tours de KECCAK-f.

Cependant, il est possible d'améliorer la complexité du distingueur pour 19 tours. Pour cela il suffit d'augmenter la dimension de W, sans en même temps augmenter le cardinal de $\mathcal{J}$. Afin de réussir cela, nous ajoutons à W un nombre de vecteurs linéairement indépendants, dont les images appartiennent à $\bigoplus_{j \in \mathcal{J}} B_j$, le même ensemble $\mathcal{J}$ que précédemment. Le nouvel espace W est généré par les lignes $0, 5, 105, 110, 209, 214$ et par les vecteurs suivants, qui sont linéairement indépendants et qui appartient au total à 40 lignes de l'état, différentes des précédentes :

e_6, e_9, e_{35}, e_{37}, e_{50}, e_{51}, e_{60}, e_{62}, e_{65}, e_{68}, e_{80}, e_{82},
e_{83}, e_{85}, e_{87}, e_{105}, e_{107}, e_{109}, e_{110}, e_{112}, e_{114}, e_{130}, e_{132}, e_{134},
e_{137}, e_{139}, e_{145}, e_{149}, e_{155}, e_{156}, e_{157}, e_{175}, e_{178} e_{210}, e_{211}, e_{213},
e_{235}, e_{236}, e_{238} e_{316}, e_{317}, e_{319}, e_{340}, e_{341}, e_{342}, e_{344}, e_{365}, e_{366},
e_{369}, e_{470}, e_{472}, e_{473}, e_{495}, e_{497}, e_{498}, e_{500}, e_{501}, e_{502}, e_{576}, e_{577},
e_{578}, e_{581}, e_{582}, e_{583}, e_{661}, e_{662}, e_{664}, e_{711}, e_{712}, e_{713}, e_{890}, e_{893},
e_{894}, e_{916}, e_{918}, e_{919}, e_{1075}, e_{1077}, e_{1078}, e_{1100}, e_{1101}, e_{1102}, e_{1135}, e_{1136},
e_{1137}, e_{1182}, e_{1183}, e_{1184}, e_{1186}, e_{1187}, e_{1188}, e_{1205}, e_{1208}, e_{1209}, e_{1275}, e_{1276},
e_{1277}, e_{1416}, e_{1418}, e_{1419}, e_{1441}, e_{1443}, e_{1444}, e_{1466}, e_{1468}, e_{1469}, e_{1570}, e_{1571},
e_{1573}, e_{1595}, e_{1596}, e_{1598}.

Nous avons vérifié que $W \subset \bigoplus_{i \in \mathcal{I}} B_i$, avec $|\mathcal{I}| = 46$ et $L^T(W) \subset \bigoplus_{j \in \mathcal{J}} B_j$, avec $|\mathcal{J}| = 115$. Comme $\dim W = 5 \times 6 + 112 = 142$, le corollaire 3.1 conduit à une nouvelle partition en sommes nulles de taille 2^{1458} pour 19 tours de KECCAK-f.

3.6.5 Partitions en sommes nulles pour 20 tours de KECCAK-f

Afin de trouver une partition en sommes nulles pour 20 tours de KECCAK-f, nous appliquons le théorème 3.4 avec $s_f = 2$, c'est-à-dire nous calculons un tour supplémentaire en

avant. Nous cherchons alors un sous-espace W tel qu'il existe trois ensembles de lignes $\mathcal{I}$, $\mathcal{J}_1$ et $\mathcal{J}_2$ avec $|\mathcal{I}| \leq 46$ et $|\mathcal{J}_2| \leq 115$ de façon que :

$$W \subset \bigoplus_{i \in \mathcal{I}} B_i, \quad L^T(W) \subset \bigoplus_{j \in \mathcal{J}_1} B_j \text{ et } L^T\left(\bigoplus_{j \in \mathcal{J}_1} B_j\right) \subset \bigoplus_{j \in \mathcal{J}_2} B_j.$$

Nous avons calculé que l'image par L^T de l'espace généré par $\bigoplus_{i=4}^{6} B_i$ appartient à l'union de 115 lignes de l'état. Nous cherchons alors un sous-espace W qui appartient au sous-espace généré par au plus 46 lignes de façon que $L^T(W)$ appartienne à l'union des lignes B_4, B_5 et B_6.

Nous avons trouvé le sous-espace W suivant généré par 5 vecteurs linéairement indépendants :

$$e_{531} \oplus e_{715}, \quad e_{25} \oplus e_{941}, \quad e_{60} \oplus e_{1126}, \quad e_{186} \oplus e_{1595}, \quad e_{730} \oplus e_{1421}.$$

Pour cet espace W, $W \subset \bigoplus_{i \in \mathcal{I}} B_i$ avec $|\mathcal{I}| \leq 10$ et $L^T(W)$ appartient aux lignes $4, 5$ et 6 de l'état. Comme $\dim W = 5$, nous déduisons du théorème 3.3 que nous avons trouvé une partition en sommes nulles de taille 2^{1595} pour 20 tours de Keccak-f.

3.7 Application à la permutation de finalisation de Hamsi-256

Nous avons également cherché des structures à somme nulle pour un autre candidat de la compétition SHA-3, à savoir Hamsi. Cette fonction a la particularité d'avoir un degré algébrique très faible, ce qui vient du fait que le nombre de tours de la fonction est petit. Cette caractéristique a conduit à plusieurs attaques algébriques sur la fonction de hachage, [Fuh10, DS11]. Dans le chapitre 6 nous allons examiner ces propriétés algébriques plus en détail. Dans cette section, nous nous concentrons sur la recherche de partitions en sommes nulles pour la permutation de finalisation de Hamsi-256. Nous allons voir, qu'en comparaison avec Keccak, les partitions en sommes nulles pour Hamsi sont de taille beaucoup plus petite.

Pour la recherche des partitions en sommes nulles pour la fonction de Hamsi, seulement une description rapide de la structure de la permutation de finalisation est nécessaire. Cependant, nous allons fournir une description détaillée de toute la fonction, car nous allons en avoir besoin pour l'attaque algébrique du chapitre 6.

3.7.1 Description de Hamsi-256

Hamsi est une famille de fonctions de hachage, conçue par Özgül Küçük [Kuc09] pour le concours SHA-3. Elle a été parmi les 14 candidats sélectionnés pour le deuxième tour de la compétition. Nous n'allons décrire que Hamsi-256, c'est-à-dire l'instance donnant des empreintes de taille 256 bits.

Cette fonction suit la construction Davies-Meyer en utilisant une expansion de message définie à l'aide de codes linéaires. La fonction de compression h est basée sur une permutation P, opère sur un état de 512 bits. Cet état peut être visualisé comme une matrice 4×4 de mots de 32 bits.

Dans Hamsi-256, le message est rembourré et séparé en blocs de 32 bits. Chaque bloc de message M_i est combiné avec une valeur de chaînage h_{i-1} de 256 bits pour donner la valeur

de chaînage h_i suivante :

$$\begin{aligned} h : \mathbf{F}_2^{32} \times \mathbf{F}_2^{256} &\to \mathbf{F}_2^{256} \\ (m_i, h_{i-1}) &\mapsto h_i. \end{aligned}$$

Nous allons présenter les étapes les plus importantes du calcul.

Expansion de message : Un code linéaire sur $\mathbf{F}_4$ est utilisé afin d'étendre un bloc de message de 32 bits à une valeur de 256 bits, $(m_0, \ldots, m_7)$, où chaque m_i est un mot de 32 bits.

Concaténation : La valeur de chaînage $(c_0, \ldots, c_7)$ est concaténée au message étendu $(m_0, \ldots, m_7)$ afin de former un état de 512 bits $s = (s_0, \ldots, s_{15})$, représenté par une matrice 4×4. L'état s, ainsi que la manière dont les mots de message et de la valeur de chaînage sont répartis dans s, peut être visualisé à la figure 3.5.

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

m_0	m_1	c_0	c_1
c_2	c_3	m_2	m_3
m_4	m_5	c_4	c_5
c_6	c_7	m_6	m_7

FIGURE 3.5 – Concaténation du message et de la valeur de chaînage pour Hamsi-256

Une permutation non-linéaire P dans $\mathbf{F}_2^{512}$ est ensuite appliquée à l'état concaténé. Elle est constituée d'une fonction de tour R qui est itérée trois fois. La permutation R est composée de trois opérations différentes. D'abord, des constantes sont ajoutées à l'état. Ensuite, une transformation non-linéaire composée des applications parallèles d'une boîte-S 4×4, S, est appliquée à l'état. À la fin, les bits de l'état sont mélangés, grâce à une fonction linéaire L.

Ajout des constantes : Des constantes $a_i \in \mathbf{F}_2^{32}$, $0 \leq i < 16$, sont ajoutées à l'état avant l'application de la couche non-linéaire, à l'aide d'une opération XOR. De plus, un compteur, c, est utilisé pour distinguer les tours entre eux. Ces deux opérations sont définies comme suit :

$$(s_0, s_1, \ldots, s_{15}) := (s_0 \oplus a_0, s_1 \oplus a_1 \oplus c, \ldots, s_{15} \oplus a_{15}).$$

La transformation non-linéaire : La couche non-linéaire de Hamsi est basée sur une boîte-S de 4 bits, S, qui est une des boîtes-S utilisées dans le chiffrement par blocs `Serpent` :

$$S[16] = \{8, 6, 7, 9, 3, 12, 10, 15, 13, 1, 14, 4, 0, 11, 5, 2\}.$$

Le degré algébrique de S ainsi que de son inverse est 3. Cette boîte-S est appelée en parallèle sur les 128 colonnes de l'état. Au premier tour, à cause de la manière dont l'état est concaténé, S mélange deux bits du message avec deux bits de la valeur de chaînage.

La transformation linéaire : La couche de diffusion dans Hamsi-256 est basée sur une fonction linéaire $L : \mathbf{F}_2^{128} \rightarrow \mathbf{F}_2^{128}$ qui s'applique aux mots de 32 bits. La fonction L est appelée 4 fois dans un tour, une fois par diagonale de l'état :

$$\begin{aligned}(s_0, s_5, s_{10}, s_{15}) &:= L(s_0, s_5, s_{10}, s_{15}) \\ (s_1, s_6, s_{11}, s_{12}) &:= L(s_1, s_6, s_{11}, s_{12}) \\ (s_2, s_7, s_8, s_{13}) &:= L(s_2, s_7, s_8, s_{13}) \\ (s_3, s_4, s_9, s_{14}) &:= L(s_3, s_4, s_9, s_{14})\end{aligned}$$

La fonction $L(a, b, c, d)$, avec $a, b, c, d \in \mathbf{F}_2^{32}$ est décrite comme suit :

$$\begin{aligned}a &:= a \lll 13 \\ c &:= c \lll 3 \\ b &:= b \oplus a \oplus c \\ d &:= d \oplus c \oplus (a \ll 3) \\ b &:= b \lll 1 \\ d &:= d \lll 7 \\ a &:= a \oplus b \oplus d \\ c &:= c \oplus d \oplus (b \ll 7) \\ a &:= a \lll 5 \\ c &:= c \lll 22\end{aligned}$$

Troncation : Après l'application de la permutation P, la moitié des bits de l'état sont tronqués. Plus précisément, la transformation $T : \mathbf{F}_2^{512} \rightarrow \mathbf{F}_2^{256}$ élimine tous les bits de la deuxième et de la quatrième ligne.

$$T(s_0, s_1, s_2, \ldots, s_{14}, s_{15}) = (s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}).$$

Feed-forward : Après la troncation, les 256 bits de l'état sont additionnés (par un XOR) à la valeur de chaînage h_{i-1} précédente, afin de former h_i.

La permutation P_f : Pour le dernier bloc du message rembourré, la permutation P est remplacée par une autre permutation, appelée P_f. La permutation P_f est presque identique à P. La seule différence consiste en les constantes appliquées à chaque tour et en le nombre de tours, qui est 6 pour P_f.

Nous allons maintenant montrer comment trouver des partitions en sommes nulles pour la permutation P_f.

Partitions en sommes nulles pour P_f

Comme un tour de la fonction est de degré 3, le degré de 3 tours de R est au plus $3^3 = 27$. C'est également vrai pour la fonction inverse. Cette remarque a été utilisée dans [ADMS09] afin de trouver des partitions en sommes nulles de taille 2^{28}. Nos techniques peuvent toutefois êtres utilisées afin de mettre en évidence des partitions en sommes nulles de plus petite taille.

Nous commençons par définir une numérotation pour les bits de l'état interne. Le bit j du mot de 32 bits qui se trouve à l'intersection de la ligne i et la colonne k de l'état, $0 \leq k < 4$, $0 \leq j < 32$ est numéroté $128k + 4j + i$. Dans le cas de Hamsi, le rôle des mots est maintenant joué par les colonnes de l'état, comme S s'applique aux colonnes et non pas aux lignes comme c'était le cas de Keccak. Pour cette raison, nous définissons un sous-espace B_i, $0 \leq i < 128$, comme l'espace engendré par la colonne i de l'état, c'est-à-dire

$$B_i = \langle e_{4i}, e_{4i+i}, e_{4i+2}, e_{4i+3} \rangle.$$

Nous choisissons des états après $t = 3$ tours de la permutation, dans l'espace de dimension 19 suivant :

$$V = \bigoplus_{i=14}^{16} B_i \oplus \langle e_{68}, e_{237}, e_{241}, e_{245}, e_{249}, e_{507}, e_{511} \rangle.$$

Nous considérons ensuite les ensembles

$$X_a = \{((R^{-1})^2 \circ S^{-1})(a + z), \quad z \in V\},$$

et nous utilisons la technique du théorème 3.3. Les sous-espaces V et $S^{-1}(V)$ peuvent être vus tous les deux comme une union de translatés de $B_{14} \oplus B_{15} \oplus B_{16}$. Comme le degré de deux itérations de R^{-1} est au plus 9, la somme de tous les éléments de X_a est nulle, puisque $\dim(\oplus_{i=14}^{16} B_i) > 9$. De plus, $\langle e_0, e_4, e_8, e_{12} \rangle \subset L(V)$ et il a été remarqué dans [ADMS09] que trois tours de R sont de degré 3 par rapport aux bits de poids faible de la première colonne de l'état. Par conséquent, comme $L(V)$ est une union de translatés de $B_f = \langle e_0, e_4, e_8, e_{12} \rangle$ et $D_{B_f} R^3(x) = 0$ pour tous les x, nous déduisons que les images de tous les éléments de X_a par six tours de R ont une somme nulle. Plusieurs partitions en sommes nulles de taille 2^{19} peuvent être construites de cette façon. En effet, la seule condition qui doit être satisfaite, est que $L(V)$ contienne le sous-espace généré par les 4 premiers bits de n'importe quel mot (colonne) de l'état interne.

Nous décrivons enfin une technique différente pour trouver des partitions en sommes nulles pour P_f. Cette méthode conduit à des partitions en sommes nulles de taille 2^{10} seulement. Pour cela, il suffit de considérer dix éléments quelconques dans un mot de 32 bits, après 3 tours de permutation et d'imposer le reste des bits de l'état à une valeur fixe. Alors, 3 tours de la permutation appliqués à un tel état sont de degré au plus 9. La raison est que puisque il existe au plus un bit par boîte-S active, tous les bits après le premier tour seront une combinaison affine des dix variables en entrée. C'est également vrai pour la permutation inverse. Comme après l'application de L^{-1} à cet état, il aura une variable par colonne et donc au plus un bit par boîte-S active.

Chapitre 4

Borne sur le degré des permutations itérées

Sommaire

4.1 Une nouvelle borne pour les constructions de type SPN 78
4.1.1 Comprendre la nouvelle borne 81
4.1.2 Lien entre la nouvelle borne et la borne triviale 82
4.2 Partitions en sommes nulles pour KECCAK-f 83
4.3 Application à la fonction de hachage *Luffa* 85
4.3.1 Spécifications de la fonction *Luffa* v1 86
4.3.2 Description de l'attaque [WHYK10] 87
4.3.3 Différentielles d'ordre supérieur pour la fonction de hachage *Luffa* v1 entière 90
4.3.4 Modifications sur la fonction *Luffa* v1 91
4.3.5 Degré de la permutation Q_j et de son inverse pour *Luffa* v2 91
4.3.6 Partitions en sommes nulles pour la permutation Q_j de *Luffa* v2 . . 93
4.3.7 Différentielles d'ordre supérieur pour la fonction de compression de *Luffa* v2 94
4.3.8 Degré algébrique de la fonction *Luffa* v2 avec valeurs initiales choisies 94
4.4 Application à l'AES 95
4.5 Application à la permutation interne de Grøstl 96
4.5.1 Borne sur le degré de la permutation P de Grøstl-256 98

Dans le chapitre précédent, nous avons examiné des structures à somme nulle. Nous avons pu voir que pour trouver des sommes nulles de la plus petite taille possible pour un nombre maximal de tours, il fallait savoir estimer correctement le degré d'une permutation itérée. L'exemple des sommes nulles n'est qu'un cas parmi toutes les situations où nous avons besoin de déterminer le degré d'une primitive symétrique après quelques itérations. Nous avons déjà vu quelques exemples dans le chapitre 2.

Pour estimer le degré algébrique, la borne généralement utilisée est la borne que nous avons appelée "triviale" (Proposition 3.3). C'est la borne qui est utilisée dans [AM09] afin de trouver les premières partitions de somme nulle sur certaines fonctions de hachage. Néanmoins, nous avons montré que cette borne reflète très mal la réalité quand le nombre de tours est élevé.

Nous avons alors rappelé une autre borne, plus efficace que la borne triviale quand le spectre de Walsh de la fonction itérée est divisible par une grande puissance de 2 (Proposition 3.2).

Dans ce chapitre, nous allons présenter une borne différente, qui s'applique assez efficacement, comme nous allons voir, sur les constructions de type SPN. Les résultats et les applications de ce chapitre, qui sont les fruits d'un travail commun avec Anne Canteaut et Christophe De Cannière, ont été présenté à la conférence FSE 2011 [BCD11].

4.1 Une nouvelle borne pour les constructions de type SPN

Nous avons vu à la section 1.4.2 que les chiffrements de type substitution-permutation, sont constitués d'une succession de couches non-linéaires et linéaires. Nous avons expliqué que pour des raisons d'implémentation, la partie non-linéaire est souvent composée d'un grand nombre de petites boîtes-S, qui sont appliquées sur l'état en parallèle. Nous allons maintenant voir de quelle manière cette construction influence le degré total de la fonction et nous allons en déduire une nouvelle borne sur le degré.

Avant de présenter le résultat principal, nous allons introduire une notion simple mais très utile pour la suite : le produit des coordonnées d'une fonction vectorielle.

Soit $F = (f_1, \ldots, f_m)$ une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Si $I \subset \{1, \ldots, m\}$, nous notons par F^I la fonction booléenne de n variables correspondant au produit des coordonnées de F qui sont indexées par I :

$$F^I = \prod_{i \in I} f_i.$$

Définition 4.1. *Soit $F = (f_1, \ldots, f_m)$ une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Notons par $\delta_k(F)$, $1 \leq k \leq m$ le degré maximal que peut avoir le produit de k coordonnées distinctes de F,*

$$\delta_k(F) = \max_{I \subset \{0,1,\ldots,m\}, |I|=k} \deg(F^I).$$

Exemple 4.1. *Soit F une fonction de $\mathbf{F}_2^3$ dans $\mathbf{F}_2^3$, définie par*

$$F(x_0, x_1, x_2) = (x_0x_2 + x_1, x_0 + x_1 + 1, x_0x_1x_2 + x_0x_1 + x_0 + x_2).$$

Si $k = 2$, alors

$$\delta_2(F) = \max(\deg(f_0f_1), \deg(f_0f_2), \deg(f_1f_2)) = 3.$$

Nous voyons également, que $\delta_1(F)$ est simplement le degré algébrique de la fonction F, puisque

$$\delta_1(F) = \max(\deg(f_0), \deg(f_1), \deg(f_2)) = \deg(F).$$

Proposition 4.1. *Soit F une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, et soit k un entier, $1 \leq k \leq m$. Alors, le poids de Hamming du produit de k coordonnées de F, $\delta_k(F)$, est 2^{n-k}.*

Démonstration. Notons $f_1, \ldots, f_m$, les m coordonnées de F. Soit I un sous-ensemble de $\{1, \ldots, m\}$ de taille k et notons F_I la fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^k$ dont les coordonnées sont les $f_i, i \in I$. Comme la fonction F_I est équilibrée, le multi-ensemble $\{F_I(x), x \in \mathbf{F}_2^n\}$ est constitué de tous les éléments dans $\mathbf{F}_2^k$, chacun avec une multiplicité 2^{n-k}. En conséquence, il existe précisément 2^{n-k} valeurs de x pour lesquelles toutes les coordonnées de F_I sont égales à 1, qui est équivalent à $\prod_{i \in I} f_i = 1$. □

Le corollaire suivant, que nous allons beaucoup utiliser pour les applications, est une conséquence directe de la proposition 4.1.

Corollaire 4.1. *Soit F une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Si $k < n$, le produit de k coordonnées de F est de degré au plus $(n-1)$.*

Démonstration. Soit I un sous-ensemble de $\{1, \ldots, m\}$ de taille $k < n$, et notons par $\pi = \prod_{i \in I} f_i$. Il suffit de montrer que le coefficient, a, du terme $x_0 \cdots x_{n-1}$ dans l'ANF de π est 0. D'après la définition 2.5, nous avons que

$$a = \bigoplus_{v \preceq (1,1,\cdots,1)} \pi(v) = \bigoplus_{x \in \mathbf{F}_2^n} \pi(x)$$

D'après la proposition 4.1, le poids de Hamming de π est pair, par conséquent, la dernière quantité s'annule. □

Nous sommes maintenant prêts pour présenter le résultat principal de ce chapitre, à savoir une borne sur le degré des permutations itérées dont la partie non-linéaire est composée de l'application parallèle de plusieurs boîtes-S.

Théorème 4.1. *Soit F une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^n$ correspondant à la concaténation de m plus petites boîtes-S équilibrées, $S_1, \ldots, S_m$, définies sur $\mathbf{F}_2^{n_0}$. Notons par $\delta_i(S) = \max_{1 \le j \le m} \delta_i(S_j)$. Alors, pour une fonction G de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^\ell$, nous avons que*

$$\deg(G \circ F) \le n - \frac{n - \deg(G)}{\gamma}, \tag{4.1}$$

où

$$\gamma = \max_{1 \le i \le n_0 - 1} \frac{n_0 - i}{n_0 - \delta_i(S)}.$$

En particulier, on en déduit

$$\deg(G \circ F) \le n - \frac{n - \deg(G)}{n_0 - 1}.$$

De plus, si $n_0 \ge 3$ et toutes les boîtes-S sont de degré au plus $n_0 - 2$, alors

$$\deg(G \circ F) \le n - \frac{n - \deg(G)}{n_0 - 2}. \tag{4.2}$$

Démonstration. Chaque coordonnée de $G \circ F$ s'écrit comme un produit d'au plus d coordonnées de F. On note donc par π le produit de d coordonnées de F. Ces d coordonnés impliquées dans le produit peuvent appartenir à différentes boîtes-S ou au contraire, certaines d'entre elles peuvent venir de la même boîte-S. Dans le cas général, pour chaque $1 \le i \le n_0$, nous notons par x_i, l'entier qui correspond au nombre de boîtes-S pour lesquelles exactement i de ses coordonnées sont impliquées dans π. Il est évident que

$$\deg(\pi) \le \max_{(x_1, \ldots, x_{n_0})} \sum_{i=1}^{n_0} \delta_i(S) x_i,$$

où la valeur maximale est prise sur tous les vecteurs $(x_1, \ldots, x_{n_0})$ qui satisfont

$$\sum_{i=1}^{n_0} i x_i = d \text{ et } \sum_{i=1}^{n_0} x_i \leq m.$$

Alors, nous avons

$$\begin{aligned}
\gamma \deg(\pi) - d &\leq \gamma \sum_{i=1}^{n_0} \delta_i(S) x_i - \sum_{i=1}^{n_0} i x_i \\
&\leq (\gamma - 1) n_0 x_{n_0} + \sum_{i=1}^{n_0 - 1} (\gamma \delta_i(S) - i) x_i \\
&\leq (\gamma - 1) n_0 \sum_{i=1}^{n_0} x_i - \sum_{i=1}^{n_0 - 1} ((\gamma - 1) n_0 - \gamma \delta_i(S) + i) x_i \\
&\leq (\gamma - 1) n - \sum_{i=1}^{n_0 - 1} ((\gamma - 1) n_0 - \gamma \delta_i(S) + i) x_i \\
&\leq (\gamma - 1) n \ ,
\end{aligned}$$

où la dernière inégalité provient du fait que tous les coefficients dans la somme sont positifs. En effet, nous avons

$$(\gamma - 1) n_0 - \gamma \delta_i + i = \gamma (n_0 - \delta_i) - (n_0 - i) \geq 0,$$

qui est vraie d'après la définition de γ. Ainsi, comme $\gamma \deg(\pi) - d \leq (\gamma - 1) n$, nous déduisons que

$$\gamma (n - \deg(\pi)) \geq n - d.$$

Nous allons maintenant montrer que si toutes les boîtes-S sont équilibrées, alors $\gamma \leq n_0 - 1$. En effet, pour tout $1 \leq i \leq n_0 - 1$, nous avons

$$\frac{n_0 - i}{n_0 - \delta_i} \leq \frac{n_0 - 1}{1},$$

puisque d'après le corollaire 4.1, le degré du produit de $n_0 - 1$ coordonnées ou moins, d'une boîte-S $n_0 \times n_0$ équilibrée ne peut pas être égal à n_0. Par conséquent, $\delta_i \leq n_0 - 1$. Nous pouvons aussi prouver que si les degrés de toutes les boîtes-S satisfont $\deg(S) \leq n_0 - 1$, alors $\gamma \leq n_0 - 2$, si $n_0 \geq 3$. En effet, pour $i = 1$, nous avons

$$\frac{n_0 - i}{n_0 - \delta_i} = \frac{n_0 - 1}{n_0 - \delta_1} \leq \frac{n_0 - 1}{2} \leq n_0 - 2$$

puisque $n_0 \geq 3$. De façon similaire, pour tout i, $2 \leq i < n_0$, grâce au corollaire 4.1 nous avons $\delta_i \leq n_0 - 1$ et donc nous déduisons que

$$\frac{n_0 - i}{n_0 - \delta_i} \leq n_0 - i \leq n_0 - 2 \ .$$

□

4.1.1 Comprendre la nouvelle borne

Le théorème 4.1 peut, à première vue, paraître compliqué. Cependant, l'idée qui se cache derrière est assez intuitive. Nous allons expliquer cette idée fondamentale à l'aide d'un exemple facile. En parallèle, cet exemple nous aidera à comprendre le rôle de la partie linéaire sur l'évolution du degré.

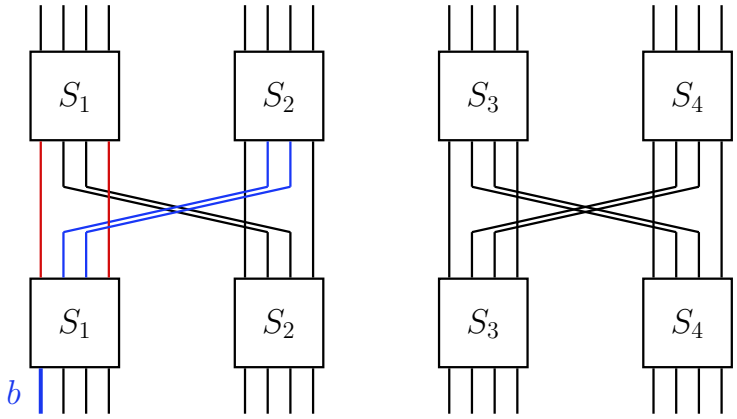

FIGURE 4.1 – La permutation P de $\mathbf{F}_2^{16}$ de type SPN, constituée de 2 tours

Nous considérons une permutation P de type SPN ayant une petite dimension et une description très simple, que nous pouvons visualiser à l'aide de la figure 4.1. Cette permutation est appliquée sur un état de 16 bits et est constituée de 2 itérations d'une permutation de tour R. La description de la fonction R est très simple. La partie linéaire est composée d'une simple permutation de bits tandis que la partie non-linéaire est constituée de l'application parallèle de quatre boîtes-S 4×4, S_1, S_2, S_3 et S_4. Nous considérons ici que $\deg(S_i)$ est égal à 3, pour tout $1 \leq i \leq 4$. Plusieurs constructions utilisent des boîtes-S ayant cette configuration. Les chiffrements par blocs, Serpent, Present, Piccolo et les fonctions de hachage Hamsi, *Luffa* et quelques instances de la famille Photon, en sont des exemples. Pour cet exemple, nous supposons plus exactement, que toutes les coordonnées de $S_1, \ldots, S_4$, sont de degré 3. En utilisant la définition 4.1 et le corollaire 4.1, nous déduisons alors que

$$\delta_1(S_i) = 3, \quad \delta_2(S_i) = 3, \quad \delta_3(S_i) = 3 \quad \text{et} \quad \delta_4(S_i) = 4, \qquad \text{pour } i = 1, 2, 3, 4.$$

Après l'application de la première couche de boîtes-S, comme toutes les coordonnées des S_i, $1 \leq i \leq 4$, sont cubiques, le degré de tous les bits de l'état est 3. Notre but est d'évaluer le degré de la permutation, $\deg(P)$, c'est-à-dire le degré de deux applications de la fonction de tour R.

En utilisant la borne triviale (proposition 3.3), comme $\deg(R) = 3$, nous en déduisons

$$\deg(P) = \deg(R \circ R) \leq \deg(R) \cdot \deg(R) \leq 3^2 = 9.$$

Néanmoins, nous allons voir que le degré de P est seulement 6. Regardons le degré du premier bit de la sortie de P, b. Comme l'application de la permutation sur l'état est symétrique, les mêmes remarques s'appliquent à tous les autres bits. Le bit b consiste en l'application d'un polynôme de degré 3 prenant en entrée un vecteur (x_0, x_1, x_2, x_3). Comme l'illustre la figure 4.1, x_0 et x_3 proviennent de S_1 tandis que x_1 et x_2 proviennent de S_2. Pour faciliter

la compréhension, nous supposons ici que l'ANF de la première coordonnée de S_1 est donnée par $x_0x_2x_3 + x_1x_3 + x_2 + 1$. Le degré de b est alors majoré par

$$\deg(b) \leq \delta_2(S_1) + \delta_1(S_2) = 3 + 3 = 6.$$

De tels calculs sont au cœur de la borne (4.1). Nous voyons alors ici que même pour un petit nombre de tours, la borne triviale peut donner des résultats assez éloignés de la situation réelle. Dans notre exemple, ceci vient du fait que la permutation linéaire que nous avons choisi est de très mauvaise qualité, puisque les bits sont très peu mélangés entre eux. En particulier, les huit premiers bits de sortie de la permutation, ne sont pas du tout affectés par les bits provenant de S_3 et S_4. Nous voyons alors que le choix de la permutation linéaire joue un rôle majeur sur l'évolution du degré de la permutation.

Pour mieux voir l'importance de la partie linéaire, nous allons maintenant donner l'exemple d'une deuxième permutation P', identique à P sauf la permutation linéaire que cette fois est choisie avec plus de soin. La permutation P' est décrite sur à la figure 4.2. Nous voyons en particulier que chaque boîte-S de la deuxième couche, est influencée par toutes les boîtes-S du premier tour.

Examinons le degré du bit b de la permutation P', exactement comme nous l'avons fait pour la permutation P. Cette fois, l'entrée (x_0, x_1, x_2, x_3) de S_1 au deuxième tour est telle que x_0 vient de S_1, x_1 de S_2, x_2 de S_3 et x_3 de S_4. En conséquence,

$$\deg(b) \leq \delta_1(S_1) + \delta_1(S_2) + \delta_1(S_3) = 3 + 3 + 3 = 9.$$

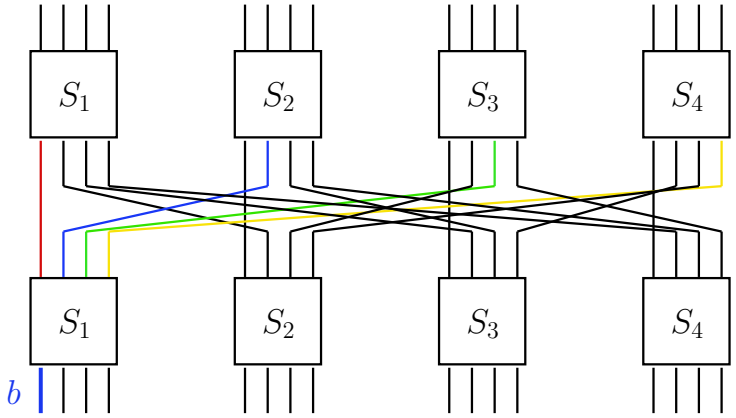

FIGURE 4.2 – La permutation P'

4.1.2 Lien entre la nouvelle borne et la borne triviale

Nous allons examiner ici la relation entre la nouvelle borne du théorème 4.1 et la borne triviale (proposition 3.3). Plus précisément, nous allons montrer que ces deux bornes sont d'une certaine façon symétriques. En effet, la borne triviale peut s'écrire

$$\frac{\deg(G \circ F)}{\deg G} \leq \max_{1 \leq i < n_0} \frac{\delta_i}{i}$$

et la borne triviale du théorème 4.1 s'écrit aussi

$$\frac{n - \deg(G \circ F)}{n - \deg G} \geq \left(\max_{1 \leq i < n_0} \frac{n_0 - i}{n_0 - \delta_i} \right)^{-1}.$$

En un sens, nous souhaitons représenter $\deg(G \circ F)$ en fonction de $\deg(G)$. L'illustration due à la borne triviale indique que le degré de $G \circ F$ est majoré par une droite qui passe par l'origine en ayant comme coefficient directeur $\deg F$. Au contraire, si nous représentons $(n - \deg(G \circ F))$ en fonction de $(n - \deg G)$, la nouvelle borne (4.1), indique que $n - \deg(G \circ F)$ est minoré par une ligne qui passe par l'origine en ayant comme coefficient γ^{-1}. Nous pouvons visualiser cela sur la figure 4.3, où les paramètres correspondent à l'inverse de la permutation KECCAK-f.

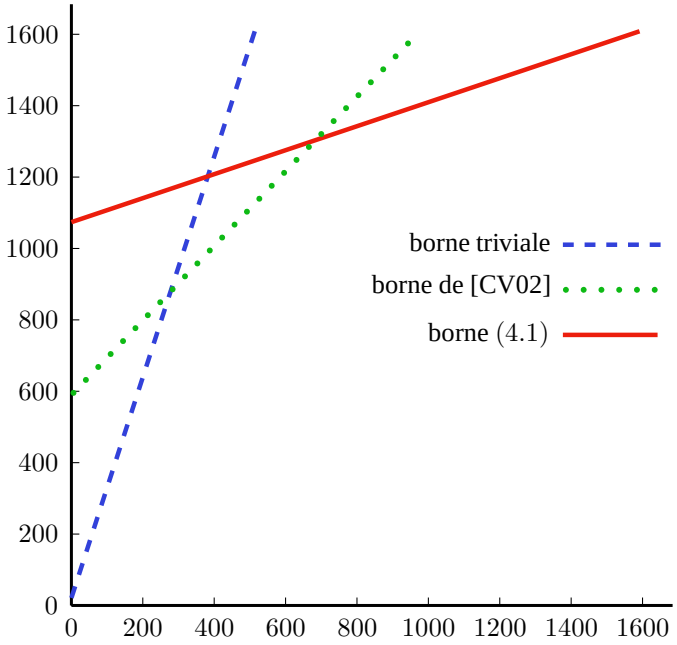

FIGURE 4.3 – Évolution du degré de $G \circ F$, où F est une fonction avec 1600 variables, composée de 320 permutations cubiques sur $\mathbf{F}_2^5$.

Nous allons ensuite voir des applications de la borne (4.1) sur certaines fonctions candidates à la compétition SHA-3. Nous commençons par montrer comment améliorer les partitions en sommes nulles que nous avons trouvées dans le chapitre 3 pour KECCAK-f.

4.2 Partitions en sommes nulles pour KECCAK-f

Nous appliquons le théorème 4.1 à la permutation de tour de KECCAK-f, qui est notée R. Puisque $n_0 = 5 \geq 3$ et que le degré de la permutation non-linéaire χ est 2, nous avons d'après la borne (4.2),

$$\deg(G \circ R) = \deg(G \circ \chi) \leq 1600 - \frac{1600 - \deg(G)}{3}, \tag{4.3}$$

pour toute fonction G. De la même façon, comme $\deg(\chi^{-1}) = 3$, nous avons pour toute fonction G,

$$\deg(G \circ R^{-1}) = \deg((G \circ L^{-1}) \circ \chi^{-1}) \leq 1600 - \frac{1600 - \deg(G)}{3}, \tag{4.4}$$

où $L = \pi \circ \rho \circ \theta$. En pratique, pour calculer une borne sur le degré de la permutation KECCAK-f après $r+1$ tours, nous prenons $G = R^r$ (respectivement $G = R^{-r}$ pour la permutation inverse).

Cependant, en 2011, Ming Duan et Xuejia Lai ont remarqué dans [DL11], qu'en multipliant deux à deux toutes les coordonnées de χ^{-1}, le degré de tous ces produits est 3. Suivant la notation introduite auparavant, cela signifie que $\delta_2(R^{-1}) = \delta_2(\chi^{-1}) = 3$. En utilisant cette remarque, nous calculons la quantité γ de l'équation (4.1) pour la fonction R^{-1}. Soit

$$\gamma_i = \frac{n_0 - i}{n_0 - \delta_i(\chi^{-1})}.$$

Nous avons que

$$\gamma_1(\chi^{-1}) = 2, \quad \gamma_2(\chi^{-1}) = 1.5, \quad \gamma_3(\chi^{-1}) = 2, \quad \text{et} \quad \gamma_4(\chi^{-1}) = 1.$$

En conséquence,

$$\gamma = \max_{1 \leq i \leq 4} \gamma_i = 2$$

et donc l'équation (4.4) devient maintenant

$$\deg(G \circ R^{-1}) \leq 1600 - \frac{1600 - \deg(G)}{2}. \tag{4.5}$$

À partir des bornes (4.3) et (4.5) nous établissons la table 4.1 sur le degré de la permutation KECCAK-f et de son inverse. Pour les premiers tours, les résultats sont obtenus à partir de la borne triviale, tandis que tous les résultats en gras sont dus à la nouvelle borne. Pour la permutation inverse, nous présentons à la deuxième colonne les résultats de la borne (4.4) et à la troisième ceux de la borne (4.5). Nous voyons à l'aide de cette table que grâce à la nouvelle borne, nous arrivons deux fois plus loin qu'avec la borne triviale. Plus précisément, nous voyons qu'au début le degré augmente d'un facteur d, où d est le degré de la permutation de tour, mais à partir d'un certain nombre de tours, à chaque tour le degré converge lentement vers le nombre de variables de la permutation.

Les auteurs de la fonction KECCAK ont comparé les résultats obtenus pour le degré de KECCAK-f[25] (version de KECCAK-f de taille de 25 bits) et de son inverse, avec les résultats provenant de la borne (4.1). Ils ont alors remarqué que les deux listes de valeurs étaient presque identiques, ce qui montre que notre borne est particulièrement pertinente dans ce cas.

Les nouvelles estimations pour le degré de KECCAK-f nous ont permis de trouver pour la première fois des partitions en sommes nulles pour la permutation entière du candidat SHA-3, défini au 2-ième tour de la compétition, c'est-à-dire avec 24 tours. Pour cela, nous avons utilisé la technique de la proposition 3.4, en considérant des états après la couche linéaire $L = \pi \circ \rho \circ \theta$ du 12^e tour. Ce découpage a été choisi en regardant la table 4.1 et l'équation (3.3). Plus précisément, nous choisissons un sous-espace V dans $\mathbf{F}_2^{1600}$ correspondant à une collection

en avant		en arrière		
# tours	borne $\deg(R^r)$	# tours	borne $\deg(R^{-r})$	borne $\deg(R^{-r})$ [DL11]
1	2	1	3	3
2	4	2	9	9
3	8	3	27	27
4	16	4	81	81
5	32	5	243	243
6	64	6	729	729
7	128	7	**1309**	**1164**
8	256	8	**1503**	**1382**
9	512	9	**1567**	**1491**
10	1024	10	**1589**	**1545**
11	**1408**	11	**1596**	**1572**
12	**1536**	12	**1598**	**1586**
13	**1578**	13	**1599**	**1593**
14	**1592**	14	**1599**	**1596**
15	**1597**	15	**1599**	**1598**
16	**1599**	16	**1599**	**1599**

TABLE 4.1 – Bornes supérieures sur le degré de plusieurs tours de KECCAK-f et de son inverse

de 315 lignes de l'état (sur un total de 320 lignes), de sorte que $\dim V = 1575$. Alors, les ensembles

$$X_a = \{(G_{11} \circ L^{-1})(a+z), z \in V\},\ a \in \mathbf{F}_2^{1600},$$

où G_{11} désigne l'inverse des onze premiers tours, forment une partition en sommes nulles de taille 2^{1575} pour la permutation entière de KECCAK.

Ces partitions distinguent les 24 tours de KECCAK-f d'une permutation aléatoire, mais n'impliquent pas un distingueur sur la fonction de hachage elle-même.

Nous montrons ensuite des applications de la borne (4.1) à une autre fonction candidate au concours SHA-3, à savoir *Luffa*.

4.3 Application à la fonction de hachage *Luffa*

Nous commençons par présenter les spécifications de la fonction *Luffa* [DSW08, DSW09], qui a été conçue par Christophe De Cannière, Hisayoshi Sato et Dai Watanabe pour le concours SHA-3. La fonction *Luffa* a été parmi les 14 candidats sélectionnés par le NIST pour participer au deuxième tour de la compétition. Les résultats que nous avons obtenus portent sur deux versions différentes de la fonction. Cela est du au fait que la version qui a été initialement soumise à la compétition en 2008 a subi des modifications un an plus tard à cause d'une attaque. Nous noterons la première version par *Luffa* v1 et la deuxième par *Luffa* v2.

Nous présenterons d'abord les spécifications de la fonction *Luffa* v1 et nous décrirons ensuite l'idée générale de l'attaque présentée sur cette version par Watanabe *et al.* [WHYK10]. Ensuite, nous analyserons nos résultats sur *Luffa* v1 et nous parlerons des changements apportés sur la fonction, qui ont conduit à la version *Luffa* v2. Finalement, nous présenterons nos résultats sur *Luffa* v2.

4.3.1 Spécifications de la fonction *Luffa* v1

Le mode opératoire de *Luffa* est une variante de la construction éponge (section 1.3.3). Son état interne est constitué de w mots de 256 bits, où w est égal à 3, 4 ou 5, pour les tailles d'empreintes 256, 384 et 512 bits respectivement. À chaque itération, un bloc de message m_i de 256 bits est introduit à l'aide d'une fonction d'injection de message linéaire, MI. Ensuite, une permutation P est appliquée à l'état. Plus précisément, l'état est divisé en w mots de 256 bits, et chaque mot j est transformé par une permutation Q_j, $0 \leq j < w$.

L'état interne de chaque application Q_j est divisé en 8 mots de 32 bits, $a_0, \ldots, a_7$. Chaque permutation est constituée d'un tweak, qui n'est appliqué qu'une seule fois au début de la permutation, et de 8 tours d'une permutation de tour, appelée `Step`. La fonction `Step` est elle-même composée d'une transformation non-linéaire, appelée `SubCrumb`, d'une transformation linéaire, appelée `MixWords` et d'une addition de constante, `AddConstant`. La fonction non-linéaire `SubCrumb` est constituée de 64 applications parallèles d'une permutation cubique 4×4.

À la fin, une fonction de finalisation C est appliquée. Si la taille du message est supérieure à la taille d'un bloc, alors cette fonction est composée de plusieurs tours à blanc, avec un message fixé à `0x0...00`, suivis d'une fonction linéaire OF. Dans le cas contraire, seulement la fonction OF est appliquée. La construction de *Luffa* est représentée à la figure 4.4.

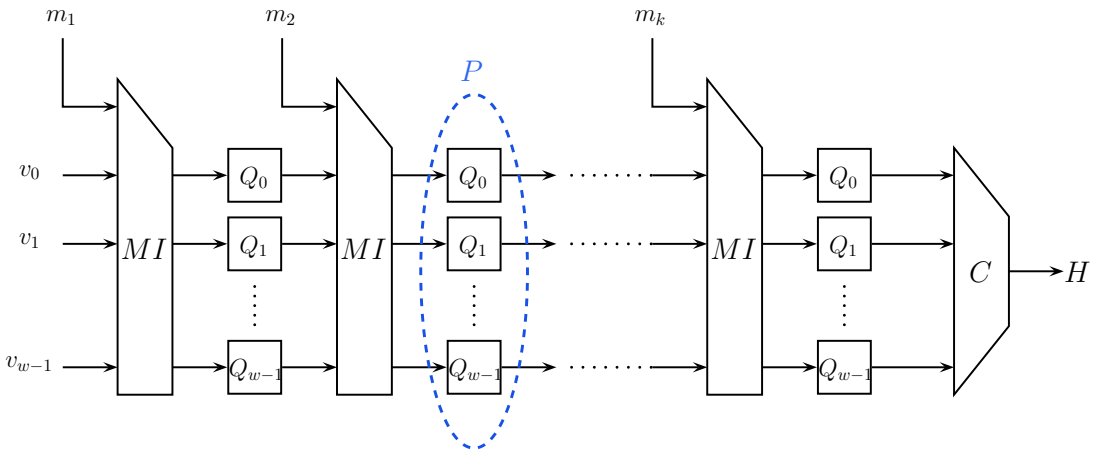

FIGURE 4.4 – La fonction de hachage *Luffa*

Nous décrivons maintenant les composantes principales de la fonction de hachage.

La permutation `SubCrumb`. L'entrée de chaque boîte-S est constituée de 4 bits, chaque bit venant d'un mot a_k différent. La boîte-S substitue les ℓ-ièmes bits, $0 \leq \ell < 32$, de a_0, a_1, a_2, a_3 (et de a_4, a_5, a_6, a_7) par une boîte-S 4×4 de degré 3. La boîte-S utilisée dans *Luffa* v1, est notée S_1 et est décrite par

$$S_1[16] = \{7, 13, 11, 10, 12, 4, 8, 3, 5, 15, 6, 0, 9, 1, 2, 14\}.$$

Watanabe *et al.* ont montré dans [WHYK10] que le degré de Q_j réduit à 5 tours (sur un total de 8) est au plus 130 et que la somme de deux premières coordonnées de Q_j après 6 tours est de degré au plus 214.

La substitution par la boîte-S dans *Luffa* v1 est donnée par

$$\begin{aligned} b_{3,\ell}||b_{2,\ell}||b_{1,\ell}||b_{0,\ell} &= S[a_{3,\ell}||a_{2,\ell}||a_{1,\ell}||a_{0,\ell}], \quad 0 \le \ell < 32, \\ b_{7,\ell}||b_{6,\ell}||b_{5,\ell}||b_{4,\ell} &= S[a_{7,\ell}||a_{6,\ell}||a_{5,\ell}||a_{4,\ell}], \quad 0 \le \ell < 32. \end{aligned}$$

La permutation `MixWord`. Cette fonction est une permutation linéaire de 2 mots de 32 bits. Si $z_0, \ldots, z_7$ sont les 8 mots de l'état après l'application de `Step`, nous avons que

$$\begin{aligned} (z_0, z_4) &= \texttt{MixWord}(b_0, b_4), \\ (z_1, z_5) &= \texttt{MixWord}(b_1, b_5), \\ (z_2, z_6) &= \texttt{MixWord}(b_2, b_6), \\ (z_3, z_7) &= \texttt{MixWord}(b_3, b_7). \end{aligned}$$

Un tour de la permutation `MixWord` peut être observé à la figure 4.5.

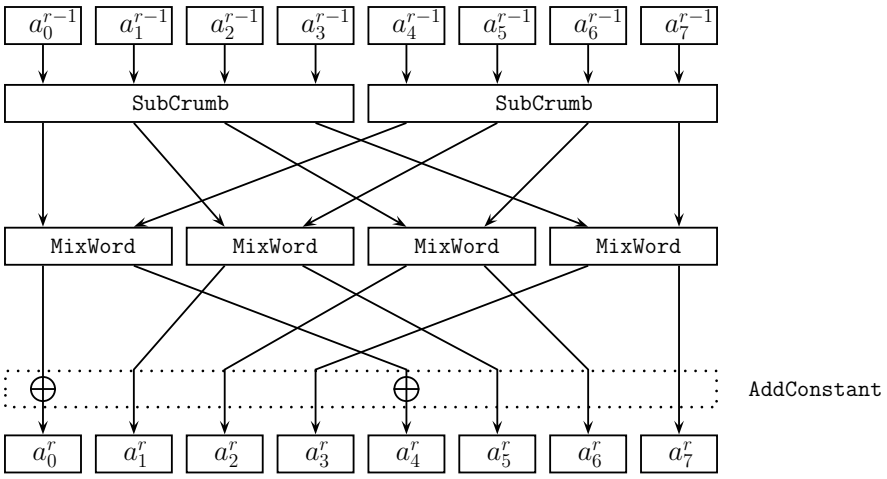

FIGURE 4.5 – La fonction `Step`

Tweak. Les quatre derniers mots de l'entrée $(b_{j,0}, b_{j,1}, \ldots, b_{j,7})$ de la permutation Q_j subissent une rotation de j bits vers la gauche :

$$\begin{aligned} a^0_{j,k,\ell} &= b_{j,k,\ell}, \quad 0 \le k < 4, \\ a^0_{j,k,\ell} &= b_{j,k,(\ell-j \mod 32)}, \quad 4 \le k < 8. \end{aligned}$$

Nous commençons notre analyse sur *Luffa* en décrivant l'attaque présentée dans [WHYK10] sur *Luffa* v1.

4.3.2 Description de l'attaque [WHYK10]

Watanabe *et al.* ont analysé dans [WHYK10] l'évolution du degré algébrique de *Luffa* v1. Ils ont remarqué qu'à cause d'une particularité de S_1, le degré n'augmente pas aussi vite que dans le cas idéal. Cette propriété vient du fait que les termes de degré 3 des trois

premières coordonnées de S_1 sont similaires. Cette propriété a été exploitée afin de montrer que le degré de Q_j n'augmente pas autant qu'imaginé. Cette observation a conduit à un distingueur différentiel d'ordre supérieur sur la fonction de hachage réduite à 7 tours. Suite à ces faiblesses, les auteurs ont modifié certaines composantes de la fonction et ont proposé une deuxième version de celle-ci, appelée *Luffa* v2.

Nous présentons ici les observations principales de ce travail. Nous allons montrer par la suite que des propriétés analogues existent également pour la version *Luffa* v2.

La description de S_1 par la forme algébrique normale de ses coordonnées est

$$\begin{aligned} y_0 &= 1 + x_2 + x_0x_1 + x_1x_3 + x_2x_3 + x_0x_1x_3 \\ y_1 &= 1 + x_0 + x_2 + x_0x_1 + x_0x_2 + x_3 + x_1x_3 + x_2x_3 + x_0x_1x_3 \\ y_2 &= 1 + x_1 + x_1x_3 + x_2x_3 + x_0x_1x_3 \\ y_3 &= x_0 + x_1 + x_2 + x_0x_1 + x_1x_2 + x_0x_1x_2 + x_1x_3 \end{aligned}$$

La remarque principale dans [WHYK10] est que dans y_0, y_1 et y_2 tous les termes non-linéaires comprenant x_3 sont égaux. Notons

$$p = x_1x_3 + x_2x_3 + x_0x_1x_3,$$

la partie commune des trois premières coordonnées, alors

$$\begin{aligned} y_0 &= p + 1 + x_2 + x_0x_1 \\ y_1 &= p + 1 + x_0 + x_2 + x_0x_1 + x_0x_2 + x_3 \\ y_2 &= p + 1 + x_1. \end{aligned}$$

Cela signifie en particulier que la somme deux à deux des éléments y_0, y_1 et y_2 est de degré 2.

Notons maintenant par q_i la valeur $y_i + p$, pour $i = 0, 1, 2$. En multipliant deux termes y_i, y_j parmi les y_0, y_1 et y_2 nous avons que

$$y_iy_j = (p + q_i)(p + q_j) = q_iq_j + (q_i + q_j + 1)p.$$

En conséquence,

$$\deg(y_iy_j) < \deg(y_i) + \deg(y_j),$$

puisque par définition $\deg(q_i) < \deg(y_i)$ pour $i = 1, 2, 3$.

Il est ensuite facile de voir que l'application de la fonction linéaire qui est constituée de quatre applications en parallèle de la fonction `MixWord`, préserve la propriété précédente. C'est-à-dire que les termes de degré élevé des trois premières coordonnées de toute boîte-S à l'entrée du tour suivant, seront les mêmes.

En utilisant ces remarques, les auteurs ont établi le degré maximal de récurrence sur le degré des mots de l'état (a_i^r) après r tours, ainsi que celui d'une autre quantité qui correspond à la somme des deux à deux des trois premières coordonnées de chaque boîte-S, notée par (d_i^r). Grâce à ces relations, ils ont pu estimer le degré de r tours de la permutation Q_j pour $1 \leq r \leq 5$. Ces résultats sont résumés à la table 4.2.

Les auteurs de [WHYK10] ont utilisé ces résultats pour attaquer la fonction de hachage *Luffa* v1, réduite à 7 tours. Dans une première étape, ils ont construit des distingueurs différentiels d'ordre supérieur pour 7 tours des permutations Q_j. Ils ont ensuite étendu ces distingueurs à la fonction de hachage, en montrant que la fonction d'insertion, les tweaks et la fonction de finalisation n'ont pas d'influence sur eux.

r	$\deg a_i^r$	$\deg d_i^r$
1	3	2
2	8	5
3	20	13
4	51	33
5	130	84
6	-	214

TABLE 4.2 – Bornes supérieures pour le degré algébrique de r itérations de la permutation `Step` de *Luffa* v1, ainsi que pour le degré de toute somme deux de trois premières coordonnées à l'entrée de chaque boîte-S, après r tours.

La partie de la sortie qui est utilisée comme distingueur est n'importe quel vecteur de 32 bits parmi les $(a_{i,k}^r + a_{i,k'}^r)$, pour $k, k' \in \{0,1,2\}, k \neq k'$ et $0 \leq i < 8$, après r tours. Comme nous pouvons le voir à la table 4.2, le degré de ces vecteurs est au plus 214 après 6 itérations de la fonction `Step`. Par conséquent, cette valeur peut être utilisée pour distinguer 6 tours de Q_j d'une permutation aléatoire. Plus précisément, il suffit de choisir des entrées dans un translaté d'un sous-espace V de dimension 215 et de calculer la différentielle d'ordre 215,

$$\sum_{v \in V} S(v+x),\ x \in \mathbf{F}_2^{256},$$

qui est nulle, où S est la fonction qui associe le vecteur de 32 bits $(a_{i,k}^6 + a_{i',k}^6)$ en sortie de 6 tours de a_j. Ceci est un événement qui ne devrait arriver qu'avec une probabilité négligeable pour une permutation aléatoire de $\mathbf{F}_2^{256}$.

Cependant, en choisissant avec attention l'espace de départ V, ce distingueur peut être étendu à 7 tours de la permutation. Pour cela, il suffit d'appliquer une technique similaire à celle que nous avons utilisée pour étendre les partitions en sommes nulles pour KECCAK à un tour supplémentaire. Plus précisément, il suffit de choisir des variables de façon à ce que l'entrée de chaque boîte-S du premier tour soit constituée de 4 variables, ou de 4 constantes. Si $a + V$ est un translaté d'un sous-espace V choisi de cette façon, alors il existe un b tel que

$$\texttt{SubCrumb}(a + V) = b + V.$$

Pour l'attaque contre *Luffa* v1, les auteurs ont choisi un sous-espace V de dimension 216, généré par les 27 premières variables de chaque mot :

$$V = \langle e_{i,\ell},\ 0 \leq i < 8, 0 \leq \ell < 27 \rangle.$$

Nous décrivons ensuite comment ce distingueur sur les permutations Q_j peut être étendu à la fonction de compression de *Luffa* v1 ainsi qu'à la fonction de hachage.

D'abord, nous voyons que, pour une valeur de chaînage fixée, la fonction linéaire d'insertion MI stabilise le sous-espace V. Ensuite, un tweak consiste en la rotation des quatre derniers mots de l'état de j bits vers la gauche. En conséquence, le nombre des boîtes-S actives et passives reste inchangé et les tweaks stabilisent alors eux aussi le sous-espace V. Enfin, comme un tour à blanc est appliqué par la fonction de finalisation seulement si le nombre de blocs est supérieur à 1, nous voyons que ce distingueur s'étend naturellement sur la fonction de hachage, quand le message ne fait qu'un seul bloc.

Nous commençons notre analyse en montrant que grâce au théorème 4.1, les résultats de la table 4.2 peuvent être étendus jusqu'à 8 tours de la permutation Q_j.

4.3.3 Différentielles d'ordre supérieur pour la fonction de hachage *Luffa* v1 entière

La fonction de tour `Step` est de type SPN ; elle est constituée d'une application linéaire et d'une couche non-linéaire composée d'une application parallèle de plusieurs boîtes-S 4×4 de degré 3. En conséquence, nous pouvons utiliser le théorème 4.1 pour étendre les résultats de la table 4.4 sur plusieurs tours. Pour $r \geq 6$, nous avons d'après le théorème 4.1 que pour toute fonction G,

$$\deg(G \circ \texttt{Step}) \leq 256 - \frac{256 - \deg(G)}{3} = \frac{512 + \deg(G)}{3}. \tag{4.6}$$

Pour tout $0 \leq i \leq 7$ et $0 \leq \ell \leq 31$ nous avons que $\deg(a^r_{i,\ell}) \leq \max_{i,\ell} \deg(a^{r-1}_{i,\ell} \circ \texttt{Step})$. De la même manière, $\deg(d^r_{i,\ell}) \leq \max_{i,\ell} \deg(d^{r-1}_{i,\ell} \circ \texttt{Step})$. Nous en déduisons alors que

$$\max_{i,\ell} \deg(a^r_{i,\ell}) \leq \frac{512 + \max_{i,\ell} \deg(a^{r-1}_{i,\ell})}{3} \text{ et } \max_{i,\ell} \deg(d^r_{i,\ell}) \leq \frac{512 + \max_{i,\ell} \deg(d^{r-1}_{i,\ell})}{3}. \tag{4.7}$$

Les nouvelles bornes sont exposées à la table 4.3.

r	$\deg a^r_{i,\ell}$	$\deg d^r_{i,\ell}$
1	3	2
2	8	5
3	20	13
4	51	33
5	130	84
6	214	198
7	242	236
8	251	249

TABLE 4.3 – Bornes supérieures sur le degré algébrique après r itérations de la fonction `Step` pour *Luffa* v1, obtenues en combinant les résultats de [WHYK10] (Table 4.2) et le théorème 4.1.

Nous voyons alors que grâce aux nouvelles estimations sur le degré des permutations Q_j, le distingueur présenté dans [WHYK10] peut être étendu à la version entière de *Luffa* v1. Plus précisément, nous voyons que le degré de certaines combinaisons linéaires de bits de sortie après 7 tours de Q_j est au plus 236. Par conséquent, nous obtenons un distingueur quand le bloc de message varie dans un sous-espace particulier de dimension 240.

En même temps, nous venons de montrer que le degré entier de la fonction de hachage, appliquée aux messages d'un bloc, est au plus 251 dans les 255 bits de message.

Suite à l'attaque de Watanabe *et. al*, les auteurs ont décidé de modifier certaines composantes de la fonction, bien que l'avantage que cette propriété exploitée pourrait offrir à un attaquant ne soit pas clair. Les raisons qui ont conduit à ces changements ainsi que leurs descriptions complètes sont données dans [DSW10]. Nous allons rapidement rappeler ici les modifications apportées à la fonction, qui a pris le nom *Luffa* v2.

4.3.4 Modifications sur la fonction *Luffa* v1

À cause des faiblesses découvertes dans [WHYK10], la boîte-S de *Luffa* v1, notée par S_1, a été remplacée par la permutation

$$S_2[16] = \{13, 14, 0, 1, 5, 10, 7, 6, 11, 3, 9, 12, 15, 8, 2, 4\}.$$

La forme algébrique normale de chacune de ses quatre coordonnées est

$$\begin{aligned}
y_0 &= 1 + x_0 + x_1 + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3 \\
y_1 &= x_0 + x_3 + x_0x_1 + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3 \\
y_2 &= 1 + x_1 + x_3 + x_0x_2 + x_1x_2 + x_1x_3 + x_2x_3 + x_0x_1x_2 + x_0x_1x_3 \\
y_3 &= 1 + x_1 + x_2 + x_0x_3 + x_0x_2 + x_1x_2 + x_1x_3 + x_2x_3 + x_0x_1x_2 + x_0x_1x_3
\end{aligned}$$

Ensuite, afin de briser les symétries dans la permutation Q_j, exploitées dans [WHYK10], l'ordre des entrées dans les 4 dernières boîtes-S a été modifié de la façon suivante :

$$\begin{aligned}
b_{3,\ell}||b_{2,\ell}||b_{1,\ell}||b_{0,\ell} &= S[a_{3,\ell}||a_{2,\ell}||a_{1,\ell}||a_{0,\ell}], \quad 0 \le \ell < 32, \\
b_{4,\ell}||b_{7,\ell}||b_{6,\ell}||b_{5,\ell} &= S[a_{4,\ell}||a_{7,\ell}||a_{6,\ell}||a_{5,\ell}], \quad 0 \le \ell < 32.
\end{aligned}$$

Le dernier changement porte sur la fonction de finalisation. Dans les spécifications de *Luffa* v2, un tour à blanc doit toujours être appliqué, quelque soit la taille du message.

4.3.5 Degré de la permutation Q_j et de son inverse pour *Luffa* v2

Nous montrons ici que malgré les changements décrits dans la section précédente, une attaque du même type que celle présentée à la section 4.3.2 s'applique à la version *Luffa* v2.

Nous commençons par démontrer que S_2 présente des faiblesses du même style que celles découvertes pour S_1. En effet, nous voyons que la somme des quatre coordonnées de S_2 est seulement de degré 2 :

$$d = y_0 + y_1 + y_2 + y_3 = 1 + x_1 + x_2 + x_0x_1 + x_0x_3. \tag{4.8}$$

Notons par $a_i^r = \left(a_{i,\ell}^r\right)_{0 \le \ell < 32}$ les mots de l'état après r itérations de la fonction `Step` et par $d_{0,\ell}^r$ (respectivement par $d_{4,\ell}^r$) la somme $a_{0,\ell}^r + a_{1,\ell}^r + a_{2,\ell}^r + a_{3,\ell}^r$ (respectivement $a_{4,\ell}^r + a_{5,\ell}^r + a_{6,\ell}^r + a_{7,\ell}^r$). Considérons maintenant la somme de deux monômes distincts de degré 3 en 4 variables, $x_i, x_j, x_k, x_{k'}$. Chaque deux monômes de ce type partagent deux variables. Notons d la somme des quatre variables, $d = x_i + x_j + x_k + x_{k'}$. Nous voyons que

$$\begin{aligned}
x_ix_jx_k + x_ix_jx_{k'} &= x_ix_jx_k + x_ix_j(x_i + x_j + x_k + d) \\
&= x_ix_jx_k + x_ix_j + x_ix_j + x_ix_jx_k + x_ix_jd \\
&= x_ix_jd.
\end{aligned}$$

Par conséquent, comme toutes les coordonnées de S_2 contiennent un nombre pair de monômes distincts de degré 3, le degré de leurs sorties, et donc le degré de la permutation Q_j après $r+1$ tours, est majoré par

$$\deg a_{i,\ell}^{r+1} \le 2 \max_{0 \le j \le 3} \deg a_{j,\ell}^r + \deg d_{0,\ell}^r, \quad \forall 0 \le i \le 3, \tag{4.9}$$

De plus, cette propriété est vraie quel que soit l'ordre des entrées et des sorties de la boîte-S. En conséquence,

$$\deg a_{i,\ell}^{r+1} \leq 2 \max_{4\leq j\leq 7} \deg a_{j,\ell}^{r} + \deg d_{4,\ell}^{r}, \quad \forall 4 \leq i \leq 7.$$

Ensuite, comme la couche linéaire est constituée de la même fonction `MixWord` appliquée aux paires (b_k, b_{k+4}) en parallèle, nous en déduisons

$$\begin{aligned} d_{0,\ell}^{r+1} &= a_{0,\ell}^{r+1} + a_{1,\ell}^{r+1} + a_{2,\ell}^{r+1} + a_{3,\ell}^{r+1} \\ &= \sum_{i=0}^{3} \texttt{MixWord}_{0,\ell}(b_i, b_{i+4}) \\ &= \texttt{MixWord}_{0,\ell}\left(\sum_{i=0}^{3} b_i, \sum_{i=0}^{3} b_{i+4}\right) \end{aligned}$$

et

$$d_{4,\ell}^{r+1} = \texttt{MixWord}_{1,\ell}\left(\sum_{i=0}^{3} b_i, \sum_{i=0}^{3} b_{i+4}\right).$$

Nous voyons alors que les degrés de $d_{0,\ell}^{r+1}$ et de $d_{4,\ell}^{r+1}$ correspondent aux degrés de la somme de quatre coordonnées des boîtes-S. Alors, d'après l'équation (4.8) nous avons

$$\deg d_{i,\ell}^{r+1} \leq 2 \max_{i\leq j\leq i+3} \deg a_{j,\ell}^{r}, i \in \{0, 4\}. \tag{4.10}$$

Nous utilisons maintenant les relations récursives (4.9) et (4.10), pour trouver des bornes supérieures pour le degré de la permutation Q_j de *Luffa* v2, en suivant le principe de [WHYK10] pour *Luffa* v1. Ces bornes sont présentées dans la table 4.4.

r	$\deg a_{i,\ell}^{r}$	$d_{i,\ell}^{r}$
1	3	2
2	8	6
3	22	16
4	60	44
5	164	120

TABLE 4.4 – Bornes supérieures pour le degré algébrique de r itérations de la permutation `Step` de *Luffa* v2.

Nous appliquons ensuite le théorème 4.1 à la permutation S_2 en nous appuyant sur les résultats de la table 4.4. Pour cela, nous utilisons les bornes (4.6) et (4.7). Les nouvelles bornes sont exposées à la table 4.5.

Il convient de noter que les mêmes bornes s'appliquent sur le degré après r itérations de la fonction inverse de `Step` dans *Luffa* v2, puisque la forme algébrique normale de l'inverse de S_2 est

$$\begin{aligned} y_0 &= x_0 + x_2 + x_3 + x_2x_3 \\ y_1 &= 1 + x_3 + x_0x_1 + x_0x_2 + x_0x_3 + x_2x_3 + x_0x_2x_3 + x_1x_2x_3 \\ y_2 &= x_1 + x_2 + x_3 + x_0x_1 + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 \\ y_3 &= x_1 + x_2 + x_3 + x_0x_2 + x_2x_3 + x_0x_1x_2 + x_0x_1x_3. \end{aligned}$$

r	$\deg x_{i,\ell}^r$	$\deg d_{i,\ell}^r$
1	3	2
2	8	6
3	22	16
4	60	44
5	164	120
6	225	210
7	245	240
8	252	250

TABLE 4.5 – Bornes supérieures sur le degré algébrique après r itérations de la fonction `Step` pour *Luffa* v2, en combinant les résultats de la table 4.4 et le théorème 4.1.

La somme des quatre coordonnées de S_2^{-1} est égale à

$$1 + x_0 + x_2 + x_1x_2 + x_1x_3 + x_2x_3,$$

qui est alors aussi de degré 2. En outre, chacune des quatre coordonnées de S_2^{-1} a un nombre pair de monômes de degré 3. La technique détaillée auparavant peut alors être appliquée de la même façon à la fonction inverse.

4.3.6 Partitions en sommes nulles pour la permutation Q_j de *Luffa* v2

Aumasson et Meier ont présenté dans [AM09] les premières partitions en sommes nulles pour la permutation Q_j de *Luffa*. Comme pour les structures de somme nulle pour KECCAK, introduites dans le même article, les auteurs ont utilisé la borne triviale, afin d'estimer l'évolution du degré. Puisque la permutation de tour est de degré 3, ils ont calculé qu'après 4 tours le degré de la permutation, ainsi que de son inverse, sera au plus $3^4 = 81$. Cela a permis de trouver des partitions en sommes nulles pour la permutation Q_j de taille 2^{82} en choisissant un translaté d'un sous-espace V de dimension 82 dans un état intermédiaire après 4 tours.

Nous allons montrer ici comment trouver des partitions en sommes nulles avec une complexité beaucoup plus faible que celle de [AM09]. Pour cela, il suffit d'une part de choisir attentivement le sous-espace de départ et d'autre part d'exploiter les nouvelles bornes sur le degré.

Nous considérons un sous-espace V qui est généré par les 23 premiers bits du mot k de l'état, $0 \leq k < 8$:

$$V = \langle e_{k,0}, e_{k,1}, \ldots, e_{k,22} \rangle.$$

Nous allons montrer que les ensembles

$$X_a = \{\texttt{Tweak}_j^{-1} \circ (\texttt{Step}^{-1})^4(a + z), z \in V\}, a \in \mathbf{F}_2^{256},$$

forment une partition en sommes nulles de taille 2^{23} pour chaque Q_j.

Commençons par le calcul en avant. Chacune des 23 variables du translaté $V + a$ entrera dans une boîte-S différente. Par conséquent, le premier tour sera linéaire. Comme 3 itérations de la fonction `Step` sont de degré au plus 22 (table 4.4), nous en déduisons que

$$\sum_{x \in X_a} Q_j(x) = 0.$$

Nous nous concentrons maintenant sur le calcul en arrière. Examinons d'abord les images de V par l'inverse de la permutation `MixWord`. Comme toutes les variables se trouvent dans un seul mot k, après l'application de l'inverse de `MixWord`, tous les mots de l'état sauf les mots k et $(k+4) \mod 8$, seront constants. Mais les bits de ces deux mots entreront ensuite tous dans des boîtes-S différentes, ce qui implique que le premier tour en arrière sera également linéaire. Comme nous l'avons déjà prouvé, le degré de trois itérations de $\texttt{Step}^{-1}$ est au plus 22. En conséquence,

$$\sum_{x \in X_a} x = 0.$$

Au total, il existe $\binom{32}{23} \times 8$ partitions de ce type pour Q_j, correspondant au choix du mot k et au choix des 23 positions actives dans le mot.

4.3.7 Différentielles d'ordre supérieur pour la fonction de compression de *Luffa* v2

La fonction de compression dans *Luffa* v2 prend en entrée une valeur de chaînage de $256w$ bits et un bloc de message de 256 bits et donne en sortie une nouvelle valeur de chaînage de $256w$ bits, où $w = 3, 4$ et 5 pour une taille d'empreinte de $256, 384$ et 512 bits respectivement. Nous avons prouvé que cette fonction est de degré au plus 252, alors qu'une telle construction devrait idéalement avoir un degré de 255.

Une première conséquence de cette remarque est l'existence des différentielles d'ordre supérieur en sommes nulles, comme celles trouvées dans [WHYK10] pour *Luffa* v1 réduite à 7 tours, et rappelées à la section 4.3.2.

Soit ℓ_0 une position parmi les 32 positions possibles dans un mot, $0 \leq \ell_0 < 32$. Nous considérons un translaté d'un sous-espace linéaire V, défini de la façon suivante :

$$V = \langle e_{i,\ell}, 0 \leq i < 8, \ell \neq \ell_0 \rangle.$$

La dimension de V est 248. Pour chaque valeur de chaînage fixée, la fonction d'insertion de message MI stabilise V, impliquant que l'entrée de chaque permutation Q_j est un translaté de celui-ci. Après l'application des tweaks à l'entrée de chaque permutation Q_j, un translaté de l'espace V se transforme à un translaté d'un espace V' qui correspond à la somme directe des sous-espaces de la forme $\langle e_{i,\ell}, 0 \leq i < 4 \rangle$ et $\langle e_{i,\ell}, 4 \leq i < 8 \rangle$. Puisque la fonction `SubCrumb` s'applique indépendamment aux quatre premiers mots et aux quatre derniers mots de l'entrée, la structure de V' est stabilisée. Par conséquent, les sorties de la première itération de la fonction `Step` dans chaque Q_j décrivent un translaté d'un sous-espace de dimension 248. Donc la somme des images correspondant par la fonction de compression est nulle quand le message décrit un translaté quelconque de V, puisque $\dim V = 248 > \deg(\texttt{Step}) = 245$.

Cette observation est valide pour toute taille d'empreinte.

4.3.8 Degré algébrique de la fonction *Luffa* v2 avec valeurs initiales choisies

L'observation précédente ne s'applique pas à la fonction *Luffa* v2, puisqu'un tour à blanc est appliqué aux messages quelle que soit leur longueur. Cependant, si nous considérons les 256 bits de la valeur initiale de *Luffa* v2 comme une entrée supplémentaire que nous pouvons choisir librement, nous pouvons alors faire quelques observations théoriques sur la fonction de hachage appliquée aux messages d'un seul bloc.

Avec cette configuration, *Luffa* est une fonction prenant en entrée $256(w+1)-1$ bits et donnant en sortie $128(w-1)$ bits, où les bits de l'entrée correspondent aux bits de la valeur initiale et aux bits du message. Mais *Luffa* v2 est composée d'une fonction linéaire d'injection de message, suivie par une fonction G de $256w$ bits dans $128(w-1)$ bits. Par conséquent, le degré de la fonction de hachage est égal au degré de G et ne peut pas dépasser $256w$. De plus, nous montrons que le degré de cette fonction est même inférieur à $256w$ à cause de la structure particulière de la permutation interne.

Cette nouvelle borne supérieure sur le degré de *Luffa* v2 vient du fait que la fonction G peut être décomposée en la permutation P, c'est-à-dire en l'application parallèle de w permutations non-linéaires Q_j en n_0 variables de degré au plus (n_0-2), suivie de quelques tours de la fonction de finalisation, C. Les premiers 256 bits du haché sont extraits après une seule application de C. En utilisant alors que la fonction de finalisation consiste en 8 itérations de la fonction `Step` et a un degré au plus 252, nous avons d'après le théorème 4.1

$$\begin{aligned}\deg(C\circ P) &\leq 256w - \frac{256w-\deg C}{254}\\ &\leq 256w - \frac{256w-252}{254}\\ &< 256w-(w-1).\end{aligned}$$

Pour la version $(128(w-1))$-bits de `Luffa` v2, nous obtenons que les 256 premiers bits du haché de *Luffa* v2 sont de degré au plus $(256w-w)$. La pertinence de cette propriété est alors estimée par la probabilité qu'elle soit vraie pour une fonction de $\mathbf{F}_2^{256(w+1)-1}$ dans $\mathbf{F}_2^{256}$, choisie aléatoirement. Une telle fonction peut être écrite comme 256 polynômes aux coefficients dans $\mathbf{F}_2$. Le nombre de ses monômes de degré supérieur au égal à $256w-w+1$ est

$$\sum_{i=0}^{256+w-1}\binom{256(w+1)-1}{i}.$$

Par conséquent, la probabilité qu'une fonction choisie aléatoirement et ayant les mêmes paramètres que *Luffa* v2-256 soit de degré au plus 765, est $2^{-2^{837}}$. Les degrés calculés pour toutes les versions ainsi que les probabilités respectives sont résumés à la table 4.6.

Taille du haché	Nombre de variables	Borne sur le degré	Probabilité
256 $(w=3)$	1024	765	$2^{-2^{837}}$
384 $(w=4)$	1280	1020	$2^{-2^{933}}$
512 $(w=5)$	1536	1275	$2^{-2^{1010}}$

TABLE 4.6 – Bornes sur le degré algébrique de toutes les versions de *Luffa* v2 avec les probabilités respectives qu'une fonction avec les mêmes paramètres choisie aléatoirement aie le même degré.

4.4 Application à l'AES

Une application simple du théorème 4.1 consiste à estimer l'évolution du degré algébrique de l'AES en fonction du nombre de tours. Dans le cas des chiffrements par blocs, les bits de la clé sont considérés comme des constantes, et n'ont donc aucune influence sur le degré.

Les résultats que nous avons obtenus pour l'AES sont assez intéressants. Nous avons par exemple prouvé que le degré de deux itérations de la fonction de tour, qui était généralement majoré par 49, est au plus 28.

Pour cela, nous avons employé la technique appelée *technique de la super-Boîte-S* ("Super Sbox technique" en anglais), qui a été introduite par Joan Daemen et Vincent Rijmen pour analyser les différentielles sur deux tours de l'AES [DR06]. Cette technique consiste à unifier deux tours du chiffrement. Nous décomposons ici deux tours de l'AES :

$$\begin{aligned} R^2 \;=\; & \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes} \circ \texttt{AddRoundKey} \circ \\ & \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes} \circ \texttt{AddRoundKey}. \end{aligned}$$

Le résultat de ces transformations est équivalent à

$$\begin{aligned} R^2 \;=\; & \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes} \circ \texttt{AddRoundKey} \circ \\ & \texttt{MixColumns} \circ \texttt{SubBytes} \circ \texttt{ShiftRows} \circ \texttt{AddRoundKey}, \end{aligned}$$

puisque deux transformations adjacentes `ShiftRows` et `MixColumns` commuent. En notant

$$\texttt{SuperSbox} = \texttt{SubBytes} \circ \texttt{AddRoundKey} \circ \texttt{MixColumns} \circ \texttt{SubBytes},$$

nous avons que deux tours s'écrivent comme

$$R^2 = \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SuperSbox} \circ \texttt{ShiftRows} \circ \texttt{AddRoundKey},$$

où la super-Boîte-S `SuperSbox` est une permutation non-linéaire de $\mathbf{F}_2^{32}$. Deux tours de l'AES peuvent alors êtres vus comme une structure de type SPN, où la partie non-linéaire est constituée de 4 applications parallèles de la permutation `SuperSbox`. Nous commençons par calculer une borne sur le degré de `SuperSbox`, et en conséquence sur le degré de deux tours de l'AES. La `SuperSbox` est composée de deux couches non-linéaires de boîtes-S de degré 7, séparées par une couche de diffusion linéaire. Nous avons alors que

$$\deg(\texttt{SuperSbox}) \leq 32 - \frac{32-7}{7} \leq 28.$$

Pour r tours de chiffrement, nous obtenons d'après le théorème 4.1 la borne suivante :

$$\deg(R^r) = \deg(R^{r-1} \circ R) \leq 128 - \frac{128 - \deg(R^{r-1})}{7}.$$

Nous appliquons ensuite cette borne sur 3 et 4 tours de l'AES. Les résultats obtenus, ainsi que les résultats donnés par la borne triviale sont résumés dans la table 4.7. Un '−' dans la table signifie que la borne obtenu correspond au degré maximal de la permutation.

4.5 Application à la permutation interne de Grøstl

Grøstl est une famille de fonctions de hachage, conçue par Gauravaram *et al.* [GKM+08] pour la compétition SHA-3. Ses instances sont construites selon le mode wide-pipe (section 1.3.3) et sont donc basées sur une fonction de compression h de grande taille. Elles peuvent produire des empreintes de taille variant de 1 jusqu'à 64 octets. L'instance retournant des hachés de taille n est notée Grøstl-n.

# Tours r	Borne triviale	Borne (4.1)
1	7	7
2	49	28
3	–	113
4	–	125

TABLE 4.7 – Bornes supérieures pour le degré algébrique de plusieurs tours de l'AES

Pour chaque variante, la fonction de compression est itérée comme suit. Le message est d'abord rembourré et divisé en k blocs de ℓ bits, $m_1, \ldots, m_k$. La valeur de ℓ est égale à 512 pour les variantes retournant jusqu'à 256 bits et vaut 1024 pour les autres versions. Ensuite, étant donnée une valeur initiale $IV = h_0$, chaque bloc de message est traité par la fonction de compression h :

$$\begin{array}{rcl} h : \mathbf{F}_2^\ell \times \mathbf{F}_2^\ell & \to & \mathbf{F}_2^\ell \\ (h_{i-1}, m_i) & \mapsto & h_i \end{array}$$

Une transformation finale Ω est finalement appliquée à h_k. La fonction de compression h est construite à partir de deux grandes permutations P et Q. Elle est donnée par

$$h(h_{i-1}, m_i) = P(h_{i-1} \oplus m_i) \oplus Q(m_i) \oplus h_{i-1},$$

et peut être visualisée à la figure 4.6.

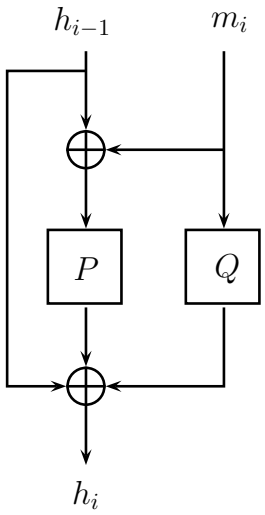

FIGURE 4.6 – La fonction de compression de Grøstl

Les fonctions P et Q sont des permutations itérées basées sur l'AES. Elles opèrent sur un état de 512 bits pour les candidats retournant des empreintes de taille au plus 256 bits et sur un état de 1024 bits pour les candidats restant. Ces deux états sont représentés par une matrice d'octets $8 \times c$ avec $c = 8$ ou 16. À chaque tour R, quatre transformations inspirées de l'AES, sont appliqués à l'état comme suit :

$$R = \texttt{MixBytes} \circ \texttt{ShiftBytes} \circ \texttt{SubBytes} \circ \texttt{AddRoundConstant}.$$

La transformation `AddRoundConstant` ajoute une constante dépendant du tour, à chaque octet de l'état. Ces constantes sont différentes pour les permutations P et Q.

La transformation `SubBytes` consiste en l'application en parallèle de la boîte-S de l'AES à chaque octet de l'état.

La transformation `ShiftRows` effectue une rotation vers la gauche sur chaque ligne de l'état. Cette transformation est différente pour P et Q.

La transformation `MixBytes` est une transformation linéaire qui s'applique en parallèle sur les colonnes de l'état.

Le nombre de tours est spécifié à 10 pour Grøstl-256 et à 14 pour Grøstl-512.

4.5.1 Borne sur le degré de la permutation P de Grøstl-256

Nous présentons ici encore une application de la borne (4.1), cette fois à la permutation P (également à Q) de Grøstl. Nous allons voir que grâce à cette borne nous pouvons efficacement majorer le degré de la permutation P jusqu'à 6 tours, tandis que avec la borne triviale, cela n'était possible que jusqu'à 3 tours.

Puisque la transformation non-linéaire de Grøstl utilise la boîte-S de l'AES, un tour R de la permutation est de degré 7. En appliquant le théorème 4.1 nous avons

$$\deg(F \circ R) \leq 512 - \frac{512 - \deg(F)}{7}$$

et

$$\deg(F \circ R^{-1}) \leq 512 - \frac{512 - \deg(F)}{7}.$$

En combinant ces résultats avec la borne triviale et en les appliquant de façon récursive, nous obtenons les bornes décrites à la table 4.8 pour le degré de $\deg(R^r)$. Les résultats en gras sont les bornes obtenues avec le théorème 4.1.

# Tours r	Borne sur $\deg(R^r)$
1	7
2	49
3	343
4	**487**
5	**508**
6	**511**

TABLE 4.8 – Bornes supérieures sur le degré algébrique de plusieurs tours de P et Q

Certainement au cas de l'AES, nous voyons que nous ne trouvons pas de résultats meilleurs pour 2 tours de la permutation, que ceux donnés par la borne triviale. Ceci est dû au fait que l'état de Grøstl est deux fois plus large, et par conséquent, les entrées de chaque boîte-S du deuxième tour peuvent provenir de 8 boîtes différentes du premier tour.

En application de ces résultats, nous pouvons construire des partitions en sommes nulles pour les permutations P et Q de Grøstl-256. Comme les seules différences entre P et Q sont dans l'addition des constantes et les décalages par la fonction `ShiftBytes`, la même

technique peut être utilisée pour les deux permutations. Choisissons un sous-espace V de $\mathbf{F}_2^{512}$ de dimension 509 après 5 tours de la permutation. Alors, les ensembles

$$X_a = \{(R^{-5}(a+z), z \in V\}, a \in \mathbf{F}_2^{512},$$

forment une partition en sommes nulles de taille 2^{509} pour les permutations de Grøstl-256.

Chapitre 5

Influence du degré algébrique de F^{-1} sur le degré de $G \circ F$

Sommaire

5.1 Une première borne impliquant le degré de la permutation inverse 102

5.2 Résultat principal 102

5.3 Quelques corollaires 104

5.4 Généralisation aux fonctions équilibrées non-injectives 106

5.5 Applications à certaines primitives symétriques 110

5.5.1 Attaque contre une variante du chiffrement par blocs $\mathcal{KN}$ 110

5.5.2 Application à Rijndael-256 114

5.5.3 Application à la fonction de hachage ECHO 114

5.5.4 Application à la fonction de hachage JH 116

Dans le chapitre précédent nous avons étudié le degré des permutations itérées dont la partie non-linéaire de la fonction de tour était constituée de plusieurs fonctions plus petites équilibrées. Ici, nous continuons d'étudier l'évolution du degré des constructions itérées, mais dans un contexte plus général. Notre résultat principal montre que le degré algébrique après plusieurs itérations dépend de l'inverse de la permutation qui est itérée. Ce résultat peut expliquer des comportements remarqués auparavant sur certaines fonctions dont nous ignorions les causes. Plus intéressant encore, nous allons voir que le degré de la fonction inverse influence le degré de la fonction itérée, même dans des situations où la fonction inverse n'est jamais utilisée dans la pratique. C'est le cas par exemple des chiffrements de Feistel ou des fonctions de hachage. Même si le degré de la fonction de tour est élevé, si le degré de la fonction inverse est bas, alors le degré du chiffrement ou de la fonction de hachage sera moins élevé que prévu.

La totalité des résultats de ce chapitre sont les fruits d'un travail commun avec Anne Canteaut. Ce travail a été accepté pour publication dans le journal *IEEE Transactions on Information Theory* [BC12b].

5.1 Une première borne impliquant le degré de la permutation inverse

Avant de présenter le résultat principal, nous allons montrer que nous pouvons déduire de la proposition 3.2 une borne sur le degré de $G \circ F$ qui implique le degré de F^{-1}. Nous rappelons que le résultat de Canteaut et Videau [CV02] présentait une borne sur le degré des permutations itérées dans le cas où le spectre de Walsh de la fonction itérée F était divisible par une puissance de 2.

De cette proposition, nous déduisons le corollaire suivant.

Corollaire 5.1. *Soit F une permutation de $\mathbf{F}_2^n$ et soit G une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors, nous avons*

$$\deg(G \circ F) \leq n - 1 - \left\lceil \frac{n-1}{\min(\deg F, \deg F^{-1})} \right\rceil + \deg G \, .$$

Démonstration. D'après la proposition 2.2, l'ensemble des coefficients de Walsh d'une permutation et de son inverse sont égaux. En outre, une borne inférieure sur la plus grande puissance de 2 qui divise les coefficients de Walsh d'une fonction booléenne peut être dérivée du théorème de Katz [Kat71]. Ce théorème énonce que, pour toute fonction F et pour tout $b \in \mathbf{F}_2^n$, nous avons

$$\sum_{x \in \mathbf{F}_2^n} (-1)^{b \cdot F(x) + a \cdot x} \equiv \sum_{x \in \mathbf{F}_2^n} (-1)^{b \cdot F(x)} \pmod{2^{\lceil \frac{n-1}{\deg F} \rceil + 1}} \, .$$

Puisque F est une permutation, toute combinaison linéaire non-triviale de ses coordonnées est équilibrée. Cela signifie que la partie droite de l'équation précédente s'annule. En conséquence, en appliquant ce résultat à F et à F^{-1} nous avons que les coefficients de Walsh de F sont divisibles par 2^ℓ avec

$$\ell \geq 1 + \left\lceil \frac{n-1}{\min(\deg F, \deg F^{-1})} \right\rceil \, .$$

□

En particulier, si F^{-1} est une fonction quadratique, nous avons d'après le corollaire 5.1 que

$$\deg(G \circ F) \leq \left\lfloor \frac{n-1}{2} \right\rfloor + \deg G \, .$$

Dans le cas où $\deg G \leq \lceil \frac{n-1}{2} \rceil$, cette borne peut être utilisée pour améliorer la borne triviale.

5.2 Résultat principal

Dans le chapitre précédent nous avons examiné le degré algébrique de la fonction Keccak. Nous rappelons ici, que la boîte-S, notée χ, utilisée dans la permutation Keccak-f, est une permutation de $\mathbf{F}_2^5$ de degré 2 ayant une inverse χ^{-1} de degré 3. Dans [DL11], Ming et Lai ont remarqué qu'en multipliant 2 coordonnées quelconques de χ^{-1}, le degré de ce produit était seulement 3. En utilisant la notation introduite dans la définition 4.1, ceci signifie que $\delta_2(\chi^{-1}) = 3$.

Nous montrons ici que ce résultat n'est pas dû au hasard, mais qu'il provient du fait que le degré algébrique de la permutation χ est faible. Plus précisément, nous démontrons le théorème suivant.

Théorème 5.1. *Soit F une permutation de $\mathbf{F}_2^n$. Alors, pour deux entiers k et ℓ nous avons*

$$\delta_\ell(F^{-1}) < n-k \text{ si et seulement si } \delta_k(F) < n-\ell. \tag{5.1}$$

Démonstration. Il suffit de montrer que si $\delta_\ell(F^{-1}) < n-k$, alors $\delta_k(F) < n-\ell$. La relation réciproque se déduit alors en échangeant les rôles de F et F^{-1}.

Soit π le produit de k coordonnés de F, $\pi : x \mapsto \prod_{i\in K} F_i(x)$, avec $|K| \leq k$. Nous allons prouver que le coefficient de tout monôme de π de degré supérieur ou égal à $n-\ell$ est nul. Pour un ensemble $L \subset \{1,\ldots,n\}$, avec $|L| \leq \ell$, nous notons a_L le coefficient du monôme $\prod_{j\notin L} x_j$ de degré $n-|L|$. Nous allons alors montrer que $a_L = 0$.

$$\begin{aligned} a_L &= \sum_{\substack{x\in\mathbf{F}_2^n \\ x_j=0, j\in L}} \pi(x) \\ &= \#\{x \in \mathbf{F}_2^n : x_j = 0, j \in L \text{ et } F_i(x) = 1, i \in K\} \mod 2 \\ &= \#\{y \in \mathbf{F}_2^n : y_i = 1, i \in K \text{ et } F_j^{-1}(y) = 0, j \in L\} \mod 2\,, \end{aligned}$$

où la dernière égalité vient du fait que F est une permutation, impliquant une correspondance biunivoque entre x et $y = F(x)$. En outre, $F_j^{-1} = 0$ pour tout $j \in L$ si et seulement si $\prod_{j\in L}(1 + F_j^{-1}(y)) = 1$. En conséquence,

$$a_L = \#\{y \in \mathbf{F}_2^n : y_i = 1, i \in K \text{ et } \prod_{j\in L}(1 + F_j^{-1}(y)) = 1\} \bmod 2\,. \tag{5.2}$$

Nous définissons maintenant la fonction booléenne suivante :

$$\begin{array}{rccc} H_{K,L}: & \{x \in \mathbf{F}_2^n : x_i = 1, i \in K\} & \to & \mathbf{F}_2 \\ & x & \mapsto & \prod_{i\in L}(1 + F_i^{-1}(x))\,. \end{array}$$

Nous avons

$$a_L = wt(H_{K,L}) \bmod 2\,.$$

$H_{K,L}$ est une fonction de $n-k$ variables de degré au plus $\delta_\ell(F^{-1})$. Par hypothèse, $\delta_\ell(F^{-1}) < n-k$ donc $H_{K,L}$ est de poids de Hamming pair. Nous en déduisons que $a_L = 0$. Par conséquent, $\delta_k(F) < n-\ell$. □

Ce théorème explique alors le comportement observé sur KECCAK-f. En effet, puisque $\delta_1(\chi) = \deg(\chi) = 2$, nous avons d'après le théorème que $\delta_2(\chi^{-1}) < 4$.

Nous présentons ensuite une borne sur le degré des permutations itérées, qui est déduite du théorème 5.1 et qui implique le degré de la permutation inverse. Cette borne a la même forme que la borne du théorème 4.1. Cependant, il faut noter que ce résultat, dont l'utilisation est sûrement plus pratique, est toutefois moins précis que le théorème 5.1.

Corollaire 5.2. *Soit F une permutation de $\mathbf{F}_2^n$ et soit G une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors nous avons,*

$$\deg(G \circ F) < n - \left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor\,.$$

Démonstration. Bien entendu, $\deg(G \circ F) \leq \delta_{\deg G}(F)$. Mais d'après le théorème 5.1 nous en déduisons que $\delta_{\deg G}(F) < n - \ell$ pour un entier ℓ si et seulement si $\delta_\ell(F^{-1}) < n - \deg G$. Cependant, en utilisant la borne triviale nous voyons que $\delta_\ell(F^{-1}) \leq \ell \deg(F^{-1})$. Il s'ensuit que $\delta_\ell(F^{-1}) < n - \deg G$ pour tout entier ℓ satisfaisant

$$\ell \leq \left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor .$$

En effet,

$$\left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor = \begin{cases} \left\lfloor \frac{n-\deg G}{\deg(F^{-1})} \right\rfloor & \text{si } n - \deg G \not\equiv 0 \pmod{\deg(F^{-1})} \\ \frac{n-\deg G}{\deg(F^{-1})} - 1 & \text{sinon.} \end{cases}$$

Par conséquent, nous avons

$$\deg(F^{-1}) \left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor < n - \deg G ,$$

impliquant que

$$\delta_\ell(F^{-1}) \leq \ell \deg(F^{-1}) \leq \deg(F^{-1}) \left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor < n - \deg G .$$

Nous en déduisons que

$$\delta_{\deg G}(F) < n - \left\lfloor \frac{n-1-\deg G}{\deg(F^{-1})} \right\rfloor .$$

□

Il est évident que la borne supérieure du théorème précédent s'améliore quand le degré de F^{-1} décroît. En outre, quand G est une fonction équilibrée, cette borne a un sens si elle améliore la borne triviale $\deg(G \circ F) < n$. C'est le cas quand $\deg G \leq n - 1 - \deg F^{-1}$.

Nous voyons alors que cette borne est généralement meilleure que celle du corollaire 5.1, car cette dernière n'améliore la borne triviale que si $\deg G < \left\lceil \frac{n-1}{\min(\deg F, \deg F^{-1})} \right\rceil$.

5.3 Quelques corollaires

Nous pouvons facilement déduire du théorème 5.1 plusieurs résultats simples, qui sont pourtant assez intéressants. Le premier résultat se déduit directement du théorème principal, en fixant simplement $k = 1$. Dans ce cas, nous avons $\deg(F^{-1}) < n - \ell$ si et seulement si $\delta_\ell(F) < n - 1$.

Corollaire 5.3. *Soit F une permutation de $\mathbf{F}_2^n$. Alors,*

$$\deg(F^{-1}) = n - \min\{k \ : \ \delta_k(F) \geq n-1\} .$$

En particulier, $\deg(F^{-1}) = n - 1$ *si et seulement si* $\deg(F) = n - 1$.

Démonstration. D'après le théorème 5.1, nous avons que $\deg(F^{-1}) = \delta_1(F^{-1}) < n - k$ pour tout entier k tel que $\delta_k(F) < n - 1$. Par conséquent, $\delta_1(F^{-1}) = n - k$, où k est le plus petit entier tel que $\delta_k(F) \geq n - 1$. □

Remarque 5.1. *Les permutations de degré maximal sont particulièrement intéressantes en pratique. De plus, nous venons de montrer que leur inverse est également de degré maximal. Une question intéressante est alors de déterminer la proportion des permutations de* $\mathbf{F}_2^n$ *qui sont de degré maximal. En réalité, presque toutes les permutations de* $\mathbf{F}_2^n$ *possèdent cette propriété. Ceci vient du fait que cette classe de permutations contient toutes les permutations de degré univarié* $(2^n - 2)$*, qui correspondent de leur côté à presque toutes les permutations [Das02, KP02, Wel69]. Par exemple chaque transposition est une involution de degré algébrique* $(n-1)$*.*

Corollaire 5.4. *Pour toute permutation* F *de* $\mathbf{F}_2^n$*, nous avons*

$$\deg(F^{-1}) \leq n - \left\lceil \frac{n-1}{\deg F} \right\rceil .$$

Démonstration. Soit un entier k tel que

$$k \leq \left\lceil \frac{n-1}{\deg F} \right\rceil - 1.$$

Nous avons alors

$$\delta_k(F) \leq k \deg F < n - 1 .$$

Nous en déduisons que

$$\min\{k \ : \ \delta_k(F) \geq n-1\} \geq \left\lceil \frac{n-1}{\deg F} \right\rceil ,$$

qui implique

$$\deg(F^{-1}) \leq n - \left\lceil \frac{n-1}{\deg F} \right\rceil .$$

□

Avec ce corollaire nous obtenons d'une façon différente la borne sur le degré de $\deg(F^{-1})$ qui peut être dérivée du théorème de Katz [Kat71] sur la divisibilité du spectre de Walsh d'une permutation. En effet, tous les coefficients de Walsh de F sont divisibles par $\lceil \frac{n-1}{\deg F} \rceil + 1$ et il est connu que le degré d'une fonction dont les coefficients de Walsh sont divisibles par 2^ℓ est au plus $(n+1-\ell)$ (voir par exemple la proposition 3 dans [CV02]).

Le corollaire 5.3 implique aussi le suivant.

Corollaire 5.5. *Soit* F *une permutation de* $\mathbf{F}_2^n$*. Le produit de* k *coordonnées de* F *est de degré* $(n-1)$ *si et seulement si* $n - \deg(F^{-1}) \leq k \leq n-1$*. En particulier,* $\delta_{n-1}(F) = n-1$*.*

Démonstration. Afin de prouver que $\delta_{n-1}(F) = n-1$ il suffit de montrer qu'à la fois $\delta_{n-1}(F) \leq n-1$ et que $\delta_{n-1}(F) \geq n-1$. La première inégalité est une conséquence directe du fait que $\delta_k(F) = n$ si et seulement si $k = n$. Pour prouver l'inégalité suivante, il suffit de remarquer que d'après le corollaire 5.3, le plus petit k tel que $\delta_k(F) \geq n-1$ est égal à $n - \deg(F^{-1})$. Mais, $n - \deg(F^{-1}) \leq n-1$. Par conséquent, $\delta_{n-1}(F) \geq n-1$. □

Une conséquence importante des résultats précédents est l'amélioration de la borne du théorème 4.1. Cette amélioration provient d'une estimation plus précise de la constante γ de cette borne. Nous avons vu, avec l'exemple de KECCAK dans le chapitre précédent, à quel point une estimation plus fine de γ pourrait améliorer les bornes obtenues.

Théorème 5.2. *Soit F une permutation de $\mathbf{F}_2^n$ correspondant à la concaténation de s permutations plus petites $S_1, \ldots, S_s$, définies sur $\mathbf{F}_2^{n_0}$. Alors, pour toute fonction G de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, nous avons*

$$\deg(G \circ F) \leq n - \frac{n - \deg(G)}{\gamma}\ , \tag{5.3}$$

où

$$\gamma = \max_{1 \leq i \leq n_0 - 1} \frac{n_0 - i}{(n_0 - \max_{1 \leq j \leq s} \delta_i(S_j))}\ .$$

En particulier, nous avons

$$\gamma \leq \max_{1 \leq j \leq s} \max \left(\frac{n_0 - 1}{n_0 - \deg(S_j)},\ \frac{n_0}{2} - 1,\ \deg(S_j^{-1}) \right)\ .$$

Démonstration. Nous notons γ_i la quantité

$$\gamma_i = \frac{n_0 - i}{n_0 - \max_{1 \leq j \leq s} \delta_i(S_j)}.$$

Nous allons calculer la valeur maximale de γ_i pour $1 \leq i \leq n_0 - 1$, c'est-à-dire γ.

Pour $i = 1$,

$$\gamma_1 = \max_{1 \leq j \leq s} \frac{n_0 - 1}{(n_0 - \deg(S_j))}.$$

Pour $2 \leq i < n_0 - \max_{1 \leq j \leq s} \deg(S_j^{-1})$, nous avons d'après le corollaire 5.5 que $\max_{1 \leq j \leq s} \delta_i(S_j) \leq n_0 - 2$, et donc

$$\gamma_i = \max_{1 \leq j \leq s} \frac{n_0 - i}{(n_0 - \delta_i(S_j))} \leq \frac{n_0 - i}{2} \leq \frac{n_0 - 2}{2}.$$

Pour les indices restant c'est-à-dire pour $i \geq n_0 - \max_{1 \leq j \leq s} \deg(S_j^{-1})$, nous avons enfin,

$$\gamma_i = \max_{1 \leq j \leq s} \frac{n_0 - i}{(n_0 - \delta_i(S_j))} \leq n_0 - i \leq \max_{1 \leq j \leq s} \deg(S_j^{-1}).$$

□

5.4 Généralisation aux fonctions équilibrées non-injectives

Dans certaines primitives symétriques, les fonctions qui constituent la partie non-linéaire ne sont pas des permutations, mais des fonctions équilibrées $F : \mathbf{F}_2^n \to \mathbf{F}_2^m$, avec $m < n$. Un exemple suivant cette conception est le premier chiffrement par blocs normalisé, à savoir le DES [FIP80]. La fonction de tour de ce chiffrement est composée de l'application en parallèle de huit boîtes-S 6×4 différentes de degré algébrique 5.

Il est alors intéressant, et très utile, de pouvoir prédire l'évolution du degré algébrique des chiffrements construits ainsi. Les résultats des sections précédentes ne s'appliquent évidemment pas tels quels sur ces constructions, puisque les boîtes-S utilisées ne sont pas inversibles. Néanmoins, nous allons voir qu'il est possible de déduire des résultats similaires.

Pour arriver à cela, nous allons essayer de nous ramener au cas des permutations, pour pouvoir appliquer les résultats précédents à une extension de la fonction. Nous introduisons alors la notion d'extension d'une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ à une permutation de $\mathbf{F}_2^n$.

Définition 5.1. *Soit $F : \mathbf{F}_2^n \to \mathbf{F}_2^m$, avec $m < n$, $F = (F_1, \ldots, F_m)$ une fonction équilibrée. Une permutation $P = (P_1, \ldots, P_n)$ de $\mathbf{F}_2^n$ est appelée* extension *de F, si ses m premières coordonnées correspondent aux coordonnées de F, c'est-à-dire pour tout i, $1 \leq i \leq m$,*

$$P_i(x) = F_i(x), \text{ pour tout } x \in \mathbf{F}_2^n.$$

En d'autres termes, la fonction F est étendue à une permutation de n variables de la façon suivante. Comme F est équilibrée, chacun des 2^m vecteurs de $\mathbf{F}_2^m$ est pris par F 2^{n-m} fois exactement. Nous complétons alors tous ces 2^{n-m} vecteurs égaux en concaténant à chacun d'entre eux un élément différent de $\mathbf{F}_2^{n-m}$ de façon à obtenir 2^{n-m} vecteurs différents de $\mathbf{F}_2^n$.

Exemple 5.1. *Soit $(n, m) = (6, 4)$ et supposons que $v = (0, 1, 1, 0)$ est un vecteur dans l'image de F, obtenu pour 4 vecteurs v_1, v_2, v_3, v_4 de $\mathbf{F}_2^6$, c'est-à-dire $F(v_1) = F(v_2) = F(v_3) = F(v_4) = v$. Une extension de F peut alors être obtenue en associant à v_1, v_2, v_3 et v_4 les quatre vecteurs différents de $\mathbf{F}_2^6$, $(0, 1, 1, 0, 0, 0), (0, 1, 1, 0, 1, 0), (0, 1, 1, 0, 0, 1), (0, 1, 1, 0, 1, 1)$. Ces 4 images sont la simple concaténation du vecteur v avec tous les éléments de $\mathbf{F}_2^2$.*

Pour une fonction $F : \mathbf{F}_2^n \to \mathbf{F}_2^m$, il existe $(2^{n-m}!)^{2^m}$ extensions différentes.

Remarque 5.2. *Dans l'exemple précédent, nous avons concaténé les éléments de $\mathbf{F}_2^2$ à la fin du vecteur v. Cette possibilité n'est évidement pas la seule pour étendre une fonction F de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ à une permutation. Pour un vecteur v dans l'image de F nous pouvons choisir $n - m$ positions quelconques $i_1, \ldots, i_{n-m}$ parmi $\{1, \ldots, n\}$ et insérer les 2^{n-m} vecteurs différents à ces positions. Dans l'exemple précédent, nous avons pris $i_1 = 5$ et $i_2 = 6$, mais nous aurons par exemple pu choisir $i_1 = 1$ et $i_2 = 4$. Dans ce cas, les quatre vecteurs v_1, v_2, v_2 et v_4 de l'exemple précédent seraient associés aux quatre vecteurs $(0, 0, 1, 0, 1, 0), (0, 0, 1, 1, 1, 0), (1, 0, 1, 0, 1, 0), (1, 0, 1, 1, 1, 0)$. Cependant, afin de simplifier la notation, nous considérons à partir de maintenant que toutes les concaténations se font comme à l'exemple 5.1. Les résultats que nous présenterons par la suite pourront être généralisés de façon triviale à tous les autres types de concaténation.*

Théorème 5.3. *Soit F une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, avec $m < n$. Soit k et ℓ deux entiers tels que $1 \leq k < m$ et $1 \leq \ell < n$. Alors, les trois propriétés suivantes sont équivalentes.*

(i) *Il existe une permutation P_F de $\mathbf{F}_2^n$ étendant F, telle que dans chaque produit de ℓ coordonnées de P_F^{-1}, tous les monômes de degré supérieur ou égal à $(n - k)$ sont de degré strictement inférieur à $(n - m)$ en les $(n - m)$ dernières variables.*

(ii) *Pour toute permutation P_F de $\mathbf{F}_2^n$ étendant F, nous avons que dans chaque produit de ℓ coordonnées de P_F^{-1}, tous les monômes de degré supérieur ou égal à $(n - k)$ sont de degré strictement inférieur à $(n - m)$ en les $(n - m)$ dernières variables.*

(iii) $\delta_k(F) < n - \ell$.

Démonstration. Soient $K \subset \{1, \ldots, m\}$ et $L \subset \{1, \ldots, n\}$. Nous désignons par π_K le produit des coordonnées F_i pour $i \in K$. Alors, le coefficient $a_{K,L}$ du monôme $\prod_{i \in \{1,\ldots,n\} \setminus L} x_i$ dans la forme algébrique normale de F est donné par

$$\begin{aligned} a_{K,L} &= \sum_{\substack{x \in \mathbf{F}_2^n \\ x_j = 0, j \in L}} \pi_K(x) \\ &= \#\{x \in \mathbf{F}_2^n : x_j = 0, j \in L \text{ et } F_i(x) = 1, i \in K\} \mod 2 \\ &= \#\{x \in \mathbf{F}_2^n : x_j = 0, j \in L \text{ et } (P_F)_i(x) = 1, i \in K\} \mod 2 , \end{aligned}$$

où la dernière égalité est valide pour toute extension P_F de F. Alors, puisque P_F est une permutation, en posant $y = P_F(x)$ nous avons

$$a_{K,L} = \#\{y \in \mathbf{F}_2^n : y_i = 1, i \in K \text{ et } (P_F^{-1})_j(y) = 0, j \in L\} \mod 2 ,$$

impliquant que $a_{K,L} = 0$ si et seulement si la fonction booléenne

$$\begin{array}{rccc} H_{K,L} : & \{x \in \mathbf{F}_2^n : x_i = 1, i \in K\} & \to & \mathbf{F}_2 \\ & x & \mapsto & \prod_{i \in L}(1 + (P_F^{-1})_i(x)) . \end{array}$$

est de degré strictement inférieur à $n - k$.

Nous démontrons d'abord que (i) implique (iii). Nous déduisons du raisonnement précédent que si la condition (i) est satisfaite, alors tout monôme de degré supérieur ou égal à $(n - k)$ dans l'ANF de la fonction booléenne à n variables

$$x \mapsto \prod_{i \in L}(1 + (P_F^{-1})_i(x))$$

n'est pas un facteur de $x_{m+1} \dots x_n$. Par conséquent, la restriction de tout monôme de ce type à n'importe quel ensemble $\{x \in \mathbf{F}_2^n : x_i = 1, i \in K\}$ avec $K \subset \{1, \dots, m\}$ est de degré strictement inférieur à $(m - k) + (n - m) = (n - k)$. Il s'ensuit que pour tout choix de $K \subset \{1, \dots, m\}$, $H_{K,L}$ est de degré strictement inférieur à $(n - k)$. Nous avons alors : (ii) $\Rightarrow$ (i) $\Rightarrow$ (iii).

Inversement, nous démontrons que (iii) implique (ii). Supposons que la condition (ii) ne soit pas satisfaite, c'est-à-dire qu'il existe une permutation P_F qui étend F et un ensemble $L \subset \{1, \dots, m\}$ tel que la fonction booléenne à n variables

$$\pi'_L : x \mapsto \prod_{i \in L}(P_F^{-1})_i(x)$$

contient un monôme de la forme $x_{m+1} \dots x_n \prod_{i \in I} x_i$ pour un ensemble $I \subset \{1, \dots, m\}$ de taille au moins $(m - k)$. Nous pouvons supposer que L est le plus petit ensemble de ce type pour l'inclusion (sinon, nous choisissons le plus petit ensemble $L' \subset L$ qui satisfait cette propriété). Choisissons $K = \{1, \dots, m\} \setminus I$ où $x_{m+1} \dots x_n \prod_{i \in I} x_i$ est le monôme de plus haut degré de cette forme dans l'ANF de π'_L. Par hypothèse, la taille de K est au plus k et elle est supérieure ou égale à 1, puisque π'_L est de degré au plus n quand $|L| < n$ (corollaire 4.1). Comme L est minimal pour l'inclusion et

$$H_{K,L}(x) = \sum_{L' \subseteq L} \prod_{i \in L'} (P_F^{-1})_i(x) ,$$

il est clair que $H_{K,L}$ est de degré au plus $(n - k)$ si et seulement si la restriction de π'_L à $\{x \in \mathbf{F}_2^n : x_i = 1, i \in K\}$ est de degré $(n - k)$. Cependant, la forme algébrique normale de π'_L contient le monôme $x_{m+1} \dots x_n \prod_{i \notin K} x_i$, impliquant que $H_{K,L}$ est de degré au moins $(n - k)$. Il s'ensuit que pour ces choix particuliers de L et K, $a_{K,L} = 1$, impliquant qu'il existe un produit d'au plus k coordonnées de F de degré supérieur ou égal à $(n - \ell)$.

Nous avons alors démontré que les trois propriétés sont équivalentes. □

Nous pouvons maintenant déduire pour le cas des fonctions équilibrées non-injectives, un corollaire similaire au corollaire 5.2.

Corollaire 5.6. *Soit F une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, avec $n \geq m$ et G une fonction de $\mathbf{F}_2^m$ dans $\mathbf{F}_2^k$. Pour n'importe quelle permutation F^* qui étend F, nous avons*

$$\deg(G \circ F) < n - \left\lfloor \frac{n-1-\deg G}{\deg(F^{*-1})} \right\rfloor .$$

Démonstration. Soit F^* une permutation qui étend F. Dans la preuve du corollaire 5.2 nous avons montré que la borne triviale implique que $\delta_\ell(F^{*-1}) < n - \deg G$ pour toute

$$\ell \leq \left\lfloor \frac{n-1-\deg G}{\deg(F^{*-1})} \right\rfloor .$$

Par conséquent, quand ℓ satisfait cette condition, le produit de ℓ coordonnées quelconques de F^{*-1} ne contient aucun monôme de degré $(n-\deg G)$. Puisque la condition (i) du théorème 5.3 est satisfaite, nous en déduisons

$$\deg(G \circ F) \leq \delta_{\deg G}(F) < n - \left\lfloor \frac{n-1-\deg G}{\deg(F^{*-1})} \right\rfloor .$$

□

Nous avons prouvé dans le corollaire 4.1 que le produit de k coordonnées d'une fonction équilibrée F à n variables est de degré n si et seulement si $k = n$. En outre, si F est une permutation, nous avons démontré dans le corollaire 5.5 que le degré de F^{-1} détermine quand le produit de plusieurs coordonnées de F est de degré $(n-1)$. Nous présentons ici un résultat similaire quand F une fonction équilibrée non-bijective.

Corollaire 5.7. *Soit F une fonction équilibrée de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$, avec $m < n$. Alors, $\delta_m(F) \leq n-2$ si et seulement si pour n'importe quel élément $y \in \mathbf{F}_2^m$, la somme des 2^{n-m} antécédents de y par F est nulle, c'est-à-dire,*

$$\sum_{x:F(x)=y} x = 0 ,$$

où la somme correspond à l'addition dans $\mathbf{F}_2^n$.

Démonstration. D'après le théorème 5.3 appliqué avec $k = m$ et $\ell = 1$, nous avons que $\delta_m(F) \leq 2$ si et seulement s'il existe une permutation P_F étendant F, telle que tout monôme de degré au moins $(n-m)$ dans l'ANF de n'importe quelle coordonnée de P_F^{-1} n'est pas un facteur de $x_{m+1} \ldots x_n$. Comme un monôme de degré strictement inférieur à $(n-m)$ ne peut pas être un facteur de $x_{m+1} \ldots x_n$, ceci de la même façon signifie que n'importe quel monôme dans l'ANF de n'importe quelle coordonnée de P_F^{-1} n'est pas un facteur de $x_{m+1} \ldots x_n$. Soit

$$\begin{array}{rccc} f: & \mathbf{F}_2^m \times \mathbf{F}_2^{n-m} & \to & \mathbf{F}_2 \\ & (x,y) & \mapsto & [P_F^{-1}(x,y)]_i , \end{array}$$

pour un entier i. Pour tout $(u,v) \in \mathbf{F}_2^m \times \mathbf{F}_2^{n-m}$, $a_{u,v}$ désigne le coefficient dans l'ANF de f du monôme $\prod_{i,u_i \neq 0} x_i \prod_{i,v_i \neq 0} x_{m+1+i}$. Notons 1_{n-m} le vecteur de $\mathbf{F}_2^{n-m}$ dont toutes les coordonnées sont égales à 1. Pour tout $x \in \mathbf{F}_2^m$ et $y \in \mathbf{F}_2^{n-m}$, nous avons

$$f(x,y) = \sum_{v \preceq y} \left[\sum_{u \preceq x} a_{u,v} \right] ,$$

où $x \preceq y$ signifie que $x_i \leq y_i$ pour tout i. Par conséquent,

$$\sum_{y \in \mathbf{F}_2^{n-m}} f(x,y) = \sum_{y \in \mathbf{F}_2^{n-m}} \sum_{v \preceq y} \left[\sum_{u \preceq x} a_{u,v} \right] \equiv \sum_{v \in \mathbf{F}_2^{n-m}} N_v \left[\sum_{u \preceq x} a_{u,v} \right] \pmod 2 ,$$

où

$$N_v = \#\{y \in \mathbf{F}_2^{n-m} \ : \ v \preceq y\} \bmod 2 = 2^{n-m-wt(v)} \bmod 2 .$$

Alors, $N_v = 0$ sauf quand v est le vecteur tout à 1. En conséquence,

$$\sum_{y \in \mathbf{F}_2^{n-m}} f(x,y) = \sum_{u \preceq x} a_{u,1_{n-m}} .$$

Nous déduisons alors que $a_{u,1_{n-m}} = 0$ pour $u \in \mathbf{F}_2^m$ si et seulement si

$$\sum_{y \in \mathbf{F}_2^{n-m}} f(x,y) = 0$$

pour tout $x \in \mathbf{F}_2^m$. Il est intéressant de constater que cette propriété est similaire à la propriété utilisée aux attaques cubes (théorème 2.2, de la section 2.2.3).

Puisque cette propriété tient pour toute coordonnée f de P_F^{-1}, la condition requise signifie de façon équivalente que pour tout $x \in \mathbf{F}_2^m$,

$$\sum_{y \in \mathbf{F}_2^{n-m}} P_F^{-1}(x,y) = 0 ,$$

où la somme est dans $\mathbf{F}_2^n$. D'après la définition de P_F, tous les éléments $P_F^{-1}(x,y)$ quand $y \in \mathbf{F}_2^{n-m}$ correspondent aux images de x par F. Cette condition peut être écrite sous la forme

$$\sum_{z:F(z)=x} z = 0 .$$

□

5.5 Applications à certaines primitives symétriques

Nous allons voir dans cette section comment les résultats de ce chapitre peuvent être utilisés pour estimer l'évolution du degré algébrique des certaines permutations itérées. Plus précisément, nous allons examiner des permutations qui sont les composantes principales de certains chiffrements par blocs et de certaines fonctions de hachage bien connus.

5.5.1 Attaque contre une variante du chiffrement par blocs $\mathcal{KN}$

Dans la section 2.2.2 nous avons décrit le chiffrement $\mathcal{KN}$, conçu par Kaisa Nyberg et Lars Knudsen en 1995. Nous avons également vu qu'à cause du degré algébrique faible de la fonction de tour, une attaque différentielle d'ordre supérieur très efficace a pu être appliquée à cette construction par Thomas Jacobsen et Lars Knudsen [JK97]. Dans l'article d'origine [NK95], les auteurs ont également proposé une variante du chiffrement, que nous allons appeler $\mathcal{KN}'$, qui

consiste à simplement remplacer la fonction de substitution S par l'inverse d'une permutation quadratique. Cette modification n'affecte pas la résistance du chiffrement contre les attaques différentielles et linéaires puisque il est connu [Nyb95] que la résistance offerte par une fonction contre ces attaques est la même que la résistance de son inverse. Cependant, une permutation et son inverse n'ont pas forcément le même degré algébrique. Par exemple, si S est une permutation puissance quadratique sur $\mathbf{F}_{2^n}$, où n est impair, c'est-à-dire $S(x) = x^{2^s+1}$ avec $\text{pgcd}(s, n) = 1$, alors le degré algébrique de S^{-1} est égal à $\frac{n+1}{2}$ [Nyb94]. Pour cette raison, le chiffrement $\mathcal{KN}'$ devrait mieux résister à l'attaque de [JK97]. De plus, comme l'inverse de la fonction de tour n'intervient ni dans la fonction de chiffrement ni dans la fonction de déchiffrement (section 1.4.3), le fait que la fonction inverse de la fonction de substitution de $\mathcal{KN}'$ soit algébriquement faible ne devrait pas avoir de conséquence sur la sécurité de la nouvelle fonction.

Nous allons voir ici que cela n'est pas vrai. Plus précisément, nous allons montrer en utilisant les résultats de ce chapitre que le fait que l'inverse de la fonction de tour de $\mathcal{KN}'$ soit de degré 2 a des conséquences sur l'évolution de son degré algébrique. En particulier, une attaque équivalente à celle présentée dans [JK97] peut être déduite pour la variante $\mathcal{KN}'$.

Puisque la complexité de l'implémentation de l'inverse de la fonction x^3 dans $\mathbf{F}_{2^{33}}$ est beaucoup trop élevée pour la plupart des applications, nous considérons la fonction non-linéaire sur $\mathbf{F}_{2^{32}}$ composée de quatre applications parallèles de la même fonction $\widetilde{\sigma}$, définie sur $\mathbf{F}_2^8$, exactement comme dans $\mathcal{KN}$:

$$\begin{array}{rccc} \widetilde{\sigma}: & \mathbf{F}_2^8 & \to & \mathbf{F}_2^8 \\ & x & \mapsto & t \circ \sigma\left(e(x)\right)) \ , \end{array}$$

où e est une expansion linéaire de $\mathbf{F}_2^8$ dans $\mathbf{F}_2^9$ de rang maximal, t une troncation de $\mathbf{F}_2^9$ dans $\mathbf{F}_2^8$ et σ l'inverse d'une permutation puissance quadratique $x \mapsto x^{2^s+1}$ sur $\mathbf{F}_{2^9}$, par exemple $\sigma(x) = x^{171}$, qui est l'inverse de x^3. Cette fonction, qui est la seule composante non-linéaire du chiffrement, est de degré algébrique 5. Il est intéressant de constater que cette fonction possède un degré univarié élevé qui est suffisant pour éviter des attaques dites *par interpolation*. La fonction de tour de $\mathcal{KN}'$ est illustrée à la figure 5.1. Elle est définie par

$$\begin{array}{rcl} \mathbf{F}_2^{32} \times \mathbf{F}_2^{32} & \to & \mathbf{F}_2^{32} \times \mathbf{F}_2^{32} \\ (x, y) & \mapsto & (y, x + \mathcal{L}' \circ \widetilde{S}\left(\mathcal{L}(x) + k_i\right)), \end{array}$$

où $\widetilde{S}$ correspond à quatre applications parallèles de $\widetilde{\sigma}$, k_i est la i-ième coordonnée de la sous-clé de 32 bits, et $\mathcal{L}$ et $\mathcal{L}'$ sont deux bijections linéaires sur $\mathbf{F}_2^{32}$.

En utilisant la borne triviale pour estimer le degré algébrique du chiffrement $\mathcal{KN}'$, nous n'obtenons aucune information suffisante permettant d'appliquer la même attaque que dans [JK97]. En effet, comme $5^3 = 125 > 64 - 1$, cette borne montre que le degré atteint après seulement 3 tours, sa valeur maximale. Nous montrons toutefois que l'attaque de Jakobsen et Knudsen s'applique également à $\mathcal{KN}'$.

Pour cela, nous étudions, exactement comme nous l'avons fait dans la section 2.2.2, le degré algébrique de la fonction qui associe à la partie gauche du texte clair, x_0, la partie gauche de la sortie du chiffrement après r tours, x_r. Par la suite, nous notons F_k la fonction sur $\mathbf{F}_2^{32}$ définie par :

$$F_k(x) = \mathcal{L}' \circ \widetilde{S}\left(\mathcal{L}(x) + k\right) \ .$$

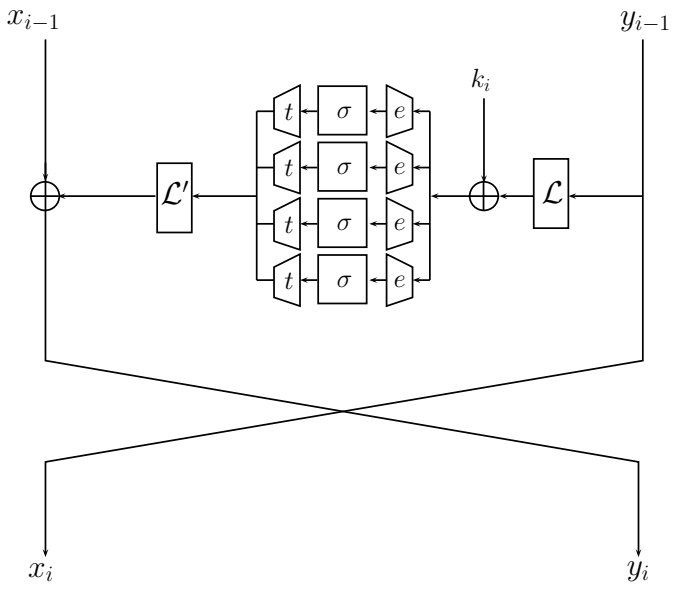

FIGURE 5.1 – Le tour i du chiffrement $\mathcal{KN}'$

Nous avons alors,

$$\begin{aligned} x_2 &= x_0 + F_{k_1}(y_0) \\ x_3 &= y_0 + F_{k_2}\left(x_0 + F_{k_1}(y_0)\right) \\ x_4 &= x_0 + F_{k_1}(y_0) + F_{k_3}\left(y_0 + F_{k_2}\left(x_0 + F_{k_1}(y_0)\right)\right) \ . \end{aligned}$$

Notons maintenant x l'élément de $\mathbf{F}_2^{36}$ qui est défini par

$$x = \mathcal{E}\left(\mathcal{L}(x_0 + F_{k_1}(y_0)) + k_2\right)$$

où $\mathcal{E}$ est l'expansion linéaire de $\mathbf{F}_2^{32}$ dans $\mathbf{F}_2^{36}$ composée de 4 applications de l'expansion e. Alors, x_0 peut être calculé en fonction de x :

$$x_0 = \mathcal{L}^{-1}\left(\mathcal{E}^\star(x) + k_2\right) + F_{k_1}(y_0) \ ,$$

où $\mathcal{E}^\star$ est la fonction de $\mathbf{F}_2^{36}$ dans $\mathbf{F}_2^{32}$ définie par $\mathcal{E}^\star\left(\mathcal{E}(x)\right) = x$ et $\mathcal{E}^\star(x) = 0$ si $x \notin Im\mathcal{E}$. Une telle fonction existe, puisque $\mathcal{E}$ est de rang maximal. Alors, x_4 peut être exprimé en fonction de x :

$$x_4 = \mathcal{L}^{-1}\left(\mathcal{E}^\star(x) + k_2\right) + F_{k_3}\left(y_0 + \mathcal{L}' \circ \mathcal{T} \circ S(x)\right) \ ,$$

où S est la permutation de $\mathbf{F}_2^{36}$ correspondant à 4 applications parallèles de σ et $\mathcal{T}$ est la fonction de $\mathbf{F}_2^{36}$ dans $\mathbf{F}_2^{32}$ définie comme 4 applications parallèles de la troncation t. Puisque,

$$x_5 = x_3 + F_{k_4}(x_4)$$

nous en déduisons

$$x_5 + x_3 = F_{k_4}\left[\mathcal{L}^{-1}\left(\mathcal{E}^\star(x) + k_2\right) + F_{k_3}\left(y_0 + \mathcal{L}' \circ \mathcal{T} \circ S(x)\right)\right] \ .$$

Le degré de x_5 en fonction de x_0 est majoré par le maximum entre le degré de x_3, qui est au plus 5 et le degré de $x_5 + x_3$, vu comme une fonction de x puisque x et de degré 1 en x_0. Nous nous concentrons sur cette dernière quantité.

Nous notons

$$x_5 + x_3 = G \circ S(x)$$

où

$$G(y) = F_{k_4}\left[\mathcal{L}^{-1}\left(\mathcal{E}^\star(S^{-1}(y)) + k_2\right) + F_{k_3}\left(y_0 + \mathcal{L}' \circ \mathcal{T}(y)\right)\right] .$$

Degré de G. Puisque F_{k_4} est de degré 5, G peut être décomposée en une somme, dont chaque terme consiste en le produit de i coordonnées de S^{-1}, multiplié par le produit d'au plus $(5-i)$ coordonnées de S. Comme, S^{-1} est de degré 2, nous obtenons

$$\deg G \leq \max_{0\leq i\leq 5}\left(2i + \delta_{5-i}(S)\right) \quad .$$

D'après le théorème 5.1, nous avons $\delta_5(S) < 36 - \lfloor \frac{30}{2} \rfloor$, et par conséquent $\delta_5(S) \leq 20$. En conséquence, nous déduisons que $\deg G \leq 2 + \delta_4(S) \leq 22$.

Degré de $G \circ S$ Nous appliquons maintenant le corollaire 5.2 pour majorer le degré de $G \circ S$, en exploitant le fait que S^{-1} est de degré 2. Alors, nous avons

$$\deg(G \circ S) < 36 - \left\lfloor \frac{35-22}{2} \right\rfloor ,$$

et donc

$$\deg(G \circ S) \leq 29 ,$$

qui signifie que x_5 est une fonction de degré au plus 29 en x_0. Cela conduit à un distingueur sur 5 tours de $\mathcal{KN}'$ de complexité en données 2^{30} qui améliore le distingueur générique (section 2.2.2). Il faut souligner que la même borne supérieure peut être dérivée du théorème 5.2, en exploitant le fait que S correspond à la concaténation de 4 permutations σ définies sur $\mathbf{F}_2^9$.

Variante avec des boîtes-S non bijectives La fonction linéaire dans $\mathcal{KN}'$ peut aussi être vue comme la concaténation de quatre fonctions équilibrées σ' de $\mathbf{F}_2^9$ dans $\mathbf{F}_2^8$. Plutôt qu'appliquer le corollaire 5.2 basé sur le degré de l'inverse de la fonction non-linéaire S, nous pouvons utiliser le fait qu'il existe une permutation S^* qui étend la boîte-S de 36×32 avec $\deg((S^*)^{-1}) = 2$. Dans ce cas nous pouvons appliquer le corollaire 5.7 qui montre lui aussi que x_5 est une fonction de degré au plus 29 en x_0.

Remarque 5.3. *Il est possible de considérer une variante de $\mathcal{KN}$ différente de celle que nous avons choisie, difficilement utilisable en pratique mais ayant un intérêt théorique. Il s'agit d'utiliser à la place de la permutation S, l'inverse de la fonction x^3 sur $\mathbf{F}_{2^{33}}$. Dans ce cas, le corollaire 5.2 conduit à un distingueur sur quatre tours de la fonction, en exploitant le fait que x_4 est de degré au plus 25. Par contre, une borne pertinente pour le degré de x_5 reste une question ouverte.*

5.5.2 Application à Rijndael-256

Nous montrons ici comment les résultats précédents s'appliquent à Rijndael-256. La description complète de cette fonction est donnée à la section 1.4.2. Nous avons vu à la section 4.4 avec l'approche superBoîte-S que le degré de 2 tours de l'AES est au plus 28. Ceci est également vrai pour Rijndael-256. Contrairement à l'AES, comme l'état de Rijndael-256 est grand, après deux itérations toutes les parties de l'état ne sont pas bien mélangées entre elles. Par conséquent, nous pouvons maintenant appliquer l'approche superBoîte-S pour trois tours de la fonction. En effet, trois tours de Rijndael-256 peuvent être vus comme l'application en parallèle de deux copies d'une fonction non-linéaire S_{128}, suivie par une application linéaire. En appliquant alors le théorème 4.1 nous obtenons

$$\deg R^3 = \deg S_{128} \leq 128 - \frac{128-28}{7} < 114.$$

Soit maintenant $F = R^2$, où R est la fonction de tour. F est une permutation de degré au plus 28 et ceci est également vrai pour son inverse. En utilisant que $R^3 \circ F = R^5$, nous obtenons une borne pour cinq tours de la fonction. D'après le théorème 5.2, nous avons que la constante γ associée à $F = R^2$ est au plus 28 et nous déduisons finalement que

$$\deg R^5 \leq 256 - \frac{256-113}{28} < 251.$$

Nous obtenons une borne similaire pour 6 itérations, en considérant $F = R^3$, qui est de degré au plus 113. Puisque $\deg F^{-1} \leq 113$, la constante γ correspondant est au plus 113. Par conséquent,

$$\deg R^6 \leq 256 - \frac{256-113}{113} < 255.$$

En conséquence, au moins 7 tours sont nécessaires afin d'atteindre le degré maximal. Ces résultats sont résumés à la table 5.1. Une comparaison avec la borne triviale ainsi qu'avec les résultats obtenus avec l'approche superBoîte-S y sont représentés.

# tours	borne triviale	superBoîte-S	ce chapitre
1	7	7	7
2	49	31	28
3	–	127	113
4	–	–	235
5	–	–	250
6	–	–	254

TABLE 5.1 – Bornes supérieures pour r itérations de la permutation de tour de Rijndael-256 obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre.

5.5.3 Application à la fonction de hachage ECHO

La fonction de hachage ECHO [BBG+09] a été conçue par Benadjila *et al.* pour la compétition SHA-3. Elle a été parmi les 14 fonctions sélectionnées pour le deuxième tour du concours.

Description de ECHO

La fonction ECHO utilise la construction HAIFA comme extension de domaine. Sa fonction de compression prend en entrée une valeur de 2048 bits qui correspond à la valeur de chaînage et au bloc de message dont les longueurs respectives dépendent de la taille de l'empreinte. Elle donne en sortie une valeur de 512 ou de 1024 bits. Elle est basée sur une grande permutation P de 2048 bits, inspirée de l'AES.

La permutation P met à jour un état de 2048 bits qui peut être vu comme un état de l'AES 4×4, composé de mots de 128 bits. À chaque tour R, trois opérations modifient l'état. Il s'agit des permutations `BIG.SubWords`, `BIG.ShiftRows` et `BIG.MixColumns`. Ces transformations sont des généralisations de trois opérations classiques de l'AES. En particulier,

- La transformation `BIG.SubWords` est une permutation non-linéaire qui s'applique de manière indépendante à chaque mot de 128 bits. Elle correspond à deux tours de l'AES.
- Les transformations `BIG.ShiftRows` et `BIG.MixColumns` sont des analogues des transformations `ShiftRows` et `MixColumns` de l'AES, à la seule différence qu'elles s'appliquent aux mots de 128 bits.

Le nombre de tours r est spécifié à 8 pour la version retournant des hachés de 256 bits, que nous notons ECHO-256.

Borne sur le degré de 4 tours de ECHO-256

Nous allons analyser ici l'évolution du degré algébrique de la permutation P. Nous allons en particulier montrer que le degré de P n'évolue pas aussi vite que prévu et attend sa valeur maximale, c'est-à-dire 2047, plus tard qu'attendu. Le degré de P était atteint être très élevé, puisque à chaque tour l'entrée doit passer deux fois par la couche des boîtes-S, de degré 7. Comme $7^4 = 2401$, deux tours semblaient suffisants pour que le degré maximal soit atteint.

La transformation `BIG.SubWords` est la seule source de non-linéarité de la permutation de tour. C'est une permutation de 128 bits correspondant à deux tours de l'AES. Son degré est alors au plus 28. Nous notons maintenant par $F = R^2$ deux itérations de la fonction de tour. Suivant l'approche superBoîte-S, F peut être décomposée en quatre applications parallèles d'une permutation S_{512}, suivies par une transformation linéaire. Nous allons déterminer le degré de chacune de ces quatre permutations. Après le premier tour de la transformation P, chaque bit de l'état correspond à un polynôme de degré au plus 28. En appliquant à cet état la première couche des boîtes-S dans chaque `BIG.SubWords`, le degré devient au plus $7 \cdot 28 = 196$. Nous pouvons maintenant appliquer le théorème 4.1, pour obtenir la borne suivante pour deux tours de P :

$$\deg \mathtt{R}^2 = \deg S_{512} \leq 512 - \frac{512 - 196}{7} < 467 \ .$$

F est alors une permutation de degré au plus 466. D'après le théorème 5.2, comme le degré de R^2 et le degré de son inverse sont tous les deux majorés par 466, la constante γ associée à cette permutation est au plus 466. Par conséquent,

$$\deg F^2 = \deg \mathtt{R}^4 \leq 2048 - \frac{2048 - 466}{466} < 2045.$$

Nous résumons ces résultats ainsi que les résultats obtenus par la borne triviale et l'approche superBoîte-S à la table 5.2, exactement comme nous l'avons fait pour Rijndael-256.

# tours	borne triviale	superBoîte-S	ce chapitre
1	49	31	28
2	–	511	466
3	–	–	1991
4	–	–	2044

TABLE 5.2 – Bornes supérieures pour r itérations de la fonction de tour de la permutation P de ECHO, obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre.

Le mêmes bornes sont valides pour la permutation inverse. Grâce à cette observation, nous pouvons distinguer la permutation P de ECHO d'une permutation aléatoire. Pour cela, nous pouvons construire plusieurs partitions en sommes nulles de taille 2^{2045}, c'est-à-dire de partitionner l'espace $\mathbf{F}_2^{2048}$ en huit ensembles $X_1 \ldots, X_8$ de taille 2^{2045}, tels que tous les éléments dans chaque X_i aient une somme nulle et que les images correspondant $P(x)$, $x \in X_i$ aient également une somme nulle, en utilisant les techniques introduites dans le chapitre 3. Plus précisément, si V est un sous-espace de $\mathbf{F}_2^{2048}$ de codimension 3 et W son espace supplémentaire, alors les huit ensembles

$$X_i = \{(\mathtt{R}^4)^{-1}(a_i + v),\ v \in V\}, a_i \in W$$

forment une partition en sommes nulles de $\mathbf{F}_2^{2048}$ de taille 2^{2045} pour P.

5.5.4 Application à la fonction de hachage JH

JH [Wu11] est une famille de fonctions de hachage, dont certaines instances ont été soumises à la compétition SHA-3. Elle a été parmi les 5 finalistes du concours.

Description de JH

La fonction de compression de la fonction JH est construite à partir d'un chiffrement par blocs à clé constante. Elle est basée sur une permutation interne, appelée E_d qui est composée de 42 itérations d'une fonction de tour R_d, où $d = 8$ pour le candidat SHA-3. La fonction de tour R_d s'applique à un état de 2^{d+2} bits, divisé en mots de 4 bits. Elle est constituée de trois couches différentes : une couche de boîtes-S, une couche linéaire et une permutation P_d.

- La *couche de boîtes-S* correspond à l'application en parallèle de 2^d boîtes-S à l'état. Deux boîtes-S différentes, S_0 et S_1 sont utilisées dans JH. Toutes les deux, comme leurs inverses, conformément au corollaire 5.5, sont de degré algébrique 3. La boîte-S qui est utilisée à chaque fois, est déterminée par la constante du tour, qui n'est pas additionnée à l'état comme c'est souvent le cas dans d'autres constructions.
- La *couche linéaire* mélange les 2^d mots de l'état deux à deux.
- La permutation P_d permute ensuite les mots de l'état.
  Deux tours de R_d pour $d = 4$ peuvent être visualisés à la figure 5.2.

Estimer le degré algébrique de JH

Un tour de permutation est de degré algébrique 3, puisque la seule source de non-linéarité de la fonction de tour sont les boîtes-S. Par conséquent, en utilisant la borne triviale nous

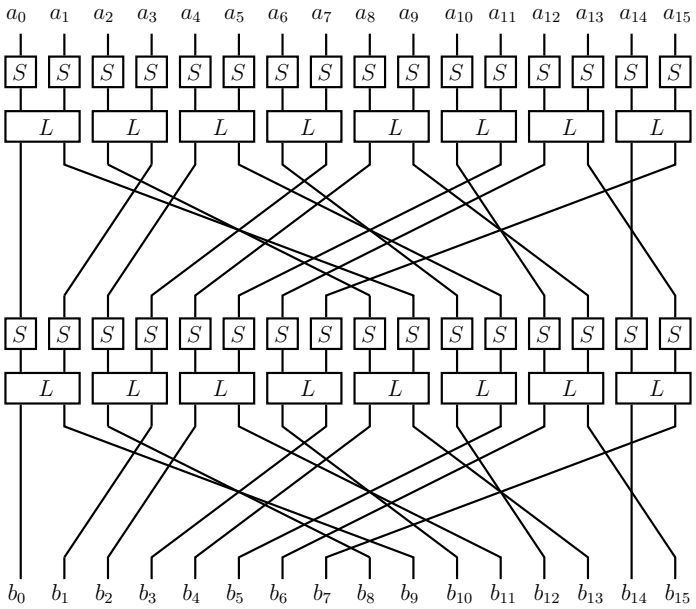

FIGURE 5.2 – Deux tours de R_4.

avons que le degré après 6 itérations est au plus $\deg R_8^6 \leq 3^6 = 729$ et donc le degré maximal semble être atteint seulement après 7 itérations. Nous montrons ici que le degré n'évolue pas aussi rapidement que prévu.

Une observation importante sur la structure de la permutation R_8 est que pour $r \leq 8$, r tours de R_8^r peuvent être vus comme la concaténation de 2^{9-r} fonctions S_r sur $\mathbf{F}_2^{2^{r+1}}$. En effet, nous voyons qu'après deux tours de permutation, chaque boîte-S de l'état est influencée par au plus 2 boîtes-S en entrée, après 3 tours, par au plus 4 boîtes-S, après 4 tours, par au plus 8 boîtes-S, etc ... En conséquence, pour $2 \leq r \leq 8$, une borne sur le degré de R_8^r peut être dérivée en utilisant le théorème 4.1 :

$$\deg(R_8^r) \leq 2^{r+1} - \frac{2^{r+1} - \deg(R_8^{r-1})}{3}.$$

Les bornes jusqu'à 8 tours de la permutation obtenues grâce à cette formule sont résumées à la table 5.3. Les mêmes bornes sont valides pour la permutation inverse.

En utilisant maintenant le théorème 5.2, nous avons que la constante $\gamma(S_8)$ de la permutation S_8 sur $\mathbf{F}_2^{512}$ est au plus 409. Nous en déduisons

$$\deg R_8^{16} \leq 1024 - \frac{1024 - \deg(R_8^8)}{\gamma(S_8)} < 1023.$$

La même technique peut être appliquée de 9 jusqu'à 16 tours et donne les résultats de la table 5.4. Une comparaison avec les bornes précédemment connues est également fournie.

# Tours	Borne sur $\deg(R_8^r)$
1	3
2	6
3	12
4	25
5	51
6	102
7	204
8	409

TABLE 5.3 – Bornes supérieures sur le degré jusqu'à 8 tours de la permutation R_8.

# tours	borne triviale	superBoîte-S	ce chapitre
1	3	3	3
2	9	7	6
3	27	15	12
4	81	31	25
5	243	63	51
6	729	127	102
7	–	255	204
8	–	511	409
9	–	–	819
10	–	–	921
11	–	–	972
12	–	–	999
13	–	–	1011
14	–	–	1017
15	–	–	1020
16	–	–	1022

TABLE 5.4 – Bornes supérieures pour r itérations de la fonction de tour de la permutation R_8 de JH, obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre.

Chapitre 6

Sur la propagation des relations linéaires à travers une boîte-S

Sommaire

6.1 La notion de la (v, w)-linéarité 120
6.1.1 Définition et propriétés générales 120
6.1.2 Lien avec la construction Maiorana-McFarland et les dérivées du deuxième ordre 121
6.1.3 Fonctions $(n-1, 1)$-linéaires 122
6.2 Application à Hamsi 128
6.2.1 Description de l'attaque de T. Fuhr 128
6.2.2 Amélioration de l'attaque de T. Fuhr 131
6.2.3 Relations affines pour trois tours de la fonction de compression . . . 132
6.3 Conclusion 137

Les fonctions non-linéaires de petite taille, appelées boîtes-S, sont des composantes qui jouent un rôle central dans la conception des cryptosystèmes symétriques. Très souvent, ces constructions sont les seules composantes non-linéaires d'un chiffrement par bloc ou d'une fonction de hachage. Sans elles, la fonction aurait été entièrement linéaire et sa sécurité serait réduite à la résolution d'un système linéaire. Après un premier passage dans la couche non-linéaire de la fonction, toutes les variables de l'état dépendent en général de façon non-linéaire des variables d'entrée. Cependant, en fixant certains bits d'entrée à une valeur bien choisie il est possible que certains bits de la sortie puissent être exprimés comme des combinaisons affines des variables de départ. Ceci est étroitement lié aux propriétés des boîtes-S utilisées.

Dans ce chapitre, nous introduisons une nouvelle notion qui exprime le fait que certaines composantes d'une boîte-S sont de degré 1 sur un espace vectoriel et tous ses translatés. L'existence de ce type de propriété est liée aux certaines caractéristiques de la boîte-S, que nous analysons par la suite. Puis, nous nous concentrons sur les permutations à 4 variables, puisque ce sont des objets qui sont largement utilisés pour la construction de cryptosystèmes symétriques. Nous montrons enfin que cette nouvelle notion peut être au coeur de certaines cryptanalyses. En particulier, de tels espaces peuvent être utilisés pour propager des relations affines à travers un système. Nous illustrons cela par l'attaque présentée par Thomas Fuhr sur Hamsi-256 [Fuh10], et nous montrons que la description de tous les espaces ayant cette propriété permet de trouver efficacement des relations de la forme souhaitée.

6.1 La notion de la (v, w)-linéarité

Nous commençons par introduire la notion de la (v, w)-linéarité et par présenter quelques propriétés générales des fonctions (v, w)-linéaires.

6.1.1 Définition et propriétés générales

Définition 6.1. *Soit S une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors, S est dite* (v, w)-linéaire *s'il existe deux sous-espaces $V \subset \mathbf{F}_2^n$ et $W \subset \mathbf{F}_2^m$ avec $\dim V = v$ et $\dim W = w$ tels que pour tout $\lambda \in W$, S_λ est de degré au plus 1 sur tous les translatés de V, où S_λ est la fonction booléenne $x \mapsto \lambda \cdot S(x)$.*

Remarque 6.1. *Il est facile de voir d'après cette définition qu'une fonction S qui est (v, w)-linéaire est également (v, i)-linéaire pour tout $1 \leq i < w$. De même, elle est (i, w)-linéaire pour tout $1 \leq i < v$.*

Cette notion est reliée à la notion de la normalité (voir section 2.1.2).

Proposition 6.1. *Soit S une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Si S est* (v, w)-linéaire*, alors S a w coordonnées v-faiblement normales. En particulier, $\mathcal{L}(S) \geq 2^v$.*

Démonstration. Voir proposition 2.4. □

Toutefois, le critère de la $(v, 1)$-linéarité est plus fort que celui de la faible v-normalité. En effet, pour toute fonction $(v, 1)$-linéaire, la composante définie par W de dimension 1 est de degré au plus 1 sur tous les translatés de V, tandis que pour une fonction v-faiblement normale cette propriété n'est exigée que pour un seul translaté.

Proposition 6.2. *Soit f une fonction booléenne à n variables et soit V un sous-espace de $\mathbf{F}_2^n$. Si f est de degré au plus d sur tous les translatés de V, alors*

$$\deg f \leq d + n - \dim V.$$

Démonstration. Nous procédons par récurrence sur $n - \dim V$. Pour $\dim V = n$, la propriété est évidente. Supposons maintenant que cette propriété est vraie pour tout sous-espace V avec $\dim V > v$ et nous allons prouver qu'elle est également vérifiée pour tout V de dimension v. Pour tout sous-ensemble V de dimension v, il existe un vecteur e_i de n bits et de poids 1 tel que $e_i \notin V$. Soit $V' = V \times \langle e_i \rangle$. Alors, la restriction de f sur un translaté quelconque de $a + V'$ peut être écrite comme

$$f_{|a+V'}(x_1, \ldots, x_n) = x_i f_{|V+e_i+a}(x_1, \ldots, x_n) + (1 + x_i) f_{|V+a}(x_1, \ldots, x_n).$$

Par hypothèse, toutes les restrictions $f_{|V+c}$, $c \in \mathbf{F}_2^n$ sont de degré au plus d, impliquant que toutes les restrictions $f_{a+V'}$ sont de degré au plus $(d+1)$. Alors, par l'hypothèse de récurrence nous avons

$$\deg f \leq (d + 1) + n - (v + 1) = d + n - v.$$

□

Nous en déduisons le corollaire suivant.

Corollaire 6.1. *Soit S une fonction de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Si S est (v,w)-linéaire alors elle possède au moins w coordonnées de degré au plus $n+1-v$.*

Il convient de noter que les deux conditions dérivées des propositions 6.1 et 6.2, c'est-à-dire $\deg(f) \leq n+1-v$ et $\mathcal{L}(f) \geq 2^v$ ne sont pas suffisantes pour garantir que f est $(v,1)$-linéaire. Par exemple, il a été prouvé dans [CDDL06] que la fonction booléenne à 14 variables $f(x) = \mathrm{Tr}(\alpha x^{57})$ avec $\alpha \in \mathbf{F}_4 \setminus \mathbf{F}_2$ n'est pas 7-faiblement normale. En conséquence, cette fonction n'est pas $(7,1)$-linéaire. Toutefois, elle est de degré 4 et satisfait $\mathcal{L}(f) = 2^7$.

6.1.2 Lien avec la construction Maiorana-McFarland et les dérivées du deuxième ordre

Il est évident que toute fonction booléenne $(v,1)$-linéaire f peut s'écrire comme

$$f(u,v) = \pi(u)\cdot v + h(u), \quad (u,v) \in U \times V,$$

où U est un espace supplémentaire de V, π est une fonction de U dans $\mathbf{F}_2^v$ et h est une fonction booléenne de U dans $\mathbf{F}_2$. Cette construction est une généralisation bien connue de la *construction Maiorana-McFarland* des fonctions courbes [McF73]. Cette classe de fonctions a été généralisée aux fonctions vectorielles [Nyb91] et a été étudiée par plusieurs autres auteurs, par exemple dans [CP04]. Il s'ensuit qu'une fonction est (u,w)-linéaire si et seulement si 2^w de ses composantes définissent une fonction qui est équivalente à une fonction vectorielle Maiorana-McFarland, dans le sens de la proposition suivante.

Proposition 6.3. *Soit S une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Il existe deux sous-espaces $V \subset \mathbf{F}_2^n$ et $W \subset \mathbf{F}_2^m$ avec $\dim V = v$ et $\dim W = w$ tels que, pour tout $\lambda \in W$, S_λ est de degré au plus 1 sur tous les translatés de V si et seulement si la fonction S_W correspondant à toutes les composantes S_λ, $\lambda \in W$ peut être écrite sous la forme*

$$S_W(u,v) = M(u)v + G(u),$$

où $U \times V = \mathbf{F}_2^n$, G est une fonction de U dans $\mathbf{F}_2^w$ et $M(u)$ est une matrice $w \times v$ binaire dont les coefficients sont des fonctions booléennes définies sur U.

Il est intéressant de noter que avec cette représentation on voit directement que le degré de S_w est au plus $(\dim U + 1) = n + 1 - v$.

Il est connu que les fonctions booléennes qui sont équivalentes à une fonction Maiorana-McFarland peuvent être caractérisées à l'aide de leurs dérivées du deuxième ordre. Ceci est similaire pour les fonctions vectorielles.

Lemme 6.1. *Soit S une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors, S est de degré au plus 1 si et seulement si toutes ses dérivées du deuxième ordre $D_\alpha D_\beta S$, $\alpha, \beta \in \mathbf{F}_2^n$ s'annulent.*

Démonstration. Il est évident que les dérivées du deuxième ordre d'une fonction de degré au plus 1 s'annulent. La réciproque repose sur le fait que toute fonction de degré 2 ou plus possède une dérivée du deuxième ordre non triviale. En effet, si $\deg S > 1$, alors S a une composante $S_\lambda : x \mapsto \lambda \cdot x$, dont la forme algébrique normale contient un monôme de la forme $x_{i_1}x_{i_2}m(x)$, où m est un monôme non-nul. Par conséquent, S_λ peut se décomposer comme suit :

$$S_\lambda(x) = x_{i_1}x_{i_2}g(x) + h(x),$$

où g est une fonction booléenne non triviale qui ne dépend pas des x_{i_1}, x_{i_2} et h une fonction booléenne qui ne contient aucune monôme multiple de $x_{i_1}x_{i_2}$. Nous voyons que la dérivée du deuxième ordre de S_λ par rapport à e_{i_1} et e_{i_2} correspond exactement à g, qui ne s'annule pas. □

Proposition 6.4. *Soit S une fonction vectorielle de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$. Alors, S est (v, w)-linéaire pour un $2 \leq v \leq n$ et un $1 \leq w \leq m$ si et seulement s'il existe un sous-ensemble de w composantes indépendantes de S, $S_W = (S_{i_1}, \ldots, S_{i_w})$ et un sous-espace V de dimension v tel que toutes les dérivées du deuxième ordre de S_W, $D_\alpha D_\beta S_W$ avec $\alpha, \beta \in V$ s'annulent.*

Démonstration. Notons U un espace supplémentaire de V. Si S_W est de degré au plus 1 sur tout translaté $u + V$, alors pour tout $u \in U$,

$$S_W(u + v) = L_u(v) + \gamma(u),$$

où L_u est une fonction linéaire de V dans $\mathbf{F}_2^w$ et $\gamma(u) \in \mathbf{F}_2^w$. En conséquence, pour tout $\alpha, \beta \in V$, nous avons

$$D_\alpha D_\beta S_W(u + v) = S_W(u + v) + S_W(u + v + \alpha) + S_W(u + v + \beta) + S_W(u + v + \alpha + \beta) = 0$$

puisque toutes les quatre entrées participant à la somme appartiennent à $u + V$.

Réciproquement, si toutes ses dérivées du deuxième ordre de S_W, $D_\alpha D_\beta S_W$, s'annulent, alors pour tout $u \in U$, la fonction F_u de V dans $\mathbf{F}_2^w$ définie par $F_u(v) = S_W(u + v)$ est telle que toutes les dérivées du deuxième ordre s'annulent. Par conséquent, pour tout $u \in U$, S_W est de degré au plus 1 sur $u + V$. □

Par la suite, nous examinons plus en détail une classe des fonctions (v, w)-linéaires, celle des fonctions $(n - 1, 1)$-linéaires.

6.1.3 Fonctions $(n - 1, 1)$-linéaires

Le cas des fonctions $(n - 1, 1)$-linéaires peut être entièrement caractérisé.

Proposition 6.5. *Soit f une fonction booléenne à n variables. Alors, f est $(n-1, 1)$-linéaire si et seulement si $\deg f \leq 2$ et $\mathcal{L}(f) \geq 2^{n-1}$. En outre, si $\deg(f) = 2$ et $\mathcal{L}(f) \geq 2^{n-1}$, il existe exactement trois hyperplans distincts H tels que f est de degré au plus 1 sur H et $\overline{H}$.*

Démonstration. Soit f une fonction $(n - 1, 1)$-linéaire. Alors, d'après la proposition 6.1 nous déduisons que sa linéarité est supérieure ou égale à 2^{n-1}. De plus, grâce à la proposition 6.2 nous avons que $\deg f \leq 1 + n - (n - 1) = 2$. Inversement, soit f une fonction booléenne quadratique (nous supposons que $\deg f = 2$ puisque le résultat est trivial pour une fonction constante ou affine). Toute fonction quadratique f satisfait $\mathcal{L}(f) = 2^{\frac{n+h}{2}}$, où $0 \leq h < n$ est la dimension de l'espace linéaire de f, $LS(f)$ (voir par exemple l'appendice 1 dans [CCCF01]), où

$$LS(f) = \{a \in \mathbf{F}_2^n : D_a f : x \mapsto f(x + a) + f(x) \text{ est constante}\}.$$

En outre, l'ensemble

$$LS^0(f) = \{a \in \mathbf{F}_2^n : D_a f = 0\},$$

est un sous-espace de $LS(f)$ de dimension $\dim LS(f)$ ou $(\dim LS(f) - 1)$. On sait que $\mathcal{L}(f) \geq 2^{n-1}$, qui est la valeur maximale pour une fonction de degré exactement 2 ($h = n - 2$). Donc,

$\mathcal{L}(f) = 2^{n-1}$ et d'après la relation de Parseval, il existe exactement 4 valeurs de α telles que $|\mathcal{F}(f+\varphi_a)| = 2^{n-1}$. Parmi ces quatre valeurs, trois ont le même signe, car la somme de tous les coefficients $\mathcal{F}(f+\varphi_\alpha)$ vaut $\pm 2^n$ (voir par exemple [CC03], proposition 1). Ces valeurs sont les éléments $\beta + LS(f)^\perp$, où $\beta = 0$ si $\dim LS^0(f) = \dim LS(f)$, et $\beta \in LS^0(f)^\perp \setminus LS(f)^\perp$ sinon. En effet, d'après le lemme V.2 dans [CCCF01] nous avons que

$$\begin{aligned}
\sum_{\alpha \in LS(f)^\perp} \mathcal{F}^2(f+\varphi_{\alpha+\beta}) &= 2^2 \sum_{e \in LS(f)} (-1)^{\beta \cdot e} \mathcal{F}(D_e f) \\
&= 2^2 \left(\sum_{e \in LS^0(f)} \mathcal{F}(D_e f) - \sum_{e \in LS(f) \setminus LS^0(f)} \mathcal{F}(D_e f) \right) \\
&= 2^{n+2+\dim LS(f)} \\
&= 2^{2n}.
\end{aligned}$$

Par conséquent, les valeurs $\mathcal{F}^2(f+\varphi_{\alpha+\beta})$ sont toutes les 4 égales à 2^{2n-2}. Puisque,

$$2\mathcal{F}((f+\varphi_\beta)|_{H_\alpha}) = \mathcal{F}(f+\varphi_\beta) + \mathcal{F}(f+\varphi_{\beta+\alpha})$$

et

$$2\mathcal{F}((f+\varphi_\beta)|_{\overline{H}_\alpha}) = \mathcal{F}(f+\varphi_\beta) - \mathcal{F}(f+\varphi_{\beta+\alpha}),$$

où H_α est l'hyperplan $\{0,\alpha\}^\perp$, $\alpha \neq 0$, nous déduisons que f est linéaire sur H_α et $\overline{H}_\alpha$ si et seulement s'il existe u_1, u_2, u_3 tels que

$$\mathcal{F}(f+\varphi_{u_1}) = \mathcal{F}(f+\varphi_{u_2}) = \mathcal{F}(f+\varphi_{u_3}) = (-1)^b 2^{n-1} \text{ et } \mathcal{F}(f+\varphi_{u_1+u_2+u_3})(-1)^{b+1} 2^{n-1}.$$

Par ailleurs, α peut être un élément quelconque dans $\{u_1+u_2, u_1+u_3, u_2+u_3\}$. En conséquence, nous avons que f est linéaire sur H_α et $\overline{H}_\alpha$ si et seulement si α est un élément non nul de $LS(f)^\perp$. □

D'après la proposition 6.1, nous avons que si une fonction S de $\mathbf{F}_2^n$ dans $\mathbf{F}_2^m$ est (v,w)-linéaire, alors $v \leq \log_2 \mathcal{L}(S)$. À cause de cette propriété nous pouvons prouver qu'il n'existe aucune permutation de $\mathbf{F}_2^n$ de non-linéarité optimale qui soit $(n-1,1)$-linéaire pour $n \geq 5$.

Corollaire 6.2. *Soit S une permutation de $\mathbf{F}_2^n$ de non-linéarité optimale, c'est-à-dire $\mathcal{L}(S) \leq 2^{\lceil \frac{n+1}{2} \rceil}$. Alors, si $n \geq 5$, S n'est pas $(n-1,1)$-linéaire.*

Démonstration. Si S possède une composante $(n-1,1)$-linéaire, alors

$$n-1 \leq \left\lceil \frac{n+1}{2} \right\rceil \leq \frac{n}{2} + 1.$$

Par conséquent, $\frac{n}{2} \leq 2$ et donc $n \leq 4$. □

Classification des permutations de $\mathbf{F}_2^4$

Les boîtes-S les plus utilisées dans les primitives symétriques sont les boîtes-S équilibrées à 4 variables. Nous allons alors nous concentrer sur ces structures particulières.

D'après la proposition précédente, une permutation S de $\mathbf{F}_2^4$ est $(3,1)$-linéaire si et seulement si elle possède une composante de degré inférieur ou égal à 2. En effet, toute permutation

S de $\mathbf{F}_2^4$ vérifie $\mathcal{L}(S) \geq 8$, borne qui correspond aux permutations de non-linéarité maximale. De plus, si S n'a aucune composante constante ou affine, alors le nombre de couples d'espaces $(H, \{0, \lambda\})$ avec $\dim H = 3$ tel que $\deg(S_\lambda) \leq 1$ sur H et $\overline{H}$ est égal à $3Q_S$, où Q_S est le nombre de composantes quadratiques de S.

Nous pouvons donc essayer de classifier ces permutations en fonction de leur nombre de composantes quadratiques. Nous commençons par classifier leurs composantes, c'est-à-dire les fonctions booléennes à 4 variables.

Définition 6.2. *Soit f et g deux fonctions booléennes de n variables. Nous disons que f est* équivalente de façon affine *à g s'il existe une matrice $n \times n$ inversible A, $a, b \in \mathbf{F}_2^n$ et $c \in \mathbf{F}_2$ tels que pour chaque $x \in \mathbf{F}_2^n$*

$$g(x) = f(Ax + a) + bx + c.$$

Il a été démontré dans [BW72] que pour cette équivalence les 2^{32} fonctions booléennes à 5 variables peuvent être réduites à 49 classes d'équivalence. De ce même article se déduit une classification similaire pour les fonctions booléennes à 4 variables.

En particulier, nous pouvons voir à la table 6.1 que chacune des 2^{15} fonctions booléennes à 4 variables de degré au plus 3 est équivalente à une des cinq fonctions suivantes : $x_1x_2x_3, x_1x_2x_3 + x_1x_4, x_1x_2, x_1x_2 + x_3x_4$ et la fonction nulle.

Ce type de classification préserve le multi-ensemble composé de tous les coefficients de Walsh pour une fonction, c'est-à-dire toutes les fonctions appartenant à la même classe d'équivalence partagent le même multi-ensemble

$$\big|\{\mathcal{F}(f + \varphi_a), a \in \mathbf{F}_2^n\}\big|.$$

		Spectre de Walsh				
classe	représentant	±16	±12	±8	±4	0
I	$x_1x_2x_3$		1		7	8
II	$x_1x_2x_3 + x_1x_4$			2	8	6
III	x_1x_2			4		12
IV	$x_1x_2 + x_3x_4$				16	0
V	0	1				15

TABLE 6.1 – Nombre d'apparitions de chaque valeur dans le spectre de Walsh de chacune des cinq classes d'équivalence pour les fonctions booléennes à 4 variables de degré au plus 3.

Il est intéressant de noter que dans une classe d'équivalence les valeurs absolues des coefficients de Walsh sont les mêmes pour toutes les fonctions de la classe, mais leurs signes peuvent varier. Les signes sont seulement préservés sous une équivalence linéaire, c'est-à-dire quand $a = 0$ et $c = 0$.

Proposition 6.6. *Soit S une permutation de $\mathbf{F}_2^4$ dont aucune composante n'est constante ou affine. Alors, S possède c_I composantes de la classe I, c_{II} composantes de la classe II et c_{III} composantes de la classe III, avec*

$$c_I + c_{II} + c_{III} = 15.$$

De plus, c_{III} est le nombre de composantes quadratiques de S et est de la forme $c_{III} = 2^r - 1$, $0 \leq r \leq 4$. Enfin, $\mathcal{L}(S) = 8$, si et seulement si $c_I = 0$.

Démonstration. Comme S est une permutation, toutes ses composantes sont de degré au plus 3 et sont équivalentes à une des cinq classes précédemment décrites. Par hypothèse, comme aucune composante de S n'est constante ou affine, S ne possède aucune composante de la classe V. De même, comme toutes les composantes non-triviales d'une permutation sont équilibrées, S ne possède aucune composante de la classe IV. Le nombre de composantes non-triviales de la permutation S est égal à 15. Par conséquent, $c_I + c_{II} + c_{III} = 15$. Comme la valeur de $\mathcal{L}(S)$ correspond à la plus grande valeur du spectre de Walsh de toutes les composantes, $\mathcal{L}(S) = 8$ si et seulement si $c_I = 0$. La classe III correspond à des fonctions quadratiques. En conséquence, c_{III} correspond au nombre de composantes quadratiques Q_S de S. Comme les valeurs de λ telles que $\deg S_\lambda \leq 2$ forment un sous-espace vectoriel de $\mathbf{F}_2^4$, Q_S est de la forme $2^r - 1$. □

Nous prouvons ensuite qu'une permutation et son inverse ont le même nombre de composantes quadratiques.

Proposition 6.7. *Soit S une permutation de $\mathbf{F}_2^4$ avec aucune composante non-triviale de degré inférieur ou égal à 1. Alors, S et son inverse ont le même nombre de composantes de degré 2.*

Démonstration. Notons comme précédemment c_I, c_{II} et c_{III} le nombre de composantes non-triviales de S dans les trois premières classes décrites dans la table 6.1. Pour $0 \leq i < 16$ notons W_i le nombre d'apparitions de i dans l'ensemble

$$\{|\mathcal{F}(S_\lambda + \varphi_\alpha)|, \alpha, \lambda \in \mathbf{F}_2^4\}.$$

D'après la table 6.1 nous avons que

$$\begin{aligned} W_{12} &= c_I \\ W_8 &= 2c_{II} + 4c_{III} = 30 - 2c_I + 2c_{III} \end{aligned}$$

en impliquant que le nombre de composantes quadratiques de S est donné par

$$c_{III} = W_{12} + \frac{1}{2}W_8 - 15.$$

Comme la permutation inverse de S a le même ensemble $(W_i)_{0 \leq i \leq 16}$ que S, les deux permutations ont le même nombre de composantes quadratiques. □

Il est intéressant de noter que le résultat précédent ne s'applique pas dans le cas général. Par exemple, si n est un nombre impair, la fonction x^3 sur le corps fini $\mathbf{F}_{2^n}$ peut être vue comme une permutation de $\mathbf{F}_2^n$. Toutes ses composantes sont de degré 2. La permutation inverse correspond à la fonction x^d avec $d = \frac{2^{n+1}-1}{3}$, impliquant que toutes ses composantes sont de degré $(n+1)/2$.

Cas des permutations de $\mathbf{F}_2^4$ avec $\mathcal{L}(S) = 8$

Dans un travail présenté en 2007 par Gregor Leander et Axel Poschmann [LP07] une analyse des boîtes-S 4×4 *optimales* est effectué. Les auteurs appellent une fonction S de $\mathbf{F}_2^4$ dans $\mathbf{F}_2^4$ *optimale* si les conditions suivantes sont vérifiées :

– S est une permutation

– $\mathcal{L}(S) = 8$.
– $\delta(S) = 4$,

où $\delta(S) = \max_{a \neq 0} \delta(a, b)$ est l'*uniformité différentielle* de S, avec

$$\delta(a, b) = \#\{x \in \mathbf{F}_2^n, S(x) + S(x + a) = b\}.$$

Dans ce travail une recherche exhaustive des boîtes-S 4×4 *optimales* est présentée. Ils ont alors prouvé que dans le sens d'équivalence que nous avons introduit dans la section précédente, il n'existe que 16 classes d'équivalence différentes pour les boîtes-S optimales. Plus précisément, il existe :

– 4 boîtes-S optimales avec $Q_S = 3$.
– 4 boîtes-S optimales avec $Q_S = 1$.
– 8 boîtes-S optimales avec $Q_S = 0$.

Par exemple, la boîte-S de la fonction de hachage Hamsi que nous allons analyser à la section suivante, est une boîte-S optimale et possède 3 composantes quadratiques.

Nous avons alors effectué une recherche exhaustive parmi toutes les permutations de $\mathbf{F}_2^4$ afin de déterminer s'il existe des permutations avec $\mathcal{L}(S) = 2^8$ et $Q_S \in \{7, 15\}$. Une seule classe de permutations a été trouvée avec 7 composantes quadratiques et 8 composantes cubiques du type II. Nous avons aussi montré que toute permutation quadratique de $\mathbf{F}_2^4$ a une composante non-triviale de degré 1. Cette recherche nous a permis par ailleurs de classifier toutes les permutations de $\mathbf{F}_2^4$ selon les classes de leurs composantes. Nous disons que deux permutations S et S' caractérisées par les tuples $(c_I, c_{II}, c_{III}, c_V)$ $((c'_I, c'_{II}, c'_{III}, c'_V)$ respectivement) sont équivalentes si et seulement si $c_I = c'_I, c_{II} = c'_{II}, c_{III} = c'_{III}$ et $c_V = c'_V$. Nous avons pu identifié 53 classes d'équivalence différentes. Ces classes, un représentant pour chaque classe, ainsi que la valeur de $\mathcal{L}(S)$ correspondant sont illustrés à la table 6.2.

Les trois classes identifiées dans [LP07] correspondent aux numéros 41, 42 et 44.

	représentant	c_I	c_{II}	c_{III}	c_V	$\mathcal{L}(S)$
1	$\{0,1,2,3,4,5,6,7,8,9,10,14,11,12,13,15\}$	12	2	0	1	16
2	$\{0,1,2,3,4,5,6,7,8,9,10,11,12,15,13,14\}$	12	0	0	3	16
3	$\{0,1,2,3,4,5,6,12,8,7,9,10,11,13,14,15\}$	10	5	0	0	12
4	$\{0,1,2,3,4,5,6,7,8,9,10,12,11,15,13,14\}$	8	6	0	1	16
5	$\{0,1,2,3,4,5,6,7,8,9,10,15,11,12,13,14\}$	8	4	2	1	16
6	$\{0,1,2,3,4,5,6,7,8,9,10,12,11,13,15,14\}$	8	4	0	3	16
7	$\{0,1,2,3,4,5,15,14,8,9,6,7,10,11,12,13\}$	8	0	7	0	12
8	$\{0,1,2,3,4,5,6,7,8,9,15,14,10,11,12,13\}$	8	0	6	1	16
9	$\{0,1,2,3,4,5,6,7,8,9,10,11,15,12,13,14\}$	8	0	4	3	16
10	$\{0,1,2,3,4,5,6,7,8,9,10,11,12,13,15,14\}$	8	0	0	7	16
11	$\{0,1,2,3,4,5,6,9,8,7,10,14,11,12,13,15\}$	7	8	0	0	12
12	$\{0,1,2,3,4,5,6,12,8,7,9,10,11,13,15,14\}$	7	7	1	0	12
13	$\{0,1,2,3,4,5,6,9,8,7,15,10,11,12,13,14\}$	6	9	0	0	12
14	$\{0,1,2,3,4,5,6,9,8,7,14,10,11,15,12,13\}$	6	8	1	0	12
15	$\{0,1,2,3,4,5,6,9,8,7,10,12,11,13,15,14\}$	6	8	0	1	16
16	$\{0,1,2,3,4,5,6,9,8,7,15,14,10,11,12,13\}$	6	6	3	0	12

17	$\{0,1,2,3,4,5,6,9,8,7,10,11,15,12,13,14\}$	6	6	2	1	16
18	$\{0,1,2,3,4,5,6,9,8,7,10,12,11,15,13,14\}$	5	10	0	0	12
19	$\{0,1,2,3,4,5,6,9,8,7,10,15,11,12,13,14\}$	5	9	1	0	12
20	$\{0,1,2,3,4,5,6,12,8,7,9,10,11,14,15,13\}$	4	11	0	0	12
21	$\{0,1,2,3,4,5,6,12,8,7,9,10,11,15,14,13\}$	4	10	1	0	12
22	$\{0,1,2,3,4,5,6,7,8,9,15,12,10,14,11,13\}$	4	10	0	1	16
23	$\{0,1,2,3,4,5,6,12,8,7,9,10,11,15,13,14\}$	4	8	3	0	12
24	$\{0,1,2,3,4,5,6,7,8,9,12,10,11,15,13,14\}$	4	8	2	1	16
25	$\{0,1,2,3,4,5,6,7,8,9,15,10,11,14,12,13\}$	4	4	6	1	16
26	$\{0,1,2,3,4,5,6,7,8,9,10,13,11,12,15,14\}$	4	4	4	3	16
27	$\{0,1,2,3,4,5,6,9,8,7,15,12,10,14,11,13\}$	3	12	0	0	12
28	$\{0,1,2,3,4,5,6,9,8,7,12,10,11,15,13,14\}$	3	11	1	0	12
29	$\{0,1,2,3,4,5,6,9,8,7,15,10,11,14,12,13\}$	3	9	3	0	12
30	$\{0,1,2,3,4,5,9,6,8,7,12,15,10,14,11,13\}$	2	13	0	0	12
31	$\{0,1,2,3,4,5,9,6,8,7,12,14,10,15,11,13\}$	2	12	1	0	12
32	$\{0,1,2,3,4,5,6,9,8,7,15,13,10,14,11,12\}$	2	12	0	1	16
33	$\{0,1,2,3,4,5,9,6,8,7,10,15,11,14,12,13\}$	2	10	3	0	12
34	$\{0,1,2,3,4,5,7,6,8,10,9,14,11,12,13,15\}$	2	10	2	1	16
35	$\{0,1,2,3,4,5,9,6,8,7,15,14,10,11,12,13\}$	2	6	7	0	12
36	$\{0,1,2,3,4,5,6,9,8,7,11,10,15,14,12,13\}$	2	6	6	1	16
37	$\{0,1,2,3,4,5,9,12,8,6,7,10,11,15,13,14\}$	1	14	0	0	12
38	$\{0,1,2,3,4,5,9,6,8,7,15,12,10,14,11,13\}$	1	13	1	0	12
39	$\{0,1,2,3,4,5,12,6,8,7,9,13,10,15,11,14\}$	1	11	3	0	12
40	$\{0,1,2,3,4,5,6,13,8,7,9,12,10,15,11,14\}$	1	7	7	0	12
41	$\{0,1,2,3,4,5,15,10,8,6,7,12,9,11,13,14\}$	0	15	0	0	8
42	$\{0,1,2,3,4,5,12,9,8,6,7,15,10,14,11,13\}$	0	14	1	0	8
43	$\{0,1,2,3,4,5,9,11,8,6,7,10,12,15,13,14\}$	0	14	0	1	16
44	$\{0,1,2,3,4,5,9,13,8,6,7,12,10,14,15,11\}$	0	12	3	0	8
45	$\{0,1,2,3,4,5,7,6,8,9,12,14,10,15,11,13\}$	0	12	2	1	16
46	$\{0,1,2,3,4,5,6,7,8,9,15,13,10,14,11,12\}$	0	12	0	3	16
47	$\{0,1,2,3,4,5,12,13,8,6,7,9,10,14,15,11\}$	0	8	7	0	8
48	$\{0,1,2,3,4,5,7,6,8,12,9,13,10,15,11,14\}$	0	8	6	1	16
49	$\{0,1,2,3,4,5,7,6,8,10,9,11,12,15,13,14\}$	0	8	4	3	16
50	$\{0,1,2,3,4,5,6,7,8,13,9,12,10,15,11,14\}$	0	0	14	1	16
51	$\{0,1,2,3,4,5,6,7,8,9,11,10,15,14,12,13\}$	0	0	12	3	16
52	$\{0,1,2,3,4,5,6,7,8,9,10,11,13,12,15,14\}$	0	0	8	7	16
53	$\{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$	0	0	0	15	16

TABLE 6.2 – Description de toutes les classes de permutations de $\mathbf{F}_2^4$.

Quelques familles infinies de fonctions booléennes de degré au plus $(n-1)$ ne possédant aucune dérivée du deuxième ordre constante, sont données dans le lemme 3 dans [CC03]. La caractérisation des permutations non-quadratiques qui n'ont pas de dérivées du deuxième ordre constantes est également mentionnée comme un problème ouvert dans [CC03] (page 2016).

Dans la section suivante nous allons montrer comme la notion de la (v, w)-linéarité peut être exploitée afin de cryptanalyser la fonction de hachage Hamsi.

6.2 Application à Hamsi

La fonction de hachage Hamsi a été conçue par Özgül Küçük [Kuc09] pour le concours SHA-3. Sa description complète se trouve à la section 3.7.1. Comme nous l'avons déjà mentionné à la section 3.7 cette fonction a la particularité d'avoir un degré algébrique très faible en combinaison avec un nombre de tours réduit. Ces faiblesses ont été exploitées par Thomas Fuhr [Fuh10] et par Itai Dinur et Adi Shamir [DS11] afin de trouver des deuxièmes préimages pour l'intégralité de la fonction de hachage. Nous proposons ici une amélioration de la méthode de Thomas Fuhr en utilisant, en partie, les résultats de la section précédente. Plus précisément, nous montrons comment généraliser la méthode de [Fuh10] afin de trouver des secondes préimages pour Hamsi-256, avec une complexité moins élevée.

Nous commençons par présenter brièvement l'attaque de Thomas Fuhr. Plus de détails sur la méthode peuvent être retrouvés dans l'article d'origine [Fuh10] ou dans la thèse de doctorat de Thomas Fuhr [Fuh11], où des améliorations sur les résultats présents dans [Fuh10] peuvent être observées.

6.2.1 Description de l'attaque de T. Fuhr

Thomas Fuhr a présenté dans [Fuh10] une méthode permettant de trouver des deuxièmes préimages pour Hamsi-256. Cette cryptanalyse, dont la complexité est égale à $2^{251.3}$ évaluations de la fonction de compression, a été la première attaque sur cette fonction ayant une complexité moins élevée que l'attaque générique, pour des messages de petite taille.

L'idée de cette cryptanalyse consiste à trouver des relations affines entre certains bits d'entrée et de sortie de la fonction de compression. Ceci a été possible dans le cas de Hamsi puisque le degré algébrique de la fonction de compression est relativement bas. Ces relations ont conduit à des préimages pour la fonction de compression de Hamsi-256. Ces préimages ont pu ensuite être transformées en secondes préimages pour la fonction de hachage en utilisant des techniques de *rencontre au milieu*.

La première phase de l'attaque consiste à trouver des relations linéaires entre les bits d'entrée et les bits de sortie de la fonction de compression.

La description de la boîte-S S utilisée dans Hamsi, en termes de forme algébrique normale de ses coordonnées est donnée par :

$$\begin{aligned} y_0 &= x_0x_2 + x_1 + x_2 + x_3 \\ y_1 &= x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2 + x_0x_3 + x_2x_3 + x_0 + x_1 + x_2 \\ y_2 &= x_0x_1x_3 + x_0x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_0 + x_1 + x_3 \\ y_3 &= x_0x_1x_2 + x_1x_3 + x_0 + x_1 + x_2 + 1. \end{aligned}$$

Fuhr a alors remarqué que

$$S(1, x, 0, 1 + x) = (1, 0, 0, x), \text{ pour tout } x \in \mathbf{F}_2, \tag{6.1}$$

où le bit de poids faible se trouve à gauche.

En prenant en compte cette propriété il est possible de choisir un ensemble de variables de la manière suivante. On rappelle que l'état interne de la fonction de compression de Hamsi est représenté par une matrice 4×4 de 16 mots de 32 bits (figure 6.1). Si $y \in \mathbf{F}_2^{32}$ nous désignons par y^j le j-ième bit de y.

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

m_0	m_1	c_0	c_1
c_2	c_3	m_2	m_3
m_4	m_5	c_4	c_5
c_6	c_7	m_6	m_7

FIGURE 6.1 – L'état interne de la fonction de compression de Hamsi-256

Si le bloc de message après l'addition des constantes du premier tour est tel que $s_0^j = 1$ et $s_8^j = 0$, alors nous pouvons définir une variable $x^j \in \mathbf{F}_2$ et fixer $s_4^j = x^j$ et $s_{12}^j = 1 + x^j$. Grâce à l'équation (6.1), après l'application de la couche non-linéaire du premier tour, seulement le bit s_{12}^j dépendra de x^j. Cela présente un intérêt particulier puisque s_{12}^j fera partie d'un mot d pendant l'application de la fonction linéaire L. Les bits des mots d diffusent moins rapidement par L que les variables qui se trouvent dans les mots a ou c. Les mêmes remarques s'appliquent à la colonne voisine de l'état, c'est-à-dire aux mots s_1, s_5, s_9, s_{13}. Si $s_1^j = 1$ et $s_9^j = 0$, nous définissons une variable $y^j \in \mathbf{F}_2$ et nous fixons ensuite $s_5^j = y^j$ et $s_{13}^j = 1 + y^j$.

Afin de monter l'attaque, un bloc de message est pris au hasard. La première étape consiste à choisir l'ensemble de variables $\mathcal{I} = X \cup Y$, où X correspond à l'ensemble de variables de la première colonne de l'état et Y aux variables de la deuxième colonne. Pour cela, les valeurs de s_0, s_1, s_8 et s_9 sont calculées. Si $s_0^j = 1$ et $s_8^j = 0$ alors la variable x^j est ajoutée à X. De la même façon, si $s_1^j = 1$ et $s_9^j = 0$, alors la variable y^j est ajoutée à Y.

Une fois l'ensemble de variables fixé, il faut chercher un ensemble de bits de sortie de la fonction de compression, $\mathcal{O}$, tel que chaque bit dans cet ensemble puisse être exprimé comme une fonction affine des variables de $\mathcal{I}$.

Grâce à la relation (6.1), après un tour de la permutation interne, tous les bits de l'état dépendent de façon linéaire des variables de l'ensemble $\mathcal{I}$. Ensuite, les seules composantes qui peuvent introduire de la non-linéarité sont les couches des boîtes-S du deuxième et troisième tour. Cependant, sous certaines conditions, certains bits de sortie d'une boîte-S peuvent toujours être exprimés comme une combinaison affine des variables de l'ensemble $\mathcal{I}$. Ces conditions, identifiées dans [Fuh10] sont les suivantes :

- Tous les bits d'entrée de la boîte-S, sauf un, sont constants. Si ce bit peut être exprimé comme une combinaison affine des variables de $\mathcal{I}$, alors ceci sera également le cas pour les quatre bits de sortie de la boîte-S.
- Si tous les bits d'entrée de la boîte-S dépendent d'au plus une variable de $\mathcal{I}$, alors tous les bits de sortie de la boîte-S dépendront également de cette variable.
- Si aucune de deux premières situations ne se produit, cela signifie qu'il existe au moins deux bits d'entrée à la boîte-S qui dépendent d'au moins deux variables différentes de $\mathcal{I}$. En examinant l'ANF des quatre coordonnées de S nous pouvons faire les remarques suivantes. Le seul terme non-linéaire du premier bit de sortie y_0 est x_0x_2. Si ce monôme est une combinaison affine des variables de $\mathcal{I}$, alors ceci sera également le cas pour y_0.

De la même façon, si $x_0x_1x_2$ et x_1x_3 sont affines en les variables de $\mathcal{I}$, il en sera de même pour y_3.

Ces propriétés ont été utilisées par Thomas Fuhr afin de chercher un ensemble de variables $\mathcal{I}$ et un ensemble de bits de sortie $\mathcal{O}$ s'exprimant comme une combinaison affine des bits d'entrée. La méthode de recherche consistait à fixer le nombre de variables $N_{var} = |\mathcal{I}|$ et à lancer ensuite une recherche automatique afin de déterminer l'ensemble de variables qui produisait le plus grand nombre de bits de sortie, N_{out}, affines en $\mathcal{I}$.

Nous allons montrer à la section 6.2.2 comment nous pouvons améliorer cette étape, c'est-à-dire la recherche des relations linéaires, en exploitant la notion de la (v, w)-linéarité.

Préimages pour la fonction de compression

L'étape suivante consiste en la recherche de préimages pour la fonction de compression de Hamsi-256. Étant donnée une variable de chaînage h^*, nous cherchons alors un bloc de message m et une variable de chaînage h de façon que $f(m, h) = h^*$, où f est la fonction de compression de Hamsi-256.

Par l'étape précédente, nous possédons N_{out} bits de sortie $z_0, \ldots, z_{N_{out}-1}$ et N_{var} bits d'entrée $x_0, \ldots, x_{N_{var}-1}$ tels que les premiers s'expriment en fonction de derniers de façon affine. L'algorithme proposé par Thomas Fuhr pour trouver maintenant des préimages pour la fonction de compression de Hamsi-256 est assez simple et peut être résumé dans les étapes suivantes.

1. Choisir un message m et une valeur de chaînage h tels que les conditions imposées par l'équation (6.1) pour les positions indiquées par les variables de $\mathcal{I}$ soient vérifiées.
2. Calculer les bits $z_0, \ldots, z_{N_{out}-1}$. Grâce à l'analyse précédente, nous sommes sûrs que ces bits sont des fonctions affines en les variables $x_0, \ldots, x_{var-1}$. Calculer les coefficients du système affine.
3. Résoudre le système affine. Si le système n'a pas de solution, choisir d'autres valeurs pour la partie constante de h (sans modifier la partie de h imposée par les conditions (6.1)) et recommencer. S'il n'y a pas de solution non plus, choisir un autre message m qui vérifie les mêmes contraintes qu'avant et recommencer.
4. Si le système affine a une solution (m, h), vérifier si $f(m, h) = h^*$. Cette équation a une solution avec probabilité $2^{N_{out}-256}$. Sinon, recommencer.

Une étude détaillée de la complexité de cet algorithme a été présentée dans [Fuh10]. Nous en rappelons certains points.

Étude de complexité

Le nombre moyen de solutions par système affine est $2^{N_{var}-N_{out}}$. Chaque solution permet de trouver une préimage avec probabilité $2^{N_{out}-256}$. Il faut alors générer en moyenne $2^{256-N_{var}}$ systèmes afin de trouver une préimage. Notons maintenant $\mathcal{C}_{build}$ la complexité de trouver les coefficients du système affine et par $\mathcal{C}_{solve}$ la complexité de la résolution du système. Enfin, notons par $\mathcal{C}_f$ la complexité de l'évaluation de la fonction de compression. La complexité de cet algorithme est alors donnée par :

$$\mathcal{C}_{preimage} = 2^{256-N_{var}}(\mathcal{C}_{build} + \mathcal{C}_{solve}) + 2^{256-N_{out}}\mathcal{C}_f. \tag{6.2}$$

Estimation de $\mathcal{C}_f$: Itai Dinur et Adi Shamir ont estimé dans [DS11] que le nombre d'opérations élémentaires effectuées pendant un appel à la fonction de compression de Hamsi-256 est 10500.

Estimation de $\mathcal{C}_{solve}$: Nous pouvons résoudre le système linéaire en utilisant l'algorithme de pivot de Gauss. La complexité de cet algorithme en nombre d'opérations est environ

$$N_{out}(N_{var}+1)N_{var}/2.$$

Ce nombre correspond à au plus $(N_{var}+1)$ additions nécessaires pour chacune des N_{out} équations et pour chaque variable du système. Le nombre moyen d'additions calculées est $N_{var}/2$.

Estimation de $\mathcal{C}_{build}$: Il s'agit de la phase la plus difficile à estimer. Trivialement $\mathcal{C}_{build} \leq (N_{var}+1)\mathcal{C}_f$ car il suffit d'évaluer la fonction de compression quand toutes les variables en entrée sont nulles pour retrouver les coefficients constants du systèmes, puis pour les mots de poids 1 en entrée pour retrouver les coefficients linéaires. Toutefois, Thomas Fuhr a montré qu'un grand nombre de ces opérations pouvaient être factorisées. Ainsi, une évaluation détaillée montre que pour $N_{var}=9$ et $N_{out}=11$, $\mathcal{C}_{build}$ est d'ordre de $2^{-3.2}\mathcal{C}_f$.

Secondes préimages pour la fonction de hachage

La dernière étape de l'attaque consiste à transformer les préimages pour la fonction de compression en deuxièmes préimages pour la fonction de hachage. L'approche la plus simple consiste à utiliser un algorithme générique de rencontre au milieu. Un tel algorithme a par exemple été proposé par Xuejia Lai et James Massey [LM93]. Si $\mathcal{C}_{preimage}$ est la complexité de construire une préimage pour la fonction de compression de Hamsi-256, l'algorithme de [LM93] donne une seconde préimage pour Hamsi-256 avec complexité $2 \cdot 2^{\frac{256}{2}}\sqrt{\mathcal{C}_{preimage}}$. Une méthode alternative a été appliquée par Thomas Fuhr dans [Fuh10] afin de tirer profit de la forme spécifique de l'attaque précédente. Cette méthode est basée sur l'observation suivante. La partie principale de la complexité de l'algorithme consiste à construire et à résoudre des systèmes linéaires. Si à la place d'essayer d'inverser des valeurs de chaînage une par une nous considérons un ensemble $\mathcal{H}$ de valeurs de chaînage et nous cherchons une préimage d'un élément quelconque de cet ensemble, alors la complexité de la recherche devient moins élevée. En effet, la génération des relations affines est identique pour tous les éléments de $\mathcal{H}$ et peut alors être effectuée en une seule fois.

6.2.2 Amélioration de l'attaque de T. Fuhr

Nous montrons maintenant comment rechercher plus efficacement des relations affines entre les bits d'entrée et les bits de sortie de la fonction de compression de Hamsi-256. Nos améliorations sont basées sur deux axes différents. Le premier axe concerne la façon dont la propagation à travers les boîtes-S du deuxième et troisième tour est gérée. La deuxième direction est liée à la façon de choisir quelles boîtes-S du premier tour seront affectées et de quelle façon. En particulier, une nouvelle propriété algébrique de la boîte-S de Hamsi est utilisée conjointement avec la relation (6.1) pour passer de façon linéaire à travers les boîtes-S du premier tour.

La (v, w)-linéarité de la boîte-S de Hamsi

Notons $x = (x_0, x_1, x_2, x_3)$ l'entrée de la boîte-S et $y = (y_0, y_1, y_2, y_3)$ sa sortie. Afin de traiter le cas quand au moins deux bits d'entrée d'une boîte-S du deuxième ou troisième tour sont affectés par au moins deux variables différentes de $\mathcal{I}$, Thomas Fuhr a exploité les deux propriétés algébriques suivantes de la boîte-S de Hamsi.

- y_0 est de degré au plus 1 si x_0x_2 est de degré au plus 1.
- y_3 est de degré au plus 1 si x_1x_3 et $x_0x_1x_2$ sont de degré du plus 1.

Ces deux propriétés peuvent être reformulées de la façon suivante, où tout vecteur x de $\mathbf{F}_2^4$ est représenté par l'entier $(\sum_{i=0}^{3} x_i 2^i)$.

- y_0 est de degré au plus 1 si x appartient à $V^\perp \subset \mathbf{F}_2^4$ avec $V = \langle 1 \rangle$, $V = \langle 4 \rangle$, ou $V = \langle 5 \rangle$, ou à un translaté quelconque d'un de ces trois hyperplans.
- y_3 est de degré au plus 1 si x appartient à un translaté quelconque de $V^\perp$, avec $V = \langle 1, 2 \rangle$, $V = \langle 2, 4 \rangle$ ou $V = \langle 2, 5 \rangle$.

Par ailleurs, il est possible de trouver d'autres sous-espaces de dimension 2 tels que certaines composantes de S sont de degré au plus 1 sur tous leurs translatés. Par la méthode fournie à la proposition 6.4 nous avons identifié tous ces sous-espaces, qui sont décrits à la table 6.3.

Nous pouvons alors remarquer que si on considère les composantes y_0 et y_3, les propriétés utilisées par Fuhr décrivent toute la liste des sous-espaces V tels que le degré de ces composantes est au plus 1 sur tous les translatés de V. Cependant, nous voyons qu'il est possible d'identifier d'autres composantes de S qui sont de degré au plus 1 sur ces mêmes sous-espaces. En particulier, nous voyons que la boîte-S de Hamsi est $(2, 3)$-linéaire. En effet, la restriction de S_λ à tout translaté de $\langle 1, 2 \rangle^\perp$ est de degré 1 pour $\lambda \in \{1, 6, 8\}$. Ces remarques peuvent être très utiles en pratique, puisque nous pouvons désormais garantir la propagation affine de certaines composantes de S, qui étaient rejetées dans [Fuh10]. Par exemple, nous pouvons observer que y_1 et y_2 sont de degré au plus 1 sur tous les translatés de trois sous-espaces différents de dimension 2. Ces cas, qui n'étaient pas traités dans [Fuh10], peuvent maintenant être utilisés afin de chercher des propagations affines à travers les boîtes-S du deuxième et troisième tour.

6.2.3 Relations affines pour trois tours de la fonction de compression

Dans [Fuh10], la propriété (6.1) a été utilisée pour garantir la propagation affine à travers la première couche des boîtes-S. Comme nous l'avons déjà mentionné, cette propriété assure qu'après le passage à travers la partie non-linéaire du premier tour, il existe au plus une variable par boîte-S *active*. Avec le terme *active* nous désignons toute boîte-S ayant en entrée au moins une variable. Dans le cas contraire, une boîte-S *non active* désignera une permutation dont le vecteur d'entrée est constant. De plus, la relation (6.1) garantit que cette unique variable appartiendra à un mot d de l'état. Cette notation provient de la façon dont nous représentons la fonction linéaire de Hamsi $L : \mathbf{F}_2^{128} \to \mathbf{F}_2^{128}$ qui prend en entrée quatre mots de 32 bits a, b, c et d. Il est facile de voir d'après la description de la fonction L que les variables qui appartiennent aux mots d se propagent beaucoup moins rapidement que les variables des

V	liste de λ
$\langle 1, 2\rangle$	$\{1, 6, 7, 8, 9, e, f\}$
$\langle 1, 4\rangle$	$\{1, e, f\}$
$\langle 1, 6\rangle$	$\{1, 4, 5, a, b, e, f\}$
$\langle 1, 8\rangle$	$\{1, e, f\}$
$\langle 1, a\rangle$	$\{1, 2, 3, c, d, e, f\}$
$\langle 1, c\rangle$	$\{1, e, f\}$
$\langle 1, e\rangle$	$\{1, e, f\}$
$\langle 2, 4\rangle$	$\{1, 8, 9\}$
$\langle 2, 5\rangle$	$\{1, 8, 9\}$
$\langle 2, 8\rangle$	$\{e\}$
$\langle 2, 9\rangle$	$\{e\}$
$\langle 2, c\rangle$	$\{f\}$
$\langle 2, d\rangle$	$\{f\}$
$\langle 3, 4\rangle$	$\{1, 6, 7\}$
$\langle 3, 5\rangle$	$\{1, 6, 7\}$
$\langle 3, 8\rangle$	$\{e\}$
$\langle 3, 9\rangle$	$\{e\}$
$\langle 3, c\rangle$	$\{f\}$
$\langle 3, d\rangle$	$\{f\}$
$\langle 4, 8\rangle$	$\{1, e, f\}$
$\langle 4, 9\rangle$	$\{1, e, f\}$
$\langle 4, a\rangle$	$\{1\}$
$\langle 4, b\rangle$	$\{1\}$
$\langle 5, 8\rangle$	$\{1, e, f\}$
$\langle 5, 9\rangle$	$\{1, e, f\}$
$\langle 5, a\rangle$	$\{1\}$
$\langle 5, b\rangle$	$\{1\}$
$\langle 6, 8\rangle$	$\{4, a, e\}$
$\langle 6, 9\rangle$	$\{4, a, e\}$
$\langle 6, a\rangle$	$\{2, d, f\}$
$\langle 6, b\rangle$	$\{3, c, f\}$
$\langle 7, 8\rangle$	$\{5, b, e\}$
$\langle 7, 9\rangle$	$\{5, b, e\}$
$\langle 7, a\rangle$	$\{2, d, f\}$
$\langle 7, b\rangle$	$\{3, c, f\}$

TABLE 6.3 – Liste de tous les $\lambda \in \mathbf{F}_2^4$ tels que $x \mapsto \lambda \cdot S(x)$ est de degré au plus 1 sur tous les translatés de $V^\perp$, pour chaque sous-espace V de dimension 2.

mots a et c. En particulier, chaque variable d'un mot d affecte au plus trois bits de l'état après l'application de la partie linéaire.

Toutefois, les variables des mots b ont la même propagation lente que les variables des mots d, et cette propriété n'a pas été exploitée dans [Fuh10]. En ce sens, la propriété suivante

de la boîte-S de Hamsi peut alors être très utile :

$$S(1, x, 0, x) = (0, x, 1, 0) \text{ pour tout } x \in \mathbf{F}_2. \tag{6.3}$$

Avant de présenter notre méthode de recherche des relations affines entre les variables d'entrée et de sortie de la fonction de compression de Hamsi-256, nous avons besoin d'une énumération des 512 bits de l'état. La notation que nous allons introduire sera ensuite utilisée pour présenter les résultats expérimentaux à la fin de cette section.

Notation : Nous désignons par 0 le bit le plus à gauche du mot s_0, par 31 le bit le plus à droite du même mot, par 32 le bit le plus à gauche du mot s_1, par 128 le bit le plus à gauche du mot s_4 etc. Cette notation est illustrée à la table 6.4. De la même façon, les boîtes-S, $S_0, \ldots, S_{127}$ sont énumérées de gauche à droite.

0.....31	32.....63	64.....95	96.....127
128 . . . 159	160 . . . 191	192 . . . 223	224 . . . 255
256 . . . 287	288 . . . 319	320 . . . 351	352 . . . 383
384 . . . 415	416 . . . 447	448 . . . 479	480 . . . 511

TABLE 6.4 – Énumération des bits de l'état de Hamsi-256

Nous cherchons un ensemble de variables $\mathcal{I}$ à l'entrée de la fonction de compression, tel que l'ensemble de bits en sortie de la fonction de compression, $\mathcal{O}$, qui sont affines en les variables de $\mathcal{I}$, soit maximisé. Le principal problème est alors la façon de choisir les boîtes-S de l'état pendant le premier tour qui seront actives. Afin de résoudre ce problème, nous suivons l'approche suivante.

Pour chacun de 256 bits de sortie b_i de la fonction de compression, nous avons étudié leur propagation en arrière à travers la fonction de compression, afin de déterminer les bits en entrée dont il dépend. Nous avons alors pu observer un phénomène intéressant. Pour chaque i il existe plusieurs bits de sortie de la couche non-linéaire du premier tour qui n'affectent pas b_i. Ce comportement constitue une faiblesse importante de la diffusion de la fonction et est principalement lié au fait que la permutation interne de Hamsi n'est constituée que de 3 tours.

Par exemple, en considérant b_0 nous pouvons identifier une liste de boîtes-S du premier tour qui ont au moins un bit de sortie qui n'affecte pas b_0. Cette liste est :

$$\{4, 8, 12, 15, 17, 23, 24, 26, 31, 45, 58, 59, 64, 66, 76, 78, 89, 103, 114, 117\}.$$

Dans cette liste, pour les boîtes-S dans $\{12, 15, 17, 26, 64, 66, 114, 117\}$ c'est la quatrième sortie y_3 qui n'influence pas b_0, tandis que pour les boîtes-S $\{4, 23, 31, 45, 58, 76, 78, 89, 103\}$, b_0 n'est pas affecté par y_1. Enfin, pour les boîtes-S $\{8, 24, 59\}$ ni la coordonnée y_1 ni y_3 n'influence b_0.

La même recherche peut être effectuée pour les 255 autres bits de sortie de la fonction de compression. Une première remarque que nous pouvons faire est qu'un bit de sortie b est rarement indépendant de la première ou troisième sortie (coordonnée y_0 et y_2 respectivement)

d'une boîte-S du premier tour. Par contre, il est courant de trouver des boîtes-S du premier tour dont la deuxième ou quatrième sortie (coordonnée y_1 et y_3 respectivement) n'affectent pas b. Ce comportement est clairement dû à la propagation plus rapide des variables se trouvant dans des mots a ou c, en comparaison des variables des mots b et d. L'idée est alors de trouver un ensemble de bits de sortie de la fonction de compression $\mathcal{O}$ qui partagent un grand nombre de boîtes-S du premier tour qui n'influencent que très peu de bits dans $\mathcal{O}$.

Nous allons cependant restreindre notre recherche aux 64 premières boîtes-S de l'état. La raison pour cela est que nous cherchons, exactement comme dans [Fuh10], une préimage h^* d'une valeur de chaînage donnée h. Par conséquent, nous choisissons des variables dans la partie de l'état interne qui, après la concaténation, contient des bits de la valeur de chaînage. En utilisant les relations (6.1) et (6.3) cette contrainte est vérifiée pour la première moitié de l'état. Au contraire, ceci n'est plus valide pour la deuxième moitié de l'état, puisque les rôles du message et de la valeur de chaînage sont inversés.

Il est évident que nous ne pouvons pas tester tous les ensembles de sortie possibles $\mathcal{O}$ pour tous les ensembles d'entrée possibles $\mathcal{I}$, à cause de la complexité trop élevée d'une telle recherche. Pour cette raison, nous avons adopté la stratégie suivante, décrite par l'algorithme 4. Nous choisissons au hasard une boîte-S du premier tour, parmi les 64 possibles, et nous éliminons tous les bits de sortie de la fonction de compression qui dépendent entièrement de chacune des sorties de cette boîte-S. Nous regardons ensuite les bits de sortie qui restent après cette première élimination et nous comptons le nombre de fois où la deuxième coordonnée de cette boîte-S n'influence pas ces bits. Nous faisons la même chose pour la quatrième coordonnée et nous choisissons le plus grand nombre parmi les deux. Nous éliminons ensuite les bits de sortie qui correspondent au plus petit nombre. Après cette nouvelle élimination, nous sommes sûrs que les bits qui restent dans notre liste ne sont pas influencés par la deuxième (ou la quatrième coordonnée) de cette boîte-S. Nous choisissons ensuite une deuxième boîte-S du premier tour et nous utilisons la même technique afin de restreindre encore un peu la liste de bits en sortie. Après cette étape, pour un nombre de variables d'entrée $N_{var} = 9$, seulement 12-14 bits restent dans notre liste. Nous notons la liste de ces bits $\mathcal{L}$. Pour $N_{var} = 9$ deux tours de cette étape sont suffisants pour avoir un nombre de bits en sortie raisonnable. Toutefois, pour un nombre plus petit de variables, nous pouvons répéter cette étape encore une ou deux fois. Au contraire, pour un nombre plus grand de variables, une seule étape aurait probablement suffit. Notons n_{et} le nombre de répétitions de cette procédure.

La deuxième étape consiste maintenant à trouver un nombre de boîtes-S du premier tour qui ont une coordonnée qui n'influence pas un grand nombre de bits de $\mathcal{L}$. Par exemple, pour 9 variables nous pouvons trouver un ensemble $\mathcal{S}$ de 20 boîtes-S du premier tour qui ont une coordonnée qui n'influence pas plus de 5 bits de $\mathcal{L}$. Pour chacune des boîtes-S dans $\mathcal{S}$, si la majorité des bits dans $\mathcal{L}$ ne sont pas influencés par la deuxième coordonnée de cette boîte-S, nous leur imposons alors la relation (6.1). Au contraire, si majoritairement les bits dans $\mathcal{L}$ ne sont pas influencés par la quatrième coordonnée de la boîte-S, nous imposons alors la relation (6.3). S'il n'est pas clair si la majorité appartient à la deuxième ou la quatrième coordonnée, nous fixons alors une des deux relations au hasard. Théoriquement, il est possible que l'indépendance provienne majoritairement d'une autre composante de la boîte-S, mais cela n'arrive en pratique que très rarement. Pour cette raison, nous ne traitons pas du tout ce cas.

Après avoir fixé une relation d'entrée pour toutes les boîtes-S dans S, nous cherchons exhaustivement une combinaison de 9 boîtes-S, c'est-à-dire de 9 variables dans $\mathcal{I}$ qui donnent le plus grand nombre de relations linéaires. Pour chaque combinaison de N_{var} boîtes-S, nous procédons comme suit. Pour chaque colonne de l'état qui correspond à une boîte-S active,

nous fixons les bits de l'état après la première couche non-linéaire selon la relation (6.1) ou la relation (6.3). Dans le premier cas, nous fixons comme variable le quatrième bit de la colonne tandis que pour le deuxième cas c'est le deuxième bit qui nous considérons comme variable. Les autres bits de la colonne sont des valeurs constantes déterminées par la relation (6.1) ou la relation (6.3) respectivement. Enfin, les bits de l'état qui correspondent aux boîtes-S non-actives, sont également considérés comme des bits constants.

Nous calculons ensuite la propagation des bits variables à travers la transformation linéaire et les tours deux et trois de la permutation. Pour chaque boîte-S de l'état du deuxième et troisième tour, nous regardons si la première relation identifiée par Thomas Fuhr ou une des relations décrites par la table 6.3 est vérifiée. Nous pouvons alors déterminer après la couche des boîtes-S de chaque tour, quels bits de l'état s'expriment de façon affine en les variables de $\mathcal{I}$. Cette étape est décrite dans l'algorithme 5.

Algorithme 5: Algorithme `Relations Affines` qui détermine pour un ensemble donné de variables d'entrée $\mathcal{I}$ le nombre de bits en sortie de la fonction de compression de Hamsi-256 qui dépendent de façon affine des variables dans $\mathcal{I}$.

Données : Un ensemble de variables $\mathcal{I}$ de taille N_{var}
Résultat : Un ensemble $\mathcal{O}$ de bits de sortie de la fonction de compression de Hamsi-256 qui dépendent de façon affine des variables de $\mathcal{I}$ et $N_{out} = |\mathcal{O}|$.

```
Calculer la propagation des variables à travers la fonction linéaire ;
pour i de 1 à 2 faire
    E ← ∅, Z ← ∅ ;
    pour j de 0 à 127 faire
        si la première relation identifiée par Thomas Fuhr ou une des relation décrites
        par la table 6.3 est vérifiée pour S_j alors
            Ajouter dans E les bits en sortie de S_j qui sont des combinaisons affines des
            bits variables de I ;
        pour k dans E faire
            Calculer la propagation du bit k à travers de la fonction linéaire ;
            Ajouter à Z les bits de l'état après le tour i + 1 qui dépendent de k ;
    I ← Z ;
O ← Z;
N_out ← |O| ;
Renvoyer O, N_out;
```

Avec cette méthodologie nous avons trouvé les résultats suivants pour $N_{var} = 9$ et pour $N_{var} = 10$.

Résultats pour 9 variables : Pour les 9 boîtes-S décrites par la liste $\{0, 7, 24, 30, 35, 37, 51, 59, 61\}$, nous avons fixé la relation (6.1) pour les boîtes-S $\{0, 30, 35, 37\}$ et la relation (6.3) pour les boîtes-S restant. Alors, les 13 bits de sortie suivants de la fonction de compression de Hamsi-256 dépendent de façon affine des neuf variables d'entrée :

$$\{6, 8, 43, 78, 262, 278, 313, 320, 343, 345, 350, 355, 380\}.$$

En particulier nous avons trouvé deux relations de plus que dans les résultats décrits dans [Fuh11] pour le même nombre de variables.

Résultats pour 10 variables : Nous avons considéré les 10 boîtes-S suivantes : $\{0, 7, 12, 16, 30, 35, 37, 51, 59, 61\}$. Nous avons fixé la relation (6.1) pour les boîtes-S dans $\{0, 16, 30, 35, 37\}$ et la relation (6.3) pour les autres. Les 11 bits suivants dépendent alors de façon affine des 10 variables en entrée :

$$\{6, 8, 43, 78, 278, 313, 320, 343, 345, 350, 380\}.$$

Également pour ce cas, nous avons identifié deux relations en plus de celles décrites dans [Fuh11].

6.3 Conclusion

Dans ce chapitre nous avons présenté une méthode de recherche de relations affines à travers une primitive symétrique. Cette analyse, a été d'abord appliquée afin d'améliorer une cryptanalyse connue de la fonction de hachage Hamsi. Puisque nous avons trouvé des paramètres N_{var} et N_{out} plus grands que dans l'attaque originale, notre analyse permet d'améliorer légèrement la complexité de celle-ci en appliquant les nouveaux paramètres à l'équation (6.2). Nous avons montré que cette attaque peut être généralisée à d'autres constructions dont la partie non-linéaire est composée de plusieurs petites boîtes-S et qui sont de degré algébrique bas. L'application de ce type d'attaque à une primitive cryptographique dépend principalement des propriétés des boîtes-S utilisées. Nous avons analysé certaines de ces propriétés afin d'isoler les caractéristiques qui rendent une boîte-S susceptible de ne pas résister à ce type de cryptanalyse. Nous avons appliqué la même attaque à des versions de Hamsi où la boîte-S ordinaire a été remplacé par une autre boîte-S à 4 variables ne possèdant aucune composante quadratique. Plus précisement, nous avons d'abord essayé avec la boîte-S representative de la classe G_3 comme décrit dans [LP07] et puis avec la boîte-S S_0 de la fonction de hachage JH. Dans les deux cas, nous avons constaté que l'attaque ne fonctionnait plus.

Algorithme 4: Méthode pour la recherche des relations affines pour Hamsi-256

Données : Un nombre N_{var} de variables, un seuil s, une constante n_{et}
Résultat : Un ensemble $\mathcal{I}$ de taille N_{var} de variables en entrée et un ensemble de $\mathcal{O}$ de bits en sortie de la fonction de compression tels que tous les bits dans $\mathcal{O}$ soient des combinaisons affines des variables de $\mathcal{I}$

Mettre $\mathcal{S} = \emptyset$ et $\mathcal{L} = \{0, \ldots, 127\} \cup \{256, \ldots, 383\}$;
pour *i de* 1 *à* n_{et} **faire**
  Choisir une boîte-S S, $S \notin \mathcal{S}$ parmi les 64 premières boîtes-S de l'état ;
  $\mathcal{S} \leftarrow S \cup \{S\}$;
  pour *j dans* $\mathcal{L}$ **faire**
    si b_j *dépend de toutes les coordonnés de* S_i **alors**
      $\mathcal{L} \leftarrow \mathcal{L} \setminus \{j\}$;
  $\text{compteur}_2 \leftarrow 0$, $\text{compteur}_4 \leftarrow 0$;
  $\mathcal{L}_2 \leftarrow \emptyset$, $\mathcal{L}_4 \leftarrow \emptyset$;
  pour *j dans* $\mathcal{L}$ **faire**
    si b_j *ne dépend pas de la deuxième coordonnée de* S_i **alors**
      $\text{compteur}_2 \leftarrow \text{compteur}_2 + 1$; $\mathcal{L}_2 \leftarrow \mathcal{L}_2 \cup \{j\}$;
    si b *ne dépend pas de la quatrième coordonnée de* S_i **alors**
      $\text{compteur}_4 \leftarrow \text{compteur}_4 + 1$; $\mathcal{L}_4 \leftarrow \mathcal{L}_4 \cup \{j\}$;
  si $compteur_2 \geq compteur_4$ **alors**
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_4$;
  sinon
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_2$;
pour *i de* 0 *à* 63 **faire**
  si $S_i \notin \mathcal{S}$ **alors**
    si *au moins une coordonnée de* S_i *n'influence pas au moins* s *bits dans* $\mathcal{L}$ **alors**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_i\}$;
$\text{compteur}_2 \leftarrow 0$, $\text{compteur}_4 \leftarrow 0$;
pour S_i *dans* $\mathcal{S}$ **faire**
  si b *ne dépend pas de la deuxième coordonnée de* S_i **alors**
    $\text{compteur}_2 \leftarrow \text{compteur}_2 + 1$
  si b *ne dépend pas de la quatrième coordonnée de* S_i **alors**
    $\text{compteur}_4 \leftarrow \text{compteur}_4 + 1$
  si $compteur_2 \geq compteur_4$ **alors**
    imposer la relation (6.3) pour S_i ;
  sinon
    imposer la relation (6.1) pour S_i ;
max $\leftarrow 0$;
pour *toute combinaison* $S_{j_1} \ldots S_{j_{N_{var}}}$ *de* N_{var} *boîtes-S parmi les* $\binom{|\mathcal{S}|}{N_{var}}$ **faire**
  $\mathcal{E} \leftarrow \emptyset$;
  pour *i de* 1 *à* N_{var} **faire**
    si *la relation (6.3) est imposée pour* S_i **alors**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{x_{128+i}\}$
    si *la relation (6.1) est imposée pour* S_i **alors**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{x_{384+i}\}$
    $(N_{out}, \mathcal{O}_{out}) \leftarrow$ `Relations Affines`$(\mathcal{E})$;
    si $N_{out} \geq max$ **alors**
      $\mathcal{I} \leftarrow \mathcal{E}$; $\mathcal{O} \leftarrow \mathcal{O}_{out}$; $max \leftarrow N_{eq}$;
Renvoyer $\mathcal{I}, \mathcal{O}$;

Deuxième partie

Securité physique des cryptosystèmes symétriques

Chapitre 7

Attaques par canaux cachés contre certains candidats SHA-3

Sommaire

7.1 Introduction 141
7.2 Attaques par canaux cachés 142
7.2.1 Bref historique 142
7.2.2 Les fuites par mesure du temps 144
7.2.3 Analyse simple du courant (SPA) 144
7.2.4 Les analyses statistiques du courant 144
7.2.5 L'analyse différentielle du courant (DPA) 146
7.3 Protection contre les analyses statistiques du courant 147
7.3.1 Désynchronisation 148
7.3.2 Autres contre-mesures au niveau matériel 148
7.3.3 Randomisation des données 148
7.3.4 Contre-mesures au niveau protocole 150
7.3.5 Cryptographie résistante aux fuites d'information 150
7.4 Le concours SHA-3 et les attaques physiques 150
7.5 Analyse CPA de Grøstl et Skein 151
7.5.1 Attaque par canaux cachés contre HMAC et *envelope* MAC 152
7.5.2 Analyse de Grøstl 153
7.5.3 Analyse de Skein 158
7.5.4 Analyse de performance pour les deux candidats 162

7.1 Introduction

Depuis plusieurs années, de plus en plus de produits industriels contenant des données confidentielles intègrent des algorithmes cryptographiques. Les implémentations de ces algorithmes au sein de dispositifs divers, comme par exemple les cartes à puce, font l'objet d'attaques à faible coût visant à récupérer les clefs secrètes dans leurs circuits. Pour cette raison, lors de la conception d'un algorithme, sa sécurité "physique" doit sérieusement être prise en compte. Même si un algorithme est prouvé sûr contre des cryptanalyses classiques,

son implantation physique peut contenir des fuites d'informations qui peuvent être exploitées par un attaquant.

La sécurité physique d'un algorithme consiste en sa protection face aux attaques dites *physiques*, dont l'objectif est d'extraire de l'information des phénomènes physiques relatifs à une implémentation particulière. Nous pouvons tout d'abord catégoriser les attaques physiques selon le niveau de l'intrusion de l'attaquant. On distingue ici majoritairement trois grands groupes : les attaques intrusives, les attaques non-intrusives et les attaques semi-intrusives. Le principe d'une attaque non-intrusive est de laisser le dispositif intact. Seuls les paramètres environnementaux peuvent être modifiés ou simplement observés. Au contraire, au cours d'une attaque intrusive, un accès direct aux composantes intérieures du dispositif est établi. La dernière classe est celle des attaques semi-intruisives, introduites par Skorobogatov et Anderson [SA03]. Elles consistent à sortir le circuit de son emballage (boîtier, carte à puce) afin de mieux observer les composantes internes mais sans établir pour autant une relation directe avec celles-ci.

D'autre part, les attaques physiques peuvent être classifiées selon l'activité de l'attaquant durant leur réalisation, c'est-à-dire selon que l'attaquant se contente d'observer le comportement de la carte pendant l'exécution de l'algorithme ou qu'il se permet de la perturber. Nous appelons les attaques du premier type *passives* tandis que les attaques du deuxième type sont appelées *actives*. D'autres classifications existent également dans la littérature (voir par exemple [KK99]).

Une classe importante d'attaques physiques sont les attaques dites *par faute*. Ce sont des attaques d'un caractère actif, visant à perturber l'exécution normale d'un algorithme et à obtenir des informations sur le secret, en comparant les données avec d'introduction des erreurs aux données correctes. Cette classe d'attaques, malgré son intérêt, ne sera pas étudiée dans ce mémoire. Une autre catégorie d'attaques très importantes et très efficaces, que nous décrivons en détail par la suite, sont les attaques *par canaux cachés* (*side-channel attacks*, en anglais). Les attaques par canaux cachés ou par canaux auxiliaires sont des attaques passives qui consistent en l'exploitation des fuites liées par exemple à la consommation d'énergie ou à la radiation électromagnétique d'un dispositif exécutant une primitive cryptographique.

7.2 Attaques par canaux cachés

Lorsqu'un algorithme cryptographique est implémenté sur un dispositif spécifique, de nombreuses fuites de nature physique peuvent être observées pendant l'exécution. Les fuites d'information qui peuvent être constatées comprennent l'information sur le temps d'exécution de certaines opérations, la consommation de courant, les rayonnements électromagnétiques, la chaleur dissipée ou encore le bruit produit. Toutes ces informations peuvent être observées, classifiées et traitées sans perturber l'exécution du dispositif. Les attaques exploitant ces fuites naturelles sont appelées attaques par canaux cachés.

7.2.1 Bref historique

Nous commençons par citer brièvement certains des premiers exemples d'utilisation de canaux auxiliaires afin d'obtenir des informations précieuses échangées pendant des communications confidentielles.

Exploitation d'ondes électromagnétiques pour le système Bell 131-B2

Le premier exemple d'une exploitation d'ondes électroniques d'une machine chiffrante concerne le périphérique de chiffrement Bell 131-B2. Ce dispositif a été utilisé par l'armée américaine pendant la Seconde Guerre Mondiale. En 1943 un chercheur des laboratoires Bell a remarqué par hasard que lorsque un caractère était chiffré par le dispositif, un pic apparaissait sur un oscilloscope situé dans un endroit éloigné du laboratoire. En examinant les pics, il était alors possible de retrouver le texte clair qui était chiffré par la machine.

Les laboratoires Bell ont ensuite étudié ce phénomène et ont identifié trois types d'émissions : les radiations électromagnétiques, les signaux induits dans les câbles et les champs magnétiques. Suite à cela ils ont proposé trois solutions. La première solution était le blindage pour bloquer les radiations et les champs magnétiques. La deuxième solution proposait le filtrage des signaux induits, tandis que la troisième concernait le masquage des signaux induits et rayonnés. Cependant, la mise en pratique de ces solutions a été difficile ; la machine est devenue lente et souffrait de surchauffe. Par conséquence la seule mesure finalement utilisée était de placer la machine dans un endroit surveillé autour d'un rayon de 30 mètres.

Cet incident, dont nous sommes au courant grâce à un document de la NSA (abréviation de National Security Agency) récemment déclassifié [NSA72] a sûrement motivé le lancement du projet TEMPEST.

Projet TEMPEST

Pendant les années 1950, le gouvernement des États Unis d'Amérique a commencé à se soucier du fait que les radiations électromagnétiques émanant des dispositifs électroniques pouvaient être capturées par les agents secrets des pays ennemis afin d'espionner les conversations confidentielles. Un projet dont le but était d'étudier et de prévenir des émanations compromettantes, a alors été lancé dans cette direction. Ce projet est connu sous le nom TEMPEST. Ce mot est aujourd'hui lié aux normes définies par ce projet qu'un dispositif chiffrant doit respecter afin d'être considéré comme sûr contre ce type de menace.

En 1985, Wim Van Eck a présenté la première étude publique [Van85] qui expliquait comment intercepter et décoder des émanations électromagnétiques des tubes cathodiques (CRT en anglais), utilisées dans les technologies vidéo.

Premières attaques par canaux cachés sur des implémentations cryptographiques

La première attaque par canaux cachés qui a cassé une implémentation d'un algorithme cryptographique est due à Paul Kocher [Koc96]. Dans ce travail l'auteur a montré comment retrouver de façon très efficace les clés utilisées dans des cryptosystèmes classiques tels que RSA ou Diffie-Hellman. Ces premières attaques exploitaient des variations dans le temps d'exécution de certaines opérations dépendant des valeurs secrètes, comme par exemple l'exponentiation modulaire.

Deux années plus tard, Paul Kocher, Joshua Jaffe et Benjamin Jun [KJJ98, KJJ99] ont décrit une attaque exploitant des fuites de l'énergie consommée pendant l'exécution d'un algorithme cryptographique. Il a été ensuite montré qu'une attaque similaire pouvait se déduire en exploitant les émanations électromagnétiques du dispositif [GMO01, QS01].

Nous présentons maintenant certaines des classes les plus importantes des attaques par canaux cachés.

7.2.2 Les fuites par mesure du temps

Le premier type de canal auxiliaire qui a été exploité contre un système cryptographique est la fuite par mesure du temps. Ce type d'attaque a été présenté pour la première fois par Kocher [Koc96] et a été réalisé pour la première fois en pratique par Dhem *et al* [DKL+00]. Une attaque par analyse du temps d'exécution consiste à mesurer le temps nécessaire pour exécuter certaines opérations. Si l'opération dépend de la clé, le temps de son exécution peut alors révéler des informations sur la valeur exacte de la clé. Par exemple, lorsque une exponentiation modulaire est implémentée avec l'algorithme *élever au carré et multiplier* (*square and multiply* en anglais), le temps peut varier selon que le bit traité est 0 ou 1.

Cependant, il est assez facile d'assurer qu'une implémentation sur une carte à puce s'exécute en temps constant. Pour cette raison, l'exploitation de ce type de canal caché n'a pas conduit à d'autres attaques (voir malgré tout [HH99]).

7.2.3 Analyse simple du courant (SPA)

L'*analyse simple du courant* ou SPA (abréviation de *Simple Power Analysis* en anglais) est la forme la plus basique d'une attaque par analyse de la consommation. Elle consiste à simplement observer la consommation de courant du dispositif et à interpréter les informations recueillies pendant les opérations cryptographiques.

La consommation de courant d'un circuit électronique est liée au changement de l'état des portes présentes sur celui-ci. Plus l'état du circuit change, plus la consommation est élevée. La consommation de courant d'un microprocesseur à un instant t résulte essentiellement de la somme des contributions des consommations de toutes ses portes à cet instant.

L'analyse simple du courant peut dans certains cas révéler des informations sur la clé secrète utilisée. Comme déjà vu avant, si l'algorithme *élever au carré et multiplier* est appliqué, dans le cas par exemple d'une exponentiation avec la clé privée dans RSA, il peut être possible de déterminer certains bits de la valeur secrète si l'élévation au carré et la multiplication n'ont pas la même consommation. De façon générale, tout algorithme comprenant des branchements conditionnels dépendant des paramètres secrets est vulnerable à ce type d'attaque.

Une simple trace de consommation peut être également utilisée afin d'observer la structure de l'algorithme exécuté et de distinguer ses différentes parties. La figure 7.1 illustre la trace de consommation d'un dispositif exécutant HMAC-Grøstl. Nous pouvons clairement y distinguer un motif répété plusieurs fois. Ce motif correspond en conséquence à un tour de la permutation de Grøstl. Même si cette observation ne conduit pas à une attaque en elle-même, elle peut être utilisée pour délimiter l'opération cible lors d'une attaque statistique. Une analyse attentive à cette étape peut aider à diminuer drastiquement les données à traiter à l'étape suivante.

7.2.4 Les analyses statistiques du courant

Une forme plus puissante des attaques par mesure du courant est l'*analyse statistique du courant*. Historiquement, la première technique d'analyse de ce type a été introduite en 1998 par Kocher, Jaffe et Jun dans [KJJ98] et s'appelle *analyse différentielle du courant*, ou DPA (abréviation de *Differential Power Analysis* en anglais).

La consommation de courant pendant l'exécution d'un algorithme cryptographique dépend de la valeur des données traitées, et donc des bits de la clé secrète. Toutefois, cette dépendance n'est généralement pas visible avec une seule trace. En particulier, pour attaquer des chiffre-

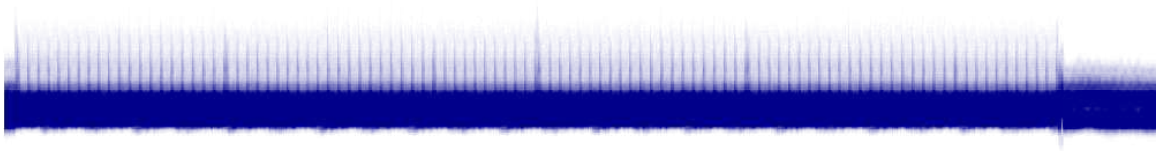

FIGURE 7.1 – Exécution de HMAC-Grøstl sur une carte à puce.

ments symétriques, l'analyse simple du courant ne suffit pas. Des méthodes statistiques doivent être utilisées.

Dans une analyse statistique, un algorithme cryptographique est exécuté plusieurs fois avec une entrée différente à chaque fois. Typiquement, dans le cas des chiffrements par blocs, on fait varier le message afin de retrouver la clé. Une trace de courant est alors capturée par exécution. Les traces obtenues sont ensuite analysées à l'aide d'outils statistiques.

Modèles de fuite Pour ces méthodes, un modèle expliquant la variation de la consommation du courant en fonction des données traitées doit être choisi. Parmi les modèles proposés, les modèles de *poids de Hamming* et de la *distance de Hamming* [BCO04] sont les plus utilisés. Tous les deux sont basés sur l'hypothèse que les plus grandes variations du courant en fonction d'une donnée sont observées lorsque cette donnée est présente sur le bus externe de la CPU, puisque les lignes du bus sont les éléments les plus consommateurs d'un microcontrôleur. Chaque ligne du bus pour laquelle une transition de 0 à 1 et de 1 à 0 se produit consomme l'énergie nécessaire pour réaliser ce basculement d'état. Il est communément supposé que la fuite dépend du nombre de bits qui basculent d'un état à l'autre à un moment précis [CKN01, CCD00]. Si on considère que la contribution de chaque bit est additive, que cette contribution est égale pour tous les bits et que l'énergie requise est la même quel que soit le sens de la transition ($1 \mapsto 0$ ou $1 \mapsto 0$), la consommation totale W peut être exprimée à l'aide du modèle de la distance de Hamming comme

$$W = a \cdot HW(P \oplus X) + b,$$

où X est la valeur de la donnée manipulée, et P est la valeur préalablement présente sur le bus et appelée *état de référence*. La variable b comprend parmi d'autres les décalages de calibration instrumentale, les composantes dépendant du temps et le bruit. Enfin, a représente le gain entre la distance de Hamming et le courant consommé. Puisque toutes les informations passent à travers le bus, cette relation peut être exploitée quand les données secrètes y sont transférées.

Un cas particulier du modèle précèdent est quand le bus est mis à zéro avant l'envoi de chaque nouvelle valeur, c'est-à-dire quand $P = 0$. Nous en déduisons le modèle de fuite le plus simple, à savoir le modèle de poids de Hamming. Ce modèle a été introduit dans [KJJ99, TSMS99]. Il est décrit par la relation

$$W = a \cdot HW(X) + b.$$

D'autres modèles prenant en considération des propriétés de certains dispositifs ont été également proposés. Un exemple est le modèle de la distance de commutation (*switching distance model* en anglais) [PSQ07].

7.2.5 L'analyse différentielle du courant (DPA)

L'analyse différentielle du courant est le premier exemple d'analyse statistique proposé. Elle peut être appliquée sur tout algorithme pour lequel une opération intermédiaire de la forme $f(m, k)$ est calculée, où k est une partie de la clé secrète, m est une valeur aléatoire mais connue de l'attaquant et f une opération quelconque mélangeant les deux quantités. La fonction f est souvent appelée *fonction de sélection*. Elle peut par exemple être une opération XOR, une addition modulaire ou une boîte-S. En générale, les fonctions non-linéaires sont préférées.

Ensuite, un seul bit, appelé *bit de sélection* ou *bit cible* doit être choisi. La fonction de sélection est alors exécutée N fois avec N messages M_i choisis aléatoirement, $1 \leq i \leq N$. Une trace de courant W_i est capturée à l'aide d'un oscilloscope à chaque exécution. La valeur de N dépend de l'architecture et de la fonction de sélection, mais elle varie de façon générale entre 50 et 10000 traces.

L'analyse DPA est une attaque du type *diviser pour régner*, puisque elle vise à récupérer les différents morceaux de la clé, que nous pouvons dans ce contexte appeler sous-clés, indépendamment un par un. Afin de retrouver une sous-clé de m bits, Ses 2^m valeurs possibles sont considérées. Pour chaque hypothèse, K_j, le bit de sélection b est utilisé afin de partitionner les traces de courant W_i, $1 \leq i \leq N$ en deux ensembles, E_0^j et E_1^j :

$$E_0^j = \{W_i : b = 0\},$$

$$E_1^j = \{W_i : b = 1\}.$$

La valeur calculée pour le bit de selection dépend à la fois du message en entrée et de l'hypothèse faite sur la valeur de la sous-clé. C'est une prédiction sur la valeur réelle de ce bit. Cette prédiction est systématiquement correcte pour la bonne valeur de la sous-clé et correcte par chance avec probabilité 1/2 pour une valeur qui n'est pas la bonne.

Une fois les courbes de courant ainsi partitionnées, on calcule pour chaque hypothèse j la courbe de courant moyenne Δ_j, $1 \leq j \leq 2^m$ de la manière suivante :

$$\Delta_j = \frac{\sum_{W_i \in E_0^j} W_i}{|E_0^j|} - \frac{\sum_{W_i \in E_1^j} W_i}{|E_1^j|}.$$

Il en résulte une *courbe de DPA* pour chaque hypothèse K_j sur la sous-clé.

Si la supposition sur la sous-clé est correcte alors le bit de sélection reflète exactement sa valeur dans la carte à chaque instant que celle-ci est manipulée. Un "pic" est alors visible à chacun de ces instants. Au contraire, pour une supposition fausse, les courbes de chaque ensemble sont indépendantes de ce qui s'est réellement passé à cet instant dans la carte. Par conséquent, les courbes DPA correspondantes sont supposées être plates, à un bruit près.

L'analyse du courant par corrélation (CPA)

Un inconvénient important de la DPA, susceptible de perturber l'attaquant, est la présence des *pics fantômes*. Il s'agit des pics correspondant à des suppositions fausses dont l'amplitude

rivalise avec celle de “vrais” pics. Leur présence rend l'identification de la vraie sous-clé difficile. Leur existence est en général due aux erreurs de modélisation. Ces erreurs sont principalement liées à deux causes. La première est la supposition que pour une clé incorrecte, la répartition de chaque courbe dans chaque ensemble est indépendante de la répartition correspondant au bon candidat. La deuxième cause est due à l'ignorance de la contribution à la consommation de courant des bits voisins du bit de sélection, même si ces derniers se trouvent dans le même mot machine. Une analyse vaste des causes de l'existence de pics fantômes peut être trouvée dans la thèse de doctorat de Christophe Clavier [Cla07].

Pour ces raisons, une alternative à la DPA a été présentée par Eric Brier, Christophe Clavier et Francis Olivier dans [BCO04]. Il s'agit de l'*analyse du courant par corrélation* ou CPA (abréviation de *Correlation Power Analysis* en anglais). Pour cette méthode, un modèle de fuite doit être préalablement choisi. Une différence importance par rapport à la DPA est que, au lieu d'analyser un seul bit, l'attaquant essaie de prédire l'intégralité du mot machine qui est manipulé à un instant précis.

Alors, si la consommation de courant suit le modèle du poids de Hamming ou de la distance de Hamming ($W = a \cdot H + b$), ce modèle linéaire implique certaines relations entre les variances de différents termes considérés comme des variables aléatoires : $\sigma_W^2 = a^2\sigma_H^2 + \sigma_b^2$. Dans un micro-processeur à m bits, une donnée qui contient m bits a un poids de Hamming qui varie entre 0 et m. Si on suppose que ces bits sont indépendants et uniformement distibués, alors le mot complet a un poids de Hamming moyen $m/2$ et une variance égale à $m/4$. Le *coefficient de corrélation de Pearson* ρ_{WH} entre la distance de Hamming et le courant mesuré est donné par :

$$\begin{aligned}\rho_{WH} &= \frac{\mathrm{cov}(W,H)}{\sigma_W\sigma_H} = \frac{\mathrm{cov}(a\cdot H+b,H)}{\sigma_W\sigma_H} = \frac{a\cdot\mathrm{cov}(H,H)}{\sigma_W\sigma_H} = \frac{a\cdot\sigma_H^2}{\sigma_W\sigma_H} \\ &= \frac{a\sigma_H}{\sigma_W} = \frac{a\sigma_H}{\sqrt{a^2\sigma_H^2+\sigma_b^2}} = \frac{a\sqrt{\frac{m}{4}}}{\sqrt{a^2\frac{m}{4}+\sigma_b^2}} = \frac{a\sqrt{m}}{\sqrt{ma^2+4\sigma_b^2}},\end{aligned}$$

et satisfait la propriété $-1 \leq \rho_{WH} \leq 1$.

Dans la pratique, ce coefficient est calculé à l'aide de N traces W_i correspondant à N prédictions sur le poids de Hamming H_i à un moment donné. Une estimation du coefficient de corrélation est alors donné par l'expression suivante :

$$\rho = \frac{N\sum_{i=1}^N W_iH_i - \sum_{i=1}^N W_i\sum_{i=1}^N H_i}{\sqrt{N\sum_{i=1}^N W_i^2 - (\sum_{i=1}^N W_i)^2}\sqrt{N\sum_{i=1}^N H_i^2 - (\sum_{i=1}^N H_i)^2}}.$$

7.3 Protection contre les analyses statistiques du courant

Nous présentons ici les contre-mesures les plus communes vis-à-vis des attaques par analyse statistique du courant. La plupart de ces contre-mesures visent à faire disparaître (ou au moins à diminuer) la dépendance entre la fuite observée et les variables sensibles traitées. Cependant, il existe également une multitude de contre-mesures dont le but est de gêner l'attaquant, en ajoutant par exemple du bruit.

Afin qu'une attaque statistique soit réussie, une condition nécessaire est que l'opération visée par l'attaquant se produise au même instant pour chaque courbe générée. La première contre-mesure est alors la *désynchronisation* des différentes exécutions.

7.3.1 Désynchronisation

La désynchronisation des différentes exécutions d'un algorithme cryptographique peut être obtenue grâce aux mécanismes matériaux, à l'aide d'horloges internes instables ou encore de délais au niveau logiciel. Une méthode consiste par exemple à insérer des morceaux de code inutile à des instants aléatoires. Pour rendre la tache plus difficile, ces instructions peuvent elles-mêmes être aléatoires et avoir une durée quelconque. Un exemple est d'utiliser des boucles où une valeur aléatoire est générée et puis décrémentée jusqu'à 0. De telles solutions ne nécessitent que très peu de code supplémentaire. Quelques méthodes avancées pour introduire des délais au niveau logiciel sont décrites dans [TB07] ou [CK09].

Une autre méthode consiste à exécuter des processus indépendants dans un ordre aléatoire. Cette contre-mesure peut par exemple être appliquée aux chiffrements dont la partie non-linéaire consiste en l'application en parallèle des boîtes-S, afin d'exécuter celles-ci dans un ordre quelconque.

L'effet de désynchronisation n'est pas une contre-mesure absolue, mais peut sérieusement gêner un attaquant, car une analyse statistique dans ces conditions est impossible ; les courbes doivent d'abord être synchronisées. Souvent, quelques milliers de traces doivent être traitées. Ce processus devient alors très dur.

7.3.2 Autres contre-mesures au niveau matériel

Une contre-mesure classique au niveau matériel consiste à brouiller la consommation en ajoutant un bruit aléatoire. Cela peut être fait à l'aide d'un générateur de bruit. L'inconvénient de cette méthode est qu'elle est accompagnée d'une augmentation significative du courant consommé. Elle ne s'applique alors que très difficilement aux dispositifs de petite taille, comme c'est par exemple le cas d'un téléphone portable. Une autre solution est d'utiliser un filtre afin d'aplatir la consommation de courant [CKN01, KK99, Mes00]. Cependant cette contre-mesure ne protège pas contre les attaques par analyse des rayonnements électromagnétiques.

Les contre-mesures décrites ci-dessus peuvent considérablement gêner l'attaquant, mais ne sont toutefois pas suffisantes contre des attaques pratiques. Afin de mieux protéger le dispositif, ces méthodes doivent être combinées avec des contre-mesures plus fondamentales que nous présentons tout de suite.

7.3.3 Randomisation des données

La randomisation des données consiste à rendre aléatoires les valeurs cibles, prédites par l'attaquant. Ces techniques s'appliquent aussi bien au niveau logiciel que matériel. Contrairement aux techniques décrites auparavant, ces méthodes ne concernent pas le dispositif entier, mais visent seulement à protéger les informations sensibles.

La méthode la plus utilisée pour protéger des primitives symétriques est le *masquage* aléatoire des données intermédiaires. Le principe du masquage consiste à ajouter pendant le calcul une ou plusieurs valeurs aléatoires à chaque variable sensible. De cette manière, la vraie valeur manipulée et donc la consommation correspondant, est indépendante de celle prédite. En conséquence, l'analyse statistique échoue. Cette approche est apparue très rapidement après la publication des premiers travaux sur l'analyse statistique du courant. L'idée était indépendamment publiée par Suresh Chari, Charanjit Jutla, Josyula Rao, et Pankaj Rohatgi dans [CJRR99] et par Louis Goubin et Jacques Patarin dans [GP99].

De façon générale, afin de protéger une valeur sensible z, on divise celle-ci en $d+1$ *parts* (*shares* en anglais), $r_0, \ldots, r_d$, de façon que

$$r_0 * r_1 * \cdots * r_d = z,$$

où $*$ est une opération de groupe quelconque. Les valeurs $r_1, \ldots, r_d$ sont en général appelées *masques*, tandis que la valeur r_0 est appelée *variable masquée*. Selon la nature des opérations, des versions différentes sont utilisées. Cependant, deux familles se distinguent : le *masquage booléen* pour lequel l'opération $*$ coïncide avec le ou-exclusif (XOR) et le *masquage arithmétique* pour lequel $*$ correspond à l'addition modulaire. Lorsque d masques sont employés afin de protéger la variable z le masquage est appelé *d'ordre d*. Dans la pratique, d est souvent fixé à 1.

Nous allons ensuite analyser les principaux schémas de masquage appliqués aux primitives symétriques (chiffrements par blocs ou codes d'authentification de message basés sur une fonction de hachage). Une grande partie des chiffrements par blocs sont composés d'opérations linéaires, d'additions des sous-clés et de boîtes-S. Le problème principal de la conception d'un schéma de masquage pour ce type de construction est le masquage des boîtes-S. Plusieurs solutions existent dans la littérature. Certaines d'entre elles sont assez génériques et s'appliquent sur une boîte-S quelconque, tandis que d'autres sont spécifiques à la boîte-S de l'AES, en exploitant en particulier sa structure mathématique [AG01, BGK04, OMPR05, OS06, CB08, CG06].

Masquage des boîtes-S

Une méthode très simple et efficace pour masquer une boîte-S quelconque, connue sous le nom *recalcul de table* (*table re-computation* en anglais), a été proposée par Thomas Messerges dans [Mes01]. Elle consiste à calculer une *table de correspondance* représentant une version masquée de la boîte-S. Afin de masquer une permutation non-linéaire S dans $\mathbf{F}_2^n$ deux masques u, v de $\mathbf{F}_2^n$ doivent être générés à l'aide d'un générateur pseudo-aléatoire. Une table masquée T, allouée en RAM, peut alors être calculée en fonction de S, u, et v de façon décrite par l'algorithme 6.

Algorithme 6: La méthode "recalcul de table"pour masquer une boîte-S

Données : Un masque d'entrée u, un masque de sortie v
Résultat : La table masquée T
pour x *de 0 à* 2^{n-1} **faire**
  $T[x] \leftarrow S(x \oplus u) \oplus v$;
Renvoyer T ;

La simplicité et l'efficacité de cette méthode font d'elle l'outil le plus utilisé en pratique pour masquer une boîte-S contre des attaques statistiques du premier ordre.

Cependant, cette méthode peut poser des problèmes si la taille des tables à masquer est grande. Pour pallier ce problème, des solutions consistant à effectuer des calculs à la volée, sans allocation de mémoire RAM, ont été proposées. Une première méthode basée sur la transformation de Fourier, proposée par Emmanuel Prouff *et al.* [PGA06] a été attaquée et puis réparée par Jean-Sébastien Coron *et al.* dans [CGPR06]. Une méthode alternative a été également proposée dans [PR08]. La thèse de doctorat de Matthieu Rivain [Riv09] présente une étude assez complète sur les différentes méthodes de masquage des boîtes-S.

Problème de conversion des masques

Plusieurs chiffrements par blocs et fonctions de hachage sont basés sur des opérations à la fois arithmétiques et booléennes. C'est le cas en particulier des fonctions ARX. Pour ce type de constructions, il est naturel d'utiliser en même temps des masques booléens ainsi qu'arithmétiques pour protéger les données sensibles. Des mécanismes de conversion doivent alors être utilisés pour transformer un masque arithmétique en un masque booléen et vice versa. La première méthode de ce type a été proposée par Thomas Messerges dans [Mes01], mais Jean-Sébastien Coron et Louis Goubin ont montré qu'elle était vulnérable à une attaque statistique du premier ordre [CG00].

Une autre méthode de conversion a été par la suite proposée par Louis Goubin [Gou01]. Il s'agit d'une méthode qui est aujourd'hui très utilisée en pratique. Dans cette approche, la conversion booléenne vers arithmétique est très efficace, mais ceci n'est pas le cas pour la conversion inverse. Une version alternative a été ensuite proposée par Alexei Tchulkine et Jean-Sébastien Coron [CT03]. Dans cette approche, la conversion problématique, c'est-à-dire la conversion arithmétique vers booléenne est remplacée par l'utilisation de tables précalculées. Le choix de l'utilisation d'une méthode ou de l'autre dépend souvent du compromis souhaité entre efficacité et mémoire. Une méthode intéressante qui combine les deux approches a été récemment présentée par Blandine Debraize [Deb12].

7.3.4 Contre-mesures au niveau protocole

Récemment, plusieurs travaux concernant des contre-mesures au niveau protocole sont apparus. La plupart de ces propositions consistent à limiter l'usage d'une seule clé à un certain nombre d'exécutions et à mettre régulièrement à jour les sous-clé utilisées [GM11, MSGR10]. Une autre idée est d'utiliser certains concepts des chiffrements par blocs avec un tweak, en utilisant le tweak pour mettre périodiquement à jour la configuration [GM11].

7.3.5 Cryptographie résistante aux fuites d'information

Une nouvelle direction consiste à concevoir des primitives cryptographiques qui sont prouvées sûres dans des modèles où la quantité d'information secrète qui fuit est bornée [DP08, SPY+09]. Ce type de cryptographie s'appelle *cryptographie résistante aux fuites d'information* ou *leakage resilient cryptography* en anglais.

7.4 Le concours SHA-3 et les attaques physiques

Les fonctions de hachage sont des algorithmes publics ne traitant aucun secret. Il est alors naturel de se demander s'il y a du sens à analyser la sécurité physique de ces fonctions. Il est évident que puisque les attaques physiques visent à récupérer une valeur secrète présente dans le dispositif, ces méthodes ne s'appliquent pas sur les fonctions de hachage lorsque elles sont utilisées de façon classique. Toutefois, les fonctions de hachage sont très souvent présentes dans les constructions d'authentification de message (MAC, voir section 1.3.6). Cette utilisation fait également d'elles une cible pour les attaques physiques. Des analyses théoriques concernant la résistance aux attaques par analyse statistique du courant des diverses constructions MAC ont été présentées dans [GO08, Oke06].

La compétition SHA-3 (voir section 1.3.7) qui a débuté en 2008, a comme but de sélectionner une fonction sûre qui pourrait être utilisée pour tout type d'application sur toute sorte de dispositif. Le NIST a alors requis que les fonctions participant soient sûres quand elles sont employées dans la construction HMAC ou dans une autre construction MAC équivalente. John Kelsey a également précisé dès le début du concours que la sécurité physique des fonctions serait prise en considération pour la décision finale [Kel08].

Il est alors important de se soucier dès maintenant de la résistance contre les attaques physiques des candidats finalistes du concours et de commencer à examiner des contre-mesures qui pourraient être appliquées dans chaque cas. En 2010, Olivier Benoît et Thomas Peyrin ont présenté la première analyse de ce genre. Plus précisément, ils ont étudié la résistance de certains candidats du deuxième tour, à savoir ECHO, Grøstl, SHAvite-3, CubeHash, BLAKE et Hamsi, contre les analyses statistiques [BP10]. Dans cette analyse les meilleures fonctions de sélection pour chaque candidat ont été déterminées. Afin de valider les hypothèses théoriques, les auteurs ont ensuite programmé les fonctions de sélection sur un *circuit logique programmable* (FPGA, abréviation de *Field-Programmable Gate Array*) et ont analysé les fuites par émanation électromagnétique. Dans un travail plus récent, la résistance contre les analyses par canaux cachés des fonctions finalistes Grøstl, JH, Keccak et Skein, a été étudié [ZKS11].

7.5 Analyse CPA de Grøstl et Skein

Dans cette section nous présentons les résultats de notre analyse sur la résistance contre les attaques par canaux cachés des finalistes du concours SHA-3 Grøstl et Skein. Ces résultats ont été obtenus en collaboration avec Sylvain Lévêque et David Vigilant. Ils ont été présenté au workshop *Trustworthy Embedded Devices-TrustED 2012* (partie de *IEEE CS Security and Privacy Workshops*) [BLV12].

Dans le cadre de ce travail nous avons étudié les fonctions finalistes Grøstl [GKM+08] et Skein [FLS+10]. Le but de cette étude était de proposer pour ces deux candidats les premières contre-mesures concrètes au niveau logiciel contre les analyses statistiques du premier ordre. En parallèle, nous avons voulu présenter une comparaison des deux fonctions, en comparant les versions simples ainsi que les versions sécurisées sous le même modèle de référence. Pour cette raison, un certain nombre de choix concernant le cadre de l'attaque a été fait.

Le premier choix concerne la construction MAC à implémenter. Les auteurs de Grøstl suggèrent l'utilisation de *envelope* MAC (voir section 1.3.6) pour transformer Grøstl en code d'authentification de message, tandis que les concepteurs de Skein proposent un mode spécifique à Skein, le Skein-MAC. Néanmoins, nous avons choisi d'implémenter pour les deux fonctions les constructions HMAC correspondant. La première raison de cette décision est que le mode HMAC est le code d'authentification de message basé sur les fonctions de hachage le plus utilisé dans l'industrie. La deuxième raison est que nous avons voulu avoir une base commune afin de pouvoir comparer les deux candidats. Nous avons toutefois analysé théoriquement les MAC proposés par les concepteurs des deux fonctions. Les résultats de cette analyse sont présentés dans la partie qui suit.

Le deuxième choix que nous avons fait concerne le code source utilisé. Puisque le but de notre travail n'était pas la conception d'implémentations optimisées et ultra-rapides, nous nous sommes contentés d'utiliser le code source de référence proposé par les concepteurs des deux fonctions. La troisième décision est liée au nombre de courbes générées. Afin d'avoir un

modèle de comparaison, nous avons fixé le nombre de traces de courant recueillies à 5000, tant pour les versions ordinaires que pour les versions sécurisées. Le choix de ce nombre dépend principalement de la carte à puce utilisée. Pour la carte que nous avons choisie pour nos expériences, 5000 courbes sont généralement suffisantes pour avoir une fuite exploitable avec des méthodes statistiques.

Nous avons implémenté les deux fonctions, ainsi que les HMAC correspondant, sur une carte à puce ARM en 32 bits, fonctionnant à une vitesse de 8 MHz. Les réglages de sécurité de la carte utilisée comprennent l'activation de tous les capteurs au niveau matériel et d'un générateur de nombres aléatoires. La fuite exploitée pour les deux fonctions est la consommation de courant et l'attaque que nous appliquons est l'analyse du courant par corrélation (CPA).

7.5.1 Attaque par canaux cachés contre HMAC et *envelope* MAC

Nous présentons dans cette section la stratégie classique à suivre lors de l'attaque d'une construction HMAC ou du mode *envelope* MAC par canaux cachés.

Commençons d'abord par analyser le cas de HMAC. Nous pouvons voir à la figure 7.2 que les entrées au premier appel de la fonction de compression pour chacun de deux calculs (CV_0^{in} et $\overline{K} \oplus ipad$ d'un côté et CV_0^{out} et $\overline{K} \oplus opad$ de l'autre) sont des valeurs constantes ne dépendant que de la valeur secrète K. Dans certaines implémentations, afin d'améliorer la performance, les valeurs

$$K_i = h(CV_0^{in}, \overline{K} \oplus ipad) \text{ et } K_o = h(CV_0^{out}, \overline{K} \oplus opad),$$

sont pré-calculées et stockées dans le dispositif.

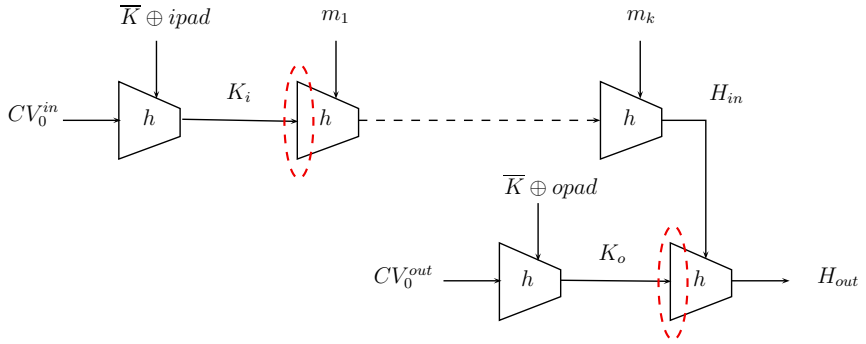

FIGURE 7.2 – Les zones d'attaque contre une construction HMAC lors d'une analyse par canaux cachés.

Les attaques classiques contre la construction HMAC visent à récupérer les valeurs K_i et K_o. Même si leur connaissance ne permet pas de récupérer la valeur secrète K, elle permet toutefois de forger entièrement le MAC. En effet, en connaissant K_i et K_o il est possible de construire des codes légitimes indépendamment du message m et de la valeur H_{in}. Les deux zones d'attaque sont visibles à la figure 7.2. Puisque les techniques utilisées pour récupérer K_i et K_o sont similaires, nous nous concentrons dans le reste de cette section sur l'attaque de $K_i = h(CV_0^{in}, \overline{K} \oplus ipad)$.

Lors d'une attaque par canaux auxiliaires contre la construction *envelope* MAC, la situation est légèrement différente. Dans le cas de HMAC, la connaissance seule de K_i et K_o était

FIGURE 7.3 – Les zones d'attaque contre une construction *envelope* MAC lors d'une analyse par canaux cachés.

suffisante pour forger le MAC. Mais pour une attaque réussie contre *envelope* MAC cela ne suffit pas ; la clé secrète K doit être impérativement récupérée. Toutefois, l'effort fourni pour une attaque statistique efficace contre *envelope* MAC n'est pas différente du cas de HMAC, puisque ici aussi deux étapes sont nécessaires. La valeur de K_i doit d'abord être récupérée, puis l'insertion de la clé au dernier appel de la fonction de compression doit être ciblée en traitant plusieurs messages différents m.

7.5.2 Analyse de Grøstl

La fonction de hachage Grøstl est parmi les 5 finalistes du concours SHA-3. Sa description complète se trouve à la section 4.5. Nous présentons ici une analyse statistique de consommation de courant pour la fonction Grøstl-256 lorsque celle-ci est utilisée dans un MAC. Pour des raisons expliquées précédemment nous avons seulement implémenté la version HMAC pour cette fonction, que nous appellerons désormais HMAC-Grøstl-256. Une analyse théorique pour l'utilisation de Grøstl dans un envelope-MAC figure à la fin de la section.

La première étape de notre étude consiste à simplement identifier les opérations qui sont sensibles vis-à-vis d'une analyse statistique du courant et doivent alors être protégées. Notre but est de récupérer les valeurs K_i et K_o. Pour cela nous commençons par déterminer les fonctions de sélection potentielles (opérations où les données secrètes et publiques sont mélangées).

Trois opérations simples de ce type ont pu être identifiées. Elles sont illustrées à la figure 7.4. La première opération est le XOR entre h_{i-1} et m_i. Lors d'une configuration HMAC, quand le premier bloc de message est traité, h_{i-1} est égal à K_i. La deuxième opération sensible est la première application de `SubBytes`. Ces deux fonctions de sélection ont également été identifiées dans [BP10]. Puisqu'une analyse par canaux cachés est plus efficace quand l'opération cible est une fonction non-linéaire (en comparaison avec une fonction linéaire), les auteurs dans [BP10] avaient choisi d'attaquer seulement l'opération `SubBytes`. Dans notre étude, les deux fonctions ont été attaquées, afin de comparer leur efficacité dans le cadre d'une vraie attaque.

Une troisième opération qui peut potentiellement devenir une opération cible lors d'une attaque par canaux cachés est le XOR entre h_{i-1}, $Q(m_i)$ et $P(h_{i-1} \oplus m_i)$ pour calculer la valeur de chaînage h_i. L'applicabilité d'une attaque sur cette opération dépend de la façon dont elle est implémentée en pratique ; trois façons différentes existent. La première consiste à calculer l'opération XOR entre $Q(m_i)$ et $P(h_{i-1} \oplus m_i)$ et de XOR-er ce résultat avec h_{i-1} :

$$\begin{aligned} \texttt{tmp} &= Q(m_i) \oplus P(h_{i-1} \oplus m_i) \\ h_i &= h_{i-1} \oplus \texttt{tmp} \ . \end{aligned}$$

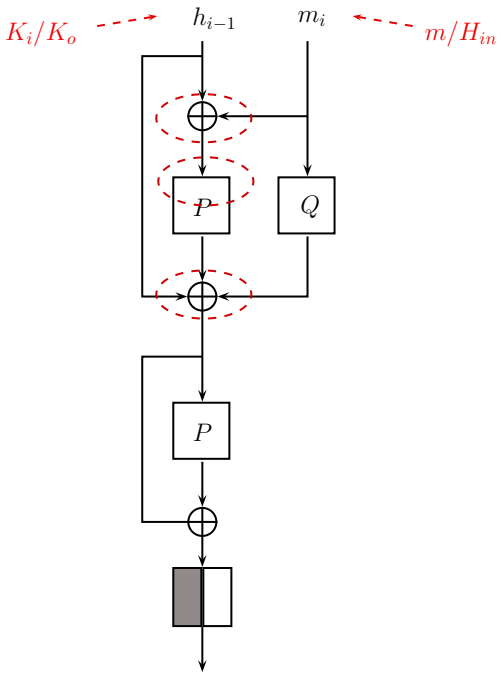

FIGURE 7.4 – Un appel à la fonction de hachage Grøstl-256 avec un bloc de message lors de son utilisation dans HMAC. Les trois opérations sensibles identifiées sont entourées.

La deuxième approche consiste à calculer l'opération XOR entre $P(h_{i-1} \oplus m_i)$ et h_{i-1} et de XOR-er cela à $Q(m_i)$:

$$\begin{aligned} \texttt{tmp} &= P(h_{i-1} \oplus m_i) \oplus h_{i-1} \\ h_i &= Q(m_i) \oplus \texttt{tmp} \ , \end{aligned}$$

tandis que la troisième manière combine d'abord h_{i-1} avec $Q(m_i)$ et ajoute ensuite $P(h_{i-1} \oplus m_i)$:

$$\begin{aligned} \texttt{tmp} &= Q(m_i) \oplus h_{i-1} \\ h_i &= P(h_{i-1} \oplus m_i) \oplus \texttt{tmp} \ . \end{aligned}$$

Les deux premières manières d'implémenter cette opération ne présentent aucune menace visible pour la fonction de hachage, puisque $P(h_{i-1} \oplus m_i)$ est une valeur inconnue et variable. Lors d'une attaque statistique de courant, l'attaquant doit être en mesure de contrôler une de deux entrées de la fonction de sélection. Or, $P(h_{i-1} \oplus m_i)$ ne peut pas être contrôlé puisque la variable h_{i-1} est inconnue. Cependant, la première opération de la troisième implémentation est similaire à l'opération XOR entre m_i et h_{i-1}, avec la seule différence que $Q(m_i)$ est utilisé à la place de m_i. Elle est alors vulnérable à une attaque par canaux cachés. Il faut souligner que dans le code source donné par les concepteurs, cette troisième implémentation problématique a été évitée.

Nous avons ensuite effectué une analyse de courant par corrélation sur HMAC-Grøstl-256 en visant les deux opérations identifiées. Même si l'attaque d'une implémentation non-protégée doit en théorie être toujours réussie, nous avons malgré tout effectué cette attaque pour les deux raisons suivantes. La première raison est pour valider nos estimations théoriques et pour

montrer qu'une fuite d'information peut en effet être observée. La deuxième raison consiste à voir à quel point l'information secrète peut être récupérée, puisque la réussite d'une telle attaque dépend souvent de l'architecture et de l'opération visée.

Une remarque importante est que les deux opérations sensibles, c'est-à-dire le XOR initial et la première couche des boîtes-S, peuvent être capturées dans la même trace de courant avec une très bonne résolution de signal. Ceci peut être observé à la figure 7.5, où les deux zones d'attaque sont clairement visibles. Du point de vue de l'attaquant, cette observation est précieuse car un seul ensemble de courbes doit être généré.

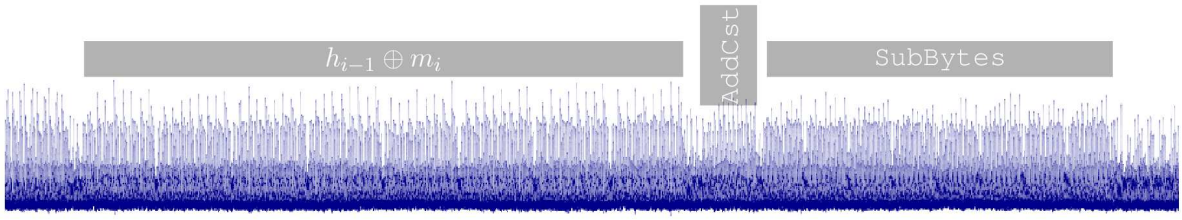

FIGURE 7.5 – Analyse simple de courant (SPA) des zones d'attaque dans HMAC-Grøstl : l'absorption du message et l'opération `SubBytes`

Nous avons capturé 5000 courbes et les avons ensuite traitées en utilisant la méthode de l'analyse par corrélation du courant (CPA). Des fuites visibles ont pu être observées pour les deux opérations et tous les octets de la clé ont pu être retrouvés. Cependant, comme attendu, les pics de corrélation pour la bonne valeur de sous-clé sont plus prononcés pour l'opération `SubBytes` que pour l'opération XOR. Les résultats de notre analyse CPA contre la fonction `SubBytes` sont illustrés à la figure 7.6.

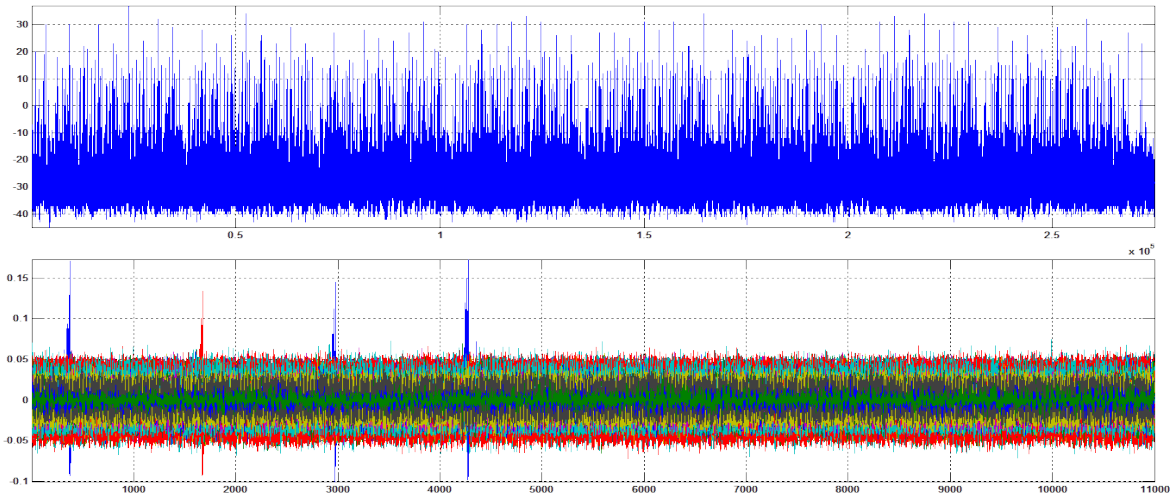

FIGURE 7.6 – Analyse CPA sur l'opération `SubBytes` de HMAC-Grøstl (4 premiers octets de la valeur cible)

Cette analyse a confirmé le besoin de protéger efficacement ces deux opérations.

Les concepteurs de Grøstl ont proposé l'utilisation de la construction *envelope* MAC afin de transformer Grøstl en code d'authentification de message. Nous avons choisi d'implémenter sur la carte seulement la construction HMAC, mais il est facile de voir qu'en termes d'analyse

par canaux cachés la procédure de l'attaque sur le HMAC est identique à une attaque sur le *envelope* MAC. La seule différence consiste en la nature de l'information récupérée lors d'une attaque réussie. En effet, en attaquant *envelope* MAC, nous voyons que durant le dernier appel de la fonction de compression les rôles des valeurs secrètes et publiques sont échangés. Puisque ces rôles sont entièrement symétriques, le scénario de l'attaque reste le même, mais la valeur obtenue est cette fois-ci la clé secrète. Si la même clé secrète est aussi utilisée pour d'autres applications de la carte, l'attaque peut alors devenir très dévastatrice, contrairement au cas de HMAC, où la clé d'origine n'est pas récupérée.

Nous avons ensuite proposé des contre-mesures pour masquer les opérations sensibles de Grøstl. Ces contre-mesures s'appliquent évidement aussi bien dans le cas de HMAC que dans le cas de *envelope* MAC.

Contre-mesures pour Grøstl-256

Nous sommes arrivés à des contre-mesures très simples pour Grøstl masquant les données sensibles tout au long du calcul du MAC. Ces contre-mesures devraient être suffisantes pour protéger la fonction contre toute analyse statistique du premier ordre. D'après l'analyse précédente, les opérations critiques sont le XOR entre le nouveau bloc de message et la valeur de chaînage ainsi que l'opération `SubBytes`.

Afin de protéger l'opération XOR, un masque booléen R de 512 bits est généré une seule fois. Ce masque est XOR-é à la valeur de chaînage et le feed-forward la réinjecte automatiquement au début de chaque fonction de compression. Ce masque R, que nous appelons *masque global* doit être supprimé en le re-XOR-ant simplement à l'état, juste avant la troncation finale. Sa propagation, tout comme le calcul non-protégé de la fonction, peuvent être observés à la figure 7.7.

En revanche, afin de passer par la couche non-linéaire, le masque global doit être supprimé au début de chaque permutation P. Par conséquent, un autre type de protection est nécessaire afin de masquer le calcul des boîtes-S et le reste de la permutation. Pour cela, nous avons choisi de masquer la table en utilisant la méthode proposée dans [Mes01] et décrite à la section 7.3.3. Nous avons alors généré au début du calcul deux masques u et v, et nous avons construit la boîte-S masquée S', à partir de la boîte-S S de l'AES, en calculant pour chaque $x \in \mathbf{F}_{2^8}$

$$S'(x+u) = S(x) + v.$$

La boîte-S S' ainsi que les valeurs u et v sont stockées dans la RAM et réutilisées lors des calculs suivants.

Il faut noter que cette contre-mesure pour la partie non-linéaire n'est pas suffisante pour protéger la fonction contre les attaques statistiques d'ordre supérieur. Pour cela un masque d'ordre supérieur devrait être utilisé.

Ce nouveau masque, que nous appelons *local*, doit être appliqué à l'état avant que le masque global soit supprimé, afin que les variables sensibles restent protégées à tout moment du calcul. À la figure 7.7, la permutation P' représente la permutation P à laquelle les contre-mesures ci-dessus sont appliquées.

Pour vérifier l'efficacité de ces contre-mesures, nous les avons implémentées sur la carte à puce et nous avons appliqué à nouveau la même attaque que pour la version non-protégée. En effet, il est possible de vérifier sur la figure 7.8 qu'aucune fuite n'est désormais observable.

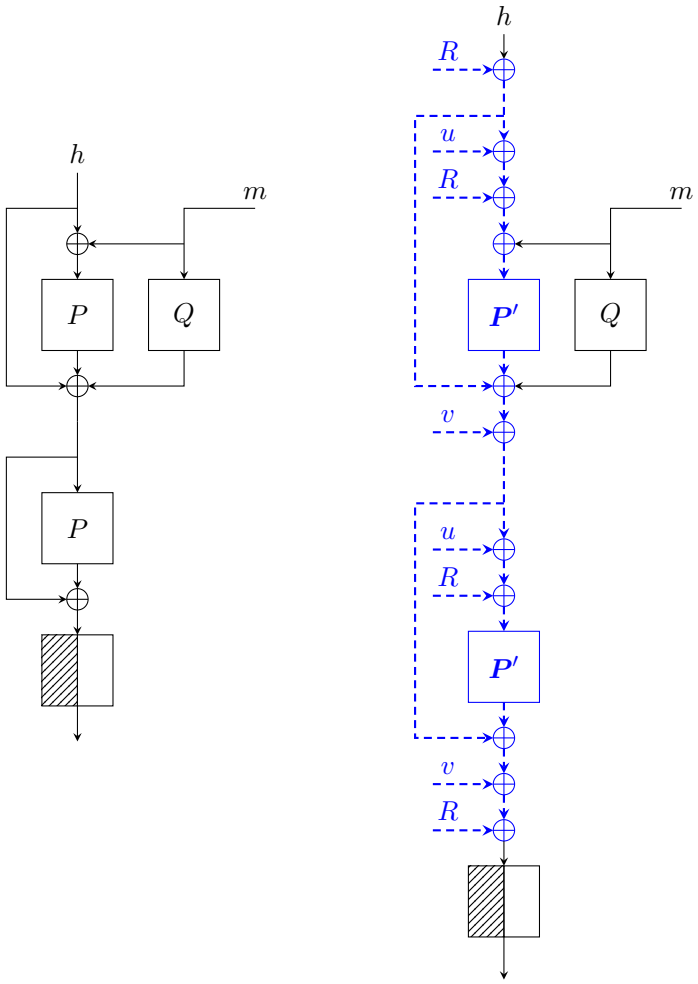

FIGURE 7.7 – Comparaison des implémentations simple et sécurisée de Grøstl

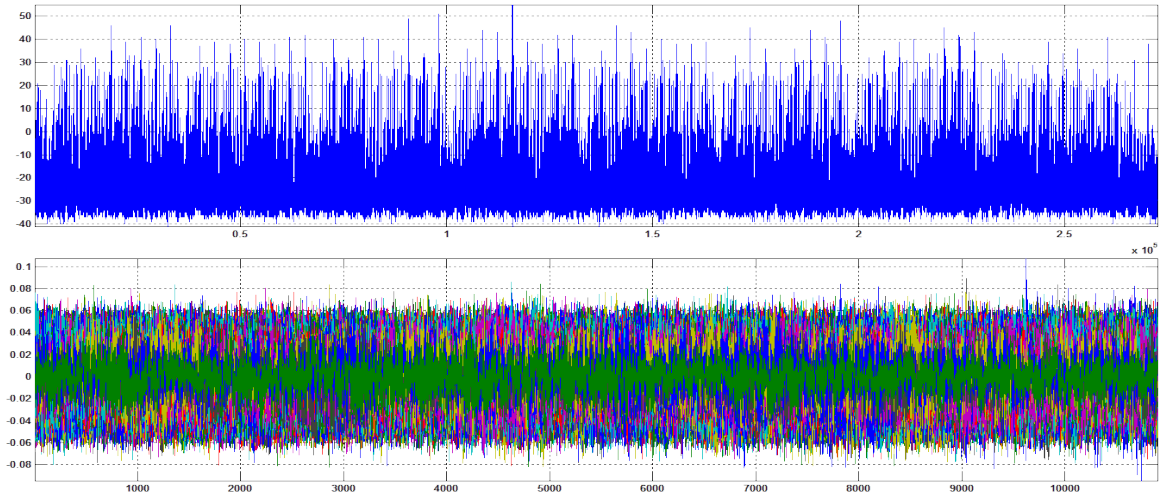

FIGURE 7.8 – Analyse CPA sur l'opération `SubBytes` de HMAC-Grøstl sécurisé (4 premiers octets de la valeur cible)

7.5.3 Analyse de Skein

Skein est une fonction de hachage finaliste du concours SHA-3, conçue par Ferguson *et al.* [FLS+10]. Elle est basée sur le chiffrement par blocs adaptable [LRW02], Threefish. L'état interne est divisé en mots de 64 bits et peut avoir une taille de 256, 512 ou 1024 bits. La taille de la clé du chiffrement est égale à la taille de l'état tandis que la taille du tweak est fixée à 128 bits pour toutes les trois versions. La valeur du tweak pour chaque bloc code le nombre d'octets déjà traités accompagné de quelques informations supplémentaires.

Dans notre étude nous avons analysé Skein-512-256, c'est-à-dire la version dont l'état interne est de 512 bits et les empreintes calculées de 256 bits. Par conséquent, nous n'allons décrire les spécifications que de cette instance.

Le chiffrement Threefish n'utilise que trois opérations mathématiques, à savoir le XOR, l'addition modulaire et les rotations par une constante sur des mots de 64 bits. Sa composante principale, appelée MIX, est décrite à la figure 7.9.

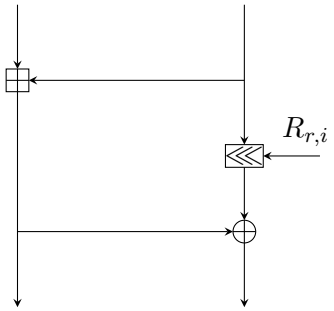

FIGURE 7.9 – L'opération MIX

Skein-512-256 est composé de 72 tours d'opérations basiques. Chaque tour est constitué de 4 applications en parallèle de la fonction MIX, suivie par une permutation sur les huit mots de l'état. Une nouvelle sous-clé est injectée tous les 4 tours du chiffrement. Quatre tours de Threefish-512 sont illustrés à la figure 7.10.

Toutes les fonctions internes opèrent sur des mots de 64 bits. L'état tout comme la clé sont composés de huit mots. Soient t_0 et t_1 les deux mots constituant le tweak et $k_0, \ldots, k_7$ les huit mots de la clé. La première sous-clé s^0 est donnée par

$$s^0 = (k_0, k_1, k_2, k_3, k_4, k_5 + t_0, k_6 + t_1, k_7),$$

où $+$ symbolise l'addition modulaire dans $\mathbb{Z}/(2^{64}-1)\mathbb{Z}$. La fonction de compression de Threefish est définie comme

$$h_i = E_{h_{i-1},T_i}(m_i) \oplus m_i,$$

où $E_{K,T}$ est le chiffrement par blocs Threefish, h_{i-1} la valeur de chaînage précédente, T_i le tweak et m_i le bloc de message.

Le mode opératoire de Skein est une variation du mode Matyas-Meier-Oseas (voir section 1.3.4), avec l'utilisation d'un tweak. Les concepteurs de Skein appellent ce mode spécifique UBI (abréviation de *Unique Block Iteration* en anglais). Pour effectuer du hachage standard un bloc de configuration est traité par la fonction de compression avant de hacher le message lui-même. Un autre appel à la fonction de compression est également effectué une fois tous les blocs de message traités. Un tweak de 128 bits unique est utilisé pour chaque appel à la fonction de compression.

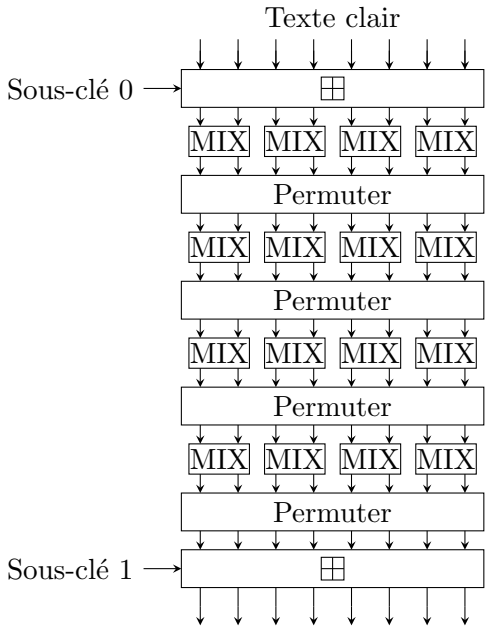

FIGURE 7.10 – Quatre tours de Threefish

Skein-MAC Dans le document de soumission [FLS+10] il est mentionné que Skein peut être naturellement utilisé comme un HMAC, mais cette application n'est pas recommandée car elle n'est pas assez efficace pour traiter des messages de petite taille. Alternativement, une méthode simple pour transformer Skein en MAC peut être visualisée à la figure 7.11. Elle consiste à traiter d'abord la clé en fixant la valeur de chaînage à 0, ensuite un bloc de configuration et puis le message.

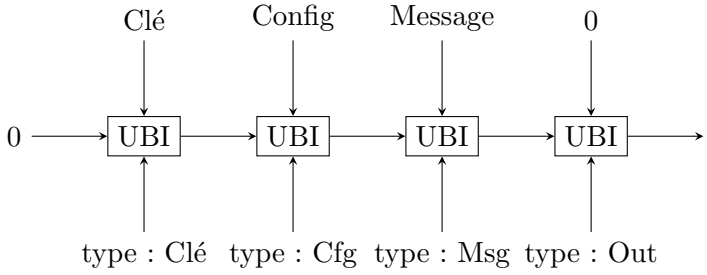

FIGURE 7.11 – La construction Skein-MAC

Analyse par canaux cachés de Skein

Nous présentons dans cette section une analyse de Skein-512-256 quand celui-ci est utilisé dans un MAC. Nous avons cherché des stratégies d'attaque pour HMAC-Skein et Skein-MAC et nous avons identifié exactement les mêmes fonctions de sélection pour les deux constructions. Cependant, les valeurs retrouvées lors d'une attaque par canaux cachés sont différentes pour les deux configurations. Pour HMAC, afin de forger le code, la récupération de K_i et K_o est nécessaire, tandis que pour Skein-MAC seulement une valeur secrète, à savoir la sortie

du deuxième appel à la fonction de compression doit être retrouvée afin que l'attaque soit réussie. Dans notre analyse, les remarques mentionnées pour HMAC s'appliqueront également à Skein-MAC.

Comme déjà mentionné dans [ZKS11], l'opération la plus facile à attaquer pour une attaque par analyse statistique du courant est l'addition modulaire entre la première sous-clé et le bloc de message. Dans une configuration HMAC, si $m_0, \ldots, m_7$ sont les huit mots de 64 bits du premier bloc de message et $k_0, \ldots, k_7$ sont les huit mots de la première sous-clé, cette opération est simplement

$$k_i \boxplus m_i, \text{ pour } i = 0, \ldots, 7.$$

La zone d'attaque est visible à la figure 7.12.

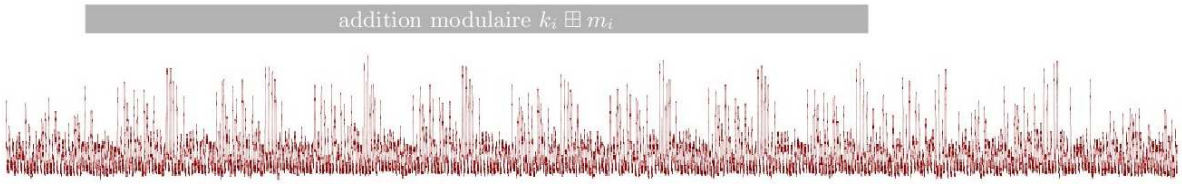

FIGURE 7.12 – Analyse simple de courant (SPA) de la zone d'attaque dans HMAC-Skein

Nous avons ensuite mené une attaque par analyse de corrélation du courant sur la version non-protégée de Skein-MAC, exactement comme nous l'avons fait auparavant pour Grøstl. Les huit additions ont été ciblées afin de récupérer la valeur secrète mot par mot. Comme précédemment, 5000 courbes ont été analysées. Cependant, comme nous pouvons le voir à la figure 7.13 les résultats de cette expérimentation sont beaucoup plus surprenants que nous l'avions imaginé. Si on note $(b_0, \ldots, b_7)$ les octets d'un mot de 64 bits de la première sous-clé, seulement les octets b_0, b_4, b_5, b_6 et b_7 ont pu être récupérés en pratique. Mais même parmi ces 5 octets, le pic le plus prononcé correspondait à la vraie valeur seulement pour b_0, tandis que pour les quatre autres octets un effort supplémentaire a été nécessaire afin de retrouver la valeur correcte parmi quelques autres faux pics.

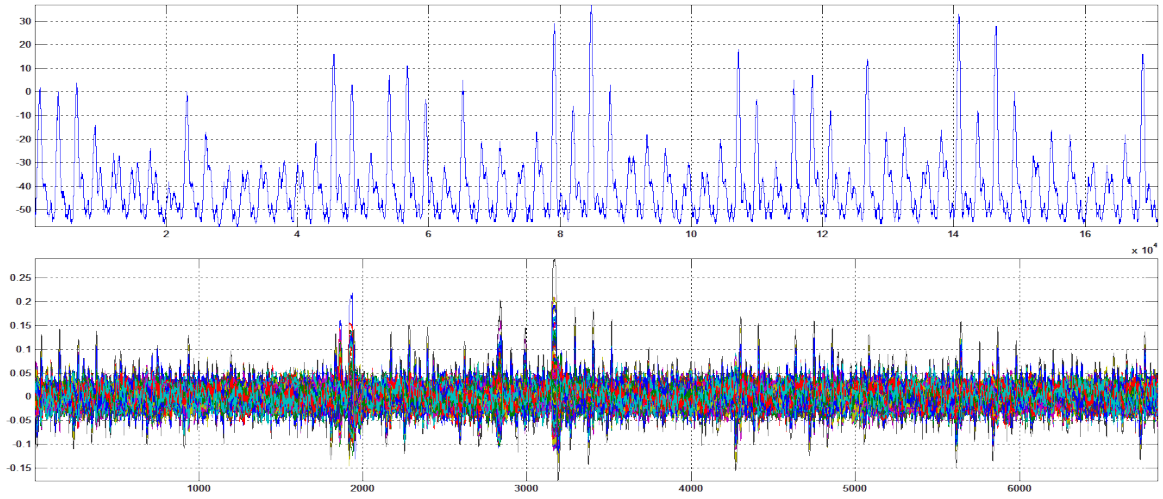

FIGURE 7.13 – CPA sur l'addition modulaire

La raison de ce comportement inattendu peut venir de la façon dont une addition modulaire entre valeurs de 64 bits est réalisée sur une architecture 32 bits. Toutefois, ce problème qui n'est probablement pas spécifique à Skein, doit être examiné en détail. Dans notre cas, même si la clé secrète n'est pas entièrement récupérée, la fuite observée est assez importante. En conséquence, la protection de cette addition modulaire est nécessaire.

Contre-mesures pour Skein

Il est clair que l'addition modulaire entre le message et la clé doit être protégée, toutefois cette protection seule n'est pas suffisante pour garantir la sécurité de la fonction entière. Cela est dû au fait que la diffusion assurée par la fonction MIX à l'intérieur d'un seul tour n'est pas assez importante. En conséquence, les opérations suivant l'addition modulaire doivent également être protégées. Cependant, la protection des fonctions de type ARX nécessite des conversions consécutives entre des masques arithmétiques et booléens. Ces conversions peuvent être très chères (voir section 7.3.3). Un compromis entre la sécurité et la performance doit alors être trouvé. Nous avons décidé que la protection des 4 premiers tours assure une bonne diffusion du secret de façon que la fonction résiste à toute les attaques par analyse statistique du premier ordre, en ayant en même temps un impact très limité sur la sécurité.

Dans la section 7.3.3 nous avons présenté les principales méthodes de conversion entre les masques booléens et arithmétiques. La première méthode, due à Louis Goubin [Gou01] comprend un algorithme très efficace pour la conversion d'un masque booléen vers un masque arithmétique et un algorithme moins rapide pour la transformation inverse. Le nombre d'opérations de ce dernier algorithme dépend de la taille des données traitées, qui sont dans notre cas des mots de 64 bits. La deuxième méthode largement utilisée est celle de Alexei Tchulkine et de Jean-Sébastien Coron [CT03]. Dans cette méthode, des tableaux de masquage de grande taille sont utilisés afin d'éviter le grand nombre d'opérations coûteuses, gagnant ainsi en la performance aux dépens de la mémoire.

Puisque Skein est un candidat assez rapide, garder les ressources en mémoire à un niveau bas en demandant un peu plus de temps de traitement est un compromis plus intéressant. Pour cette raison, nous avons décidé d'implémenter la solution de Goubin [Gou01] pour protéger 4 tours de Skein.

Dans la permutation de Skein, les indices pairs et impairs ne sont pas mélangés entre eux. Pour cette raison nous pouvons utiliser un seul masque arithmétique R_o pour les mots impairs et le même masque arithmétique R_e peut être utilisé pour les mots pairs. De cette façon, le nombre d'appels au générateur pseudo-aléatoire ainsi que la mémoire RAM pour stocker les masques sont réduits.

Comme nous l'avons déjà mentionné, la conversion coûteuse est la conversion d'un masque arithmétique à un masque booléen. Dans notre implémentation, le temps nécessaire pour cette conversion est approximativement 16 fois plus élevé que le temps pour faire la conversion inverse, en utilisant l'algorithme de [Gou01]. Dans une approche directe, puisque l'insertion de la clé est une addition modulaire, il semble naturel d'utiliser deux masques arithmétiques. Cette implémentation naïve qui est illustrée à la figure 7.14, nécessite deux conversions arithmétique vers booléen pour chaque transformation MIX, c'est-à-dire 16 transformations pour chaque tour. Afin d'éviter ce grand nombre de conversions nous proposons d'appliquer un tweak juste avant la première couche des opérations MIX.

Soit R_o le masque arithmétique protégeant l'insertion de sous-clé des mots impairs. Juste avant l'opération MIX, une conversion arithmétique vers booléen est appliquée pour la branche

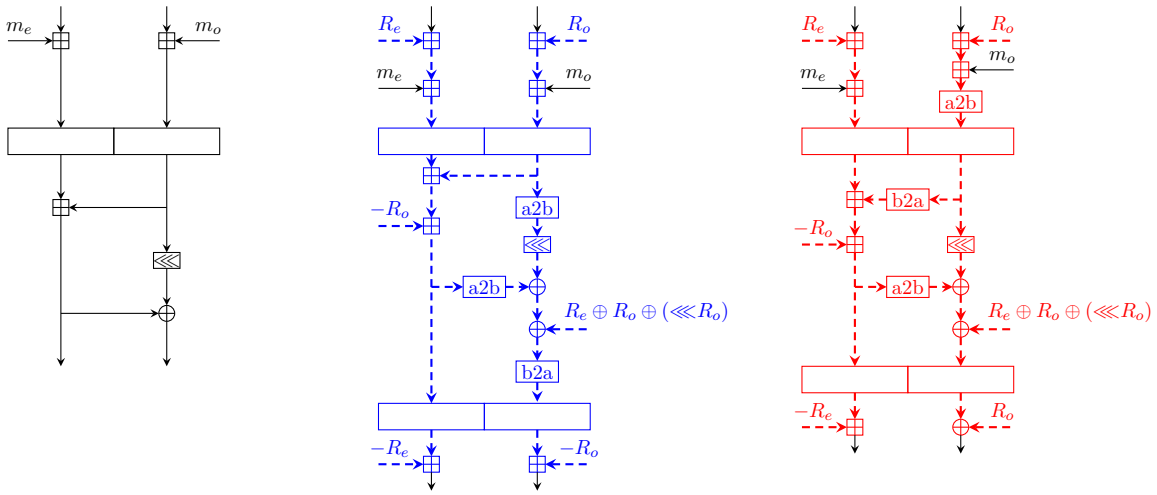

FIGURE 7.14 – Comparaison de l'implémentation simple et sécurisée de Skein (sans et avec le tweak)

impaire de chaque opération MIX. De cette manière, la branche impaire de chaque opération MIX est protégée par un masque arithmétique, tandis que la branche paire est protégée par un masque booléen. Les opérations suivantes sont alors effectuées pour les 32 transformations MIX des quatre premiers tours.

Une conversion arithmétique vers booléen est appliquée à la branche droite de la transformation MIX avant l'addition modulaire et la conversion inverse est appliquée à la branche gauche, comme nous pouvons le voir à la figure 7.14. Avec cette méthode, une seule conversion arithmétique vers booléen est effectuée à l'intérieur de chaque MIX. En conséquence, avec cette optimisation, 8 conversions arithmétique vers booléen sont effectuées avant la première transformation de tour et 32 sont faites pendant les 4 premiers tours, à la place des 64 conversion effectuées auparavant. Grâce à ce tweak, une gain de performance de 30% pour le calcul du HMAC complet est observée.

Afin de vérifier l'efficacité de nos contre-mesures nous avons essayé de monter une attaque par analyse par corrélation de courant contre l'implémentation protégée de HMAC-Skein. Les résultats de cette analyse ne montrent que l'apparition d'un seul pic, correspondant à la clé nulle, c'est-à-dire à la corrélation avec le message.

7.5.4 Analyse de performance pour les deux candidats

Afin de pouvoir comparer les deux candidats, nous avons choisi d'implémenter le mode HMAC, même si ce n'est le mode le plus performant pour aucune de deux fonctions. La différence de temps d'exécution pour chaque HMAC quand un bloc de message est traité peut être un exemple de comparaison. Ces mesures peuvent être observées à la table 7.1. Dans cette table, la quantité de RAM nécessaire afin d'implémenter les paramètres de sécurité y est également mentionnés, ainsi que la taille de code supplémentaire.

Ces mêmes résultats peuvent également être visualisés à la figure 7.15. Elle fournit les courbes de consommation de courant produites par un oscilloscope pendant le calcul de HMAC-Grøstl et HMAC-Skein pour un seul bloc de message, pour des implémentations simples et sécurisées. En particulier, nous pouvons voir qu'avant la sécurisation HMAC-Grøstl

Algorithme	Mesure du temps à 8MHz		RAM supplémentaire		Code supplémentaire
	code de réf.	code sécurisé	statique	tas	
HMAC-Grøstl	453 ms	486 ms (+7.2%)	+325 oct.	0	+688 oct.
HMAC-Skein	77.7 ms	155 ms (+100%)	0	+32 oct.	+3484 oct.

TABLE 7.1 – Surcoût du code sécurisé en termes de temps, de consommation de RAM et de la taille du code quand des messages d'un seul bloc sont traités.

est 6 fois plus lent que HMAC-Skein, tandis qu'après l'application des contre-mesures, le HMAC-Grøstl n'est que 3 fois plus lent.

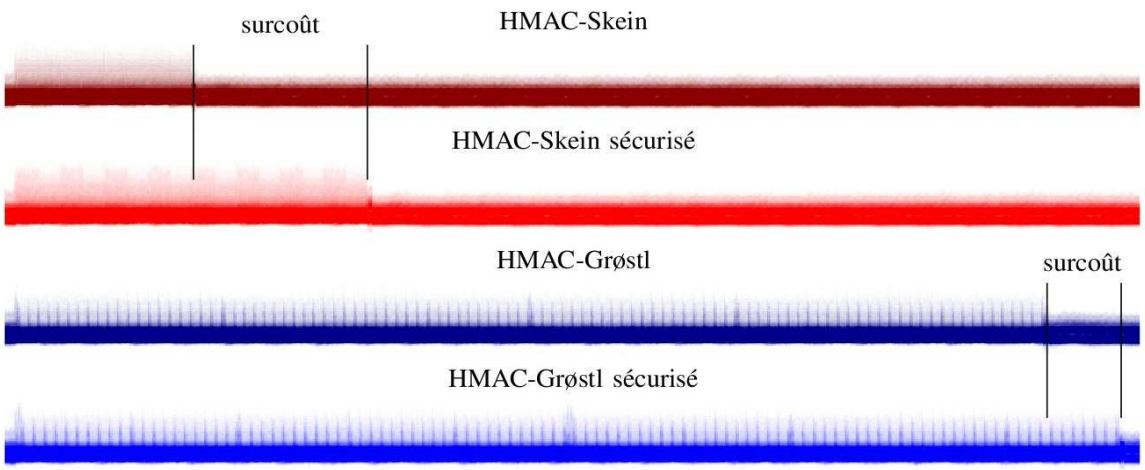

FIGURE 7.15 – Consommation de courant pour l'implémentation de référence et l'implémentation sécurisée pour HMAC-Skein et HMAC Grøstl

En conclusion, nous pouvons noter que les contre-mesures que nous avons proposées pour Grøstl sont très efficaces et protègent entièrement HMAC-Grøstl (et en conséquence Grøstl envelope MAC) avec un surcoût de seulement 7 % sur la performance totale. Au contraire, pour Skein la situation est beaucoup plus compliquée, puisque un grand nombre de conversions de masques est nécessaire afin de protéger la fonction efficacement.

Conclusions

Dans la première partie de cette thèse nous avons analysé des propriétés algébriques des primitives symétriques qui sont au coeur de plusieurs fonctions de hachage. Tout d'abord, nous avons étudié et formalisé un nouveau type de distingueur, le *distingueur à sommes nulles*. En particulier, nous avons vu que ce type de distingueur est lié à la fois aux composantes non-linéaires et linéaires de la fonction. Nous avons ensuite étudié la façon dont évolue le degré algébrique des permutations itérées et nous avons établi deux nouvelles bornes pour le degré de ces constructions. Ces bornes reposent sur l'exploration des propriétés des permutations de petite taille présentes dans la partie non-linéaire des primitives symétriques et connues sous le nom de *boîtes-S*. Cette contribution est assez importante puisque ce sujet a été très peu étudié jusqu'à présent. Nous nous sommes également intéressés à un autre type de propriétés des boîtes-S, également lié à leur degré algébrique, qui permet de trouver des relations affines entre les entrées et les sorties d'une permutation itérée. Cette analyse nous a permis d'améliorer une cryptanalyse connue de la fonction de hachage Hamsi, mais également d'identifier les caractéristiques des boîtes-S qui sont responsables de ce type de faiblesses.

Dans la deuxième partie de cette thèse nous avons analysé la résistance contre les attaques par canaux cachés de deux fonctions finalistes du concours SHA-3. Ce sujet complémentaire à la sécurité mathématique est également assez important, puisque la fonction sélectionnée par le NIST sera implémenté dans tout sorte de dispositif et sera alors la cible d'une multitude d'attaques physiques.

Les sujets que j'ai abordés pendant ces trois ans laissent des questions ouvertes, qui demandent à être explorées. Une première voie est d'examiner à quel point les bornes sur le degré que nous avons établies sont proches du degré réel d'une fonction et si elles peuvent être améliorées dans certains cas. On peut aussi se demander si l'attaque que nous avons présentée au chapitre 5 contre cinq tours du chiffrement $\mathcal{KN}'$ peut s'étendre à plus de tours. Une piste également intéressante consiste à étudier si la permutation inverse de la fonction interne dans un schéma de Feistel influence la sécurité du chiffrement d'une manière autre qu'à travers le degré algébrique. Au chapitre 6 nous avons analysé des propriétés algébriques des boîtes-S à 4 variables qui jouent un rôle dans la propagation des relations affines entre les bits d'entrée et de sortie d'une primitive symétrique. Cependant, il existe plusieurs systèmes symétriques qui intègrent des permutations non-linéaires d'une taille différente. Une approche alors naturelle est d'étendre notre étude à toutes ces autres permutations.

Jusqu'à présent, je me suis principalement intéressée à la sécurité des fonctions de hachage et il s'agit d'une voie que j'aimerais continuer à explorer par la suite. Toutefois, j'aimerais également m'investir à une autre branche de la cryptographie symétrique, celle des chiffrements par blocs. Ces deux familles de constructions symétriques sont très proches et les avancées dans ces deux domaines sont très fortement liées. Récemment, plusieurs constructions nouvelles de chiffrements par blocs ont vu le jour. La plupart de ces fonctions ont été

conçues pour être implantées dans des environnements restreints, comme les téléphones portables ou les étiquettes RFID. La sécurité de ces chiffrements par blocs, connus sous le nom de *chiffrements à bas coût*, n'a été que très peu étudiée. De plus, les contraintes de construction de ces algorithmes imposent l'utilisation des structures très particulières. Par exemple, il est commun de retrouver dans ces algorithmes des boîtes-S de très petite taille ou des boîtes-S qui sont des involutions, puisque leur implémentation est peu coûteuse en termes de mémoire. Pour la même raison, la partie linéaire de ces chiffrements est souvent très allégée, composée par exemple d'une simple permutation des bits. Ces constructions sont alors susceptibles de posséder de fortes propriétés algébriques. Il est alors très intéressant d'étudier l'impact de l'utilisation de ces briques particulières sur la sécurité de ces fonctions. En particulier, on peut se demander si les résultats des chapitres 4, 5 et 6 peuvent être améliorés dans ce type de cas particuliers.

Bibliographie

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier et Adi Shamir : Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. *Dans* Orr Dunkelman, éditeur : *FSE'09*, volume 5665 de *Lecture Notes in Computer Science*, pages 1–22. Springer, 2009. 2.2.3, 3.7.1

[AG01] Mehdi-Laurent Akkar et Christophe Giraud : An Implementation of DES and AES, Secure against Some Attacks. *Dans* Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs : *CHES'01*, volume 2162 de *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001. 7.3.3

[AKK+10] Jean-Philippe Aumasson, Emilia Käsper, Lars R. Knudsen, Krystian Matusiewicz, Rune Steinsmo Ødegård, Thomas Peyrin et Martin Schläffer : Distinguishers for the Compression Function and Output Transformation of Hamsi-256. *Dans* Ron Steinfeld et Philip Hawkes, éditeurs : *ACISP'10*, volume 6168 de *Lecture Notes in Computer Science*, pages 87–103. Springer, 2010. 3.2.2

[AM09] Jean-Philippe Aumasson et Willi Meier : Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi, 2009. Comment on the NIST SHA-3 Hash Competition. (document), 3, 3.1, 3.4, 3.4.1, 3.4.1, 3.4.2, 3.6.1, 3.6.2, 3.6.3, 4, 4.3.6, 7.5.4

[BBG+09] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw et Yannick Seurin : SHA-3 Proposal : ECHO. Submission to NIST (Round 2), 2009. 5.5.3

[BC10] Christina Boura et Anne Canteaut : A zero-sum property for the Keccak-f permutation with 18 rounds. *Dans ISIT'10*, pages 2488–2492. IEEE, 2010. (document), 3.2

[BC11a] Christina Boura et Anne Canteaut : On the Algebraic Degree of Iterated Permutations. *Dans Finite Fields and Applications - Fq10, Gent, Belgium*, July 2011. (document)

[BC11b] Christina Boura et Anne Canteaut : Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256. *Dans* Alex Biryukov, Guang Gong et Douglas R. Stinson, éditeurs : *Selected Areas in Cryptography-SAC'10*, volume 6544 de *Lecture Notes in Computer Science*, pages 1–17. Springer, 2011. (document), 3.2, 3.4.2

[BC12a] Christina Boura et Anne Canteaut : On the Algebraic Degree of some SHA-3 Candidates. *Dans Proceedings of the Third SHA-3 Candidate Conference, Washington D.C, March 22-23*, 2012. (document)

[BC12b] Christina Boura et Anne Canteaut : On the influence of the algebraic degree of F^{-1} on the algebraic degree of $G \circ F$. *IEEE Transactions on Information Theory*, 2012. À parraître. (document), 5

[BCD11] Christina Boura, Anne Canteaut et Christophe De Cannière : Higher-Order Differential Properties of Keccak and *Luffa*. *Dans* Antoine Joux, éditeur : *FSE'11*, volume 6733 de *Lecture Notes in Computer Science*, pages 252–269, 2011. (document), 3.2, 4

[BCK96] Mihir Bellare, Ran Canetti et Hugo Krawczyk : Keying Hash Functions for Message Authentication. *Dans* Neal Koblitz, éditeur : *CRYPTO'96*, volume 1109 de *Lecture Notes in Computer Science*, pages 1–15, 1996. 1.3.6

[BCO04] Eric Brier, Christophe Clavier et Francis Olivier : Correlation Power Analysis with a Leakage Model. *Dans* Marc Joye et Jean-Jacques Quisquater, éditeurs : *CHES'04*, volume 3156 de *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. 7.2.4, 7.2.5

[BD06] Eli Biham et Orr Dunkelman : A Framework for Iterative Hash Functions : HAIFA . Dans Proceedings of Second NIST Cryptographic Hash Workshop, "http://csrc.nist.gov/groups/ST/hash/second_workshop.html", Août 2006. 1.3.3

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : On the Indifferentiability of the Sponge Construction. *Dans* Nigel P. Smart, éditeur : *EUROCRYPT'08*, volume 4965 de *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. 1.3.3, 1.3.3

[BDPV09] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Keccak sponge function family main document. Submission to NIST (Round 2), 2009. 3.6

[BDPV10] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Note on zero-sum distinguishers of Keccak-f. Public comment on the NIST Hash competition, available at http://keccak.noekeon.org/NoteZeroSum.pdf, 2010. 3.2.2

[BGK04] Johannes Blömer, Jorge Guajardo et Volker Krummel : Provably Secure Masking of AES. *Dans* Helena Handschuh et M. Anwar Hasan, éditeurs : *Selected Areas in Cryptography-SAC'04*, volume 3357 de *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004. 7.3.3

[BJMM12] Anja Becker, Antoine Joux, Alexander May et Alexander Meurer : Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding. *Dans* David Pointcheval et Thomas Johansson, éditeurs : *EUROCRYPT'12*, volume 7237 de *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. 3.2.2

[BLR90] Manuel Blum, Michael Luby et Ronitt Rubinfeld : Self-Testing/Correcting with Applications to Numerical Problems. *Dans* Harriet Ortiz, éditeur : *STOC'90*, pages 73–83. ACM, 1990. 2.2.3

[BLV12] Christina Boura, Sylvain Lévêque et David Vigilant : Side-channel Analysis of Grøstl and Skein. *Dans TrustED, IEEE Computer Society Security and Privacy Workshops-SPW'12*, pages 16–26. IEEE, 2012. (document), 7.5

[BM97] Mihir Bellare et Daniele Micciancio : A New Paradigm for Collision-Free Hashing : Incrementality at Reduced Cost. *Dans* Walter Fumy, éditeur : *EUROCRYPT'97*, volume 1233 de *Lecture Notes in Computer Science*, pages 163–192. Springer, 1997. 3.2.2

[BP10] Olivier Benoît et Thomas Peyrin : Side-Channel Analysis of Six SHA-3 Candidates. *Dans* Stefan Mangard et François-Xavier Standaert, éditeurs : *CHES'10*, volume 6225 de *Lecture Notes in Computer Science*, pages 140–157. Springer, 2010. 7.4, 7.5.2

[BR06] Mihir Bellare et Thomas Ristenpart : Multi-Property-Preserving Hash Domain Extension and the EMD Transform. *Dans* Xuejia Lai et Kefei Chen, éditeurs : *ASIACRYPT'06*, volume 4284 de *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006. 1.3.3

[BRJ+02] Paulo S. L. M. Barreto, Vincent Rijmen, Jorge Nakahara Jr., Bart Preneel, Joos Vandewalle et Hae Yong Kim : Improved SQUARE Attacks against Reduced-Round HIEROCRYPT. *Dans* Mitsuru Matsui, éditeur : *FSE'01*, volume 2355 de *Lecture Notes in Computer Science*, pages 165–173. Springer, 2002. 2.2.4

[BRS02] John Black, Phillip Rogaway et Thomas Shrimpton : Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. *Dans* Moti Yung, éditeur : *CRYPTO'02*, volume 2442 de *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002. 1.3.4

[BS91] Eli Biham et Adi Shamir : Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991. 2.2.2

[BS93] Eli Biham et Adi Shamir : Differential Cryptanalysis of the Full 16-Round DES. *Dans* Ernest F. Brickell, éditeur : *CRYPTO'92*, volume 740 de *Lecture Notes in Computer Science*, pages 487–496, 1993. 2.2.2

[BT93] Andries E. Brouwer et Ludo M. G. M. Tolhuizen : A sharpening of the Johnson bound for binary linear codes and the nonexistence of linear codes with Preparata parameters. *Des. Codes Cryptography*, 3(2):95–98, May 1993. 3.2.1

[BW72] Elwyn R. Berlekamp et Lloyd R. Welch : Weight distributions of the cosets of the (32,6) Reed-Muller code. *IEEE Transactions on Information Theory*, 18(1): 203–207, 1972. 6.1.3

[CB08] David Canright et Lejla Batina : A Very Compact "Perfectly Masked" S-Box for AES. *Dans* Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis

et Moti Yung, éditeurs : *ACNS'08*, volume 5037 de *Lecture Notes in Computer Science*. Springer, 2008. 7.3.3

[CC98] Anne Canteaut et Florent Chabaud : A New Algorithm for Finding Minimum-Weight Words in a Linear Code : Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. 3.2.2

[CC03] Anne Canteaut et Pascale Charpin : Decomposing bent functions. *IEEE Transactions on Information Theory*, 49(8):2004–2019, 2003. 2.3, 6.1.3, 6.1.3

[CCCF01] Anne Canteaut, Claude Carlet, Pascale Charpin et Caroline Fontaine : On cryptographic properties of the cosets of $R(1, m)$. *IEEE Transactions on Information Theory*, 47(4):1494–1513, 2001. 2.4, 6.1.3

[CCD00] Christophe Clavier, Jean-Sébastien Coron et Nora Dabbous : Differential Power Analysis in the Presence of Hardware Countermeasures. *Dans* Çetin Kaya Koç et Christof Paar, éditeurs : *CHES'00*, volume 1965 de *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000. 7.2.4

[CCZ98] Claude Carlet, Pascale Charpin et Victor Zinoviev : Codes, Bent Functions and Permutations Suitable For DES-like Cryptosystems. *Des. Codes Cryptography*, 15(2):125–156, 1998. 3.2.1

[CDDL06] Anne Canteaut, Magnus Daum, Hans Dobbertin et Gregor Leander : Finding nonnormal bent functions. *Discrete Applied Mathematics*, 154(2):202–218, 2006. 6.1.1

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud et Prashant Puniya : Merkle-Damgård Revisited : How to Construct a Hash Function. *Dans* Victor Shoup, éditeur : *CRYPTO'05*, volume 3621 de *Lecture Notes in Computer Science*, pages 430–448, 2005. 1.3.3

[CG00] Jean-Sébastien Coron et Louis Goubin : On Boolean and Arithmetic Masking against Differential Power Analysis. *Dans* Çetin Kaya Koç et Christof Paar, éditeurs : *CHES'00*, volume 1965 de *Lecture Notes in Computer Science*, pages 231–237. Springer, 2000. 7.3.3

[CG06] Nicolas Courtois et Louis Goubin : An Algebraic Masking Method to Protect AES Against Power Attacks. *Dans* Dongho Won et Seungjoo Kim, éditeurs : *ICISC'05*, volume 3935 de *Lecture Notes in Computer Science*. Springer, 2006. 7.3.3

[CGPR06] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff et Matthieu Rivain : Attack and Improvement of a Secure S-Box Calculation Based on the Fourier Transform. *Dans* Louis Goubin et Mitsuri Matsui, éditeurs : *CHES'06*, volume 4249 de *Lecture Notes in Computer Science*, pages 216–230. Springer, 2006. 7.3.3

[Cha04] Pascale Charpin : Normal Boolean functions. *J. Complexity*, 20(2-3):245–265, 2004. 2.1.2

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao et Pankaj Rohatgi : Towards Sound Approaches to Counteract Power-Analysis Attacks. *Dans* Michael J. Wiener, éditeur : *CRYPTO'99*, volume 1666 de *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. 7.3.3

[CK09] Jean-Sébastien Coron et Ilya Kizhvatov : An Efficient Method for Random Delay Generation in Embedded Software. *Dans* Christophe Clavier et Kris Gaj, éditeurs : *CHES'09*, volume 5747 de *Lecture Notes in Computer Science*, pages 156–170. Springer, 2009. 7.3.1

[CKN01] Jean-Sébastien Coron, Paul C. Kocher et David Naccache : Statistics and Secret Leakage. *Dans* Yair Frankel, éditeur : *Financial Cryptography'00*, volume 1962 de *Lecture Notes in Computer Science*, pages 157–173. Springer, 2001. 7.2.4, 7.3.2

[CL05] Carlos Cid et Gaëtan Leurent : An Analysis of the XSL Algorithm. *Dans* Bimal K. Roy, éditeur : *ASIACRYPT'05*, volume 3788 de *Lecture Notes in Computer Science*, pages 333–352. Springer, 2005. 2.2.1

[Cla07] Christophe Clavier : *De la sécurité physique des crypto-systèmes embarqués*. Thèse de doctorat, Université de Versailles Saint-Quentin, 2007. 7.2.5

[Cop02] Don Coppersmith : Comments on Crypto-Gram Newsletter, October 2002. 2.2.1

[CP02] Nicolas Courtois et Josef Pieprzyk : Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. *Dans* Yuliang Zheng, éditeur : *ASIACRYPT'02*, volume 2501 de *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002. 2.2.1

[CP04] Claude Carlet et Emmanuel Prouff : Vectorial Functions and Covering Sequences. *Dans* Gary L. Mullen, Alain Poli et Henning Stichtenoth, éditeurs : *7th International Conference on Finite Fields and Applications-Fq7*, volume 2948 de *Lecture Notes in Computer Science*, pages 215–248. Springer, 2004. 6.1.2

[CP05] Christophe De Cannière et Bart Preneel : Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher, 2005. 2.1, 2.2.3

[CT03] Jean-Sébastien Coron et Alexei Tchulkine : A New Algorithm for Switching from Arithmetic to Boolean Masking. *Dans CHES'03*, volume 2779 de *Lecture Notes in Computer Science*, pages 89–97. Springer, 2003. 7.3.3, 7.5.3

[CV02] Anne Canteaut et Marion Videau : Degree of Composition of Highly Nonlinear Functions and Applications to Higher Order Differential Cryptanalysis. *Dans* Lars R. Knudsen, éditeur : *EUROCRYPT'02*, volume 2332 de *Lecture Notes in Computer Science*, pages 518–533. Springer, 2002. 3.4.2, 3.2, 5.1, 5.3

[Dam90] Ivan Damgård : A Design Principle for Hash Functions. *Dans* Gilles Brassard, éditeur : *CRYPTO'89*, volume 435 de *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990. 1.3.3, 1.3.3

[Das02] Pinaki Das : The Number of Permutation Polynomials of a Given Degree Over a Finite Field. *Finite Fields and Their Applications*, 8(4):478–490, 2002. 5.1

[DBRP99] Carl D'Halluin, Gert Bijnens, Vincent Rijmen et Bart Preneel : Attack on Six Rounds of Crypton. *Dans* Lars R. Knudsen, éditeur : *FSE'99*, volume 1636 de *Lecture Notes in Computer Science*, pages 46–59. Springer, 1999. 2.2.4

[Deb12] Blandine Debraize : Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking. *Dans* Emmanuel Prouff et Patrick Schaumont, éditeurs : *CHES'12*, volume 7428 de *Lecture Notes in Computer Science*, pages 107–121. Springer, 2012. 7.3.3

[Dil74] John F. Dillon : *Elementary Hadamard Difference sets*. Thèse de doctorat, University of Maryland, 1974. 2.2.2

[DKL+00] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater et Jean-Louis Willems : A Practical Implementation of the Timing Attack. *Dans* Jean-Jacques Quisquater et Bruce Schneier, éditeurs : *CARDIS'98*, volume 1820 de *Lecture Notes in Computer Science*, pages 167–182. Springer, 2000. 7.2.2

[DKR97] Joan Daemen, Lars R. Knudsen et Vincent Rijmen : The Block Cipher Square. *Dans* Eli Biham, éditeur : *FSE'97*, volume 1267 de *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. 2.2.4

[DL11] Ming Duan et Xuejia Lai : Improved zero-sum distinguisher for full round Keccak-f permutation. IACR ePrint Report 2011/023, janvier 2011. http://eprint.iacr.org/2011/023. 4.2, ??, 5.2

[Dob95] Hans Dobbertin : Construction of Bent Functions and Balanced Boolean Functions with High Nonlinearity. *Dans* Bart Preneel, éditeur : *FSE'94*, volume 1008 de *Lecture Notes in Computer Science*, pages 61–74. Springer, 1995. 2.1.2

[DP08] Stefan Dziembowski et Krzysztof Pietrzak : Leakage-Resilient Cryptography. *Dans FOCS'08*, pages 293–302. IEEE Computer Society, 2008. 7.3.5

[DR99] Joan Daemen et Vincent Rijmen : Aes proposal : Rijndael, 1999. 2.2.4

[DR00a] Joan Daemen et Vincent Rijmen : Rijndael for AES. *Dans AES Candidate Conference*, pages 343–348, 2000. 1.4.2

[DR00b] Joan Daemen et Vincent Rijmen : The Block Cipher Rijndael. *Dans* Jean-Jacques Quisquater et Bruce Schneier, éditeurs : *CARDIS'98*, volume 1820 de *Lecture Notes in Computer Science*, pages 277–284. Springer, 2000. 1.2, 1.4.2

[DR06] Joan Daemen et Vincent Rijmen : Understanding Two-Round Differentials in AES. *Dans* Roberto De Prisco et Moti Yung, éditeurs : *SCN'06*, volume 4116 de *Lecture Notes in Computer Science*, pages 78–94. Springer, 2006. 4.4

[DS09] Itai Dinur et Adi Shamir : Cube Attacks on Tweakable Black Box Polynomials. *Dans* Antoine Joux, éditeur : *EUROCRYPT'09*, volume 5479 de *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. 2.2.3, 2.2.3, 2.2, 2.2.3

[DS11] Itai Dinur et Adi Shamir : An Improved Algebraic Attack on Hamsi-256. *Dans* Antoine Joux, éditeur : *FSE'11*, volume 6733 de *Lecture Notes in Computer Science*, pages 88–106. Springer, 2011. 3.7, 6.2, 6.2.1

[DSW08] Chistophe De Cannière, Hisayoshi Sato et Dai Watanabe : Hash Function Luffa : Specification. Submission to NIST (Round 1), 2008. 4.3

[DSW09] Christophe De Cannière, Hisayoshi Sato et Dai Watanabe : Hash Function Luffa : Specification. Submission to NIST (Round 2), 2009. 4.3

[DSW10] Christophe De Cannière, Hisayoshi Sato et Dai Watanabe : The reasons for the change of Luffa. Supplied with the second round package, 2010. 4.3.3

[DZ84] Stefan M. Dodunekov et Victor Zinoviev : A note on Preparata codes. *Dans Proceedings of the 6th Intern. Symp. on Information Theory, Moscow-Tashkent Part 2*, pages 78–80, 1984. 3.2.1

[ECR] ECRYPT II- European Network of Excellence in Cryptology II : The SHA-3 Zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo. 1.3.7

[Fau99] Jean-Charles Faugère : A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, June 1999. 2.2.1

[Fau02] Jean Charles Faugère : A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). *Dans Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM. 2.2.1

[FIP80] FIPS 81 : DES Modes of Operation. Federal Information Processing Standards Publication 81, 1980. U.S. Department of Commerce/National Bureau of Standards. 5.4

[FIP99] FIPS 46-3 : Data Encryption Standard. National Institute for Standards and Technology, Gaithersburg, MD, USA, Octobre 1999. 1.2

[FIP01] FIPS 197 : Announcing the Advanced Encryption Standard (AES). National Institute for Standards and Technology, Gaithersburg, MD, USA, November 2001. 1.2, 1.4.2

[FIP02] FIPS 198 : The Keyed-Hash Message Authentication Code (HMAC) . National Institute for Standards and Technology, Gaithersburg, MD, USA, Mars 2002. 1.3.6

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner et Doug Whiting : Improved Cryptanalysis of Rijndael. *Dans* Bruce Schneier, éditeur : *FSE'00*, volume 1978 de *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000. 2.2.4

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas et Jesse Walker : The Skein Hash Function Family. Submission to NIST (Round 3), 2010. 7.5, 7.5.3, 7.5.3

[FS09] Matthieu Finiasz et Nicolas Sendrier : Security Bounds for the Design of Code-Based Cryptosystems. *Dans* Mitsuru Matsui, éditeur : *ASIACRYPT'09*, volume 5912 de *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. 3.2.2

[Fuh10] Thomas Fuhr : Finding Second Preimages of Short Messages for Hamsi-256. *Dans* Masayuki Abe, éditeur : *ASIACRYPT'10*, volume 6477 de *Lecture Notes in Computer Science*, pages 20–37. Springer, 2010. (document), 3.7, 6, 6.2, 6.2.1, 6.2.1, 6.2.1, 6.2.1, 6.2.2, 6.2.3, 6.2.3

[Fuh11] Thomas Fuhr : *Conception, preuves et analyse de fonctions de hachage cryptographiques*. Thèse de doctorat, Télécom ParisTech, 2011. 6.2, 6.2.3, 6.2.3

[GBGS12] Vincent Grosso, Christina Boura, Benoît Gérard et François-Xavier Standaert : A Note on the Empirical Evaluation of Security Margins against Algebraic Attacks (with Application to Low Cost Ciphers LED and Piccolo). *Dans Proceedings of the 33rd WIC Symposium on Information Theory in the Benelux, Boekelo, The Netherlands*, pages 52,59, May 2012. (document)

[GKM+08] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer et Søren S. Thomsen : Grøstl-a SHA-3 candidate. Submission to NIST (Round 1), 2008. 4.5, 7.5

[GM00] Henri Gilbert et Marine Minier : A Collision Attack on 7 Rounds of Rijndael. *Dans AES Candidate Conference*, pages 230–241, 2000. 2.2.4

[GM11] Jorge Guajardo et Bart Mennink : On Side-Channel Resistant Block Cipher Usage. *Dans* Gene Tsudik, Spyros S. Magliveras et Ivana Ilic, éditeurs : *ISC'10*, volume 6531 de *Lecture Notes in Computer Science*, pages 254–268. Springer, 2011. 7.3.4

[GMO01] Karine Gandolfi, Christophe Mourtel et Francis Olivier : Electromagnetic Analysis : Concrete Results. *Dans* Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs : *CHES'01*, volume 2162 de *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001. 7.2.1

[GO08] Praveen Gauravaram et Katsuyuki Okeya : Side Channel Analysis of Some Hash Based MACs : A Response to SHA-3 Requirements. *Dans* Liqun Chen, Mark Dermot Ryan et Guilin Wang, éditeurs : *ICICS'08*, volume 5308 de *Lecture Notes in Computer Science*, pages 111–127. Springer, 2008. 7.4

[Gou01] Louis Goubin : A Sound Method for Switching between Boolean and Arithmetic Masking. *Dans* Çetin Kaya Koç, David Naccache et Christof Paar, éditeurs : *CHES'01*, volume 2162 de *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001. 7.3.3, 7.5.3

[GP99] Louis Goubin et Jacques Patarin : DES and Differential Power Analysis - The Duplication Method. *Dans* Çetin Kaya Koç et Christof Paar, éditeurs : *CHES'99*, volume 1717 de *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999. 7.3.3

[GS94] Marc Girault et Jacques Stern : On the Length of Cryptographic Hash-Values Used in Identification Schemes. *Dans* Yvo Desmedt, éditeur : *CRYPTO'94*, volume 839 de *Lecture Notes in Computer Science*, pages 202–215. Springer, 1994. 1.3.3

[HH99] Helena Handschuh et Howard M. Heys : A Timing Attack on RC5. *Dans* Stafford E. Tavares et Henk Meijer, éditeurs : *Selected Areas in Cryptography-SAC'98*, volume 1556 de *Lecture Notes in Computer Science*, pages 306–318. Springer, 1999. 7.2.2

[HJM07] Martin Hell, Thomas Johansson et Willi Meier : Grain : a stream cipher for constrained environments. *Int. J. Wire. Mob. Comput.*, 2(1):86–93, mai 2007. 2.2.3

[HLL+02] Kyungdeok Hwang, Wonil Lee, Sungjae Lee, Sangjin Lee et Jongin Lim : Saturation attacks on reduced round skipjack. *Dans* Joan Daemen et Vincent Rijmen, éditeurs : *FSE'02*, volume 2365 de *Lecture Notes in Computer Science*, pages 100–111. Springer, 2002. 2.2.4

[JK97] Thomas Jakobsen et Lars R. Knudsen : The Interpolation Attack on Block Ciphers. *Dans* Eli Biham, éditeur : *FSE'97*, volume 1267 de *Lecture Notes in Computer Science*, pages 28–40. Springer, 1997. 2.2.2, 2.2.2, 5.5.1, 5.5.1

[Jou04] Antoine Joux : Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. *Dans* Matthew K. Franklin, éditeur : *CRYPTO'04*, volume 3152 de *Lecture Notes in Computer Science*, pages 306–316, 2004. 1.3.3

[Kal00] Burton S. Kaliski : PKCS #5 : Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational) `http://www.ietf.org/rfc/rfc2898.txt`, September 2000. 1.3.2

[Kat71] Nicholas M. Katz : On a theorem of Ax. *American Journal of Mathematics*, 93:485–499, 1971. 5.1, 5.3

[KBC97] H. Krawczyk, M. Bellare et R. Canetti : HMAC : Keyed-Hashing for Message Authentication. RFC 2104 (Informational), février 1997. Updated by RFC 6151. 1.3.6

[Kel08] John Kelsey : How to Choose SHA-3. `http://www.lorentzcenter.nl/lc/web/2008/309/presentations/Kelsey.pdf`, 2008. 7.4

[KJJ98] Paul C. Kocher, Joshua Jaffe et Benjamin Jun : Introduction to Differential Power Analysis and Related Attacks. Rapport technique, Cryptography Research Inc., 1998. 7.2.1, 7.2.4

[KJJ99] Paul C. Kocher, Joshua Jaffe et Benjamin Jun : Differential Power Analysis. *Dans* Michael J. Wiener, éditeur : *CRYPTO'99*, volume 1666 de *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. 7.2.1, 7.2.4

[KK99] Oliver Kömmerling et Markus G. Kuhn : Design Principles for Tamper-Resistant Smartcard Processors. *Dans Proceedings of the USENIX Workshop on*

Smartcard Technology, WOST'99, pages 9–20, Berkeley, CA, USA, 1999. USENIX Association. 7.1, 7.3.2

[Knu95] Lars R. KNUDSEN : Truncated and Higher Order Differentials. *Dans* Bart PRENEEL, éditeur : *FSE'94*, volume 1008 de *Lecture Notes in Computer Science*, pages 196–211. Springer, 1995. 2.2.2

[Koc96] Paul C. KOCHER : Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *Dans* Neal KOBLITZ, éditeur : *CRYPTO'96*, volume 1109 de *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 7.2.1, 7.2.2

[KP02] Sergei KONYAGIN et Francesco PAPPALARDI : Enumerating Permutation Polynomials over Finite Fields by Degree. *Finite Fields and Their Applications*, 8(4):548–553, 2002. 5.1

[KR07] Lars R. KNUDSEN et Vincent RIJMEN : Known-Key Distinguishers for Some Block Ciphers. *Dans* Kaoru KUROSAWA, éditeur : *ASIACRYPT'07*, volume 4833 de *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007. 3, 3.1, 3.4.1

[KS05] John KELSEY et Bruce SCHNEIER : Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. *Dans* Ronald CRAMER, éditeur : *EUROCRYPT'05*, volume 3494 de *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005. 1.3.4

[Kuc09] Özgül KUCÜK : The Hash Function Hamsi. Submission to NIST (Round 2), 2009. 3.7.1, 6.2

[KW02] Lars R. KNUDSEN et David WAGNER : Integral Cryptanalysis. *Dans* Joan DAEMEN et Vincent RIJMEN, éditeurs : *FSE'02*, volume 2365 de *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002. 2.2.4, 2.2.4

[Lai94] Xuejia LAI : Higher order derivatives and differential cryptanalysis. *Dans Proc. "Symposium on Communication, Coding and Cryptography", in honor of J. L. Massey on the occasion of his 60'th birthday*. Kluwer Academic Publishers, 1994. 2.2.2, 2.6

[Lat05] Joel LATHROP : Cube Attacks on Cryptographic Hash Functions. Mémoire de Master, Rochester Institute of Technology, 2005. 2.2.3

[LM93] Xuejia LAI et James L. MASSEY : Hash Function Based on Block Ciphers. *Dans* Rainer A. RUEPPEL, éditeur : *EUROCRYPT'92*, volume 658 de *Lecture Notes in Computer Science*, pages 55–70. Springer, 1993. 6.2.1

[LN83] Rudolf LIDL et Harald NIEDERREITER : *Finite fields*, volume 20 de *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1983. 2.1, 2.1

[LP07] Gregor LEANDER et Axel POSCHMANN : On the Classification of 4 Bit S-Boxes. *Dans* Claude CARLET et Berk SUNAR, éditeurs : *WAIFI'07*, volume 4547 de *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007. 6.1.3, 6.3

[LRW02] Moses Liskov, Ronald L. Rivest et David Wagner : Tweakable Block Ciphers. *Dans* Moti Yung, éditeur : *CRYPTO'02*, volume 2442 de *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002. 7.5.3

[Luc02] Stefan Lucks : The Saturation Attack - A Bait for Twofish. *Dans* Mitsuru Matsui, éditeur : *FSE'01*, volume 2355 de *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. 2.2.4

[Luc05] Stefan Lucks : A Failure-Friendly Design Principle for Hash Functions. *Dans* Bimal K. Roy, éditeur : *ASIACRYPT'05*, volume 3788 de *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005. 1.3.3

[McF73] Robert L. McFarland : A Family of Difference Sets in Non-cyclic Groups. *Journal of Combinatorial Theory*, 15(1):1–10, 1973. 6.1.2

[Mer90] Ralph C. Merkle : One Way Hash Functions and DES. *Dans* Gilles Brassard, éditeur : *CRYPTO'89*, volume 435 de *Lecture Notes in Computer Science*, pages 428–446. Springer, 1990. 1.3.3, 1.3.3

[Mes00] Thomas S. Messerges : *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. Thèse de doctorat, University of Illinois, 2000. 7.3.2

[Mes01] Thomas S. Messerges : Securing the AES Finalists Against Power Analysis Attacks. *Dans* Bruce Schneier, éditeur : *FSE'00*, volume 1978 de *Lecture Notes in Computer Science*, pages 150–164. Springer, 2001. 7.3.3, 7.3.3, 7.5.2

[MMO85] Stephen M. Matyas, Carl H. Meyer et J. Oseas : Generating strong one-way functions with cryptographic algorithms . *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985. 1.3.4

[MMT11] Alexander May, Alexander Meurer et Enrico Thomae : Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$. *Dans* Dong Hoon Lee et Xiaoyun Wang, éditeurs : *ASIACRYPT'11*, volume 7073 de *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. 3.2.2

[MOI90] Shoji Miyaguchi, Kazuo Ohta et Masahiko Iwata : 128-bit hash function (N-hash). *NTT Review*, 2:128–132, 1990. 1.3.4

[Mor11] Pawel Morawiecki : Preimages and Collisions for Keccak[$r = \{240, 640, 1440\}, c = 160, nr = 1, 2$]. Keccak website `http://keccak.noekeon.org/crunchy_contest.html`, 2011. 2.2.1

[MS78] Florence J. MacWilliams et Neil J.A. Sloane : *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 1978. 3.2.1, 3.1

[MS10] Pawel Morawiecki et Marian Srebrny : A SAT-based preimage analysis of reduced KECCAK hash functions. *Dans Second SHA-3 Candidate Conference, Santa Barbara*, 2010. 2.2.1

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Grossschädl et Francesco Regazzoni : Fresh Re-keying : Security against Side-Channel and

Fault Attacks for Low-Cost Devices. *Dans* Daniel J. Bernstein et Tanja Lange, éditeurs : *AFRICACRYPT'10*, volume 6055 de *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010. 7.3.4

[MvOV97] A.J. Menezes, P.C. van Oorshot et S.A. Vanstone : *Handbook of Applied Cryptography*. CRC Press, 1997. Disponible sur http://www.cacr.math.uwaterloo.ca/hac/. 1.3.1

[Nat] National Institute of Standards and Technology : Cryptographic Hash Algorithm Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/. 1.3.7

[NK95] Kaisa Nyberg et Lars R. Knudsen : Provable Security Against a Differential Attack. *J. Cryptology*, 8(1):27–37, 1995. 2.2.2, 2.2.2, 5.5.1

[NSA72] NSA : TEMPEST : A Signal Problem. http://www.nsa.gov/public_info/_files/cryptologic_spectrum/tempest.pdf, 1972. Declassified in 2007. 7.2.1

[Nyb91] Kaisa Nyberg : Perfect Nonlinear S-Boxes. *Dans* Donald W. Davies, éditeur : *EUROCRYPT'91*, volume 547 de *Lecture Notes in Computer Science*, pages 378–386. Springer, 1991. 6.1.2

[Nyb94] Kaisa Nyberg : Differentially Uniform Mappings for Cryptography. *Dans* Tor Helleseth, éditeur : *EUROCRYPT'93*, volume 765 de *Lecture Notes in Computer Science*, pages 55–64. Springer, 1994. 2.2.2, 2.1, 5.5.1

[Nyb95] Kaisa Nyberg : S-boxes and Round Functions with Controllable Linearity and Differential Uniformity. *Dans* Bart Preneel, éditeur : *FSE'94*, volume 1008 de *Lecture Notes in Computer Science*, pages 111–130, 1995. 5.5.1

[Nyb12] Kaisa Nyberg : "Provable" Security Against Differential and Linear Cryptanalysis. *Dans* Anne Canteaut, éditeur : *Fast Software Encryption - FSE 2012*, volume 7549 de *Lecture Notes in Computer Science*, pages 1–8. Springer, 2012. 2.2.2

[Oke06] Katsuyuki Okeya : Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions. *Dans* Lynn Margaret Batten et Reihaneh Safavi-Naini, éditeurs : *ACISP'06*, volume 4058 de *Lecture Notes in Computer Science*, pages 432–443. Springer, 2006. 7.4

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller et Vincent Rijmen : A Side-Channel Analysis Resistant Description of the AES S-Box. *Dans* Henri Gilbert et Helena Handschuh, éditeurs : *FSE'05*, volume 3557 de *Lecture Notes in Computer Science*. Springer, 2005. 7.3.3

[OS06] Elisabeth Oswald et Kai Schramm : An Efficient Masking Scheme for AES Software Implementations. *Dans* JooSeok Song, Taekyoung Kwon et Moti Yung, éditeurs : *WISA'05*, volume 3786 de *Lecture Notes in Computer Science*. Springer, 2006. 7.3.3

[Pat04] Jacques Patarin : Security of Random Feistel Schemes with 5 or More Rounds. *Dans* Matthew K. Franklin, éditeur : *CRYPTO'04*, volume 3152 de *Lecture Notes in Computer Science*, pages 106–122, 2004. 2.2.2

[PGA06] Emmanuel Prouff, Christophe Giraud et Sébastien Aumônier : Provably Secure S-Box Implementation Based on Fourier Transform. *Dans* Louis Goubin et Mitsuru Matsui, éditeurs : *CHES'06*, volume 4249 de *Lecture Notes in Computer Science*, pages 216–230. Springer, 2006. 7.3.3

[PGV94] Bart Preneel, René Govaerts et Joos Vandewalle : Hash Functions Based on Block Ciphers : A Synthetic Approach. *Dans* Douglas R. Stinson, éditeur : *CRYPTO'03*, volume 773 de *Lecture Notes in Computer Science*, pages 368–378. Springer, 1994. 1.3.4

[Pol75] John Pollard : Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975. 1.3.1

[PR08] Emmanuel Prouff et Matthieu Rivain : A Generic Method for Secure SBox Implementation. *Dans* Sehun Kim, Moti Yung et Hyung-Woo Lee, éditeurs : *WISA'07*, volume 4867 de *Lecture Notes in Computer Science*, pages 227–244. Springer, 2008. 7.3.3

[Pre93] Bart Preneel : *Analysis and design of cryptographic hash functions*. Thèse de doctorat, Katholieke Universiteit Leuven (Belgium), Janvier 1993. 1.3.4

[PSQ07] Eric Peeters, François-Xavier Standaert et Jean-Jacques Quisquater : Power and electromagnetic analysis : Improved model, consequences and comparisons. *Integration*, 40(1):52–60, 2007. 7.2.4

[PvO96] Bart Preneel et Paul C. van Oorschot : On the Security of Two MAC Algorithms. *Dans* Ueli M. Maurer, éditeur : *EUROCRYPT'96*, volume 1070 de *Lecture Notes in Computer Science*, pages 19–32. Springer, 1996. 1.3.6

[QD90] Jean-Jacques Quisquater et Jean-Paul Delescaille : How Easy is Collision Search. New Results and Applications to DES. *Dans* Gilles Brassard, éditeur : *CRYPTO'89*, volume 435 de *Lecture Notes in Computer Science*, pages 408–413. Springer, 1990. 1.3.1

[QS01] Jean-Jacques Quisquater et David Samyde : ElectroMagnetic Analysis (EMA) : Measures and Counter-Measures for Smart Cards. *Dans* Isabelle Attali et Thomas P. Jensen, éditeurs : *E-smart'01*, volume 2140 de *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001. 7.2.1

[Rab78] Michael O. Rabin : Digitalized signatures. *Foundations of Secure Computation*, pages 155–168, 1978. 1.3.3

[Riv08] Ronald L. Rivest : The MD6 hash function – A proposal to NIST for SHA-3. Submission to NIST, 2008. 2.2.3

[Riv09] Matthieu Rivain : *On the Physical Security of Cryptographic Implementations*. Thèse de doctorat, Université du Luxembourg, 2009. 7.3.3

[RS04] Phillip Rogaway et Thomas Shrimpton : Cryptographic Hash-Function Basics : Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *Dans* Bimal K. Roy et Willi Meier, éditeurs : *FSE'04*, volume 3017 de *Lecture Notes in Computer Science*. Springer, 2004. 1.3.1

[SA03] Sergei P. Skorobogatov et Ross J. Anderson : Optical Fault Induction Attacks. *Dans* Burton S. Kaliski Jr., Çetin Kaya Koç et Christof Paar, éditeurs : *CHES'02*, volume 2523 de *Lecture Notes in Computer Science*, pages 2–12. Springer, 2003. 7.1

[Sha49] Claude Shannon : Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949. 1.4.1, 2.2.1

[SK96] Bruce Schneier et John Kelsey : Unbalanced Feistel Networks and Block Cipher Design. *Dans* Dieter Gollmann, éditeur : *FSE'96*, volume 1039 de *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996. 1.4.3

[SKPI07] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret et Hideki Imai : Algebraic Cryptanalysis of 58-Round SHA-1. *Dans* Alex Biryukov, éditeur : *FSE'07*, volume 4593 de *Lecture Notes in Computer Science*, pages 349–365. Springer, 2007. 2.2.1

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall et Niels Ferguson : Twofish : A 128-Bit Block Cipher. *Dans First Advanced Encryption Standard (AES) Conference*, 1998. 2.2.4

[SPY+09] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung et Elisabeth Oswald : Leakage Resilient Cryptography in Practice. *IACR Cryptology ePrint Archive*, 2009. 7.3.5

[TB07] Michael Tunstall et Olivier Benoît : Efficient Use of Random Delays in Embedded Software. *Dans* Damien Sauveron, Constantinos Markantonakis, Angelos Bilas et Jean-Jacques Quisquater, éditeurs : *WISTP'07*, volume 4462 de *Lecture Notes in Computer Science*, pages 27–38. Springer, 2007. 7.3.1

[TSMS99] Ezzy A. Dabish Thomas S. Messerges et Robert H. Sloan : Investigations of power analysis attacks on smartcards. *Dans Proceedings of the USENIX Workshop on Smartcard Technology*, WOST'99, pages 151–162, Berkeley, CA, USA, 1999. USENIX Association. 7.2.4

[Tsu92] Gene Tsudik : Message Authentication with One-Way Hash Functions. *Dans INFOCOM'92*, pages 2055–2059, 1992. 1.3.6

[Van85] Wim Van Eck : Electromagnetic radiation from video display units : an eavesdropping risk ? *Computers & Security*, 4(4):269–286, décembre 1985. 7.2.1

[vOW99] Paul C. van Oorschot et Michael J. Wiener : Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999. 1.3.1

[Wag02] David Wagner : A Generalized Birthday Problem. *Dans* Moti Yung, éditeur : *CRYPTO'02*, volume 2442 de *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002. 1.3.1, 3.2.2

[Wel69] Charles Wells : The degrees of permutation polynomials over finite fields. *Journal of Combinatorial Theory*, 7(1):49–55, 1969. 5.1

[WHYK10] Dai Watanabe, Yasuo Hatano, Tsuyoshi Yamada et Toshinobu Kaneko : Higher Order Differential Attack on Step-Reduced Variants of *Luffa* v1. *Dans* Seokhie Hong et Tetsu Iwata, éditeurs : *FSE'10*, volume 6147 de *Lecture Notes in Computer Science*, pages 270–285. Springer, 2010. 4, 4.3, 4.3.1, 4.3.1, 4.3.2, 4.3.2, 4.3, 4.3.3, 4.3.4, 4.3.5, 4.3.7, 7.5.4, 7.5.4

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen et Xiuyuan Yu : Cryptanalysis of the Hash Functions MD4 and RIPEMD. *Dans* Ronald Cramer, éditeur : *EUROCRYPT'05*, volume 3494 de *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005. 1.3.7

[Wu11] Hongjun Wu : The hash function JH. Submission to NIST (Round 3) available at http://www3.ntu.edu.sg/home/wuhj/research/jh/, 2011. 5.5.4

[WY05] Xiaoyun Wang et Hongbo Yu : How to Break MD5 and Other Hash Functions. *Dans* Ronald Cramer, éditeur : *EUROCRYPT'05*, volume 3494 de *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. 1.3.7, 2.2.1

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin et Hongbo Yu : Finding Collisions in the Full SHA-1. *Dans* Victor Shoup, éditeur : *CRYPTO'05*, volume 3621 de *Lecture Notes in Computer Science*, pages 17–36, 2005. 1.3.7

[WYY05b] Xiaoyun Wang, Hongbo Yu et Yiqun Lisa Yin : Efficient Collision Search Attacks on SHA-0. *Dans* Victor Shoup, éditeur : *CRYPTO'05*, volume 3621 de *Lecture Notes in Computer Science*, pages 1–16, 2005. 1.3.7

[Yas07] Kan Yasuda : "Sandwich" Is Indeed Secure : How to Authenticate a Message with Just One Hashing. *Dans* Josef Pieprzyk, Hossein Ghodosi et Ed Dawson, éditeurs : *ACISP'07*, volume 4586 de *Lecture Notes in Computer Science*, pages 355–369, 2007. 1.3.6

[YPK02] Yongjin Yeom, Sangwoo Park et Iljun Kim : On the Security of CAMELLIA against the Square Attack. *Dans* Joan Daemen et Vincent Rijmen, éditeurs : *FSE'02*, volume 2365 de *Lecture Notes in Computer Science*, pages 89–99. Springer, 2002. 2.2.4

[Yuv97] G. Yuval : How to Swindle Rabin. *Cryptologia*, 3(3):187–189, 1997. 1.3.1

[ZKS11] Michael Zohner, Michael Kasper et Marc Stöttinger : Side Channel Evaluation of SHA-3 Candidates. First International Workshop on Trustworthy Embedded Devices TRUSTED, 2011. 7.4, 7.5.3

[ZMI90] Yuliang Zheng, Tsutomu Matsumoto et Hideki Imai : On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses.

Dans Gilles BRASSARD, éditeur : *CRYPTO'89*, volume 435 de *Lecture Notes in Computer Science*, pages 461–480. Springer, 1990. 1.4.3

Table des figures

1.1 Construction de Merkle-Damgård . . . 11
1.2 Une 8-multi-collision . . . 13
1.3 Construction *wide pipe* . . . 14
1.4 La construction éponge . . . 15
1.5 Les schémas les plus connus pour la construction d'une fonction de compression à partir d'un chiffrement par blocs . . . 17
1.6 La construction HMAC pour une fonction de hachage H construite selon l'algorithme de Merkle-Damgård avec une fonction de compression h. . . . 19
1.7 La construction *envelope* MAC pour une fonction de hachage H construite selon l'algorithme de Merkle-Damgård avec une fonction de compression h. . . 19
1.8 Le chiffrement itératif par blocs . . . 22
1.9 Fonction de tour d'un réseau substitution-permutation . . . 23
1.10 Les états de Rijndael-256 et Rijndael-128 . . . 24
1.11 La transformation `SubBytes` pour Rijndael-128 . . . 25
1.12 La transformation `ShiftRows` pour Rijndael-128 . . . 26
1.13 La transformation `MixColumns` pour Rijndael-128 . . . 26
1.14 Un tour d'un chiffrement d'un réseau de type Feistel . . . 27

2.1 Une différentielle pour une fonction F . . . 39
2.2 Le tour i du chiffrement par blocs $\mathcal{KN}$ . . . 42
2.3 L'attaque différentielle d'ordre supérieur sur le chiffrement $\mathcal{KN}$ . . . 44
2.4 Distingueur pour 3 tours de l'AES, avec $\mathcal{S} = 0$ . . . 49

3.1 Une integrale pour 7 tours de l'AES utilisant 2^{56} textes clairs. . . . 52
3.2 Méthode pour construire une partition en sommes nulles pour une permutation P, composée de r tours . . . 58
3.3 Méthode pour étendre une partition en sommes nulles à un tour supplémentaire 62
3.4 Méthode pour étendre une partition en sommes nulles à deux tours supplémentaires 64
3.5 Concaténation du message et de la valeur de chaînage pour Hamsi-256 . . . 73

4.1 La permutation P de $\mathbf{F}_2^{16}$ de type SPN, constituée de 2 tours . . . 81
4.2 La permutation P' . . . 82
4.3 Évolution du degré de $G \circ F$, où F est une fonction avec 1600 variables, composée de 320 permutations cubiques sur $\mathbf{F}_2^5$. . . . 83
4.4 La fonction de hachage *Luffa* . . . 86
4.5 La fonction `Step` . . . 87
4.6 La fonction de compression de Grøstl . . . 97

5.1 Le tour i du chiffrement $\mathcal{KN}'$ . . . 112
5.2 Deux tours de R_4. . . . 117

6.1 L'état interne de la fonction de compression de Hamsi-256 . . . 129

7.1 Exécution de HMAC-Grøstl sur une carte à puce. . . . 145
7.2 Les zones d'attaque contre une construction HMAC lors d'une analyse par canaux cachés. . . . 152
7.3 Les zones d'attaque contre une construction *envelope* MAC lors d'une analyse par canaux cachés. . . . 153
7.4 Un appel à la fonction de hachage Grøstl-256 avec un bloc de message lors de son utilisation dans HMAC. Les trois opérations sensibles identifiées sont entourées. . . . 154
7.5 Analyse simple de courant (SPA) des zones d'attaque dans HMAC-Grøstl : l'absorption du message et l'opération `SubBytes` . . . 155
7.6 Analyse CPA sur l'opération `SubBytes` de HMAC-Grøstl (4 premiers octets de la valeur cible) . . . 155
7.7 Comparaison des implémentations simple et sécurisée de Grøstl . . . 157
7.8 Analyse CPA sur l'opération `SubBytes` de HMAC-Grøstl sécurisé (4 premiers octets de la valeur cible) . . . 157
7.9 L'opération MIX . . . 158
7.10 Quatre tours de Threefish . . . 159
7.11 La construction Skein-MAC . . . 159
7.12 Analyse simple de courant (SPA) de la zone d'attaque dans HMAC-Skein . . . 160
7.13 CPA sur l'addition modulaire . . . 160
7.14 Comparaison de l'implémentation simple et sécurisée de Skein (sans et avec le tweak) . . . 162
7.15 Consommation de courant pour l'implémentation de référence et l'implémentation sécurisée pour HMAC-Skein et HMAC Grøstl . . . 163

Liste des Algorithmes

1 Recherche des k-multi-collisions 13

2 Méthode générique pour la recherche des sommes nulles 55
3 Méthode générique pour la recherche des partitions en sommes nulles 57

5 Algorithme `Relations Affines` qui détermine pour un ensemble donné de variables d'entrée $\mathcal{I}$ le nombre de bits en sortie de la fonction de compression de Hamsi-256 qui dépendent de façon affine des variables dans $\mathcal{I}$. 136
4 Méthode pour la recherche des relations affines pour Hamsi-256 138

6 La méthode "recalcul de table"pour masquer une boîte-S 149

Liste des tableaux

1.1 Complexité des meilleures attaques génériques. 7
1.2 Les 14 candidats du deuxième tour de la compétition SHA-3 21
1.3 Nombre de tours r pour Rijndael en fonction de la taille de l'état N_b et de la taille de la clé N_k. 24
1.4 Longueur de décalage pour chaque ligne en fonction de la taille de l'état interne. 25

2.1 La table de vérité d'une fonction booléenne à 3 variables. 30

3.1 Valeurs de translations pour la transformation ρ de Keccak 68

4.1 Bornes supérieures sur le degré de plusieurs tours de Keccak-f et de son inverse 85
4.2 Bornes supérieures pour le degré algébrique de r itérations de la permutation `Step` de *Luffa* v1, ainsi que pour le degré de toute somme deux de trois premières coordonnées à l'entrée de chaque boîte-S, après r tours. 89
4.3 Bornes supérieures sur le degré algébrique après r itérations de la fonction `Step` pour *Luffa* v1, obtenues en combinant les résultats de [WHYK10] (Table 4.2) et le théorème 4.1. 90
4.4 Bornes supérieures pour le degré algébrique de r itérations de la permutation `Step` de *Luffa* v2. 92
4.5 Bornes supérieures sur le degré algébrique après r itérations de la fonction `Step` pour *Luffa* v2, en combinant les résultats de la table 4.4 et le théorème 4.1. . 93
4.6 Bornes sur le degré algébrique de toutes les versions de *Luffa* v2 avec les probabilités respectives qu'une fonction avec les mêmes paramètres choisie aléatoirement aie le même degré. 95
4.7 Bornes supérieures pour le degré algébrique de plusieurs tours de l'AES . . . 97
4.8 Bornes supérieures sur le degré algébrique de plusieurs tours de P et Q . . . 98

5.1 Bornes supérieures pour r itérations de la permutation de tour de Rijndael-256 obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre. 114
5.2 Bornes supérieures pour r itérations de la fonction de tour de la permutation P de ECHO, obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre. 116
5.3 Bornes supérieures sur le degré jusqu'à 8 tours de la permutation R_8. 118
5.4 Bornes supérieures pour r itérations de la fonction de tour de la permutation R_8 de JH, obtenues respectivement avec la borne triviale, l'approche superBoîte-S et les résultats de ce chapitre. 118

6.1 Nombre d'apparitions de chaque valeur dans le spectre de Walsh de chacune des cinq classes d'équivalence pour les fonctions booléennes à 4 variables de degré au plus 3. 124
6.2 Description de toutes les classes de permutations de $\mathbf{F}_2^4$. 127
6.3 Liste de tous les $\lambda \in \mathbf{F}_2^4$ tels que $x \mapsto \lambda \cdot S(x)$ est de degré au plus 1 sur tous les translatés de $V^\perp$, pour chaque sous-espace V de dimension 2. 133
6.4 Énumération des bits de l'état de Hamsi-256 134

7.1 Surcoût du code sécurisé en termes de temps, de consommation de RAM et de la taille du code quand des messages d'un seul bloc sont traités. 163

Table des matières

Remerciements i

Overview v

Résumé ix

Introduction générale xi

I Analyse de primitives symétriques 1

1 Introduction 3
- 1.1 Introduction à la cryptologie . . . 3
- 1.2 La cryptographie symétrique . . . 4
- 1.3 Les fonctions de hachage . . . 5
  - 1.3.1 Fonctions de hachage cryptographiques . . . 6
  - 1.3.2 Utilisations . . . 8
  - 1.3.3 Les modes opératoires . . . 11
  - 1.3.4 Fonctions de compression fondées sur un algorithme de chiffrement par blocs . . . 15
  - 1.3.5 Conception des primitives internes . . . 17
  - 1.3.6 Code d'authentification ou MAC . . . 18
  - 1.3.7 Le concours SHA-3 . . . 20
- 1.4 Les chiffrements par blocs . . . 21
  - 1.4.1 Les chiffrements par blocs itératifs . . . 22
  - 1.4.2 Réseaux de substitution-permutation . . . 23
  - 1.4.3 Réseaux de Feistel . . . 27

2 Propriétés algébriques 29
- 2.1 Fonctions booléennes . . . 29
  - 2.1.1 Spectre de Walsh et non-linéarité d'une fonction booléenne . . . 32
  - 2.1.2 Restriction d'une fonction booléenne et fonctions booléennes normales 35
- 2.2 Quelques attaques exploitant un degré algébrique faible . . . 36
  - 2.2.1 Attaques algébriques . . . 37
  - 2.2.2 Attaques différentielles d'ordre supérieur . . . 38
  - 2.2.3 Attaque cube . . . 45
  - 2.2.4 Attaque intégrale . . . 47

3 Distingueurs à somme nulle **51**
3.1 Une application sur l'AES . . . 51
3.2 Structures à somme nulle et leurs propriétés . . . 53
3.2.1 Taille minimale d'une somme nulle . . . 53
3.2.2 Algorithmes génériques . . . 54
3.3 Les partitions en sommes nulles . . . 56
3.3.1 Algorithme générique pour trouver des partitions en sommes nulles . . . 56
3.4 Exploiter le degré algébrique de la partie non-linéaire . . . 57
3.4.1 Des partitions en sommes nulles à partir des différentielles d'ordre supérieur 58
3.4.2 Trouver une borne supérieure pour le degré . . . 59
3.5 Exploiter la structure de la partie linéaire . . . 61
3.5.1 Propriété pour un tour . . . 61
3.5.2 Propriété pour plusieurs tours . . . 62
3.6 Application à la permutation interne de KECCAK . . . 67
3.6.1 Description de la permutation KECCAK-f . . . 68
3.6.2 Les partitions en sommes nulles dans [AM09] . . . 69
3.6.3 Partitions en sommes nulles pour 18 tours de KECCAK-f . . . 70
3.6.4 Partitions en sommes nulles pour 19 tours de KECCAK-f . . . 71
3.6.5 Partitions en sommes nulles pour 20 tours de KECCAK-f . . . 71
3.7 Application à la permutation de finalisation de Hamsi-256 . . . 72
3.7.1 Description de Hamsi-256 . . . 72

4 Borne sur le degré des permutations itérées **77**
4.1 Une nouvelle borne pour les constructions de type SPN . . . 78
4.1.1 Comprendre la nouvelle borne . . . 81
4.1.2 Lien entre la nouvelle borne et la borne triviale . . . 82
4.2 Partitions en sommes nulles pour KECCAK-f . . . 83
4.3 Application à la fonction de hachage *Luffa* . . . 85
4.3.1 Spécifications de la fonction *Luffa* v1 . . . 86
4.3.2 Description de l'attaque [WHYK10] . . . 87
4.3.3 Différentielles d'ordre supérieur pour la fonction de hachage *Luffa* v1 entière . . . 90
4.3.4 Modifications sur la fonction *Luffa* v1 . . . 91
4.3.5 Degré de la permutation Q_j et de son inverse pour *Luffa* v2 . . . 91
4.3.6 Partitions en sommes nulles pour la permutation Q_j de *Luffa* v2 . . . 93
4.3.7 Différentielles d'ordre supérieur pour la fonction de compression de *Luffa* v2 . . . 94
4.3.8 Degré algébrique de la fonction *Luffa* v2 avec valeurs initiales choisies 94
4.4 Application à l'AES . . . 95
4.5 Application à la permutation interne de Grøstl . . . 96
4.5.1 Borne sur le degré de la permutation P de Grøstl-256 . . . 98

5 Influence du degré de F^{-1} **101**
5.1 Une première borne impliquant le degré de la permutation inverse . . . 102
5.2 Résultat principal . . . 102
5.3 Quelques corollaires . . . 104
5.4 Généralisation aux fonctions équilibrées non-injectives . . . 106

5.5 Applications à certaines primitives symétriques . . . 110
5.5.1 Attaque contre une variante du chiffrement par blocs $\mathcal{KN}$ . . . 110
5.5.2 Application à Rijndael-256 . . . 114
5.5.3 Application à la fonction de hachage ECHO . . . 114
5.5.4 Application à la fonction de hachage JH . . . 116

6 Sur la propagation des relations linéaires 119
6.1 La notion de la (v, w)-linéarité . . . 120
6.1.1 Définition et propriétés générales . . . 120
6.1.2 Lien avec la construction Maiorana-McFarland et les dérivées du deuxième ordre . . . 121
6.1.3 Fonctions $(n-1, 1)$-linéaires . . . 122
6.2 Application à Hamsi . . . 128
6.2.1 Description de l'attaque de T. Fuhr . . . 128
6.2.2 Amélioration de l'attaque de T. Fuhr . . . 131
6.2.3 Relations affines pour trois tours de la fonction de compression . . . 132
6.3 Conclusion . . . 137

II Securité physique des cryptosystèmes symétriques 139

7 Attaques par canaux cachés 141
7.1 Introduction . . . 141
7.2 Attaques par canaux cachés . . . 142
7.2.1 Bref historique . . . 142
7.2.2 Les fuites par mesure du temps . . . 144
7.2.3 Analyse simple du courant (SPA) . . . 144
7.2.4 Les analyses statistiques du courant . . . 144
7.2.5 L'analyse différentielle du courant (DPA) . . . 146
7.3 Protection contre les analyses statistiques du courant . . . 147
7.3.1 Désynchronisation . . . 148
7.3.2 Autres contre-mesures au niveau matériel . . . 148
7.3.3 Randomisation des données . . . 148
7.3.4 Contre-mesures au niveau protocole . . . 150
7.3.5 Cryptographie résistante aux fuites d'information . . . 150
7.4 Le concours SHA-3 et les attaques physiques . . . 150
7.5 Analyse CPA de Grøstl et Skein . . . 151
7.5.1 Attaque par canaux cachés contre HMAC et *envelope* MAC . . . 152
7.5.2 Analyse de Grøstl . . . 153
7.5.3 Analyse de Skein . . . 158
7.5.4 Analyse de performance pour les deux candidats . . . 162

Conclusions 165

Bibliographie 182

Index 183

Table des figures 184

Liste des algorithmes 185

Liste des tableaux 188