Mathématiques discrètes appliquées à la cryptographie symétrique

Yann Rotella

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Yann ROTELLA

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Mathématiques discrètes appliquées à la cryptographie symétrique

soutenue publiquement le 19 septembre 2018 devant le jury composé de :

Mme Anne Canteaut	Inria de Paris	Directrice
M. Joan Daemen	Radboud University	Rapporteur
M. Henri Gilbert	ANSSI	Rapporteur
M. Pierre-Alain Fouque	Université de Rennes 1	
M. Antoine Joux	Sorbonne Université	
Mme Sihem Mesnager	Université de Paris 8	
Mme María Naya-Plasencia	Inria de Paris	
M. Sondre Rønjom	University of Bergen	

Remerciements

La production de ce manuscrit n'aurait pu se faire sans l'accompagnement de tout mon entourage proche qui a admirablement supporté mon caractère pendant ces années de thèse.

À la rédaction de ces remerciements, mes premières pensées sont naturellement tournées vers Anne Canteaut qui m'a donné goût à la cryptographie symétrique bien avant de m'avoir pris sous son aile en tant que thésard. Anne, je ne te remercierai jamais assez pour m'avoir apporté cette passion de la recherche et pour m'avoir accordé ta confiance dès le début. La liberté que tu m'as laissée a formé un excellent équilibre avec ton implication et ton intérêt sans cesse présents. Je garderai un excellent souvenir de ces années passées à tes côtés et de tous les moments devant ton tableau à profiter de ta rigueur scientifique et de tes réflexions ainsi qu'à m'écouter, toujours avec une grande attention. Ton caractère toujours chaleureux, prévenant et amical à mon égard ainsi que ton intérêt pour ma vie personnelle m'ont aussi permis de me sentir extrêmement bien pendant toutes ces années et m'ont largement donné confiance. Pour toutes ces raisons, ces années ont été uniquement joyeuses et harmonieuses en toutes circonstances : Anne, je ne peux donc que t'en être profondément reconnaissant.

Ensuite, je voudrais remercier chaleureusement Henri Gilbert pour avoir relu mon manuscrit avec une incroyable attention et pour participer à mon jury de thèse. Pour ces mêmes raisons, mais aussi pour m'accueillir chez lui en post-doc à Nijmegen, je tiens à remercier Joan Daemen.

D'ailleurs, je tiens à remercier l'ensemble des autres membres du jury de thèse pour avoir accepté de m'écouter parler : Pierre-Alain Fouque, Antoine Joux, Sihem Mesnager, María Naya-Plasencia et Sondre Rønjom.

Sihem, je te remercie aussi pour porter un intérêt certain sur mes travaux depuis le début : tu as d'ailleurs évalué mon travail au comité de suivi doctoral avec Olivier Rioul, que je remercie au passage.

Je souhaite aussi remercier l'ensemble des personnes avec lesquelles j'ai pu discuter, plaisanter et travailler lors de réunions, de séminaires et de conférences : les journées Codage et Cryptographie, Fast Software Encryption, CRYPTO, l'ANR BLOC... Une pensée toute particulière pour l'ANR BRUTUS pour avoir financé ma thèse et m'avoir permis d'en apprendre beaucoup au contact de ces réunions. En parlant de BRUTUS, je remercie María Naya-Plasencia et Thomas Fuhr : c'était chouette de travailler avec vous. De la même manière je souhaite

dire merci à Claude Carlet, Pierrick Méaux, Virginie Lallemand, Sébastien Duval, Geoffroy Couteau, Mélissa Rossi et Aurélien Dupin pour les travaux menés à vos côtés. Tomer Ashur, Maria Eichlseder, Gaëtan Leurent, Brice Minaud, Yu Sasaki et Benoît Viguier : thanks for all the work at Leiden ! Ich möchte auch Christof und Gregor danken : Die Zusammenarbeit mit Ihnen hat wirklich Spaß gemacht.

D'ailleurs je remercie Christina Boura pour son invitation, ainsi que Luca de Feo, Jean-Pierre Tillich pour sa confiance et son implication (ainsi que les cours de Babyfoot), Jean-Pierre Flori, Serge Vaudenay et Michael Bages pour leurs invitations respectives.

À l'Inria, dans l'équipe-projet SECRET, je me suis senti dans mon élément. Cela a naturellement pu être le cas grâce à la bonne humeur qui règne au sein de cette équipe. Je tiens donc à remercier l'ensemble des personnes que j'ai rencontrées ici, chez SECRET (en espérant n'oublier personne [1]) : Adrien, Anaïs, André, André, Andrea, Anirudh, Anne, Anthony, Antoine, Audrey, Aurélie, Christelle, Christina, Christof, Ferdinand, Florian, Gaëtan, Ghazal, Irene, Jean-Pierre, Joëlle, Julia, Kaushik, Kévin, Léo, Marìa, Mariem, Mathilde, Matthieu, Nicky, Nicolas, Pascale, Rémi, Rodolfo, Sébastien, Thomas, Thomas, Valentin, Valentin, Victoire, Virginie, Vivien et Xavier.

Plus généralement, je voudrais tous vous remercier pour : les pots, les gâteaux, les bières prises au QG (depuis que l'on a déménagé), le bowling, le lancer de haches, les restaurants, votre bonne humeur, le babyfoot [2], les discussions politiques enrichissantes, les échanges scientifiques incroyablement formateurs et l'ambiance de travail saine et détendue.

J'aimerais adresser une pensée supplémentaire à celles et ceux qui ont partagé le bureau C201 (anciennement bureau 1 à Rocquencourt) : Virginie, Sébastien, Xavier, Christof et Florian. En particulier je tiens à remercier mon "frère académique" Sébastien dont les multiples discussions que nous avons pu avoir depuis notre premier jour de thèse ainsi que les constructions au tableau ont égayé mes journées tout au long de ces trois dernières années. Enfin, Christina, Christelle, Julia, Thomas, Matthieu, Kévin, Vivien et Philip Morris, je tiens à vous rendre grâce pour tous les moments de détente au $5^{\text{ème}}$.

Je remercie aussi les jeunes cryptographes d'ailleurs avec qui j'ai pu avoir des échanges vifs et amusants, Julien, Colin, Victor,... Merci Cécile pour avoir rendu possible la jolie impression de ce manuscrit !

Je tiens aussi à citer ici un de mes premiers professeurs qui a suivi mon travail et mes études depuis longtemps : Jean-Luc Sarrabayrouse.

Parce que ils ou elles m'ont tous et toutes porté une attention particulière, mais surtout parce que tous ces moments passés sont incroyablement précieux, je voudrais exprimer ici mes sentiments à toutes les personnes suivantes, sans qui ces années n'auraient jamais été aussi joyeuses.

Au Tartiflette Crew, je voudrais dire que vous rencontrer a été déterminant pour moi, et extrêmement enrichissant. J'ai une petite pensée pour Camille et

1. J'espère que tu m'excuseras, toi qui as la patience de lire ce manuscrit et que j'ai lamentablement oublié

2. tmtc Sébastien on a le beau jeu.

son engouement pour les jeux et le papotage, Vicky et Caro pour ces moments musicaux intenses, Maud pour me comprendre avec brio, Chloé pour son énergie, Julia, Clairou, Mathias, Tempo, Yanis, Marianne, Paul, Raphie, Laura, Cécile, Bryan pour leur présence qui ont animé ces nombreuses soirées. Karen, je voudrais faire plus que te remercier en te souhaitant surtout du courage pour l'année à venir. Matthieu, je te fais surtout plein de câlins. Enfin Elina, merci pour les sorties culturelles et nos discussions qui m'ont apporté bien plus que tu ne le penses.

Je remercie Nico, Baptiste, Henri, Léa, Dom-dom, Daudau, Stéphanie, mais aussi Charles, Romain, Félix, Léo ainsi que Patrick, Ponyo, Geoffroy. Plus généralement je fais un Big Up à la LudoTech et à Télécom BariTech pour les grands moments passés ensemble. En tout cas, merci pour les jeux, les soirées, vos invitations et la bonne ambiance.

Je souhaite remercier toute l'Académie Parisienne d'Aïkido du 20$^{\text{ème}}$ et Sensei De Nussac dont l'exigence et la perfection ont su me guider.

Je salue l'humour rarement égalé et l'intelligence mais surtout le jeu des Kadavreskys et de tous les anciens de la Porte Bleue. Merci de me faire toujours rire avec autant de finesse.

Je veux aussi remercier les bars de la Butte aux Cailles ainsi que ceux de la place Stravinsky pour leur accueil chaleureux et amical, ainsi qu'Une Petite Mousse pour les quelques minutes de bonheur par soir.

Andréa, t'as pas perdu ta bonne humeur depuis la seconde et je te remercie d'égayer à chaque fois nos levers de coude.

À Nono et Adé, j'envoie aussi des bisous.

Cyrielle, un grand merci pour les Zeldas et ton accueil chaleureux.

Je voudrais aussi citer dans ces remerciements Agathe, Guilhem, Caro, Judith, Sonia, Marie, Rodo, Laeti, Tom, Godo, Romain, Solène, Élodie, Loris, Maud et Ugo (en espérant ne pas avoir oublié qui que ce soit) pour me faire rire et pour leur ambiance bon enfant.

Nous arrivons au point des remerciements où j'ai forcément oublié quelqu'un. À toi, oui toi qui as ce document entre tes mains, je te remercie par avance pour l'effort que tu réalises et m'excuse platement de t'avoir oublié [3].

À tout le groupe du LOL : Joyce, Hugo, Damien, Robin, Lucas, Thomas, Philippe, Kévin, Mariam, Naïs et Florian, je ne peux que vous dire que je vous aime, et que, comme Katerine, je vous fais des bisous. Vous êtes toujours là depuis 8 ans déjà et je ne peux que vous remercier pour toutes les aventures que nous avons entreprises, les jeux, les voyages, les randonnées, votre finesse d'esprit, vos traits d'humour et surtout votre amitié et votre amour. Hugo et Robin, merci de plus pour la relecture on ne peut plus efficace de ce manuscrit.

À Benjamin et Eric, là je n'aurai probablement jamais assez de mots pour exprimer correctement mon ressenti. Ayant été toujours présents pour écouter mes doutes et partager les vôtres mais aussi là pour s'amuser, vous ne pouviez pas ne pas avoir votre propre paragraphe ici. Nos interminables discussions politiques ainsi que votre amitié et votre amour présents depuis les bacs à sable de l'école

3. Tu n'as qu'à me demander un exemplaire paraphé.

maternelle Henri Wallon ont joué et jouent encore un immense rôle dans ma vie personnelle. Benji et Eric, que la force soit avec vous, à tout jamais !

Aux trois hommes de ma vie avec lesquels nous formons l'inénarrable Quatuor de l'Enfer : j'ai nommé HP, Maxou ainsi que le beau Thomas, je dédie ces quelques mots. Le temps passé avec vous, nos aventures, votre humour déjanté, votre perspicacité, votre état d'esprit et votre bonne humeur pour ne citer qu'eux forment un cocktail à consommer sans modération. Pour me faire rire avec autant de délicatesse et d'esprit mais aussi pour m'accompagner avec autant d'intérêt et d'attention, je tiens à vous rendre grâce ici et maintenant de l'amour que je vous porte.

Qui plus est, je voudrais remercier toute ma famille : Hélène pour sa gentillesse ainsi que pour son parcours scientifique qui m'a indubitablement inspiré sans oublier naturellement Clément et Emma, Didier, dont j'admire l'état d'esprit ainsi que sa remarquable passion pour la musique qu'il m'a faite partager. À Irène et Frédéric, j'aimerais aussi adresser quelques mots, bien que mon vocabulaire restreint ne suffise probablement pas. Pour l'écoute attentive, l'amour, la compréhension, l'ouverture d'esprit mais aussi pour avoir suivi de très près ma thèse et m'avoir toujours soutenu, je tiens à vous remercier du fond du coeur. D'ailleurs, c'est un peu à vous que je dédie ce manuscrit.

Enfin, Camille, mes derniers mots ne pouvaient pas être adressés à quelqu'un d'autre. Merci. Merci de me comprendre mieux que quiconque sans même que je parle. Merci de voir toujours les choses qui comptent avec autant de justesse. Merci de me soutenir, peu importe les circonstances. Je pourrais m'étaler bien plus, mais les mots ne pourraient pas faire ressortir avec justesse mon ressenti. Je paraphraserai donc Montaigne : parce que c'est toi, parce que c'est moi, je ne peux que te remercier d'être toi, juste toi, et d'accepter d'entremêler nos vies.

Organisation de la thèse et aperçu des contributions

Afin que le.a lecteur.rice trouve une certaine logique dans la présentation de mes travaux, ni le résumé des résultats ni le document lui-même ne suivent l'ordre chronologique. Ce chapitre fait office de description générale du manuscrit et est là pour décrire l'ensemble de mes contributions. Le.a lecteur.rice prendra soin de se souvenir de l'ensemble de mes coauteur.e.s, car c'est grâce à un grand nombre de discussions avec ces personnes et bien d'autres encore que ces résultats ont pu voir le jour. Chaque chapitre du document correspond à un article publié ou en cours de publication.

Chapitre 1. Introduction. Dans l'introduction, nous faisons un bref résumé de l'historique de la cryptographie. Nous décrivons les problèmes auxquels le.a cryptographe fait face, afin de comprendre le contexte des travaux et des résultats présentés dans ce document.

Chapitre 2. Préliminaires. Ce chapitre passe rapidement sur des notions indispensables à la compréhension du document, principalement sur les corps finis et les fonctions booléennes utilisées en cryptographie.

Chapitre 3. Attaques par invariant sur les chiffrements par bloc. Depuis quelques années, les cryptographes s'intéressent à la construction de chiffrements à bas coût, afin de satisfaire certaines contraintes d'implémentation. Dans de nombreux chiffrements par bloc à bas coût proposés récemment, les concepteur.rice.s ont fait le choix de diminuer drastiquement la complexité de l'algorithme de cadencement de clefs, celui-ci se réduisant à l'addition de la clef-maître avec une constante de tour, souvent creuse, ou arbitrairement choisie. Sur certains de ces algorithmes de chiffrement, il apparaît alors un phénomène étrange : un sous-ensemble en entrée du chiffrement est envoyé sur lui-même, mais ceci pour un espace de clefs parfois de taille non-négligeable. Ce phénomène est évidemment indésirable du point de vue de la sécurité de l'algorithme.

Nous expliquons pourquoi ce phénomène apparaît dans certains chiffrements : c'est une mauvaise interaction entre la couche linéaire et le choix des constantes de

tour. Nous donnons des conditions nécessaires sur l'existence d'espaces invariants à la fois par la couche linéaire et la couche de substitution. Finalement, nous expliquons aussi comment choisir les constantes de tour de manière approriée en fonction de la couche linéaire, et combien de tours il faut pour assurer la non-existence d'espaces invariants. Toutes ces propriétés sont reliées à la forme canonique rationnelle de la matrice représentant la couche linéaire du chiffrement.

Les résultats décrits dans ce chapitre sur les invariants ont été publiés à *CRYPTO 2017* : Christof Beierle, Anne Canteaut, Gregor Leander et Yann Rotella : Proving resistance against invariant attacks : How to choose the round constants, *In* Jonathan Katz et Hovav Shacham, éditeurs,*CRYPTO 2017*, *Part II*, volume 10402 de *LNCS*, pages 647 à 678. Springer, Heidelberg, août 2017.

Chapitre 4. Attaques sur les registres filtrés exploitant la structure de corps fini. Dans ce chapitre nous nous focalisons sur les registres à décalage et à rétroaction linéaire (LFSR) filtrés. En 2010, Sondre Rønjom et Carlos Cid ont mis en évidence une équivalence non-linéaire entre les registres filtrés que nous avons appelée équivalence monomiale. Avec Anne Canteaut, nous avons étudié cette équivalence entre les registres filtrés et nous avons observé que cette équivalence ne modifiait pas la complexité linéaire de la suite produite, et donc la complexité des attaques algébriques.

De plus, nous avons montré que le critère pertinent qui mesure la résistance aux attaques par corrélation rapides n'est pas la non-linéarité, mais est en fait la non-linéarité généralisée. Ce critère avait été introduit en 2001 par Gong et Youssef, mais sans motivation cryptographique particulière et nous le relions ici à la résistance à ces attaques. Enfin et surtout, nous avons pu montrer comment il était possible de réaliser des attaques par corrélation classiques en découpant l'état interne du registre dans les sous-groupes multiplicatifs du corps fini $\mathbb{F}_{2^n}$. Cette attaque nous permet de réaliser une technique de type "diviser pour mieux régner" afin de retrouver l'état initial du registre. Il en découle la nécessité de définir un nouveau critère sur les fonctions booléennes, afin de se prémunir contre ce type d'attaque.

Les travaux décrits en détail dans ce chapitre ont été publiés à *Fast Software Encryption 2016* : Anne Canteaut et Yann Rotella : Attacks against filter generators exploiting monomial mappings. *In* Thomas Peyrin, éditeur : *FSE 2016*, volume 9783 de *LNCS*, pages 78-98. Springer, Heidelberg, mars 2016.

Chapitre 5. Cryptanalyse de FLIP. L'algorithme de chiffrement à flot FLIP [MJSC16], publié à *EUROCRYPT 2016* par Pierrick Méaux, Claude Carlet, Anthony Journault et François-Xavier Standaert est un chiffrement symétrique adapté aux contraintes particulières du chiffrement hybride complètement homomorphe. Cependant, avec Sébastien Duval et Virginie Lallemand, nous avons pu monter une attaque de type "supposer et déterminer" en 2^{54} (respectivement 2^{68}) opérations élémentaires, pour une sécurité revendiquée de 80 bits (respectivement 128 bits). Le chiffrement FLIP souffre principalement d'un trop petit nombre de

monômes dans la représentation multivariée de la fonction booléenne de filtrage, le système à résoudre devenant alors trop structuré, en particulier trop creux.

L'article a été publié à *CRYPTO 2016* : Sébastien Duval, Virginie Lallemand et Yann Rotella : Cryptanalysis of the FLIP family of stream ciphers. *In* Matthew Robshaw et Jonathan Katz, éditeurs : *CRYPTO 2016*, *Part I*, volume 9814 de *LNCS*, pages 457-475. Springer, Heidelberg, août 2016.

Chapitre 6. Fonctions booléennes à entrées restreintes. Suite à une longue discussion avec les concepteurs de FLIP, nous nous sommes rendu compte que ces derniers n'avaient pas analysé correctement la sécurité de leur chiffrement. En effet, FLIP est composé d'un registre dans lequel la clef est stockée. À chaque instant, une permutation publique choisie à l'aide d'un générateur pseudo-aléatoire est appliquée aux bits du registre, et une fonction booléenne est utilisée comme fonction de filtrage sur la clef permutée. Ainsi, les entrées de la fonction sont toujours de poids de Hamming égal à celui de la clef. Utiliser les critères classiques comme l'immunité algébrique ou la non-linéarité n'a donc plus de sens car l'entrée de la fonction n'est pas uniformément distribuée. Il convient alors de revisiter les critères classiques sur les fonctions booléennes cryptographiques en considérant des entrées restreintes à un sous-ensemble donné. Dans le cas de FLIP, il faut considérer les entrées ayant un poids de Hamming fixé. Nous avons donc réalisé une étude sur les fonctions booléennes à entrées restreintes, et nous avons vu à quel point les propriétés cryptographiques des fonctions booléennes pouvaient varier lorsque l'entrée était restreinte.

L'article a été publié dans le journal *IACR Transactions on Symmetric Cryptology* : Claude Carlet, Pierrick Méaux et Yann Rotella : Boolean functions with restricted input and their robustness ; application to the FLIP cipher. *IACR Transactions on Symmetric Cryptology*, 2017(3) :192-227, 2017.

Chapitre 7. Cryptanalyse du PRG de Goldreich. Le PRG (générateur pseudo-aléatoire) de Goldreich [Gol00] est une construction théorique très célèbre permettant d'étendre une source d'*alea* (graine) en une suite de taille plus grande. La sécurité repose sur l'absence d'attaque fonctionnant en temps polynomial et pose la question de l'existence de PRG dont les bits de sortie ne dépendraient que d'un faible nombre de bits en entrée. Cette construction a une structure proche de celle de FLIP. On peut donc lui appliquer une attaque similaire à celle que nous avons proposée sur FLIP. Ceci conduit alors à une attaque sur ce PRG, avec la complexité de $\mathcal{O}(2^{\sqrt{n}})$, où n est la taille de la graine [CDM+18].

Nous montrons qu'un niveau de sécurité minimal (par exemple 80 bits) dans le générateur de Goldreich impose une taille de graine très importante (au moins 10000 bits), ce qui rend l'utilisation de ce PRG très peu pratique voire impossible. Finalement, cette étude remet sévèrement en cause la sécurité du PRG de Goldreich qui reposait sur l'absence d'attaque en temps polynomial, puisque nous avons proposé un algorithme en temps sous-exponentiel qui s'avère extrêmement puissant pour des tailles raisonnables. De plus, nous concluons à l'impossibilité de construire un générateur pseudo-aléatoire ayant une petite

localité, c'est-à-dire dont la sortie ne dépend que de peu de bits de l'entrée, ce qui affine les connaissances sur cette construction théorique.

Les résultats détaillés dans ce chapitre sont publiés à *ASIACRYPT 2018* et ont été réalisés en collaboration avec Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux et Mélissa Rossi.

Chapitre 8. Cryptanalyse de Ketje. Ketje [BDP+14] est un chiffrement authentifié proposé par les auteurs du standard de hachage SHA-3 [BDPA13] et inspiré de ce dernier. Ketje suit une construction itérative appelée éponge qui absorbe une partie du message clair et produit une partie du texte chiffré à chaque tour. Pour un état interne secret de 200 bits, nous avons accès à une partie de l'information sur cet état après chaque application de la fonction de tour. L'attaque consiste à exploiter l'information issue de l'état interne à des instants différents et à combiner cette information intelligemment de manière à en déduire la valeur complète de l'état à un instant donné. Dans le cas de Ketje, nous parvenons dans un premier temps à exploiter de l'information sur deux tours consécutifs et dans un second temps, nous utilisons des techniques non-triviales d'algorithmes de fusion de listes, en criblant avec des équations non-linéaires pour exploiter de l'information sur 3 tours consécutifs.

Notre cryptanalyse [FNR18] s'applique à une version affaiblie de Ketje dans laquelle le ratio (paramètre critique de la construction en éponge) est augmenté (32 ou 40 bits au lieu de 16). Si elle ne contredit pas la sécurité revendiquée par les concepteurs, notre attaque apporte un éclairage nouveau sur Ketje car elle met en évidence une faiblesse jusqu'ici non-exploitée, issue de la possibilité d'obtenir des équations creuses.

L'article détaillant notre cryptanalyse a été publié dans le journal *IACR Transactions on Symmetric Cryptology* : Thomas Fuhr, María Naya-Plasencia et Yann Rotella : State-recovery attacks on modified Ketje Jr. *IACR Transactions on Symmetric Cryptology*, 2018(1) :29-56, 2018.

Conclusion et perspectives. L'ensemble de mes travaux montre qu'une vision structurelle des objets mathématiques mis en jeu dans les systèmes cryptographiques peut avoir un apport important en cryptanalyse. Cette vision permet de découvrir de nouvelles attaques et de mettre en évidence des nouveaux critères de conception. Il est donc nécessaire de faire des allers-retours entre la conception de chiffrements, la cryptanalyse et les notions mathématiques fondamentales exploitées dans les attaques afin de déterminer précisément le niveau de sécurité pratique des algorithmes de chiffrement.

Table des matières

Chapitre 1

Introduction

1.1 Principes de la cryptographie

Un des principaux objectifs de la cryptographie est de permettre à deux interlocuteur.rice.s (Alice et Bob), grâce à un procédé de chiffrement, de pouvoir communiquer de manière confidentielle sur un canal de communication non sécurisé. Plus généralement, la cryptographie permet d'assurer la *confidentialité*, mais aussi l'*authenticité* et l'*intégrité* des messages transmis.

La confidentialité d'une communication est assurée par un algorithme de chiffrement. Cet algorithme transforme un message initial appelé le *clair* en un message indéchiffrable, le *chiffré*, pour toute entité ne connaissant pas la *clef* (privée ou secrète) qui permet d'inverser le chiffrement afin de *déchiffrer* le message.

Nous pouvons distinguer deux types de chiffrement : les chiffrements à clef publique (dits *asymétriques*) et ceux à clef secrète (dits *symétriques*).

La cryptographie asymétrique. Dans le contexte de la cryptographie à clef publique, Alice et Bob possèdent chacun.e une paire de clefs publique/privée. La clef privée est connue seulement de son propriétaire et la clef publique est connue de tous. Quand Bob souhaite envoyer un message à Alice, il le chiffre en utilisant la *clef publique d'Alice*. Ensuite, Alice peut déchiffrer en utilisant sa clef privée. En des termes un peu moins formels, tout le monde peut chiffrer des message en utilisant la clef publique, mais seul le détenteur de la clef privée associée à la clef publique peut déchiffrer.

La cryptographie symétrique. Dans le contexte de la cryptographie à clef secrète, Alice et Bob doivent au préalable se mettre d'accord sur une valeur : la clef secrète. Cette clef leur permet alors de chiffrer et de déchiffrer.

La cryptographie hybride. Les algorithmes de chiffrement symétriques sont les seuls qui aient des performances acceptables pour la plupart des applications

en termes de débit, de taille de clef ou de complexité d'implémentation matérielle. Mais, l'inconvénient de la cryptographie symétrique est la nécessité d'échanger la clef secrète de manière sécurisée. Ainsi, dans la plupart des applications, nous combinons les deux types de cryptographie : on utilise un système asymétrique pour échanger une clef secrète, qui paramètre ensuite un chiffrement symétrique. On parle alors de chiffrement hybride.

1.2 Quelques repères historiques

Si le principe d'assurer la confidentialité d'un message remonte à l'Antiquité (Plutarque fait état de l'utilisation par Lysandre de Sparte d'une scytale en 404 avant J-C), avant le XIX$^{\text{ème}}$ siècle, en revanche, les messages codés ne sont généralement obtenus qu'avec des transpositions et substitutions alphabétiques. La cryptographie comme on l'entend aujourd'hui est nettement plus élaborée et a pu voir le jour grâce à l'expansion de l'informatique et de la théorie de l'information.

Voici donc quelques dates historiques importantes qui nous précipitent jusqu'à l'avènement de la cryptographie moderne. La cryptographie est une science relativement jeune, avec des applications évidentes dans un monde de plus en plus connecté et qui pose actuellement des questions théoriques se situant au croisement de l'informatique, des mathématiques et plus récemment de la physique avec la probabilité grandissante de voir un jour arriver un ordinateur quantique.

Principe de Kerckhoffs. En 1883, Auguste Kerckhoffs énonce un principe fondamental de la cryptographie [Ker83] :

La sécurité d'un cryptosystème doit reposer exclusivement sur le secret de la clef.

Ce principe de base de la cryptographie s'oppose à une sécurité par obscurité. Il convient alors de faire en sorte que la partie secrète, *i.e.* la clef, soit la plus facile et la moins coûteuse à changer. En effet, si le système entier est compromis à un moment, il faut le remplacer dans sa totalité. Il est important de noter que ce principe a été énoncé à la fin du XIX$^{\text{ème}}$ siècle à une époque où les algorithmes de chiffrement étaient mécaniques.

Le chiffrement de Vernam. Cet algorithme de cryptographie inventé par Gilbert Vernam en 1917 est aussi appelé masque jetable ou *one-time pad*. C'est un chiffrement parfait (inconditionnellement sûr), *i.e.* théoriquement impossible à "casser". Cependant il est impossible à mettre en pratique puisqu'il nécessite l'utilisation d'une clef composée d'un suite aléatoire de caractères au moins aussi longue que le message. De plus, chaque clef ne doit être utilisée qu'une seule fois, d'où l'impraticabilité de ce chiffrement. L'opération de chiffrement consiste à combiner par l'opération booléenne XOR ("ou" logique exclusif) les bits du message clair avec les bits de la suite aléatoire correspondant à la clef.

Enigma et les "bombes" de Turing. Pendant la seconde guerre mondiale, les Allemands utilisaient une machine à chiffrer appelée Enigma dont le nombre de clefs possibles était de l'ordre de 10^{16} c'est-à-dire environ 10 millions de milliards, à une époque où les ordinateurs n'existaient pas encore. La taille de l'espace des clefs possibles de la machine Enigma rendait toute recherche exhaustive manuelle hors de portée. C'est avec les "bombes" d'Alan Turing, des machines automatisant les calculs que ce dernier a pu "casser" Enigma : à partir de messages chiffrés interceptés par les Alliés, il pouvait retrouver la clef et donc déchiffrer les messages.

Claude Shannon et la notion de sécurité. Claude Shannon [Sha49] a prouvé que le chiffrement de Vernam était inconditionnellement sûr. Ce dernier étant impraticable, Shannon a théorisé la notion de sécurité des cryptosystèmes et a introduit la notion de sécurité pratique.

Définition 1.1 (Sécurité inconditionnelle)**.** *La connaissance du message chiffré n'apporte aucune information sur le message clair.*

Pour avoir une sécurité inconditionnelle, il faut que la clef soit au moins aussi longue que le message. Cela est évidemment impossible à mettre en place sauf dans quelques applications particulières. Donc, dans toute la suite du document, quand nous parlerons de sécurité, nous parlerons de sécurité pratique : tous les chiffrements utilisés actuellement sont théoriquement cassables.

Définition 1.2 (Sécurité pratique)**.** *La connaissance du message chiffré et de certains couples clairs/chiffrés ne permet de retrouver ni la clef ni le message en un temps raisonnable.*

De plus, Shannon a introduit les notions de *confusion* et de *diffusion*, nécessaires à la sécurité d'un algorithme de chiffrement. La confusion signifie que la relation entre les bits du texte clair, les bits du chiffré et les bits de la clef doit être suffisamment compliquée. La diffusion exprime elle le fait que changer un bit dans le texte clair doit changer un grand nombre de bits dans le texte chiffré. En d'autres termes, un bit en entrée du chiffrement doit impacter plusieurs bits en sortie, et la relation entre les bits doit être suffisamment compliquée.

Data Encryption Standard (DES - 1973) L'algorithme de chiffrement DES est un chiffrement symétrique standardisé qui souffre principalement de la petite taille des clefs (56 bits). Il est actuellement complètement cassé, principalement à cause de l'accroissement de la puissance des ordinateurs. Pour donner quelques ordres de grandeur, en juin 1997, en utilisant 78000 ordinateurs, le DES était cassé en 96 jours ; 6 mois plus tard, en distribuant les calculs sur plusieurs millions de processeurs, récupérer une clef secrète utilisée dans le DES prenait 39 jours. Encore 6 mois plus tard, pour un coût de 250000 $, on arrivait à casser le DES en une dizaine d'heures...

Les tailles de clef actuelles sont, en cryptographie symétrique, souvent de 128 bits ou 256 bits.

Cependant, tous les systèmes décrits précédemment sont des algorithmes de chiffrement symétrique, ce qui impose que deux entités qui veulent communiquer en utilisant ces algorithmes doivent se mettre d'accord au préalable sur le secret commun à utiliser, sans que celui-ce ne soit compromis.

Cryptographie asymétrique, Diffie-Hellman (1976). Le premier protocole d'échange de clef au travers d'un canal non-sécurisé est le protocole de Diffie-Hellman [DH76]. C'est le début de la cryptographie asymétrique. Le principe est le suivant : Alice choisit un groupe fini et publie un générateur g de ce groupe. Elle choisit ensuite un entier a au hasard, qu'elle garde secret, élève g à la puissance a et transmet cette valeur à Bob. Bob choisit de même un entier b au hasard et envoie g^b à Alice. Finalement, Alice élève g^b à la puissance a et Bob fait de même pour g^a et chacun retrouve g^{ab}, qui devient le secret commun partagé par Alice et Bob. Ce protocole repose sur la difficulté de calculer g^{ab} à partir uniquement de g, g^a et g^b sans avoir connaissance des valeurs de a et de b.

RSA : Rivest Shamir Addleman (1978). RSA [RSA78] est le chiffrement asymétrique le plus connu et le plus utilisé actuellement. Ce cryptosystème repose lui sur la factorisation du produit de deux nombres premiers, problème supposé difficile.

AES : Advanced Encryption Standard (2000). Le standard AES [DR02] est le chiffrement symétrique le plus connu et le plus utilisé actuellement et considéré dans la communauté comme le plus sûr.

1.3 Les algorithmes de chiffrement symétriques

Une des difficultés en cryptographie est la possibilité de pouvoir traiter des messages de longueur arbitraire. Pour réaliser cela il existe deux manières de traiter le problème, qui divisent les algorithmes de chiffrements symétriques en deux grandes familles : les chiffrements à flot et les chiffrements par bloc.

Les chiffrements à flot. La première technique consiste à imiter le chiffrement de Vernam. Celui-ci nécessite une suite aléatoire au moins aussi longue que le message. Pour imiter le fonctionnement de ce chiffrement inconditionnellement sûr, nous utilisons des chiffrements à flot. L'idée ici consiste à utiliser un générateur *pseudo*-aléatoire, et d'utiliser la suite engendrée comme "clef" d'un chiffrement de Vernam.

Un chiffrement à flot repose donc sur un générateur pseudo-aléatoire public, c'est-à-dire un algorithme qui permet de dériver, à partir d'une graine aléatoire,

une suite binaire de plus grande taille ayant de bonnes propriétés statistiques [1]. La suite ainsi produite est appelée la *suite chiffrante*.

Dans ces conditions, Alice et Bob initialisent l'état interne du générateur pseudo-aléatoire avec la clef (typiquement de taille 128 bits) ainsi qu'une valeur publique (IV - vecteur d'initialisation) afin que chacun.e d'eux engendre la même suite binaire, employée ensuite pour chiffrer comme dans le *one-time-pad*. Ainsi, Alice et Bob peuvent échanger des messages de taille arbitraire, en utilisant le nombre de bits nécessaires correspondant dans la suite chiffrante.

Les chiffrements à flot et leur sécurité seront étudiés tout au long de cette thèse à partir du chapitre 4.

Les chiffrements par bloc. L'autre grande famille d'algorithmes de chiffrement symétriques est la famille des chiffrements par bloc. L'idée ici consiste à "découper" le message en plusieurs blocs de même longueur (typiquement 64 ou 128 bits). Naturellement, les messages étant de taille arbitraire, il est nécessaire de définir, au préalable, une règle inversible de "rembourrage" du message clair, de manière à ce que la taille de celui-ci devienne un multiple de la taille des blocs considérés.

Il est ensuite nécessaire de définir comment traiter ces blocs. C'est là qu'intervient le chiffrement par bloc. Un chiffrement par bloc est une transformation qui prend en entrée un bloc de taille n (souvent 64 ou 128 bits) ainsi qu'une clef k de taille κ bits et renvoie en sortie un bloc de même taille.

Pour pouvoir déchiffrer, il est nécessaire que, pour toute clef, la transformation du message soit inversible. Un chiffrement par bloc est donc défini comme une famille de 2^κ permutations et choisir une clef revient à choisir une permutation dans cette famille.

Enfin, l'utilisation d'un chiffrement par bloc nécessite la définition d'un *mode opératoire* qui précise comment les blocs du message clair et les blocs du message chiffré sont combinés, afin de déterminer le message qui sera transmis. Les chiffrements par bloc sont décrits plus en détail au chapitre 3.

1.4 Sécurité et cryptanalyse

La sécurité des systèmes asymétriques repose sur des problèmes difficiles (factorisation, logarithme discret, recherche de plus court vecteur dans un réseau euclidien,...), c'est-à-dire des problèmes pour lesquels on ne connaît pas d'algorithme de résolution en temps polynomial [2]. Comme on parle de sécurité asymptotique, cela signifie que le système peut "passer à l'échelle" quand la puissance de calcul des ordinateurs évolue. Par exemple, les clefs utilisées actuellement dans le cryptosystème RSA sont des entiers de 2048 bits.

1. c'est-à-dire qu'il est difficile de trouver des propriétés distinguantes sur ladite suite.
2. Sous réserve de la véracité de la conjecture du millénaire $\mathcal{P} \neq \mathcal{NP}$ pour les problèmes $\mathcal{NP}$ difficiles.

Toutefois, dans les spécifications pratiques d'un chiffrement, les paramètres sont fixés, et à partir de ce moment, la complexité asymptotique des problèmes sous-jacents n'est plus un argument pertinent. Il est alors nécessaire d'analyser concrètement les paramètres instanciés afin de cerner correctement la sécurité de l'algorithme de chiffrement. Pour la cryptographie symétrique, il n'y a pas de réduction à des problèmes difficiles, tout simplement car les algorithmes sont instanciés concrètement, avec des tailles de bloc et de clefs fixées.

Niveau de sécurité. Le niveau de sécurité d'un chiffrement est défini comme le nombre minimal d'opérations nécessaires pour "casser" le chiffrement, *i.e.* retrouver la clef, une partie du message clair,... La protection apportée par un algorithme de chiffrement est liée à la longueur de la clef, qui peut s'exprimer en bits. En fait, la longueur de la clef quantifie le nombre maximal d'opérations nécessaires au décryptage : c'est le coût de la recherche exhaustive qui consiste à tester toutes les clefs possibles. Pour une clef de κ bits, le coût de la recherche exhaustive est de 2^κ opérations : c'est donc une borne supérieure sur la sécurité du système.

Si l'attaque la plus performante sur un système est de 2^i opérations, on dira que le système a un niveau de sécurité de i bits. Le niveau de sécurité est donc toujours inférieur ou égal à la taille de la clef utilisée. Si c'est égal, cela revient à dire que l'attaque la plus performante est la recherche exhaustive.

En l'état actuel, il est recommandé d'utiliser des clef d'au moins 128 bits pour les systèmes symétriques (rapport du 21 février de l'Agence Nationale de la Sécurité des Systèmes d'Information [3]). Pour les clefs asymétriques, le problème est moins simple : le nombre maximal d'opérations est obtenu pour l'essai systématique de toutes les clefs possibles. Or, pour les systèmes asymétriques, qui sont le plus souvent construits sur des problèmes relevant de l'arithmétique, il existe toujours des moyens largement moins coûteux en temps, à savoir la résolution du problème arithmétique sous-jacent. Concrètement, si on considère le système RSA, basé sur le problème de la factorisation, pour une clef de 2048 bits il n'est pas nécessaire de tester les 2^{2048} clefs, il "suffit" de factoriser un nombre dont l'écriture binaire comporte 2048 bits, ce qui est largement plus simple, bien que toujours totalement infaisable de nos jours. De plus, la complexité de résolution des problèmes difficiles fait l'objet d'études constantes, raffinant *de facto* la sécurité *réelle* des systèmes asymétriques.

Dans tous les cas, que ce soit la cryptographie symétrique ou la cryptographie asymétrique, il est absolument indispensable d'analyser la sécurité de l'ensemble des constructions cryptographiques : c'est la cryptanalyse. En effet, la sécurité des primitives cryptographiques repose essentiellement sur leur étude approfondie : ce n'est pas parce que toutes les attaques existantes fonctionnent en un temps plus coûteux que la recherche exhaustive qu'il ne pourrait pas y en avoir une plus efficace dans 10 ans. Il est donc indispensable d'étudier au plus près les primitives pour s'assurer que les nouveaux schémas apportent bien le niveau de

3. Disponible sur `https://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf`

sécurité revendiqué par les auteur.e.s, puisque la sécurité des systèmes ne repose que sur l'absence d'attaque performante. Le rôle de la cryptanalyse est donc absolument fondamental et constitue le seul moyen d'évaluation du niveau de sécurité des chiffrements.

De plus, à l'issue de chaque nouvelle cryptanalyse, les cryptographes définissent de nouveaux critères cryptographiques et peuvent alors construire des systèmes de plus en plus sûrs, ce que nous réalisons dans cette thèse. L'intérêt de la cryptanalyse est donc double dans ce sens.

La cryptanalyse. Tout d'abord, nous devons nous positionner dans un certain modèle d'attaque, qui définit le "pouvoir" que l'on donne à l'attaquant.e. De manière classique les quatre modèles sont les suivants.

- Les *attaques à chiffré seul*, où l'on considère que l'attaquant.e n'a accès qu'à des textes chiffrés. C'est le modèle le moins "fort" du point de vue du concepteur ou de la conceptrice.
- Les *attaques à clair connu*, où l'on considère que l'attaquant.e a accès à des couples clairs/chiffrés.
- Les *attaques à clair choisi*, où l'on considère que l'attaquant.e peut avoir accès à des messages chiffrés de textes clairs qu'il choisit.
- Les *attaques à chiffré choisi*, où l'on considère que l'attaquant.e peut avoir accès au déchiffrement de messages chiffrés qu'il choisit.

A priori, l'attaquant.e ne connaît pas la clef secrète utilisée. Cependant, il existe aussi d'autres modèles : les *attaques à clefs liées* et les *attaques à clef connue*.

Dans le premier modèle, on considère que l'on a accès à de l'information provenant de chiffrements utilisant le même algorithme mais plusieurs clefs qui sont liées par une certaine relation. Ce modèle semble peu réaliste, cependant, il peut être approprié quand on évalue la sécurité de protocoles particuliers. Il peut amener à des attaques concrètes, comme nous l'avons présenté à la rump session de *FSE 2018* en décrivant une attaque sur le chiffrement à flot LILLE [BIM18] en réduisant le coût de la recherche exhaustive d'un facteur 2^5.

Dans le modèle à clef connue, introduit par Lars Knudsen et Vincent Rijmen [KR07], on suppose que l'attaquant.e connaît la clef et essaie de trouver des propriétés distinguantes sur la permutation engendrée par cette clef. Même si le modèle parait extrêmement irréaliste d'un point de vue pratique, si le distingueur qui en découle est suffisamment puissant et fonctionne pour un grand nombre de clefs, il se pourrait qu'une attaque concrète en découle.

Finalement, la cryptanalyse théorique des primitives cryptographiques met en avant des failles pouvant être exploitées en pratique dans la suite et pouvant avoir des répercussions bien plus importantes, comme ont pu par exemple le montrer Karthikeyan Bhargavan et Gaëtan Leurent dans [BL16] ou encore Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini et Yarik Markov avec la collision sur la fonction de hachage SHA-1 [SBK+17]. Ainsi, la cryptanalyse des algorithmes est largement nécessaire, à la fois pour la compréhension fondamentale

de ce qui est possible de faire et de ce qui ne l'est pas, mais aussi pour mettre en évidence certaines failles pouvant avoir des répercussions concrètes plus tard, même si les coûts des attaques peuvent parfois paraître démesurés.

1.5 La cryptographie aujourd'hui

Nous disposons actuellement d'outils relativement solides en cryptographie. Cependant, le coût d'implémentation de nos algorithmes peut parfois être trop grand au regard de certaines applications qui ont des contraintes extrêmes.

Depuis une dizaine d'années, l'apparition de nouvelles applications et la multiplication des objets connectés ont imposé aux algorithmes de chiffrement de nouvelles contraintes portant par exemple sur la consommation d'énergie ou la taille du circuit implémentant l'algorithme. La conception de chiffrements à bas coût est donc actuellement une voie de recherche importante qui répond à une demande industrielle pressante. Les travaux sur la cryptographie légère devraient notamment aboutir à un processus de standardisation qui sera probablement initié par le NIST (National Institute of Standards and Technology) en fin d'année 2018. Les travaux mentionnés dans le chapitre 3 permettent de répondre à certaines attaques existant sur certains types de chiffrement à bas coût.

Nous pouvons aussi mentionner les algorithmes de chiffrement conçus pour être combinés avec des systèmes asymétriques de chiffrement complètement homomorphe, comme c'est le cas du chiffrement FLIP que nous cryptanalysons au chapitre 5.

L'intérêt du chiffrement authentifié (qui assure l'authenticité des messages) a grandi dans la communauté cryptographique ces dernières années, avec pour conséquence la tenue de la compétition CAESAR, ayant pour but de sélectionner les meilleurs algorithmes de chiffrement authentifiés, proposés par l'ensemble de la communauté cryptographique internationale. Cette compétition est censée se terminer par l'annonce des vainqueurs à la fin de l'année. Dans cette direction, nous avons analysé le chiffrement Ketje (chapitre 8), toujours en lice lors du troisième tour de la compétition CAESAR et nous avons participé à un travail collectif [AEL+18] de cryptanalyse du candidat MORUS [WH17].

Comme nous allons le voir tout au long de ce document, il y a aussi un très grand nombre de questions mathématiques sous-jacentes aux cryptanalyses, qui restent difficiles et demandent des réponses. Une autre direction de la cryptographie se trouve peut-être ici : recommencer à résoudre des problèmes fondamentaux de mathématiques discrètes, qui nous permettraient de bien mieux cerner les difficultés auxquelles le monde cryptographique fait face, afin d'assurer la sécurité des primitives cryptographiques plus facilement et agrémenter significativement notre connaissance intrinsèque des structures mathématiques sous-jacentes.

Chapitre 2

Préliminaires mathématiques

Ce chapitre rappelle quelques notions fondamentales nécessaires à la compréhension du document, principalement sur les fonctions booléennes appliquées à la cryptographie ainsi que sur les corps finis de caractéristique 2 dont nous avons besoin. Pour le.a lecteur.rice averti.e et connaisseur.se, il est tout à fait possible de passer directement au chapitre suivant et de rentrer dans le corps du document.

2.1 L'espace vectoriel $\mathbb{F}_2^n$

Comme les systèmes cryptographiques sont voués à être déployés et implémentés sur des outils informatiques, nous travaillons généralement sur l'espace vectoriel $\mathbb{F}_2^n$. Dans ces conditions, nous utilisons les opérations classiques AND ("et") et XOR ("ou exclusif"), qui sont des opérations sur des bits (des éléments du corps $\mathbb{F}_2$). Les opérations XOR et le AND sont données par les tables de vérité suivantes.

x_1	x_2	AND(x_1, x_2)
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	XOR(x_1, x_2)
0	0	0
0	1	1
1	0	1
1	1	0

Le XOR sera noté $+$, car on travaille sur $\mathbb{F}_2$, et $\mathrm{AND}(x_1, x_2)$ sera contracté par $x_1 x_2$.

Pour $x, y \in \mathbb{F}_2^n$, on note aussi $x \cdot y$ le produit scalaire usuel entre x et y, *i.e.* $x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$ où les x_i (respectivement y_i) sont les coordonnées de x (respectivement de y), *i.e.* les bits de x (respectivement de y).

2.2 Les corps finis de caractéristique 2

Nous verrons dans cette thèse qu'il est parfois pratique d'identifier les éléments de $\mathbb{F}_2^n$ à des éléments de $\mathbb{F}_{2^n}$, le corps fini à 2^n éléments. Nous rappelons donc quelques notions utiles sur les corps finis de caractéristique 2.

Définition 2.1 (Endomorphisme de Frobenius)**.** *L'application de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_{2^n}$ qui à X associe X^2 est un endomorphisme de corps.*

Plus précisément, en caractéristique 2, on a, pour tout $a, b \in \mathbb{F}_{2^n}$, $(a+b)^2 = a^2 + b^2$.

Nous savons que les corps finis sont déterminés de manière unique, à isomorphisme près, par leur cardinalité, qui doit être une puissance d'un nombre premier. Le corps premier de cardinal 2 est noté classiquement $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$.

Définition 2.2 (Fonction trace)**.** *Pour chaque entier d et r qui divise d, on définit classiquement la fonction trace de $\mathbb{F}_{2^d}$ dans $\mathbb{F}_{2^r}$, notée par $\mathsf{Tr}_r^d(\cdot)$ de la manière suivante :*

$$\mathsf{Tr}_r^d(x) = \sum_{i=0}^{\frac{d}{r}-1} x^{2^{ir}} = x + x^{2^r} + x^{2^{2r}} + \cdots + x^{2^{d-r}}$$

Lorsque le contexte sera clair, les indices d et r pourront être omis. De plus, la fonction Tr_r^d est une forme linéaire surjective sur $\mathbb{F}_{2^r}$, ce qui nous permet de considérer que le produit scalaire usuel peut être aussi décrit à partir de la fonction trace.

Définition 2.3 (Indicatrice d'Euler)**.** *Soit k un entier, alors $\phi(k)$ est le nombre d'entiers x compris entre 1 et k tels que x est premier avec k. On appelle alors $\phi(k)$ l'indicatrice d'Euler de k.*

Comme on travaille sur un corps fini, on sait que $\mathbb{F}_{2^n}^*$ est un groupe pour la multiplication. Ce groupe est d'ordre $(2^n - 1)$, et le nombre de générateurs de ce groupe est donné par l'indicatrice d'Euler.

Définition 2.4 (Éléments primitifs)**.** *Les générateurs du groupe multiplicatif $\mathbb{F}_{2^n}^*$ sont appelés éléments primitifs.*

Un polynôme P est dit primitif si l'anneau quotient $\mathbb{F}_2[X]/P(X)$ a une structure de corps. Il y a alors un parallèle naturel entre les polynômes primitifs et les éléments primitifs du corps. Plus précisément, soit P un polynôme de

degré n primitif. Soit α une racine de P, alors α est un générateur du groupe multiplicatif $\mathbb{F}_{2^n}^*$. Dans ces conditions, les éléments du corps fini peuvent être identifiés à $\{0, 1 = \alpha^0, \alpha, \alpha^2, \alpha^3, \alpha^4, \ldots, \alpha^{2^n-2}\}$.

Par exemple, le polynôme $X^6 + X + 1$ est primitif et nous pouvons alors définir $\mathbb{F}_{2^6}$ comme extension du corps $\mathbb{F}_2$ à l'aide de ce polynôme.

Avec ces notations, nous savons que l'ensemble des éléments primitifs est $\{\alpha^i | \mathsf{pgcd}(i, 2^n - 1) = 1\}$. P étant de degré n, il admet donc n racines distinctes dans le corps fini $\mathbb{F}_{2^n}$ défini comme anneau des polynômes quotienté par P. Les n racines de P sont conjuguées entre elles par l'application de l'endomorphisme de Frobenius, *i.e.* les racines de P sont $\alpha, \alpha^2, \alpha^4, \alpha^8, \ldots, \alpha^{2^{n-1}}$. L'application du Frobenius revient alors à multiplier par 2 modulo (2^n-1) les entiers correspondant aux exposants de α.

Définition 2.5 (Classes cyclotomiques)**.** *L'opération qui consiste à multiplier par* 2 *les entiers modulo* $(2^n - 1)$ *divise les entiers inférieurs à* $(2^n - 1)$ *en sous-ensembles appelés classes cyclotomiques modulo* $(2^n - 1)$*. La classe cyclotomique contenant* k *est donc définie par*

$$c(k) := \{k2^i \mod (2^n - 1) : 0 \leq k \leq m_k - 1\}$$

où $m_k = \min\{j | k2^j \equiv k \mod (2^n - 1)\}\}$ *. De plus, le plus petit entier de chaque classe cyclotomique est appelé le représentant de la classe cyclotomique. L'ensemble des représentants des classes cyclotomiques est noté* Γ*.*

Si k est premier avec l'ordre du groupe multiplicatif $\mathbb{F}_{2^n}^*$, *i.e.* avec $(2^n - 1)$, alors la taille de sa classe cyclotomique est nécessairement n. Si k n'est pas premier avec $(2^n - 1)$, alors la taille de sa classe cyclotomique peut être n ou un diviseur de n. Dans le cas où la taille de la classe cyclotomique de k divise strictement n, cela signifie que α^k est dans un sous-corps de $\mathbb{F}_{2^n}$.

Par exemple, pour le corps fini $\mathbb{F}_{2^6}$, les classes cyclotomiques sont :

— $c(0) := \{0\}$;
— $c(1) := \{1, 2, 4, 8, 16, 32\}$;
— $c(3) := \{3, 6, 12, 24, 48, 33\}$;
— $c(5) := \{5, 10, 20, 40, 17, 34\}$;
— $c(7) := \{7, 14, 28, 56, 49, 35\}$;
— $c(9) := \{9, 18, 36\}$;
— $c(11) := \{11, 22, 44, 25, 50, 37\}$;
— $c(13) := \{13, 26, 52, 41, 19, 38\}$;
— $c(15) := \{15, 30, 60, 57, 51, 39\}$;
— $c(21) := \{21, 42\}$;
— $c(23) := \{23, 46, 29, 58, 53, 43\}$;
— $c(27) := \{27, 54, 45\}$;
— $c(31) := \{31, 62, 61, 59, 55, 47\}$.

L'ensemble des représentants des classes cyclotomiques modulo 63 est donc $\Gamma := \{0, 1, 3, 5, 7, 9, 11, 13, 15, 21, 23, 27, 31\}$. La classe cyclotomique $\{0\}$ correspond à l'élément neutre pour la multiplication. Avec l'élement neutre pour l'addition (0), cette classe forme le sous-groupe premier, *i.e.* $\mathbb{F}_2$. Si on y rajoute la classe de longueur 2 on trouve le sous-corps $\mathbb{F}_{2^2}$. Si en revanche on y rajoute les deux classes cyclotomiques de taille 3, on retrouve le sous-corps $\mathbb{F}_{2^3}$.

2.3 Fonctions booléennes

Définition 2.6 (Fonctions booléennes)**.** *Une fonction booléenne à n variables est une fonction de $\mathbb{F}_2^n$ à valeurs dans $\mathbb{F}_2$. L'ensemble des fonctions booléennes à n variables est noté $\mathcal{B}_n$.*

Définition 2.7 (Vecteur des valeurs)**.** *Le vecteur des valeurs d'une fonction booléenne f à n variables est le vecteur binaire v_f de longueur 2^n composé des valeurs de la fonction : $f(x)$ pour $x \in \mathbb{F}_2^n$.*

Par exemple, la table 2.1 donne le vecteur des valeurs d'une fonction à 3 variables.

x_1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1
$f_1(x_1, x_2, x_3)$	0	0	1	1	0	1	1	1

Table 2.1 – Table de vérité d'une fonction à 3 variables.

Ainsi, une fonction booléenne est entièrement définie par sa table de vérité, de taille 2^n. Le nombre de fonctions booléennes à n variables est donc 2^{2^n}.

Nous définissons aussi le support d'une fonction booléenne.

Définition 2.8 (Support)**.** *Soit f une fonction booléenne à n variables, alors le support de f, noté $\mathsf{supp}(f)$ est défini par*

$$\mathsf{supp}(f) := \{x | f(x) = 1\} .$$

Dans notre exemple, on a $\mathsf{supp}(f_1) = \{010, 110, 101, 011, 111\}$.

De plus, les fonctions booléennes se représentent sous forme d'un polynôme multivarié à n variables. On utilise alors la notation suivante pour désigner les monômes.

Notation 2.9 (Monômes)**.** *Pour tout $u = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$, x^u désigne le monôme dans l'anneau $\mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 + x_1, \ldots, x_n^2 + x_n)$ défini par*

$$\prod_{i=1}^{n} x_i^{u_i} .$$

Ces monômes forment une famille libre et génératrice de l'espace des fonctions booléennes, et nous pouvons alors représenter une fonction booléenne comme un polynôme multivarié : la *forme algébrique normale*.

Théorème 2.10 (Forme algébrique normale)**.** *Soit f une fonction booléenne à n variables. Alors il existe un unique polynôme multivarié dans l'anneau $\mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 + x_1, \ldots, x_n^2 + x_n)$ tel que*

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_u x^u \ , \ \textit{avec } a_u \in \mathbb{F}_2 \ .$$

Ce polynôme multivarié est appelé la forme algébrique normale (ANF) de f.

De plus, les coefficients de l'ANF et les valeurs de f satisfont

$$a_u = \sum_{x \preceq u} f(x) \qquad \textit{et} \qquad f(u) = \sum_{x \preceq u} a_x,$$

où les sommes sont calculées sur $\mathbb{F}_2$ et $x \preceq y$ si et seulement si $x_i \leq y_i$ pour tout $1 \leq i \leq n$.

En reprenant notre exemple de la table 2.1, on a $f_1(x_1, x_2, x_3) = x_2 + x_1x_3 + x_1x_2x_3$.

Définition 2.11 (Poids de Hamming)**.** *Le poids de Hamming d'un élément de $\mathbb{F}_2^n$ (respectivement d'une fonction booléenne) est le nombre de* 1 *dans sa représentation binaire (respectivement dans son vecteur des valeurs). On note le poids de Hamming $\mathsf{w_H}(\cdot)$.*

Alors, le degré algébrique d'une fonction booléenne est défini comme le degré du plus grand monôme dans sa forme algébrique normale, *i.e.*,

$$\deg f = \max_{u \in \mathbb{F}_2^n : a_u \neq 0} \mathsf{w_H}(u) \ .$$

Le degré algébrique de notre fonction f_1 vaut 3.

Nous avons déjà vu que nous pouvons représenter les fonctions booléennes soit en utilisant le vecteur des valeurs soit en utilisant la forme algébrique normale. Il existe une troisième manière de représenter les fonctions booléennes. En effet, comme les fonctions booléennes prennent leurs entrées dans l'espace $\mathbb{F}_2^n$, on peut considérer que ces entrées sont à valeur dans le corps fini $\mathbb{F}_{2^n}$. Alors, dans ces conditions, les fonctions booléennes peuvent être représentées à l'aide de la fonction trace.

Proposition 2.12 (Représentation Trace)**.** *Pour toute fonction booléenne à n variables, nous pouvons écrire f de la forme*

$$f(x) = \sum_{k \in \Gamma(n)} \mathsf{Tr}_{m_k}^n(\lambda_k x^k) + \varepsilon_1 + \varepsilon_2 x^{2^n - 1}$$

où Γ est l'ensemble des représentants des classes cyclotomiques modulo $(2^n - 1)$ et m_k est la taille de la classe cyclotomique de k, et $\lambda_k \in \mathbb{F}_{2_k^m}$ et $\varepsilon_1, \varepsilon_2 \in \mathbb{F}_2$.

Les restrictions sur λ_k ainsi que sur le fait de regrouper les termes en fonction des classes cyclotomiques permettent d'assurer l'unicité de cette représentation. ε_1 correspond au coefficient constant dans la forme algébrique normale, *i.e.* le monôme de degré 0 et ε_2 est obtenu en faisant la somme de tous les monômes de degré 1 ou plus, *i.e.* $x^{2^n-1} = 1$ pour tout $x \neq 0$. Ainsi si $f(0) = 0$, alors $\varepsilon_1 = 0$ et si la fonction est équilibrée, alors nécessairement $\varepsilon_2 = 0$.

En reprenant notre exemple, on a $f_1(x) = \mathsf{Tr}_1^3((a+1)x^3) + \mathsf{Tr}_1^3((a^2+a)x) + x^7$.

2.4 Propriétés cryptographiques

2.4.1 Transformée de Walsh

En cryptographie, on s'intéresse à générer des suites ne pouvant pas être distinguées d'une suite aléatoire. Pour cela, il est notamment nécessaire que les suites que l'on construit ne soient pas *biaisées*, *i.e.* qu'elles soient équilibrées. Donc, nous définissons la notion de biais d'une fonction booléenne.

Définition 2.13 (Biais, corrélation)**.** *Soit f une fonction booléenne à n variables, alors le biais est défini par*

$$\varepsilon = \frac{\mathcal{E}(f)}{2^n}$$

où

$$\mathcal{E}(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} = 2^n - 2\mathsf{w_H}(f) \ .$$

La fonction f_1 n'est pas équilibrée. Plus précisément son biais vaut $\varepsilon = \frac{\mathcal{E}(f_1)}{2^n} = -0.25$.

Dans la littérature, la notion de corrélation est souvent définie comme deux fois le biais. De manière générale, si l'on a deux suites binaires s et σ, alors pour quantifier la corrélation entre ces deux suites, nous nous intéresserons principalement à la quantité

$$\sum_t (-1)^{s_t + \sigma_t} \ .$$

Cette notion est naturellement liée à la transformée de Walsh, qui est un bon outil pour étudier les fonctions booléennes et mesurer la distance aux fonctions affines.

Définition 2.14 (Transformée de Walsh)**.** *Soit f une fonction booléenne à n variables. La transformée de Walsh de la fonction f est la fonction*

$$\begin{array}{rcl} \mathbb{F}_2^n & \to & \mathbb{Z} \\ a & \mapsto & \mathcal{E}(f + \phi_a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + a \cdot x} \end{array}$$

où ϕ_a est la fonction booléenne linéaire définie par $\phi_a(x) = a \cdot x$. La valeur $\mathcal{E}(f + \phi_a)$ est appelée le coefficient de Walsh de f au point a et l'ensemble des coefficients de la transformée de Walsh est appelé le spectre de Walsh.

Par exemple, le spectre de Walsh de f_1 est le suivant :

a	000	001	010	011	100	101	110	111
$\mathcal{E}(f_1 + \phi_a)$	-2	2	6	2	2	-2	2	-2

où, quand $a = 001$, $\phi_a(x_1, x_2, x_3) = x_1$,...

Calculer la transformée Walsh se fait en $n2^n$ opérations. De plus, cette transformée possède les propriétés mathématiques d'une transformée de Fourier. Par exemple, c'est une involution, à un facteur constant près.

Proposition 2.15. *Soit f une fonction booléenne à n variables. Pour tout $b \in \mathbb{F}_2^n$, on a*

$$\sum_{a \in \mathbb{F}_2^n} (-1)^{a \cdot b} \mathcal{E}(f + \phi_a) = 2^n (-1)^{f(b)} .$$

Nous avons de plus l'identité de Parseval :

$$\sum_{a \in \mathbb{F}_2^n} [\mathcal{E}(f + \phi_a)]^2 = 2^{2n} .$$

Comme la cryptanalyse linéaire mais aussi les attaques par corrélation exploitent l'existence d'approximations linéaires, il est impératif de considérer la distance aux fonctions de degré 1. Nous définissons alors la linéarité, au moyen du spectre de Walsh.

Définition 2.16 (Linéarité d'une fonction booléenne)**.** *Soit f une fonction booléenne à n variables. La linéarité de f est le plus grand coefficient de Walsh en valeur absolue, i.e.,*

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

Par exemple, la linéarité de f_1 vaut 6.

Nous définissons aussi la non-linéarité, qui donne la distance (de Hamming) à toutes les fonctions linéaires :

Définition 2.17 (Non-linéarité d'une fonction booléenne)**.** *Soit f une fonction booléenne à n variables. La non-linéarité de f est définie par*

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

Par exemple, la non-linéarité de f_1 vaut 1.

Naturellement, l'identité de Parseval implique que la linéarité optimale vaut $2^{\frac{n}{2}}$, et est atteinte pour les fonctions dites *courbes* [1] qui n'existent que lorsque

1. "Bent functions" en anglais.

n est pair et qui ne sont pas équilibrées. Lorsque n est impair, la borne n'est pas atteinte, puisque $2^{\frac{n}{2}}$ n'est pas un entier, et nous ne connaissons pas la non-linéarité optimale pour un nombre impair de variables supérieur ou égal à 9. De la même manière, les fonctions courbes ne peuvent pas être équilibrées, et la linéarité optimale pour une fonction équilibrée n'est pas connue dès que $n \geq 8$.

2.4.2 Résilience

Certaines attaques de type "diviser pour mieux régner" [Sie85] exploitent elles, l'existence d'une approximation non-nécessairement linéaire, mais avec moins de variables. Alors, il est nécessaire d'utiliser des fonctions ayant un grand ordre de résilience, au sens de la définition suivante.

Définition 2.18 (Ordre d'immunité aux corrélations [Sie84])**.** *Une fonction booléenne f à n variables est sans corrélation d'ordre t si le biais de f reste inchangé en fixant t variables quelconques en entrée. De plus, une fonction équilibrée et sans corrélation d'ordre t est appelée résiliente d'ordre t.*

En plus que d'avoir un ordre de résilience suffisamment élevé pour éviter certaines attaques, il est aussi nécessaire que les fonctions booléennes utilisées comme blocs de construction de chiffrements aient un degré algébrique suffisamment élevé. Or, il y a un compromis entre le degré algébrique et l'ordre de résilience.

Proposition 2.19. *[Sie84] Soit f une fonction booléenne à n variables. Alors son ordre d'immunité aux corrélations satisfait*

$$t + \deg(f) \leq n \ .$$

Si de plus f est équilibrée et $t < n - 1$, alors

$$t + \deg(f) \leq n - 1 \ .$$

2.4.3 Immunité algébrique

Alors que le degré algébrique des fonctions booléennes était considéré comme un bon critère pour résister aux attaques algébriques, il est apparu en 2003 [Cou03] que ce n'était pas le critère pertinent pour résister aux attaques algébriques, et que le critère approprié était l'*immunité algébrique*, au sens de la définition suivante.

Définition 2.20 (Immunité algébrique)**.** *Soit f une fonction booléenne à n variables, alors l'immunité algébrique de f, notée $\mathsf{AI}(f)$ est définie par*

$$\mathsf{AI}(f) = \min\{\deg g, g \in \mathcal{B}_n, g \neq 0, gf = 0 \text{ ou } g(f+1) = 0\} \ .$$

L'idée sous-jacente à ce critère est qu'une équation de haut degré peut en "cacher une autre". Prenons par exemple notre fonction f_1, qui est de degré algébrique 3. On a

$$f(x_1, x_2, x_3) = x_2 + x_1x_3 + x_1x_2x_3 \ .$$

Supposons de plus que l'on a accès à une valeur en sortie de cette fonction, par exemple 1, et que l'on cherche à établir une relation entre les valeurs en entrées. A priori, l'équation

$$x_2 + x_1x_3 + x_1x_2x_3 = 1 \tag{2.1}$$

est de degré 3. Or, la relation

$$(1 + x_1 + x_2 + x_1x_2)(x_2 + x_1x_3 + x_1x_2x_3) = 0$$

est satisfaite pour tout triplet $(x_1, x_2, x_3) \in \mathbb{F}_2^3$. Donc, l'équation (2.1) implique

$$0 = 1 + x_1 + x_2 + x_1x_2$$

qui est une relation de degré 2.

Il est bien connu que l'immunité algébrique optimale est $\lceil n/2 \rceil$ quand n est le nombre de variables de la fonction booléenne [CM03].

2.4.4 Construction en somme directe

Une fois que nous avons défini les critères cryptographiques, il est nécessaire de construire des fonctions ayant de bonnes propriétés. Une manière relativement simple et que nous utiliserons parfois dans le document est de construire des fonctions booléennes à l'aide de la somme directe, en ajoutant deux fonctions dont les variables sont indépendantes.

Définition 2.21 (Somme directe)**.** *Soit f une fonction booléenne à n variables et g une fonction booléenne à m variables, alors la somme directe de f et de g est la fonction booléenne F à $n + m$ variables définie par*

$$F(x, y) = f(x) + g(y)$$

où $x \in \mathbb{F}_2^n$ et $y \in \mathbb{F}_2^m$.

Cette construction permet d'assurer que la fonction F hérite à la fois des bonnes propriétés de f et des bonnes propriétés de g pour la non-linéarité et l'immunité algébrique. Par exemple, si f ou g est équilibrée, alors F est équilibrée. Plus généralement, on a les propriétés suivantes [Car07, MJSC16]. Soit F la somme directe de f à n variables et g à m variables, alors

- $\mathsf{NL}(F) = 2^m\mathsf{NL}(g) + 2^n\mathsf{NL}(g) - 2\mathsf{NL}(f)\mathsf{NL}(g)$;
- $\mathsf{AI}(f) + \mathsf{AI}(g) \geq \mathsf{AI}(F) \geq \max(\mathsf{AI}(f), \mathsf{AI}(g))$;
- $\mathsf{res}(F) = \mathsf{res}(f) + \mathsf{res}(g) + 1$,

où res désigne l'ordre de résilience.

Un grand nombre de questions restent ouvertes sur les fonctions booléennes, au regard de ces propriétés cryptographiques. Il existe pléthore de résultats, de bornes et de constructions de fonctions booléennes appliquées à la cryptographie. Pour le.a lecteur.rice qui s'intéresse au sujet, nous renvoyons au livre de C. Carlet : *Boolean Functions for Cryptography and Error Correcting Codes* [Car07], qui rassemble un grand nombre de résultats connus.

Nous verrons dans ce document comment revisiter plusieurs propriétés cryptographiques au regard des diverses faiblesses que nous allons mettre en évidence sur certains types de systèmes.

Chapitre 3

Attaques par invariant sur les chiffrements par bloc

Dans ce chapitre, nous investiguons un nouveau type d'attaque sur les chiffrements par bloc appelé *attaques par invariant*, apparu récemment sur un certain nombre de chiffrements par bloc légers. Les résultats obtenus dans ce chapitre ont été publiés à *CRYPTO* en 2017 dans [BCLR17].

3.1 Les chiffrements par bloc

Les chiffrements par bloc forment une famille d'algorithmes de chiffrement symétrique classiques et très largement utilisés actuellement. L'AES (Advanced Encryption Standard) est le standard actuel (FIPS 197) de chiffrement par bloc [DR02] et est utilisé dans de nombreuses applications. Pour transformer un chiffrement par bloc en un schéma générique qui permet de traiter des messages de longueur arbitraire, ces derniers sont découpés en plusieurs parties (blocs) de taille identique, puis ces blocs sont combinés d'une certaine manière au moyen d'un mode opératoire.

Les modes opératoires. En cryptographie, les modes opératoires décrivent la manière d'enchaîner les appels au chiffrement par bloc pour chiffrer un message de taille arbitraire. Selon le contexte, certains modes opératoires sont plus adaptés que d'autres. On peut citer parmi les plus utilisés, OFB (OutPut FeedBack), CFB (Cipher FeedBack) ou CBC (Cipher Feedback) qui sont standardisés par le NIST [1] (FIPS 81).

Habituellement, la sécurité des modes opératoires est analysée indépendamment des algorithmes de chiffrement par bloc. Ainsi, certains modes opératoires sont vulnérables à des attaques intrinsèques, qui sont à prendre en compte lors

1. National Institute of Standards and Technology

du choix dudit mode opératoire. Finalement, la sécurité globale du schéma de chiffrement utilisé dépend à la fois du chiffrement par bloc et du mode opératoire.

3.1.1 Conception des chiffrements par bloc

Un chiffrement par bloc est défini de la manière suivante.

Définition 3.1 (Chiffrement par bloc)**.** *Un chiffrement par bloc E de taille de clef κ opérant sur des blocs de n bits est défini comme une famille de 2^κ permutations de $\mathbb{F}_2^n$ notées $(E_k)_{k\in\mathbb{F}_2^\kappa}$.*

En d'autres termes, à chaque clef secrète $k \in \mathbb{F}_2^\kappa$, on associe une permutation qui est notée E_k.

D'une manière générale, le concepteur d'un chiffrement par bloc doit faire en sorte qu'il soit difficile de différencier chaque permutation E_k pour $k \in \mathbb{F}_2^\kappa$ d'une permutation aléatoire. Les principes qui régissent la construction d'un chiffrement par bloc sont les notions de *confusion* et de *diffusion* introduits par C. Shannon [Sha49]. Pour des raisons évidentes d'implémentation ainsi que pour éviter certaines vulnérabilités, le cryptologue travaille principalement sur des tailles de bloc de 64, 128 ou 256 bits. Il est donc hors de portée de décrire les permutations E_k avec leur table de vérité et ce pour chaque clef secrète k. De plus, chaque permutation est "compliquée" [2] puisqu'elle doit être indistinguable d'une permutation choisie aléatoirement.

Une solution à ce problème consiste à définir une permutation facile à décrire ayant des propriétés distinctives, puis de l'itérer, afin de rendre l'algorithme suffisamment compliqué. Cette permutation itérée est appelée la fonction de tour, et le nombre de tours correspond au nombre d'itérations. À chaque itération, la fonction de tour dépend d'une clef secrète dont la valeur change avec l'indice du tour. L'ensemble de ces clefs de tour est dérivé de la clef-maître par un algorithme de cadencement de clef.

Actuellement, tous les chiffrements par bloc sont conçus sur ce modèle itératif. Celui-ci est décrit à la figure 3.1.

Les deux principales structures itératives sont les réseaux de Feistel et les réseaux de substitution-permutation (SPN - Substitution Permutation Network).

3.1.2 Les réseaux de Feistel

Le principe des réseaux de Feistel est de découper un bloc de taille fixe en deux parties (droite et gauche). Si X est un mot de $2n$ bits, on note X_g et X_d ses parties gauche et droite : $X = X_g||X_d$. On applique ensuite une fonction F_k non-nécessairement bijective sur la partie droite et le résultat d'un tour de Feistel est défini par $Y = X_d||F_k(X_d) \oplus X_g$. Ce schéma est décrit à la figure 3.2

Par construction, un tour de Feistel est toujours une permutation de $\mathbb{F}_2^n$ quel que soit le choix de la fonction F_k. En effet, il s'agit d'une involution à une permutation près des deux moitiés.

2. degré algébrique élevé, pas de structure particulière a priori visible,...

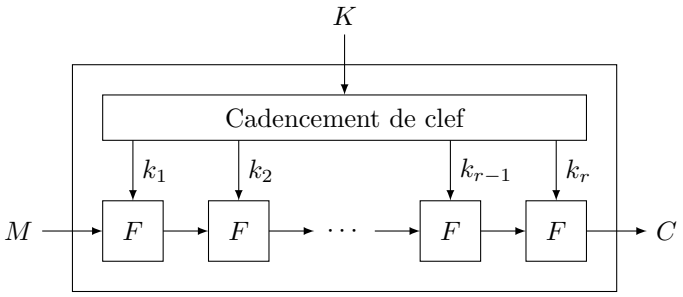

Figure 3.1 – Schéma d'un chiffrement par bloc E_K construit de manière itérative.

3.1.3 Les réseaux de substitution-permutation

Une autre façon de concevoir un chiffrement par bloc est d'utiliser un réseau de substitution-permutation, construction décrite à la figure 3.3. Ici, la fonction de tour consiste en l'addition d'une clef de tour, puis d'une permutation sans paramètre, définie par l'application en parallèle de petites substitutions opérant souvent sur 4 ou 8 bits appelées boîtes-S apportant la *confusion* au chiffrement. Cette permutation est ensuite composée avec une opération linéaire apportant la *diffusion*. Afin d'assurer la bijectivité du chiffrement, il faut que chaque élément soit inversible. Il existe de nombreux critères de sécurité sur l'application linéaire et sur les boîtes-S qui garantissent que le chiffrement résiste à certaines attaques. Ainsi, pour assurer une bonne diffusion, on utilise des applications linéaires définies par un code linéaire ayant une distance minimale et une distance duale élevée, par exemple un code MDS (Maximum Distance Separable). Par ailleurs, l'emploi de boîtes-S ayant une bonne uniformité différentielle (respectivement une bonne non-linéarité) assurera une certaine résistance aux attaques différentielles (respectivement linéaires).

3.1.4 Attaques connues sur les chiffrements par bloc.

Les deux principales cryptanalyses sur les chiffrements par bloc sont la cryptanalyse différentielle et la cryptanalyse linéaire. Pour la cryptanalyse différentielle, l'attaquant.e essaye de trouver un biais statistique dans la distribution de la différence entre deux chiffrés correspondant à des clairs ayant une différence fixée. Pour la cryptanalyse linéaire, l'attaquant.e essaye de trouver de bonnes approximations affines faisant intervenir les bits du texte clair et du texte chiffré.

À chaque attaque, on associe des critères cryptographiques garantissant la résistance à ce type de cryptanalyse. Par exemple, pour la cryptanalyse différentielle, le critère associé est d'avoir une faible uniformité différentielle, au sens de la définition 3.2 suivante.

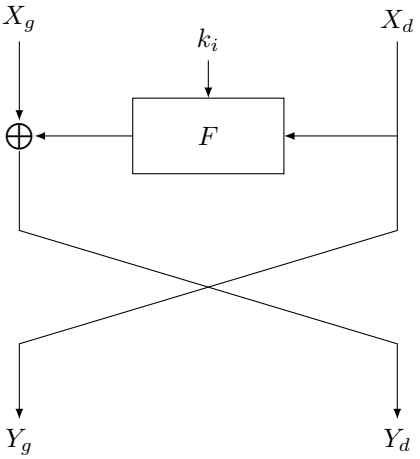

Figure 3.2 – Schéma d'un tour de Feistel où k_i la clef de tour, dérivée de la clef maître par le cadencement de clef.

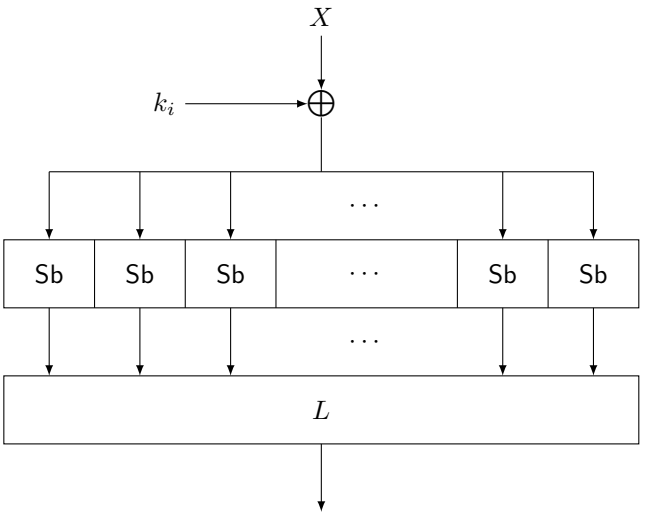

Figure 3.3 – Schéma d'un tour d'un réseau de substitution/permutation où S est la couche de substitution définie comme la concaténation des boîtes-S Sb et L la couche linéaire et k_i la clef de tour, dérivée de la clef maître par le cadencement de clef.

Définition 3.2 (Uniformité différentielle)**.** *Soit F une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$, l'uniformité différentielle $\delta(F)$ de F est la quantité*

$$\delta(F) = \max_{\substack{(a,b)\in\mathbb{F}_2^n\times\mathbb{F}_2^n \\ a\neq 0}} \#\{x \in \mathbb{F}_2^n, F(x+a) + F(x) = b\}$$

Cryptographie à bas coût. Dans un souci de réduction du coût d'implémentation des algorithmes cryptographiques, certaines composantes des chiffrements par bloc sont réduites à leur substantifique moëlle : on réduit le nombre de XOR dans la couche linéaire [3] ; on utilise des boîtes-S sur 4 bits plutôt que sur 8 bits ; on réduit le cadencement de clef à la seule addition d'une constante de tour (souvent très creuse). En revanche, afin de continuer à assurer une certaine sécurité, on augmente parfois le nombre de tours. Nous pouvons citer par exemple les algorithmes de chiffrement LED [GPPR11], PRESENT [BKL+07], Simon & Speck [BSS+13],...

Parmi tous ces chiffrements à bas coût relativement récents [4], beaucoup se sont retrouvés confrontés à un nouveau type d'attaque, sans lien a priori avec les attaques différentielles ou linéaires : les attaques par sous-espace invariant [LAAZ11, LMR15] ainsi que leur généralisation proposée par Todo, Leander et Sasaki appelée "attaque par invariants non-linéaires" [TLS16]. Ces attaques ont mis en évidence des vulnérabilités dans bon nombre de ces chiffrements, notamment PRINTcipher [KLPR10, LAAZ11], Midori64 [BBI+15, GJN+16, TLS16], iSCREAM [HCJ02, LMR15] et SCREAM [HCJ02, TLS16], NORX v2.0 [AJN14, CFG+17], la famille incluant Simpira v1 [Røn16, GM17] et Haraka v.0 [KLMR16, Jea16]. Dans la suite, nous appellerons ces deux attaques sous la terminologie attaques par invariant. Avec Christof Beierle, Anne Canteaut et Gregor Leander, nous avons expliqué pourquoi certains chiffrements étaient vulnérables aux attaques par invariant, et avons montré pour la première fois comment les constantes de tour devaient être choisies en fonction de la couche linéaire afin de s'en prémunir.

Organisation du chapitre. La première section de ce chapitre est consacrée à des notions classiques nécessaires à la compréhension de notre travail, ainsi qu'à certaines observations simples sur les invariants. Dans la deuxième section, nous nous intéressons à prouver l'absence (ou l'existence) d'invariants à la fois pour la couche linéaire et la couche de substitution dans un chiffrement de type SPN. Nous appliquons cette technique générale à plusieurs chiffrements à bas coût proposés récemment. Enfin, dans la dernière section, nous montrons comment choisir les constantes afin de se prémunir contre les attaques par invariant.

3. Trouver des matrices MDS avec le plus petit nombre de XOR est un problème difficile et d'actualité [KLSW17, DL18]

4. Une vue d'ensemble des chiffrements à bas coût, accompagnée des meilleures attaques connues sur chacun est disponible sur le site Web `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`

3.2 Préliminaires et premières observations

3.2.1 Structures linéaires

On note $\mathcal{B}_n$ l'ensemble des fonctions booléennes à n variables. Les fonctions booléennes constantes sont notées $\mathbf{0}$ et $\mathbf{1}$. La fonction dérivée de $f \in \mathcal{B}_n$ dans la direction $\alpha \in \mathbb{F}_2^n$ est notée $\Delta_\alpha f$ et est définie par $\Delta_\alpha f := x \mapsto f(x+\alpha) + f(x)$.

Définition 3.3 (Structures linéaires)**.** *Un élément $\alpha \in \mathbb{F}_2^n$ est une structure linéaire de $f \in \mathcal{B}_n$ si $\Delta_\alpha f$ est une fonction booléenne constante.*

Proposition 3.4 (Structures linéaires)**.** *L'ensemble des structures linéaires d'une fonction booléenne f forme un sous-espace vectoriel de $\mathbb{F}_2^n$ et est appelé l'espace linéaire de f. On note $\mathsf{LS}(f)$ l'ensemble des structures linéaires de f :*

$$\mathsf{LS}(f) := \{\alpha \in \mathbb{F}_2^n | \Delta_\alpha f = \boldsymbol{c},\ \boldsymbol{c} \in \{\boldsymbol{0}, \boldsymbol{1}\}\}$$

3.2.2 Principe des attaques par invariant

On considère un chiffrement par bloc E qui opère sur des blocs de taille n et dont la taille de la clef est κ :

$$E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n \tag{3.1}$$

$$(x, k) \mapsto E_k(x) \tag{3.2}$$

L'idée de l'attaque par invariant est d'identifier un sous-ensemble non trivial A de $\mathbb{F}_2^n$ tel que la partition $(A, \mathbb{F}_2^n \backslash A)$ est préservée par l'application du chiffrement pour un nombre non-négligeable de clefs k : on cherche donc $A \subseteq \mathbb{F}_2^n$ tel que $E_k(A) = A$ ou $E_k(A) = \mathbb{F}_2^n \backslash A$.

Définition 3.5 (Sous-ensemble invariant)**.** *Soit F une permutation de $\mathbb{F}_2^n$ et $A \subset \mathbb{F}_2^n$. On dit que A est un invariant pour F si la partition $(A, \mathbb{F}_2^n \backslash A)$ est préservée par F.*

Le cas particulier où A est un sous-espace affine correspond aux attaques par sous-espaces invariants.

Définition 3.6 (Invariants triviaux)**.** *Les invariants triviaux sont l'ensemble vide $\emptyset$ et l'espace tout entier $\mathbb{F}_2^n$.*

Les clefs pour lesquelles A est un invariant pour E_k sont donc des clefs faibles du chiffrement, au regard des attaques par invariant. En effet, si une clef faible k_0 est utilisée, alors l'attaquant.e est capable de distinguer la permutation E_{k_0} d'une permutation aléatoire, en exploitant l'existence d'un invariant pour cette clef. Dit autrement, ce distingueur peut aussi être utilisé sur E_k, afin de décider si la clef utilisée est faible ou pas, ce qui donne directement de l'information à l'attaquant.e sur la clef secrète. Si l'ensemble des clefs faibles est petit face au nombre total de clef, alors l'avantage de l'attaquant.e n'est pas si grand, puisque la probabilité qu'une clef faible soit utilisée est trop petite. Dans tous les cas, l'existence d'invariants n'est pas une propriété souhaitable pour un chiffrement par bloc.

3.2.3 Lien entre invariants et fonctions booléennes

Afin de travailler dans l'espace $\mathcal{B}_n$, on identifiera les sous-ensembles de $\mathbb{F}_2^n$ avec leurs fonctions indicatrices, et on désignera de manière équivalente par l'expression *invariant pour* F la fonction booléenne qui lui est associée. Chercher des sous-ensembles invariants pour E_k de $\mathbb{F}_2^n$ est donc équivalent à chercher toutes les fonctions booléennes g telles que $g + g \circ E_k$ est constante. En effet, si $A \subseteq \mathbb{F}_2^n$ est un invariant pour E_k, alors sa fonction indicatrice vérifie $g \circ E_k(x) = g(x)$ pour tout $x \in \mathbb{F}_2^n$ si $E_k(A) = A$ et $g \circ E_k(x) = g(x) + 1$ pour tout $x \in \mathbb{F}_2^n$ si $E_k(A) = \mathbb{F}_2^n \backslash A$.

Définition 3.7 (Invariant)**.** *Soit F une permutation de $\mathbb{F}_2^n$ et $g \in \mathcal{B}_n$, on dit que g est un invariant pour F si la fonction $g + g \circ F$ est constante.*

Définition 3.8 (Ensemble des invariants)**.** *Soit F une permutation de $\mathbb{F}_2^n$, alors l'ensemble des invariants $\mathcal{U}(F) \subseteq \mathcal{B}_n$ est défini par*

$$\mathcal{U}(F) := \{g \in \mathcal{B}_n | g + g \circ F \text{ est constante}\} .$$

Corollaire 3.9. *L'ensemble des invariants est un sous-espace vectoriel de $\mathcal{B}_n$.*

Démonstration. Les deux invariants triviaux ($\emptyset$ et $\mathbb{F}_2^n$) correspondent aux deux fonctions g constantes ($\mathbf{0}$ et $\mathbf{1}$). De plus, si f_1 et f_2 appartiennent à $\mathcal{U}(F)$, alors $f_1 + f_2$ appartient à $\mathcal{U}(F)$. En effet, soit c_1 et c_2 appartenant à $\mathbb{F}_2$, telles que pour tout $x \in \mathbb{F}_2^n$, $f_1(x) + f_1 \circ F(x) = c_1$ et $f_2(x) + f_2 \circ F(x) = c_2$, alors on a $(f_1 + f_2)(x) + (f_1 + f_2)(F(x)) = c_1 + c_2$ pour tout $x \in \mathbb{F}_2^n$ et est donc constante, donc $f_1 + f_2 \in \mathcal{U}(F)$. □

3.2.4 Décomposition en cycles des permutations

Quand il s'agit d'étudier les invariants d'une permutation, il est tout naturel de représenter cette dernière en fonction de ses cycles disjoints. En effet, n'importe quel sous-ensemble invariant ne peut être qu'une réunion de cycles disjoints [TLS16].

3.2.4.1 Représentation en cycles

Proposition 3.10 (Dimension de $\mathcal{U}(F)$ [TLS16])**.** *Soit F une permutation de $\mathbb{F}_2^n$ et $c_1 \circ c_2 \circ \cdots \circ c_m$ sa décomposition en cycles disjoints, alors*

$$\dim(\mathcal{U}(F)) = m .$$

Proposition 3.11 (Cycle de taille impaire [TLS16])**.** *Soit F une permutation de $\mathbb{F}_2^n$ possédant au moins un cycle de longueur impaire, alors tout invariant g pour F vérifie $g(x) = g \circ F(x)$ pour tout $x \in \mathbb{F}_2^n$.*

3.2.4.2 Le cas des involutions

Lorsque l'on a des contraintes très fortes sur la taille du circuit que l'on veut utiliser pour le chiffrement, il peut être très utile de concevoir un algorithme de chiffrement qui utilise le même circuit à la fois pour le chiffrement et pour le déchiffrement. Évidemment, le chiffrement n'est pas une involution sinon il y aurait un distingueur évident, notamment sur le nombre de points fixes moyen d'une involution [FS09], mais chaque composante du chiffrement peut être une involution. C'est le cas par exemple du chiffrement Noekeon proposé par Joan Daemen, Michaël Peeters, Gilles Van Assche et Vincent Rijmen [DPAR00].

On déduit directement de la proposition 3.10 le résultat suivant.

Proposition 3.12 (Invariants des involutions)**.** *Soit F une involution de $\mathbb{F}_2^n$, alors* $\dim(\mathcal{U}(F)) \geq 2^{n-1}$.

Ainsi, si l'on considère une boîte-S involutive opérant sur 4 bits, alors la dimension de l'espace des invariants vaut au moins $2^3 = 8$, et la dimension de l'espace des fonctions booléennes à 4 variables de degré inférieur ou égal à 2 vaut $1 + 4 + 6 = 11$. Comme la dimension de l'espace vectoriel des fonctions booléennes est 16, l'intersection des invariants et des fonctions quadratiques est nécessairement non-vide. Dans ces conditions, on peut affirmer que toute involution sur 4 bits admet au moins un invariant quadratique.

En revanche, dès que l'on considère des permutations sur $\mathbb{F}_2^{128}$ ou même $\mathbb{F}_2^{64}$, il devient très rapidement hors de portée de décrire l'espace des invariants sans faire certaines hypothèses fortes sur ces permutations. Même si l'on considère que nos permutations sont la concaténation de plusieurs petites permutations opérant sur 4 ou 8 bits (ce qui correspond à la couche de substitution), décrire les invariants en fonction des invariants des boîtes-S utilisées est ardu.

Comme il semble actuellement algorithmiquement infaisable d'étudier les invariants de toute la fonction de tour, il est tout naturel de regarder indépendamment les invariants de la couche linéaire et de la couche de substitution.

3.3 Prouver l'absence d'invariants sur des SPN

3.3.1 Propriétés générales

Dans tout ce qui suit, on considère des chiffrements par bloc conçus comme des réseaux de substitution-permutation (figure 3.3). Habituellement, les techniques utilisées pour les attaques par invariant consistent à chercher des sous-ensembles qui sont invariants à la fois par la couche de substitution et par la couche linéaire, c'est-à-dire par chaque composante du chiffrement séparément. L'algorithme décrit par Leander et al. dans [LMR15] permet de chercher des espaces invariants sur toute la fonction de tour, mais il ne fonctionne que sur des sous-espaces vectoriels de grande dimension relativement à la taille de bloc. Dans ces conditions, nous nous intéressons à des sous-ensembles invariants dont la structure et la taille

sont arbitraires, mais qui sont invariants à la fois par la couche linéaire composée avec l'addition de la clef de tour ($\mathsf{Add}_{k_i} \circ L$) et par la couche de substitution (S).

Cependant, nous avons montré qu'imposer l'invariance par la couche linéaire à chacun des tours implique l'existence de structures fortes sur ces invariants.

Proposition 3.13 (Condition sur les invariants par $\mathsf{Add}_{k_i} \circ L$)**.** *Soit L une permutation de $\mathbb{F}_2^n$ et soit $g \in \mathcal{B}_n$ un invariant à la fois pour $\mathsf{Add}_{k_i} \circ L$ et pour $\mathsf{Add}_{k_j} \circ L$ avec k_i et k_j deux clefs de tours différentes. Alors $\mathsf{LS}(g)$ est un espace linéaire invariant par L qui contient $k_i + k_j$.*

Démonstration. Par définition de g, il existe a et b dans $\mathbb{F}_2$ tels que pour tout $x \in \mathbb{F}_2^n$,

$$g(x) = g(L(x) + k_i) + a \text{ et } g(x) = g(L(x) + k_j) + b \, .$$

Cela implique que, pour tout $x \in \mathbb{F}_2^n$,

$$g(L(x) + k_i) + g(L(x) + k_j) = a + b \, ,$$

Comme L est inversible, on a de manière équivalente :

$$g(y + k_i + k_j) + g(y) = a + b, \forall y \in \mathbb{F}_2^n$$

ce qui signifie que $(k_i + k_j) \in \mathsf{LS}(g)$. Il reste à montrer que l'espace des structures linéaires de g est invariant par L.

Soit $s \in \mathsf{LS}(g)$, alors il existe une constante $c \in \mathbb{F}_2$ telle que $g(x) = g(x+s)+c$. Comme g est un invariant pour $\mathsf{Add}_{k_i} \circ L$, on a

$$g(L(x) + k_i) + a = g(x) = g(x + s) + c = g(L(x) + L(s) + k_i) + (a + c) \, .$$

Par le changement de variable $y := L(x) + k_i$, on obtient que

$$g(y) = g(y + L(s)) + c, \forall y \in \mathbb{F}_2^n \, , \tag{3.3}$$

ce qui signifie que $L(s) \in \mathsf{LS}(g)$. □

Cette proposition nous donne une première condition nécessaire sur l'existence d'un invariant à la fois pour L et S. Cette propriété a été observée pour la première fois dans [Ava17] mais dans le contexte particulier de l'attaque par sous-espace invariant. Ainsi, l'attaquant.e doit trouver un invariant pour la couche de substitution dont l'espace linéaire est invariant par L et contient l'ensemble des différences entre les clefs de tour. Ces différences entre les clefs de tour sont a priori dépendantes de la clef maître, et donc secrètes. Cependant, dans la plupart des chiffrements à bas coût (Noekeon [DPAR00], Midori [BBI+15], Rectangle [ZBL+14] pour n'en citer que quelques uns), le cadencement de clef consiste uniquement en l'addition d'une constante de tour (RC_i) à la clef maître :

$$\forall 1 \leq i \leq t, \, k_i = \mathsf{RC}_i + k \, .$$

Dans ce cas particulier, les différences entre les clefs de tour sont égales aux différences entre les constantes de tour qui sont des quantités publiques. Notre

condition nécessaire sur les invariants g de la couche de substitution devient donc indépendante de la clef de chiffrement. Plus précisément, $\mathsf{LS}(g)$ est un espace vectoriel qui doit être invariant par l'application de L et qui doit contenir les différences $(\mathsf{RC}_i + \mathsf{RC}_j)$ pour toute paire de constantes de tour.

Comme $\mathsf{LS}(g)$ est un espace vectoriel, nous nous intéressons au plus petit espace vectoriel qui contient les différences entre les constantes de tour.

Définition 3.14 (Espace $W_L(c)$)**.** *Soit L une permutation linéaire sur $\mathbb{F}_2^n$. Pour tout $c \in \mathbb{F}_2^n$, le plus petit sous-espace de $\mathbb{F}_2^n$ invariant par L qui contient c, noté $W_L(c)$ est*

$$\langle L^i(c), i \geq 0 \rangle \ .$$

Démonstration. Tout d'abord, il est clair, par construction, que l'espace $\langle L^i(c), i \geq 0\rangle$ est inclus dans $W_L(c)$ puisque $W_L(c)$ est un sous-espace vectoriel de $\mathbb{F}_2^n$ invariant par L par définition. De plus, pour tout λ_1, $\lambda_2 \in \mathbb{F}_2$ et toute paire (i, j),

$$L(\lambda_1 L^i(c) + \lambda_2 L^j(c)) = \lambda_1 L^{i+1}(c) + \lambda_2 L^{j+1}(c)$$

et donc appartient à l'espace vectoriel défini par $\langle L^i(c), i \geq 0\rangle$. On peut donc conclure que ce sous-espace de $\mathbb{F}_2^n$ est le plus petit espace invariant par L et qui contient c. □

On peut naturellement généraliser cette définition à plusieurs éléments. Soit D un sous-ensemble de $\mathbb{F}_2^n$, on définit $W_L(D) \subseteq \mathbb{F}_2^n$ comme étant le plus petit sous-espace vectoriel invariant par L contenant D :

$$W_L(D) := \sum_{c \in D} \langle L^i(c), i \geq 0\rangle = \sum_{c \in D} W_L(D) \ .$$

Dans notre contexte, D sera le sous-ensemble défini par les différences entre les clefs de tour (*i.e.* entre les constantes de tour). Pour que l'attaque par invariant puisse fonctionner, il faut donc trouver un invariant g pour la couche de substitution tel que $W_L(D) \subseteq \mathsf{LS}(g)$. Intuitivement, le concepteur du chiffrement devra donc faire en sorte pour que la dimension de $W_L(D)$ soit la plus grande possible : plus celle-ci est élevée, moins nous avons de degrés de liberté pour choisir g.

3.3.1.1 Quelques exemples

Afin de comprendre plus en détail ce qui se passe, nous regardons plusieurs algorithmes de chiffrement par bloc ayant les spécificités décrites ci-dessus, c'est-à-dire des clefs de tour qui sont égales à la clef maître à l'addition d'une constante de tour près.

Skinny. Skinny est une famille de chiffrement adaptable [5] proposée par Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas

5. Tweakable Block Cipher en anglais

Peyrin, Yu Sasaki, Pascal Sasdrich et Siang Meng Sim dans [BJK+16] dans le but d'être le plus "léger" possible, sans pour autant sacrifier de la sécurité. Les boîtes-S de Skinny sont de taille 4 (respectivement 8) pour la version à 64 bits (respectivement 128 bits). L'opération ShiftRows est identique à celle de l'AES à l'exception de la direction de la rotation et la matrice correspondant à MixColumns est la suivante :

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

et a la particularité d'être binaire (à la différence de celle de l'AES). En composant cette matrice par ShiftRows, on se retrouve avec un étage linéaire que l'on peut représenter par une matrice carrée binaire de taille 16 qui opère sur les 16 quartets (respectivement octets) de l'état de 64 (respectivement 128) bits.

Skinny-64. Pour la version sans "tweak" de Skinny-64-64, les mêmes clefs de tour sont réutilisées tous les 16 tours à une constante près. Donc les différences entre les sous-clefs $(k_i + k_{i+16})$ pour tout i sont indépendantes de la clef maître. En définissant le sous-ensemble D suivant :

$$D := \{\mathsf{RC}_1 + \mathsf{RC}_{17}, \mathsf{RC}_2 + \mathsf{RC}_{18}, \mathsf{RC}_3 + \mathsf{RC}_{19}, \mathsf{RC}_4 + \mathsf{RC}_{20}, \mathsf{RC}_5 + \mathsf{RC}_{21}\}$$

on obtient que $W_L(D) = \mathbb{F}_2^{64}$.

Skinny-128. Pour Skinny-128, les constantes de tour sont toutes de la forme suivante :

$$\begin{bmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

où c_0 est à valeurs dans l'ensemble $\{\mathsf{0x00}, \ldots, \mathsf{0x0f}\}$, c_1 est à valeurs dans $\{\mathsf{0x00}, \ldots, \mathsf{0x03}\}$ et $c_2 = \mathsf{0x02}$. Ainsi, les 4 premiers bits de chaque octet de l'état ne sont pas affectés par les constantes de tour. Comme la matrice représentant la couche linéaire est binaire, on en déduit que $W_L(D)$ est un sous-espace de $\mathbb{F}_2^{128}$ de dimension au plus 64.

Prince. Prince est un SPN dont le but est d'avoir une faible latence. Il a été conçu par Julia Borghoff, Anne Canteaut, Tim Guneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen et Tolga Yalcın [BCG+12]. Prince utilise 10 clefs de tours différentes $(k_i)_{1 \leq i \leq 10}$, de la forme $k_i = k + \mathsf{RC}_i$. De plus, une propriété particulière au chiffrement (appelée α-réflection) impose la condition suivante : pour tout $1 \leq i \leq 10$, $k_i + k_{11-i} = \alpha$ où α est une constante fixée (dérivée de π). L'intérêt de cette propriété est de

permettre d'utiliser le même algorithme pour déchiffrer que pour chiffrer, en utilisant simplement $k + \alpha$ au lieu de k. Ainsi, nous devons considérer l'ensemble suivant :

$$D := \{\alpha, \mathsf{RC}_1 + \mathsf{RC}_2, \mathsf{RC}_1 + \mathsf{RC}_3, \mathsf{RC}_1 + \mathsf{RC}_4, \mathsf{RC}_1 + \mathsf{RC}_5\}$$

On obtient finalement que $\dim W_L(D) = 56$.

Mantis. Cet algorithme de chiffrement par bloc a été proposé dans le même article que Skinny, mais ressemble assez à Prince, et a le même but : avoir une faible latence. Tout comme pour Prince, les clefs de tours de Mantis_7 suivent aussi la propriété d'α-réflection. Ainsi, on considère de la même manière l'ensemble suivant :

$$D := \{\alpha, \mathsf{RC}_1 + \mathsf{RC}_2, \mathsf{RC}_1 + \mathsf{RC}_3, \mathsf{RC}_1 + \mathsf{RC}_4, \mathsf{RC}_1 + \mathsf{RC}_5, \mathsf{RC}_1 + \mathsf{RC}_6, \mathsf{RC}_1 + \mathsf{RC}_7\}$$

et on obtient que $\dim W_L(D) = 42$.

Midori-64. Cet algorithme de chiffrement par bloc a été proposé par Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita et Francesco Regazzoni [BBI+15] et est conçu dans un souci de réduction de consommation d'énergie. C'est aussi un SPN, et la couche linéaire qui est la composition de ShuffleCell et de MixColumn donne une application linéaire qui se représente par une matrice carrée binaire de taille 16 qui interagit mal avec les constantes de tour : la dimension de $W_L(D)$ est beaucoup plus petite. En effet, les constantes de tour sont ajoutées seulement au bit de poids faible de chaque moitié d'octet, ce qui implique que $W_L(D) = \{0000, 0001\}^{16}$.

3.3.2 Un cas trivial

On suppose maintenant que la dimension de l'espace vectoriel $W_L(D)$ est au moins égale à $n - 1$ où n est la taille des blocs.

Proposition 3.15. *On suppose que la dimension de $W_L(D)$ est au moins égale à $n - 1$. Alors, toute fonction $g \in \mathcal{B}_n$ telle que $W_L(D) \subseteq \mathsf{LS}(g)$ est linéaire ou constante. Par conséquent, il n'existe pas d'invariant non-trivial g pour la couche de substitution telle que $W_L(D) \subseteq \mathsf{LS}(g)$, sauf si la couche de substitution possède une composante de degré algébrique égal à 1.*

Démonstration. D'après la proposition 14 dans [Car07],

$$\dim \mathsf{LS}(g) \geq k \Leftrightarrow \deg(g) \leq \begin{cases} n - k & \text{si } k \neq n \\ 1 & \text{si } k = n \end{cases}.$$

Ceci implique que, si $\dim W_L(D) \geq n - 1$, g doit être linéaire ou constante. Si g est constante, cela correspond aux deux ensembles invariants triviaux : $\emptyset$ et $\mathbb{F}_2^n$. En revanche, si g est linéaire, cela signifie qu'il existe une approximation linéaire d'une composante de S avec probabilité 1, ou de manière équivalente que la boîte-S utilisée a une composante de degré 1. □

Il est évident que l'on peut écarter ce dernier cas, puisque les boîtes-S sont choisies avec la plus grande non-linéarité possible. Une approximation linéaire avec probabilité 1 est donc le pire cas possible pour la cryptanalyse linéaire. Nous pouvons donc naturellement écarter ce cas particulier de notre analyse.

Ainsi, pour le chiffrement Skinny-64, comme la dimension de $W_L(D)$ vaut 64, nous pouvons affirmer qu'il n'existe pas d'invariant non-trivial à la fois pour la couche linéaire et pour la couche de substitution et ce pour n'importe quel choix *raisonnable* de boîte-S, *i.e.* qui ne serait pas trivialement cassé par la cryptanalyse linéaire. Cependant, quand la dimension de $W_L(D)$ est bien plus petite que la taille du chiffrement, nous n'avons pas d'argument particulier pour décider de l'existence ou de l'absence d'invariants non-triviaux.

3.3.3 Le cas général

Dans cette partie, nous nous intéressons au cas où la dimension de $W_L(D)$ est strictement inférieure à $(n-1)$ où n est la taille des blocs. Dans ces conditions, nous ne pouvons plus prouver l'absence d'invariants non-triviaux en exploitant uniquement la couche linéaire et son interaction avec les constantes de tour. Il convient donc de considérer les propriétés des boîtes-S utilisées. Plus précisément, quand $n - \dim W_L(D)$ est "suffisamment petit", il est possible de regarder les invariants $g \in \mathcal{B}_n$ non-triviaux potentiels pour la couche de substitution qui admettent certaines structures linéaires particulières appelées *structures linéaires invariantes* ou structures linéaires 0.

Définition 3.16 (Structures linéaires invariantes)**.** *Soit $f \in \mathcal{B}_n$ et α une structure linéaire de f, alors α est une structure linéaire invariante (ou structure linéaire 0) si la fonction dérivée $\Delta_\alpha f$ est la fonction booléenne nulle. L'ensemble des structures linéaires 0 de f forme un sous-espace vectoriel de $\mathsf{LS}(f)$ et est noté $\mathsf{LS}_0(f)$. Les éléments $\beta \in \mathsf{LS}(f)$ tels que $\Delta_\beta(f) = \mathbf{1}$ sont appelés structures linéaires 1 de f.*

Il est évident et bien connu [DW97] que $\dim \mathsf{LS}_0(f) \geq \dim \mathsf{LS}(f) - 1$.

3.3.3.1 Utilisation des structures linéaires invariantes

L'idée principale ici est de considérer un sous-espace noté Z, donné de l'espace des structures linéaires 0 des invariants potentiels, et d'appliquer la couche de substitution afin d'augmenter la dimension de ce sous-espace.

Proposition 3.17. *Soit S une permutation de $\mathbb{F}_2^n$, g un invariant pour S et $Z \subset \mathbb{F}_2^n$ tels que $\mathsf{LS}_0(g) \supseteq Z$. Alors*

- *g est constante sur chaque translaté de Z ;*
- *g est constante sur $S(Z)$.*

Démonstration. Comme $Z \subseteq \mathsf{LS}_0(g)$, pour tout $a \in \mathbb{F}_2^n$ et pour tout $z \in \mathsf{LS}_0(g)$, on a $g(a+z) = g(a)$, c'est-à-dire que g est constante sur tous les ensembles

$(a + Z)$, $a \in \mathbb{F}_2^n$. De plus, g étant un invariant pour S, il existe $c \in \mathbb{F}_2$ tel que $g \circ S(x) = g(x) + c$. Comme g est constante sur Z, on en déduit que g est constante sur $S(Z)$. □

À l'aide de ces observations nous construisons un algorithme qui vérifie si les invariants à la fois pour la couche de substitution et la couche linéaire de chaque tour sont nécessairement triviaux. L'idée principale de l'algorithme décrit par l'algorithme 1 est d'évaluer la couche de substitution S sur certaines valeurs prises aléatoirement dans un sous-espace Z. En appliquant S, on augmente la taille de l'ensemble R qui doit être inclus dans l'espace des structures linéaires invariantes. Si la dimension de l'espace Z de départ est suffisamment proche de n, alors nous pouvons espérer toucher tous les translatés de Z en appliquant S. Si ce phénomène apparaît, alors on en déduit l'absence d'invariants non-triviaux.

Algorithme 1 Vérifier que $\mathcal{U}(S) \cap \{g \in \mathcal{B}_n \mid Z \subseteq \mathsf{LS}_0(g)\}$ est trivial.

1: $R = \{\}$
2: **Répéter**
3: $\quad z \xleftarrow{\$} Z$
4: $\quad$ Calculer $S(z)$
5: $\quad$ Ajouter à R un représentant du translaté de Z défini par $S(z)$
6: **tant que** $|R| < 2^{n-\dim Z}$

Plus précisément, cet algorithme calcule les différents translatés sur lesquels g doit être invariant. Finalement, lorsque le nombre de translatés est suffisamment grand, il en découle que g doit nécessairement être constante.

3.3.3.2 Déterminer Z

Jusqu'à maintenant, nous avons supposé la connaissance préalable d'un sous-espace Z qui puisse servir de point de départ à l'algorithme 1. Or, Z doit être un sous-espace de $\mathsf{LS}_0(g)$ qui dépend (évidemment) d'un invariant potentiel déjà connu. On pourrait donc penser que le serpent se mord la queue, mais c'est sans compter sur les observations faites au début de cette section sur l'espace $W_L(D)$. En revanche, nous nous restreignons aux structures linéaires invariantes et non à toutes les structures linéaires. Or, chaque élément de l'espace $W_L(D)$ peut tout aussi bien être une structure linéaire 0 ou une structure linéaire 1. Cependant certaines structures linéaires invariantes peuvent être obtenues en utilisant les deux approches suivantes.

Première approche. D'après l'équation (3.3) de la preuve de la proposition 3.13, on obtient le lemme qui suit.

Lemme 3.18. *Soit $g \in \mathcal{B}_n$ un invariant pour $\mathsf{Add}_{k_i} \circ L$ pour une clef de tour k_i et soit V un sous-espace de $\mathsf{LS}(g)$ qui est invariant par L. Alors, pour tout $v \in V$, $(v + L(v))$ est une structure linéaire invariante de g.*

Démonstration. Soit $v \in V$. De la même manière qu'à la preuve de la proposition 3.13 à la page 27, nous considérons un invariant g pour $\mathsf{Add}_{k_i} \circ L$. Alors, comme $V \subset \mathsf{LS}(g)$, il existe une constante $c \in \mathbb{F}_2$ telle que pour tout $x \in \mathbb{F}_2^n$,

$$g(x) = g(x+v) + c\,.$$

Comme g est un invariant pour $\mathsf{Add}_{k_i} \circ L$, il existe une constante $a \in \mathbb{F}_2$ telle que pour tout $x \in \mathbb{F}_2^n$,

$$g(L(x)+k_i) + a = g(x) = g(x+v) + c = g(L(x+v)+k_i) + a + c\,.$$

En posant $x' = L(x) + k_i$, on obtient, pour tout $y \in \mathbb{F}_2^n$,

$$g(x') = g(x' + L(v)) + c\,.$$

Donc, pour tout $x \in \mathbb{F}_2^n$, on a

$$g(x+v) = g(x + L(v))\,,$$

ce qui implique, en posant $y := x + v$, que, pour tout $y \in \mathbb{F}_2^n$,

$$g(y) = g(y + v + L(v))\,,$$

ce qui signifie que $(v + L(v)) \in \mathsf{LS}_0(g)$ et termine la preuve. □

Pour utiliser ce lemme, une première façon de faire est tout simplement d'appliquer l'algorithme 1 sur l'espace $Z = W_L(D')$ où $D' = \{d + L(d), d \in D\}$. Cependant, la dimension de Z peut être trop petite (et donc l'algorithme inefficace). Dans ce cas, il est nécessaire d'utiliser une approche différente : faire tourner l'algorithme plusieurs fois, en considérant maintenant tous les choix possibles pour les structures linéaires invariantes parmi tous les éléments de l'espace D. Plus précisément, soit $D = \{d_1, d_2, \ldots, d_m, \ldots, d_t\}$ où les éléments $d_1, \ldots, d_m$ sont tous des structures linéaires 1, et $d_{m+1}, \ldots, d_t$ des structures linéaires 0 pour un invariant g, avec la condition $W_L(D) \subseteq \mathsf{LS}(g)$, alors la technique naturelle consiste à s'intéresser à l'ensemble D' défini par :

$$D' = \{d_1 + L(d_1), d_2 + L(d_2), \ldots, d_m + L(d_m), d_{m+1}, \ldots, d_t, d_1 + d_2, \ldots, d_1 + d_m\}$$

Par construction de l'ensemble D', nous augmentons significativement la dimension du sous-espace vectoriel $W_L(D')$ en ajoutant la somme des structures linéaires 1. Les éléments de D' sont donc (toujours par construction) des éléments appartenant à $\mathsf{LS}_0(g)$. Ainsi $W_L(D') \subseteq \mathsf{LS}_0(g)$ et nous pouvons alors appliquer l'algorithme 1 sur $Z = W_L(D')$. Cependant, nous ne savons pas, au préalable, quels éléments de D sont des structures linéaires 0. Il convient donc de faire tourner l'algorithme 1 sur tous les choix possibles de structures linéaires 0 ou de structures linéaires 1, c'est-à-dire 2^t fois où t est la taille de l'ensemble D.

Deuxième approche. Dans la première approche, nous utilisons plutôt la partie linéaire, mais il est possible (sous certaines conditions) d'utiliser les propriétés dérivées de la représentation en cycles de la boîte-S.

Proposition 3.19. *Soit $g \in \mathcal{U}(S)$ où S est une permutation de n bits possédant un cycle de taille impaire. Alors toute structure linéaire de g qui appartient à l'ensemble des images de l'application $(S + \mathsf{Id}_n)$, i.e. à $\{S(x) + x | x \in \mathbb{F}_2^n\}$, est une structure linéaire invariante de g.*

Démonstration. Si la couche de substitution S possède un cycle de longueur impaire, alors d'après la proposition 3.11, n'importe quel $g \in \mathcal{U}(S)$ vérifie nécessairement la propriété $g(x) = g(S(x))$ pour tout $x \in \mathbb{F}_2^n$. Soit $g \in \mathcal{U}(S)$ et $a \in \mathsf{LS}(g)$. Cette structure linéaire a appartient à l'ensemble image de l'application $(S + \mathsf{Id}_n)$ s'il existe $x_0 \in \mathbb{F}_2^n$ tel que $S(x_0) = x_0 + a$. On en déduit alors que

$$g(x_0) = g(S(x_0)) = g(x_0 + a) \ ,$$

ce qui implique que a est une structure linéaire invariante de g. □

En utilisant cette propriété particulière, on se rend compte que si l'on trouve suffisamment d'éléments $a \in W_L(D) \cap \mathsf{Im}(S + \mathsf{Id}_n)$, alors il nous suffit d'appliquer l'algorithme 1 sur l'ensemble qui en découle. Nous utilisons cette approche sur le chiffrement Mantis_7.

3.3.4 Résultats sur certains SPN à bas coût

Prince. Pour ce chiffrement, nous utilisons la première méthode sur l'ensemble $D' = \{d + L(d), d \in D\}$, où

$$D = \{\alpha, \mathsf{RC}_1 + \mathsf{RC}_2, \mathsf{RC}_1 + \mathsf{RC}_3, \mathsf{RC}_1 + \mathsf{RC}_3, \mathsf{RC}_1 + \mathsf{RC}_4, \mathsf{RC}_1 + \mathsf{RC}_5\} \ .$$

Dans ce cas la dimension de $W_L(D')$ vaut 51. En faisant tourner l'algorithme 1 sur cet espace, nous prouvons l'absence d'invariants non-triviaux à la fois pour la couche de substitution et la couche linéaire en quelques minutes (sur un ordinateur standard).

Mantis. Pour ce qui est de Mantis_7, la dimension de $W_L(D)$ vaut 42. La construction de $W_L(D')$ impose que sa dimension ne peut être plus grande que celle de $W_L(D)$. Comme $|D| = 7$, il faudrait faire tourner notre algorithme 2^7 fois sur un sous-espace de codimension 23 (ou 22 selon les cas), ce qui nous coûterait beaucoup trop cher. Il convient donc d'exploiter la deuxième approche (proposition 3.19).

La couche de substitution de Mantis_7 est l'application en parallèle de la boîte-S suivante `Sb` qui opère sur 4 bits.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathtt{Sb}(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6
$x + \mathtt{Sb}(x)$	c	b	f	0	a	e	9	0	0	0	b	e	c	f	a	9

La couche de substitution possède un cycle de longueur impaire puisque la boîte-S utilisée dans $\mathsf{Mantis_7}$ a un point fixe. De plus, l'ensemble image de l'application $(\mathtt{Sb} + \mathsf{Id}_4)$ est composée de 7 valeurs différentes : $\mathtt{0}, \mathtt{9}, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e}, \mathtt{f}$. Les différentes valeurs $a \in W_L(D)$ pour lesquelles chaque sous-mot de 4 bits est égal à une valeur comprise dans l'ensemble $\mathsf{Im}(\mathtt{Sb} + \mathsf{Id}_4)$ sont des structures linéaires invariantes. De plus, pour un élément a pris aléatoirement dans $\mathbb{F}_2^{64}$, chaque sous-mot de 4 bits appartient à $\mathsf{Im}(\mathtt{Sb} + \mathsf{Id}_4)$ avec une probabilité de $(7/16)$. On a de plus $(7/16)^{16} \approx 2^{-19.082}$. En pratique, nous trouvons donc en temps raisonnable suffisamment d'éléments $c \in W_L(D)$ qui engendrent tout l'espace $W_L(D)$, ce qui implique immédiatement que $W_L(D) \subseteq \mathsf{LS}_0(g)$ pour n'importe quel invariant g à la fois pour S et L. Nous pouvons ainsi appliquer l'algorithme 1 sur l'espace complet $Z = W_L(D)$. L'algorithme se termine, ce qui nous permet d'assurer qu'il n'existe pas d'invariants non-triviaux à la fois pour L et S dans le chiffrement $\mathsf{Mantis_7}$.

3.3.5 Les invariants de Midori

Pour ce chiffrement, $W_L(D) = \{000, 0001\}^{16}$ et est donc de dimension 16 seulement. Ainsi, les techniques utilisées ci-dessus ne fonctionnent pas, ce qui est normal puisqu'il existe des invariants à la fois pour la couche de substitution et la couche linéaire. En revanche, nous pouvons décrire les supports de *tous* les invariants pour Midori-64.

Afin de prouver les conditions nécessaires et suffisantes qui décrivent les invariants de Midori, nous avons besoin au préalable du lemme suivant :

Lemme 3.20. *Soit S une permutation de $\mathbb{F}_2^n$ dans lui-même, et W un sous-espace de $\mathbb{F}_2^n$. Si $g \in \mathcal{U}(S)$ et $W \subseteq \mathsf{LS}(g)$, alors g est constante sur les ensembles*

$$\mathcal{A}_{S,W}(x) = \bigcup_{i \in \mathbb{N}} \tilde{S}_W^i(\{x\}),\ x \in \mathbb{F}_2^n$$

où

$$\begin{array}{rcl} \tilde{S}_W : \mathcal{P}(\mathbb{F}_2^n) & \to & \mathcal{P}(\mathbb{F}_2^n) \\ X & \mapsto & \bigcup_{w \in W} \{S(S(x) + w) + w,\ \ x \in X\}\ . \end{array}$$

De plus, si S est une involution, alors il existe $k \in \mathbb{N}$ tel que $\mathcal{A}_{S,W}(x) = \tilde{S}_W^k(\{x\})$ pour tout $x \in \mathbb{F}_2^n$.

Démonstration. Soit $g \in \mathcal{U}(S)$. Alors il existe $c_1 \in \mathbb{F}_2$ tel que $g(S(x)) = g(x) + c_1$ pour tout $x \in \mathbb{F}_2^n$. Soit $w \in W$ où $W \subseteq \mathsf{LS}(g)$. Alors w est une structure linéaire c_2 de g, pour $c_2 \in \mathbb{F}_2$. Ainsi, nous avons que, pour tout $x \in \mathbb{F}_2^n$,

$$\begin{array}{rcl} g(S(S(x) + w) + w) & = & g(S(S(x) + w)) + c_2 \\ & = & g(S(x) + w) + c_1 + c_2 \\ & = & g(S(x)) + c_2 + c_1 + c_2 \\ & = & g(x) + c_1 + c_2 + c_1 + c_2 \\ & = & g(x) \end{array}$$

Ceci signifie que g est constante sur les ensembles $\mathcal{A}_{S,W}(x)$. Si S est une involution, tout ensemble X satisfait $X \subseteq \tilde{S}_W(X)$ pour tout ensemble X. En effet, comme $0 \in W$, la propriété suivante est vérifiée :

$$\tilde{S}_W(X) \supseteq \{S(S(x)), x \in X\} = X \ .$$

Ainsi, la suite d'ensembles $(\tilde{S}^i_W(\{x\}))_{i\in\mathbb{N}}$ est croissante au sens de l'inclusion. Comme nous sommes en dimension finie, il existe nécessairement un entier k_x tel que $\mathcal{A}_{S,W}(x) = \tilde{S}^{k_x}_W(\{x\})$. On obtient alors le résultat voulu en choisissant $k = \max_x k_x$. □

Ce qui nous simplifie grandement la tâche pour Midori, c'est que les ensembles $\mathcal{A}_{S,W}(x)$ sont assez simples à décrire, puisque la couche de substitution est l'application, en parallèle, de 16 boîtes-S opérant chacune sur 4 bits. On note $\mathtt{Sb}$ cette permutation sur 4 bits. Ainsi, notre espace W correspond alors à V à la puissance 16 (au sens du produit cartésien) : $W = V^{16}$, où $V = \{0000, 0001\}$, ce qui nous permet de décrire complètement les invariants pour la couche linéaire et la couche de substitution de Midori.

Proposition 3.21. *Soit S la couche de substitution utilisée dans Midori-64 et $W = \{0000, 0001\}^{16}$. Soit $g \in \mathcal{B}_n$. Alors $g \in \mathcal{U}(S)$ et $W \subseteq \mathsf{LS}(g)$ si et seulement si le support de g est défini par*

$$\mathsf{Supp}(g) = \bigcup_{b_0,\ldots,b_{31}\in\mathsf{Supp}(h)} H_{b_0b_1} \times H_{b_2b_3} \times \cdots \times H_{b_{30}b_{31}}$$

où h est une fonction booléenne à 32 variables telle que $\{00, 10\}^{16} \subseteq \mathsf{LS}(h)$ et les ensembles H_{ab} sont définis par

$$H_{00} = \{8\}, H_{10} = \{9\}, H_{01} = \{0, 3, 5, 6, \mathtt{b}, \mathtt{c}, \mathtt{f}\} \text{ and } H_{11} = \{1, 2, 4, 7, \mathtt{a}, \mathtt{d}, \mathtt{e}\} \ .$$

Par exemple, l'invariant g_1 utilisé dans l'attaque par invariant dans [GJN+16] peut être défini par $\mathsf{Supp}(g_1) = \{8, 9\}^{16}$ et correspond alors à la fonction booléenne h définie par

$$h(b_0, \ldots, b_{31}) = \prod_{i=0}^{15}(1 + b_{2i+1}) \ .$$

Démonstration. Comme S consiste en l'application de 16 fois $\mathtt{Sb}$ et que $W = V^{16}$, avec $V = \{0000, 0001\}$, il en découle que, pour tout $x_0, \ldots, x_{15} \in \mathbb{F}_2^4$,

$$\begin{aligned} & \tilde{S}_W(\{(x_0, \ldots, x_{15})\}) \\ = \ & \{\mathtt{Sb}(\mathtt{Sb}(x_0) + w_0) + w_0, \ldots, \mathtt{Sb}(\mathtt{Sb}(x_{15}) + w_{15}) + w_{15},\ w_i \in V\} \\ = \ & \tilde{\mathtt{Sb}}_V(\{x_0\}) \times \ldots \times \tilde{\mathtt{Sb}}_V(\{x_{15}\}) \ . \end{aligned}$$

Alors, pour tout $k \in \mathbb{N}$,

$$\tilde{S}^k_W(\{(x_0, \ldots, x_{15})\}) = \tilde{\mathtt{Sb}}^k_V(\{x_0\}) \times \ldots \times \tilde{\mathtt{Sb}}^k_V(\{x_{15}\}) \ .$$

De plus, la boîte-S utilisée est une involution, donc on déduit du lemme précédent que

$$\mathcal{A}_{S,W}(x_0,\ldots,x_{15}) = \mathcal{A}_{\mathtt{Sb},V}(x_0) \times \ldots \times \mathcal{A}_{\mathtt{Sb},V}(x_{15}) \ .$$

Pour le cas de la boîte-S de Midori, on remarque rapidement que chaque ensemble $\mathcal{A}_{\mathtt{Sb},V}(x)$ correspond à l'un des ensembles suivants :

$$H_{00} = \{8\}, H_{10} = \{9\}, H_{01} = \{0,3,5,6,\mathtt{b},\mathtt{c},\mathtt{f}\} \text{ and } H_{11} = \{1,2,4,7,\mathtt{a},\mathtt{d},\mathtt{e}\} \ .$$

Ainsi, pour tout $x = (x_0,\ldots,x_{15})$ dans $\mathbb{F}_2^{64}$, il existe $b \in \mathbb{F}_2^{32}$ tel que

$$\mathcal{A}_{S,W}(x) = H_{b_0b_1} \times H_{b_2b_3} \times \cdots \times H_{b_{30}b_{31}} \ .$$

Sans oublier le fait que chaque invariant g pour la couche de substitution avec $W \subseteq \mathsf{LS}(g)$ doit être nécessairement constant sur chaque ensemble $\mathcal{A}_{S,W}(x)$, c'est-à-dire que son support doit être une union de ces ensembles, *i.e.*

$$\mathsf{Supp}(g) = \bigcup_{b_0\ldots b_{31}\in\mathcal{B}} H_{b_0b_1} \times H_{b_2b_3} \times \ldots \times H_{b_{30}b_{31}}$$

où $\mathcal{B}$ est un sous-ensemble de $\mathbb{F}_2^{32}$ qu'il nous reste à déterminer. Les ensembles H_{ab} étant disjoints, il en est de même pour le produit cartésien, ce qui, de manière équivalente signifie qu'un invariant g pour la couche de substitution de Midori est une combinaison linéaire des 2^{32} fonctions booléennes g_b pour $b \in \mathbb{F}_2^{32}$ définies par

$$g_b = \prod_{i=0}^{15} f_{b_{2i}b_{2i+1}} \text{ où } \mathsf{Supp} f_{b_{2i}b_{2i+1}} = H_{b_{2i}b_{2i+1}} \ .$$

Afin de déterminer tous les invariants possibles (ou de manière équivalente les ensembles $\mathcal{B}$ possibles), il faut prendre en compte les seuls qui nous assurent que $W \subseteq \mathsf{LS}(g)$.

Soit h une fonction booléenne à 32 variables dont le support est $\mathcal{B}$. On observe que, pour chaque $b_0b_1 \in \mathbb{F}_2^2$, on a $H_{b_0b_1} + 0001 = H_{b_0+1,b_1}$. Ainsi, en notant $\mathcal{H}_b = H_{b_0b_1} \times \cdots \times H_{b_{30}b_{31}}$, on obtient que pour tout mot de 32 bits b, l'image de la translation de $\mathcal{H}_b$ par $w \in W$ est égal à $\mathcal{H}_{b+\pi(w)}$, où $\pi(w) \in \mathbb{F}_2^{32}$ défini par $\pi(w)_i = 00$ si $w_i = 0000$ et $\pi(w)_i = 10$ si $w_i = 0001$ pour tout $0 \leq i \leq 15$. Donc, si $w \in W$ est une structure linéaire invariante pour g, alors $b \in \mathbb{F}_2^{32}$ appartient au support de h si et seulement si $b + \pi(w)$ appartient au support de h aussi. En d'autres termes : $\pi(w)$ est une structure linéaire invariante pour h, et de manière équivalente pour les structures linéaires-1. Ainsi, $\pi(W) = \{00,10\}^{16} \subseteq \mathsf{LS}(h)$. Réciproquement, soit g une fonction booléenne définie par

$$\mathsf{Supp}(g) = \bigcup_{b\in\mathsf{Supp}(h)} H_{b_0b_1} \times \cdots \times H_{b_{30}b_{31}}$$

où $\pi(W) \subseteq \mathsf{LS}(h)$. Chaque ensemble $H_{b_{2i}b_{2i+1}}$ est invariant par la boîte-S, cette propriété étant préservée par l'addition implique que g est un invariant pour toute la couche de substitution de Midori. De plus, chaque $w \in W^{16}$ est une structure linéaire pour g puisque $\pi(w)$ est une structure linéaire pour h. □

Comme déjà dit avant la preuve, l'invariant g_1 utilisé dans l'attaque par invariant dans [GJN+16] est défini par $\mathsf{Supp}(g_1) = \{8, 9\}^{16}$, ce qui correspond à la fonction booléenne h définie par

$$h(b_0, \ldots, b_{31}) = \prod_{i=0}^{15} (1 + b_{2i+1}) \ .$$

Dans ce cas particulier, les éléments dans $\pi(W) = \{00, 10\}^{16}$ sont des structures linéaires invariantes pour h, donc tous les éléments de $W_L(D)$ sont eux aussi des structures linéaires invariantes pour l'invariant g_1. Finalement, en notant les bits de la j-ième cellule de Midori-64 par $x_{j,3}$, $x_{j,2}$, $x_{j,1}$ et $x_{j,0}$ le bit de poids faible, on peut exprimer la forme algébrique normale de l'invariant g_1 :

$$g_1(x) = \prod_{j=1}^{16} (x_{j,1}x_{j,2}x_{j,3} + x_{j,1}x_{j,3} + x_{j,2}x_{j,3} + x_{j,3}) \ .$$

Pour l'attaque par invariant non-linéaire décrite dans [TLS16], Todo et ses coauteurs ont utilisé l'invariant quadratique suivant :

$$g_2(x) = \sum_{j=1}^{16} (x_{j,3}x_{j,2} + x_{j,2} + x_{j,1} + x_{j,0}) \ .$$

cet invariant correspond au choix de la fonction booléenne à 32 variables h définie par $h(b) = \sum_{i=0}^{15} b_{2i}$. Ici, seulement les éléments dans $W_L(D)$ avec un nombre pair de quartets non-nuls sont des structures linéaires invariantes pour g_2. La somme des ces deux fonctions booléennes $(g_1 + g_2)$ nous donne alors un nouvel invariant de degré 48. Cependant, cet invariant n'admet pas de nouvelles clefs faibles, ce qui n'améliore pas l'attaque. En revanche, l'accès à plusieurs invariants peut diminuer la quantité de données nécessaires pour mener à bien une attaque par invariant comme nous le verrons à la fin de ce chapitre.

3.4 Critère de conception sur la couche linéaire et les constantes de tour

Dans cette section, nous analysons les propriétés de l'espace $W_L(D)$. Nous pouvons alors expliquer plus précisément d'où viennent les observations faites à la section 3.3.1.1 sur les chiffrements par bloc existant et la dimension des espaces $W_L(D)$ associés. Principalement, $W_L(D)$ dépend du choix des constantes de tour et de l'application linéaire L, et c'est un mauvais choix de constantes de tour, combiné avec des propriétés intrinsèques de la couche linéaire qui rendent la dimension de l'espace $W_L(D)$ trop petite pour assurer une résistance aux attaques par invariant. Afin de simplifier la lisibilité de cette section, nous commencerons par une étude sur un ensemble D réduit à un unique élément arbitraire non nul $(D = \{c\})$, puis nous généraliserons les résultats à un ensemble quelconque.

Par souci de simplicité, mais aussi pour mieux contextualiser nos problèmes, nous restreignons nos résultats sur la caractéristique 2 seulement, mais quelques résultats qui suivent se généralisent à toute opération linéaire sur $\mathbb{F}_q$ où q est une puissance d'un nombre premier.

3.4.1 Les propriétés de $W_L(c)$

Dans toute la suite, c est un élément a priori arbitraire de $\mathbb{F}_2^n$, où n désigne toujours la taille des blocs du chiffrement et L est une application linéaire bijective sur $\mathbb{F}_2^n$ (couche de permutation du chiffrement). $W_L(c)$ est le sous-espace défini par le lemme 3.14 comme le plus petit sous-espace de $\mathbb{F}_2^n$ contenant c et invariant par L.

3.4.1.1 Première observation

Pour un unique élément c appartenant à $\mathbb{F}_2^n$, la dimension de $W_L(c)$ est bornée supérieurement par le degré du polynôme minimal de L.

Définition 3.22 (Page 176, [DF04])**.** *Soit L une permutation linéaire de $\mathbb{F}_2^n$. Le polynôme minimal de L est le polynôme unitaire $\mathsf{Min}_L(X) = \sum_{i=0}^{d} p_i X^i \in \mathbb{F}_2[X]$ de plus petit degré tel que*

$$\mathsf{Min}_L(L) = \sum_{i=0}^{d} p_i L^i = 0 \,.$$

De manière équivalente, le polynôme minimal annulateur relativement à L d'un élément $c \in \mathbb{F}_2^n$ (appelé simplement polynôme minimal de c) est le polynôme unitaire noté $\mathsf{ord}_L(c)(X) = \sum_{i=0}^{d} p'_i X^i \in \mathbb{F}_2[X]$ de plus petit degré tel que

$$\sum_{i=0}^{d} p'_i L^i(c) = 0$$

Proposition 3.23 (Dimension de $W_L(c)$)**.** *Soit L une permutation linéaire de $\mathbb{F}_2^n$. Pour tout élément non nul $c \in \mathbb{F}_2^n$, la dimension de $W_L(c)$ est le degré du polynôme minimal de c.*

Démonstration. Par le lemme 3.14, on sait que $W_L(c)$ est l'espace vectoriel engendré par les vecteurs $L^i(c)$ pour $i \geq 0$. Soit d le plus petit entier tel que $c, L(c), \ldots, L^{d-1}(c)$ soient linéairement indépendants. Par définition, d correspond au degré du polynôme minimal de c. En effet, exprimer le fait que $L^d(c)$ appartienne à l'espace vectoriel $\langle L^i(c), 0 \leq i < d \rangle$ est équivalent à exprimer l'existence de d éléments de $\mathbb{F}_2$ notés $p'_0, \ldots, p'_{d-1}$ tels que $L^d(c) = \sum_{i=0}^{d-1} p'_i L^i(c)$, c'est-à-dire $P(L)(c) = 0$, avec $P(X) = X^d + \sum_{i=0}^{d-1} p'_i X^i$. Ainsi $d \leq \dim W_L(c)$.

Il nous reste à montrer ici que $d = \dim W_L(c)$, *i.e.* que tout élément $L^{d+t}(c)$ pour tout $t \geq 0$ appartient à l'espace vectoriel engendré par l'ensemble $\{c, L(c), \ldots, L^{d-1}(c)\}$. On montre cette propriété par récurrence sur

t. Par construction de d, la propriété est vraie pour $t = 0$. En supposant que la propriété est vraie pour un certain $t \geq 0$, *i.e.* il existe $\lambda_0, \ldots, \lambda_{d-1} \in \mathbb{F}_2$ tels que $L^{d+t} = \sum_{i=0}^{d-1} \lambda_i L^i(c)$, on obtient que

$$L^{d+t+1}(c) = L(L^{d+t}(c)) = L\left(\sum_{i=0}^{d-1} \lambda_i L^i(c)\right) = \sum_{i=0}^{d-1} \lambda_i L^{i+1}(c) \ .$$

Comme $L^d(c) \in \langle c, \ldots, L^{d-1}(c)\rangle$, on déduit que $L^{d+t+1} \in \langle c, \ldots, L^{d-1}(c)\rangle$, ce qui prouve la propriété au rang $t+1$ et qui conclut la preuve. □

Au vu des définitions des polynômes minimaux à la fois de L et de c, il est clair que le polynôme minimal de c (pour n'importe quelle valeur de c) est un diviseur du polynôme minimal de L, ce qui nous donne directement une borne supérieure pour la dimension de $W_L(c)$: le degré du polynôme minimal.

Corollaire 3.24. *Soit L une permutation linéaire de $\mathbb{F}_2^n$. Pour tout $c \in \mathbb{F}_2^n$, la dimension de $W_L(c)$ est au plus le degré du polynôme minimal de L.*

3.4.1.2 Une meilleure connaissance de $W_L(c)$

Plus généralement, les valeurs possibles pour la dimension de $W_L(c)$ sont exactement les degrés des diviseurs du polynôme minimal de L. Ici, nos résultats sont obtenus grâce à une méthode constructive, en utilisant des matrices compagnons, puis en généralisant à tous types de matrices inversibles en utilisant leur représentation canonique.

Définition 3.25 (Matrices compagnons)**.** *Soit $g(X) = X^d + \sum_{i=0}^{d-1} g_i X^i$ un polynôme unitaire de $\mathbb{F}_2[X]$. La matrice compagnon associée au polynôme g est la matrice carrée de taille d définie de la manière suivante*

$$C(g) = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \ldots & 1 \\ g_0 & g_1 & g_2 & \cdots & g_{d-1} \end{pmatrix}$$

Quand $\deg \mathsf{Min}_L = n$. Dire que $\deg \mathsf{Min}_L = n$ est équivalent à dire que le polynôme minimal de la matrice est égal à son polynôme caractéristique. Dans ce cas particulier, on sait que l'on peut trouver une base de l'espace vectoriel dans laquelle la matrice qui représente L est une matrice compagnon, dont le polynôme associé est Min_L (*e.g.* [Her75]). Ainsi, en exploitant cette représentation particulière de notre couche linéaire sous forme de matrice compagnon, on peut alors montrer la proposition suivante.

Proposition 3.26. *Soit L une permutation linéaire de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ telle que $\deg \mathsf{Min}_L = n$, i.e. telle qu'il existe une base dans laquelle la représentation de*

L est une matrice compagnon $C(Q)$ avec $Q \in \mathbb{F}_2[X]$ de degré n. Alors, pour tout diviseur non-constant P de Q dans l'anneau des polynômes sur $\mathbb{F}_2$, il existe $c \in \mathbb{F}_2^n$ tel que $\mathsf{ord}_L(c) = P$.

Démonstration. Quand il existe une base pour laquelle la représentation matricielle de la permutation linéaire est une matrice compagnon $C(Q)$, nous pouvons considérer que les éléments $c, L(c), L^2(c), \ldots$ sont des valeurs successives d'états internes d'un LFSR (Linear Feedback Shift Register) de taille n, de polynôme caractéristique Q et d'état initial c. Dans ces conditions, $\mathsf{ord}_L(c)$ correspond exactement au polynôme minimal de la suite engendrée par le LFSR de taille n, de polynôme caractéristique Q et d'état initial c [LN83, Théorème 8.51]. Par ailleurs, il est bien connu qu'il existe une bijection entre les suites $(s_t)_{t\geq 0}$ produites par le LFSR de polynôme caractéristique Q et l'ensemble des polynômes $C \in \mathbb{F}_2[X]$ de degré plus petit que celui de Q [LN83, Théorème 8.40]. En effet, la fonction génératrice d'une suite engendrée par un LFSR s'écrit de la manière suivante

$$\sum_{t\geq 0} s_t X^t = \frac{C(X)}{Q^*(X)} ,$$

où Q^* est le polynôme réciproque de Q, c'est-à-dire $Q^* = X^{\deg Q} Q(1/X)$, et C est défini par l'état initial du LFSR.

Maintenant, nous choisissons pour P n'importe quel diviseur non-constant de Q : $Q(X) = P(X)R(X)$, avec $P \neq 1$. Alors le polynôme réciproque vérifie $Q^*(X) = P^*(X)R^*(X)$. Ainsi, pour $C(X) = R^*(X)$, on a

$$\frac{C(X)}{Q^*(X)} = \frac{1}{P^*(X)} .$$

Donc la suite engendrée par l'état initial défini par $C = R^*$ admet P comme polynôme minimal, donc le polynôme minimal de cet état initial est P, ce qui termine la preuve. □

Cependant, ceci est un cas particulier. En effet, le polynôme minimal d'une matrice n'est pas forcément égal à son polynôme caractéristique. De manière générale, toute matrice inversible est équivalente à une matrice carrée diagonale par blocs, dont les blocs sont des matrices compagnons. Finalement, le résultat précédent se généralise naturellement à l'aide de la proposition qui suit.

Proposition 3.27. *Soit L une permutation linéaire de $\mathbb{F}_2^n$ et Min_L son polynôme minimal. Alors, pour tout diviseur P de Min_L, il existe $c \in \mathbb{F}_2^n$ tel que la dimension de $W_L(c)$ est égal au degré de P. En particulier,*

$$\max_{c\in\mathbb{F}_2^n} \dim W_L(c) = \deg \mathsf{Min}_L .$$

Afin de prouver cette proposition, nous avons besoin du théorème bien connu suivant ainsi que de son corollaire.

Théorème 3.28 (Diviseurs élémentaires quand $\mathsf{Min}_L = Q^e$)**.** *[Her75, 6.7.1 page 307] Soit L une application linéaire de $\mathbb{F}_2^n$ qui a pour polynôme minimal $P = Q^e$, où Q est unitaire et irréductible dans $\mathbb{F}_2[X]$, alors il existe une base de $\mathbb{F}_2^n$ dans laquelle la matrice de L est de la forme*

$$\begin{pmatrix} C(Q^{e_1}) & & & \\ & C(Q^{e_2}) & & \\ & & \ddots & \\ & & & C(Q^{e_r}) \end{pmatrix}$$

où $e = e_1 \geq e_2 \geq \cdots \geq e_r$.

Corollaire 3.29 (Diviseurs élémentaires)**.** *[Her75, Page 308] Soit L une application linéaire de $\mathbb{F}_2^n$ qui a pour polynôme minimal $P = Q_1^{\ell_1} Q_2^{\ell_2} \cdots Q_k^{\ell_k}$ où $Q_1, \ldots, Q_2$ sont irréductibles et distincts dans $\mathbb{F}_2[X]$, alors il existe une base de $\mathbb{F}_2^n$ dans laquelle la matrice de L est de la forme*

$$\begin{pmatrix} R_1 & & & \\ & R_2 & & \\ & & \ddots & \\ & & & R_k \end{pmatrix}$$

où pour tout $1 \leq i \leq k$, R_i est de la forme

$$\begin{pmatrix} C(Q_i^{e_1^i}) & & & \\ & C(Q_i^{e_2^i}) & & \\ & & \ddots & \\ & & & C(Q_i^{e_{r_i}^i}) \end{pmatrix}$$

où $\ell_i = e_1^i \geq e_2^i \geq \cdots \geq e_{r_i}^i$.

Par exemple, la couche linéaire de Prince [BCG+12] a pour polynôme minimal

$$\begin{aligned} \mathsf{Min}_L(X) &= X^{20} + X^{18} + X^{16} + X^{14} + X^{12} + X^8 + X^6 + X^4 + X^2 + 1 \\ &= (X^4 + X^3 + X^2 + X + 1)^2 (X^2 + X + 1)^4 (X + 1)^4 \,. \end{aligned}$$

En notant $A(X) = X^4 + X^3 + X^2 + X + 1$, $B(X) = X^2 + X + 1$ et $C(X) = X + 1$, la couche linéaire se décompose de la manière suivante

$$\begin{pmatrix} R_1 & & \\ & R_2 & \\ & & R_3 \end{pmatrix}$$

où R_1, R_2 et R_3 sont les trois matrices suivantes.

$$\begin{pmatrix} C(A^2) & 0 \\ 0 & C(A^2) \end{pmatrix}$$

$$\begin{pmatrix} C(B^4) & 0 & 0 & 0 \\ 0 & C(B^4) & 0 & 0 \\ 0 & 0 & C(B^2) & 0 \\ 0 & 0 & 0 & C(B^2) \end{pmatrix}$$

$$\begin{pmatrix} C(C^4) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C(C^4) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C(C^4) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C(C^4) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C(C^2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C(C^2) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & C(C^2) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C(C^2) \end{pmatrix}$$

Démonstration. Si le polynôme P est constant (*i.e.* de degré 0), on choisit alors $c = 0$, ce qui nous permet d'écarter ce cas particulier, et de considérer dans la suite de cette preuve que le degré de P est strictement positif. On considère alors la factorisation du polynôme minimal de L :

$$\mathsf{Min}_L(X) = M_1(X)^{e_1} M_2(X)^{e_2} \cdots M_k(X)^{e_k}$$

où $M_1, \ldots, M_k$ sont des polynômes distincts et irréductibles sur $\mathbb{F}_2$. D'après le théorème 3.28 et le corollaire 3.29 dans [Her75], l'espace vectoriel $\mathbb{F}_2^n$ peut être décomposé en une somme de sous-espaces vectoriels invariants par l'application linéaire L :

$$\mathbb{F}_2^n = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{r_i} V_{i,j}$$

de manière à ce que la matrice induite par l'application L mais appliquée sur le sous-espace $V_{i,j}$ soit la matrice compagnon de polynôme $M_i^{\ell_{i,j}}$ où chaque $\ell_{i,j}$ est un entier tel que $\ell_{i,1} = e_i$. Les polynômes $M_i^{\ell_{i,j}}$ sont appelés les diviseurs élémentaires de L. Soit maintenant P un polynôme non-constant qui divise le polynôme minimal de L, Min_L. Ainsi, sans perdre de généralité (par réordonnancement des polynômes dans la décomposition en facteurs irréductibles), nous pouvons supposer qu'il existe κ entiers non nuls $a_1, a_2, \ldots, a_\kappa$, tels que $a_i \leq e_i$, pour tout $1 \leq i \leq \kappa$, tels que

$$P(X) = M_1(X)^{a_1} M_2(X)^{a_2} \cdots M_\kappa(X)^{a_\kappa} \ .$$

Comme tous les facteurs $M_i^{a_i}$ de P sont non-constants, on sait d'après la proposition 3.26 qu'il existe u_i appartenant au sous-espace vectoriel $V_{i,1}$ tel que $\mathsf{ord}_{L_i}(u_i) = M_i^{a_i}$, où L_i est définie comme l'application linéaire L mais appliquée sur l'espace $V_{i,1}$. Maintenant, on définit l'élément c appartenant à l'espace $\bigoplus_{i=1}^{\kappa} V_{i,1}$ défini par $c = \sum_{i=1}^{\kappa} u_i$. Soit alors d coefficients de $\mathbb{F}_2$ notés $p_0, \ldots, p_{d-1}$ tels que le polynôme associé à ces coefficients $R(X) = X^d + \sum_{t=0}^{d-1} p_t X^t$ soit égal

au polynôme minimal de c relativement à L. Par définition de ce polynôme, nous avons

$$L^d(c) = \sum_{t=0}^{d-1} p_t L^t(c) \ .$$

De plus, $c = \sum_{i=1}^{\kappa} u_i$, donc $L^t(c) = \sum_{i=1}^{\kappa} L^t(u_i)$ pour tout $t \geq 1$. Qui plus est, chaque u_i appartient à $V_{i,1}$, et les sous-espaces $V_{i,1}$ sont en somme directe, ce qui nous implique l'égalité suivante pour tout $1 \leq i \leq \kappa$:

$$L^d(u_i) = \sum_{t=0}^{d-1} p_t L^t(u_i) \ .$$

Ainsi, le polynôme R est un multiple de chaque polynôme minimal de u_i relativement à l'application de L. Donc R doit être un multiple du plus petit commun multiple de ces polynômes. Or ces polynômes sont, par choix de u_i, de la forme $M_i^{a_i}$ où les M_i sont distincts et irréductibles sur $\mathbb{F}_2$, donc $\mathsf{ppcm}(M_1^{a_1}, M_2^{a_2}, \ldots, M_\kappa^{a_\kappa}) = P$. Finalement, par construction de l'élément c, on sait que $P(L(c)) = 0$, ce qui implique alors que $P = R$ et donc que le polynôme minimal de c par rapport à L est P, ce qui termine la preuve. □

3.4.1.3 Des exemples concrets

LED. LED est aussi un SPN proposé par Jian Guo, Thomas Peyrin, Axel Poschmann et Matthew J. B. Robshaw en 2011 à CHES [GPPR11] et le polynôme minimal de la couche linéaire utilisée dans LED est le polynôme suivant :

$$\mathsf{Min}_L(X) = (X^8 + X^7 + X^5 + X^3 + 1)^4 (X^8 + X^7 + X^6 + X^5 + X^2 + X + 1)^4 \ .$$

Comme le degré de ce polynôme est égal à 64, qui est la taille des blocs du chiffrement, il existe une constante $c \in \mathbb{F}_2^{64}$ telle que $W_L(c)$ couvre tout l'espace.

Skinny. La couche linéaire de Skinny prend en entrée un état de taille $s \times 16$ où s vaut 4 ou 8. L'application L est une permutation $\mathbb{F}_{2^s}$-linéaire sur $(\mathbb{F}_{2^s})^{16}$ définie par une matrice carrée M de taille 16 mais dont les coefficients sont à valeur dans le corps fini à deux éléments. De plus, l'ordre multiplicatif de cette matrice vaut 16, *i.e.* $M^{16} = \mathsf{Id}_{16}$ et pour tout $e < 16$, $(M + \mathsf{Id}_{16})^e \neq 0$, donc on sait que le polynôme minimal de l'application linéaire dans Skinny est

$$\mathsf{Min}_L(X) = X^{16} + 1 = (X + 1)^{16} \ .$$

Dans ces conditions, on sait qu'il existe des éléments c non nuls dans $(\mathbb{F}_{2^s})^{16}$ tels que la dimension de l'espace $W_L(c)$ prenne n'importe quelle valeur entre 1 et 16. De plus, il est relativement simple de trouver de tels éléments. En effet, en reprenant la preuve de la proposition 3.26, on sait que la matrice décrite dans Skinny est équivalente à la matrice compagnon dont le polynôme est $X^{16} + 1$, c'est-à-dire qu'il existe U une matrice inversible de taille 16 à coefficients dans $\mathbb{F}_2$

telle que $M = U \times C(X^{16}+1) \times U^{-1}$. On considère alors maintenant des éléments c appartenant à $(\mathbb{F}_{2^s})^{16}$, mais où seulement le bit de poids faible dans chaque cellule $(\mathbb{F}_{2^s})$ peut être non-nul (on peut donc les représenter par des vecteurs de taille 16). Pour Skinny, il est très facile d'exhiber les éléments qui donnent une dimension faible puisque la matrice compagnon $C(X^{16}+1)$ correspond à un simple décalage ! La matrice de changement de base U a la forme suivante :

$$U = \begin{pmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Pour donner quelques exemples qui illustrent le propos, nous avons calculé des éléments qui donnent des espaces $W_L(c)$ de petite dimension. Les valeurs réelles des constantes associées sont à prendre *avant* l'application de la matrice U qui correspond à un changement de base.

$U^{-1} \times b$	b	$\dim W_L(c)$
1111111111111111	0011001100110011	1
1010101010101010	1111111111111111	2
1100110011001100	1001100110011001	3
1000100010001000	1011101110111011	4
1000000000000000	1111010111110001	16

Prince. Le polynôme minimal de la couche linéaire de Prince est de degré 20 et se décompose de la manière suivante :

$$\begin{aligned}
\mathsf{Min}_L(X) &= X^{20} + X^{18} + X^{16} + X^{14} + X^{12} + X^8 + X^6 + X^4 + X^2 + 1 \\
&= (X^4 + X^3 + X^2 + X + 1)^2 (X^2 + X + 1)^4 (X + 1)^4 \,.
\end{aligned}$$

Ainsi, la dimension maximale possible pour $W_L(c)$ est 20 et la factorisation particulière du polynôme nous permet d'affirmer qu'il est possible d'obtenir toutes les dimensions possibles entre 1 et 20 pour $W_L(c)$.

Mantis et Midori-64. Ces deux chiffrements ont la même couche linéaire, dont le polynôme minimal est

$$\mathsf{Min}_L(X) = (X+1)^6 \ .$$

Donc les dimensions possibles pour $W_L(c)$ se situent entre 1 et 6, ce qui est très peu par rapport à la taille des blocs.

3.4.2 Avec plusieurs constantes de tour

Jusqu'à maintenant, nous avons considéré un seul élément c (c'est-à-dire 2 constantes de tour puisque c'est la différence entre les constantes de tour qui est à prendre en compte pour les attaques par invariant). Il est donc naturel de se demander ce qui se produit lorsque l'on considère un nombre arbitraire de constantes, c'est-à-dire un nombre arbitraire de tours dans la conception du chiffrement. Nous cherchons donc dans cette partie à analyser la dimension possible de $W_L(D)$ lorsque D est un sous-ensemble de $\mathbb{F}_2^n$ de taille arbitraire.

Pour situer plus précisément l'intérêt de l'étude qui va suivre, il est nécessaire de se rappeler que le cryptographe qui conçoit un chiffrement cherche à s'assurer que la dimension de l'espace de $W_L(D)$ est la plus grande possible, sachant que si celle-ci est égale à la taille des blocs, alors il n'existe pas d'invariant à la fois pour l'étage linéaire et pour la couche de substitution, et cela quelque soit le choix (non-trivial) de la boîte-S. Il est donc nécessaire de garder en tête que le cas optimal pour le concepteur est celui où $W_L(D) = \mathbb{F}_2^n$.

Nous considérons donc que dans toute la suite $D = \{c_1, \ldots, c_t\}$. La valeur de la dimension maximale de $W_L(D)$ s'avère être liée à la *forme canonique rationnelle* de l'étage linéaire (et au cardinal de D évidemment). Ainsi, il est nécessaire ici de faire un rappel d'algèbre linéaire.

Proposition 3.30 (Forme canonique rationnelle, facteurs invariants)**.** *[DF04, Page 476] Soit L une permutation linéaire sur $\mathbb{F}_2^n$. Il existe une base de $\mathbb{F}_2^n$ dans laquelle la matrice de L est de la forme*

$$\begin{pmatrix} C(Q_r) & & & \\ & C(Q_{r-1}) & & \\ & & \ddots & \\ & & & C(Q_1) \end{pmatrix}$$

où les Q_i sont des polynômes tels que $Q_r | Q_{r-1} | \cdots | Q_1$. De plus, le polynôme Q_1 est égal au polynôme minimal de L. Dans cette décomposition, les Q_i sont appelés les facteurs invariants de L.

La forme canonique rationnelle est à différencier de la forme utilisée dans la preuve de la proposition 3.27 où l'on factorise en fonction des facteurs irréductibles du polynôme minimal, ce qui est *fondamentalement* différent de la forme canonique rationnelle.

Par exemple, la forme canonique rationnelle de la couche linéaire utilisée dans Prince est la suivante :

$$\begin{pmatrix} C(Q_8) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C(Q_7) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C(Q_6) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C(Q_5) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C(Q_4) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C(Q_3) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & C(Q_2) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C(Q_1) \end{pmatrix}$$

avec

$$\begin{aligned} Q_1(X) &= Q_2(X) = \mathsf{Min}_L(X) \\ &= X^{20} + X^{18} + X^{16} + X^{14} + X^{12} + X^8 + X^6 + X^4 + X^2 + 1 \\ Q_3(X) &= Q_4(X) = X^8 + X^6 + X^2 + 1 = (X+1)^4(X^2 + X + 1)^2 \\ Q_5(X) &= Q_6(X) = Q_7(X) = Q_8(X) = (X+1)^2 \, . \end{aligned}$$

Un de nos résultats principaux est que les facteurs invariants de la couche linéaire déterminent la dimension maximale de $W_L(c_1, \ldots, c_t)$, au travers du théorème fondamental suivant.

Théorème 3.31. *Soit $Q_1, \ldots, Q_r$ les facteurs invariants d'une permutation linéaire L sur $\mathbb{F}_2^n$ et $t \leq r$. Alors*

$$\max_{c_1,\ldots,c_t \in \mathbb{F}_2^n} \dim W_L(c_1, \ldots, c_t) = \sum_{i=1}^{t} \deg Q_i \, .$$

Ce théorème implique la nécessité de prendre au moins $r+1$ constantes de tour différentes dans le chiffrement où r est le nombre de facteurs invariants de L afin de pouvoir engendrer tout l'espace et de se prémunir des attaques par invariant.

La preuve de ce théorème étant assez longue, nous la séparons en plusieurs lemmes. Tout d'abord, nous commençons par représenter L sous sa forme canonique rationnelle définie à la proposition 3.30. On note alors $V_1, \ldots, V_r$ les sous-espaces invariants par L qui sont en somme directe, tels que $\mathbb{F}_2^n = \bigoplus_{i=1}^{r} V_i$ et que l'application linéaire induite par L restreinte à chaque V_i (application notée $L_{|V_i}$) pour i allant de 1 à r soit représentée par la matrice compagnon $C(Q_i)$ où les Q_i sont les facteurs invariants de L. De plus, pour tout $1 \leq i \leq r$, on définit $\mathbf{e}_{V_i}$ comme un vecteur de V_i tel que V_i est engendré par e_{V_i} et l'action de L, *i.e.* $V_i = \langle L^k(e_{V_i}), 0 \leq k < \deg Q_i \rangle$, et $\mathsf{ord}_{L_{|V_i}}(e_{V_i}) = Q_i$ (l'existence de tels éléments est assurée par la proposition 3.26).

Dans un premier temps, nous pouvons prouver le lemme suivant.

Lemme 3.32. *Soit $1 \leq t \leq r$, L une application linéaire inversible, alors*

$$\max_{c_1,\ldots,c_t \in \mathbb{F}_2^n} \dim W_L(c_1, \ldots, c_t) \geq \sum_{i=1}^{t} \deg Q_i$$

Démonstration. En choisissant $c_1 = e_{V_1}$, on obtient que $W_L(c_1) = V_1$. Par définition de V_1, on a donc $\dim W_L(c_1) = \deg Q_1$. Pour choisir c_2, on s'intéresse alors à l'application $L_{|V_2 \oplus \cdots \oplus V_r}$. Celle-ci est de polynôme minimal Q_2 par construction, ce qui nous permet de choisir la constante $c_2 = e_{V_2}$ de manière à ce que $W_L(c_1, c_2) = V_1 \oplus V_2$. En itérant ce processus, on construit pas à pas un espace vectoriel $W_L(c_1, \ldots, c_t)$ qui est égal à $\bigoplus_{i=1}^{t} W_L(c_i) = \bigoplus_{i=1}^{t} V_i$ et qui est donc de dimension $\sum_{i=1}^{t} \deg Q_i$. □

Cependant, il nous reste à prouver l'égalité, c'est-à-dire l'inégalité dans l'autre sens ; pour cela nous utilisons les deux lemmes suivants.

Lemme 3.33. *Soit c appartenant à $\mathbb{F}_2^n = \bigoplus_{i=1}^{r} V_i$ décrit, grâce à la somme directe comme $c = \sum_{i \in \mathcal{I}} u_i$ avec $\mathcal{I} \subset \{1, \ldots, r\}$ et $u_i \in V_i \backslash \{0\}$. Alors*

$$W_L(c) \subseteq W_L(\bar{c})$$

où $\bar{c} = \sum_{i \in \mathcal{I}} e_{V_i}$.

Démonstration. Soit v un vecteur de $W_L(c)$. Alors

$$\begin{aligned} v &= \sum_{j \in \mathbb{N}} \alpha_j L^j(c) = \sum_{j \in \mathbb{N}} \alpha_j L^j(\sum_{i \in \mathcal{I}} u_i) = \sum_{j \in \mathbb{N}} \sum_{i \in \mathcal{I}} \alpha_j L^j(u_i) \\ &= \sum_{j \in \mathbb{N}} \sum_{i \in \mathcal{I}} \alpha_j L^j(\sum_{k \in \mathbb{N}} \beta_k L^k(\mathbf{e_{V_i}})) = \sum_{j \in \mathbb{N}} \sum_{i \in \mathcal{I}} \sum_{k \in \mathbb{N}} \alpha_j \beta_k L^{j+k}(\mathbf{e_{V_i}}) \\ &= \sum_{j \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha_j \beta_k L^{j+k}(\sum_{i \in \mathcal{I}} \mathbf{e_{V_i}}) = \sum_{j \in \mathbb{N}} \sum_{k \in \mathbb{N}} \alpha_j \beta_k L^{j+k}(\bar{c}) \in W_L(\bar{c}). \end{aligned}$$

□

En appliquant le même raisonnement à $v = \sum_{k=1}^{t} \sum_{j \in \mathbb{N}} \alpha_j L^j(c_k)$ quand on a plusieurs constantes, on obtient le même résultat, c'est-à-dire que pour tout ensemble de vecteurs $c_1, \ldots, c_t$ de $\mathbb{F}_2^n$, on a $W_L(c_1, \ldots, c_t) \subseteq W_L(\bar{c_1}, \ldots, \bar{c_t})$. Comme il s'agit de prouver l'inégalité dans l'autre sens, nous pouvons supposer, sans perte de généralité que $c_k = \sum_{i=1}^{r} \lambda_{k,i} e_{V_i}$ pour tout k allant de 1 à t avec $\lambda_{k,i} \in \mathbb{F}_2$. De plus, à chaque t-uplet, avec $t \leq r$, $(c_1, \ldots, c_t) \in (\mathbb{F}_2^n)^t$, on associe la matrice de taille $t \times r$ notée $\mathbf{M}_{(c_1, \ldots, c_t)}$ telle que $[\mathbf{M}_{(c_1, \ldots, c_t)}]_{k,i} = \lambda_{k,i}$ pour tout i allant de 1 à r et k allant de 1 à t. Ceci nous permet de montrer le lemme suivant :

Lemme 3.34. *Soit $(c_1, \ldots, c_t)$ un t-uplet d'éléments de $\mathbb{F}_2^n$ tel que*

$$c_k = \sum_{i=1}^{r} \lambda_{k,i} e_{V_i}$$

et soit $\mathbf{M}_{(c'_1, \ldots, c'_t)}$ n'importe quelle matrice obtenue par opérations élémentaires sur les colonnes de $\mathbf{M}_{(c_1, \ldots, c_t)}$. Alors pour $(c'_1, \ldots, c'_t)$ le t-uplet correspondant à $\mathbf{M}_{(c'_1, \ldots, c'_t)}$, on a

$$W_L(c'_1, \ldots, c'_t) = W_L(c_1, \ldots, c_t)$$

Démonstration. Pour une matrice binaire, une opération élémentaire peut être :

— soit un échange de deux colonnes ;

— soit une addition d'une colonne à une autre.

Pour la première opération élémentaire qui consiste à échanger deux colonnes arbitraires indexées par $k_1 < k_2$, cela revient à considérer la matrice $\mathbf{M}_{(c_1,\dots,c_{k_1},\dots,c_{k_2},\dots,c_t)}$ et la matrice $\mathbf{M}_{(c_1,\dots,c_{k_2},\dots,c_{k_1},\dots,c_t)}$. Par définition même de $W_L(D)$, il est clair que

$$W_L(c_1,\dots,c_{k_1},\dots,c_{k_2},\dots,c_t) = W_L(c_1,\dots,c_{k_2},\dots,c_{k_1},\dots,c_t)\ .$$

Il nous reste donc à montrer que la propriété est vérifiée pour l'opération qui consiste à additionner une colonne à une autre. Comme

$$W_L(c_1,\dots,c_t) = \sum_{k=1}^{t} W_L(c_k)\ ,$$

il nous suffit de montrer que pour tout c_{k_1} et c_{k_2} dans $\mathbb{F}_2^n$, $W_L(c_{k_1}) + W_L(c_{k_2}) = W_L(c_{k_1} + c_{k_2}) + W_L(c_{k_2})$.

Soit $u \in W_L(c_{k_1}) + W_L(c_{k_2})$, alors

$$\begin{aligned} u &= \sum_{j\in\mathbb{N}} (\alpha_j L^j(c_{k_1}) + \beta_j L^j(c_{k_2})) \\ &= \sum_{j\in\mathbb{N}} (\alpha_j L^j(c_{k_1}) + \alpha_j L^j(c_{k_2}) + \alpha_j L^j(c_{k_2}) + \beta_j L^j(c_{k_2})) \\ &= \sum_{j\in\mathbb{N}} (\alpha_j L^j(c_{k_1} + c_{k_2}) + (\alpha_j + \beta_j) L^j(c_{k_2})) \in W_L(c_{k_1} + c_{k_2}) + W_L(c_{k_2}). \end{aligned}$$

L'inclusion réciproque se montre de la même manière, ce qui conclut la preuve de ce lemme. □

Grâce à ces trois lemmes, nous sommes finalement en mesure de prouver le théorème 3.31.

Démonstration. (Théorème 3.31) Grâce au lemme 3.32, on a déjà l'inégalité $\geq$. Il nous reste donc à montrer l'inégalité dans l'autre sens ($\leq$). Soit $c_1,\dots,c_t$ t éléments appartenant à $\mathbb{F}_2^n$, avec $t \leq r$ où r est le nombre de facteurs invariants. D'après le lemme 3.33, on a $W_L(c_1,\dots,c_t) \subseteq W_L(\bar{c_1},\dots,\bar{c_t})$ pour $\bar{c_k} = \sum_{i=1}^{r} \lambda_{ki} e_{V_i}$ avec $\lambda_{ki} \in \mathbb{F}_2$.

On considère alors la matrice binaire $\mathbf{M}_{(\bar{c_1},\dots,\bar{c_t})}$ définie comme précédemment. En utilisant les opérations élémentaires sur les colonnes de cette matrice, on peut se ramener à une matrice notée $\mathbf{M}_{(\tilde{c_1},\dots,\tilde{c_t})}$ sous la *forme échelonnée réduite*. D'après le lemme 3.34, on sait que

$$W_L(\bar{c_1},\dots,\bar{c_t}) = W_L(\tilde{c_1},\dots,\tilde{c_t})\ .$$

De plus, on a

$$W_L(\tilde{c_1}, \ldots, \tilde{c_t}) = \sum_{k=1}^{t} W_L(\tilde{c_k}) \ .$$

Comme la matrice $\mathbf{M}_{(\tilde{c_1},\ldots,\tilde{c_t})}$ est sous sa forme échelonnée réduite, on a, pour tout $1 \leq k \leq t$, $\tilde{c_k} = \sum_{i=1}^{r} \tilde{\lambda_{ki}} e_{V_i}$, avec $\tilde{\lambda_{ki}} = 0$ pour tout $k < i$ ce qui implique que pour tout k allant de 1 à t, $W_L(\tilde{c_k}) = W_{|L_{V_k \oplus \cdots \oplus V_r}}(\tilde{c_k})$.

Finalement,

$$W_L(\tilde{c_1}, \ldots, \tilde{c_t}) = \sum_{k=1}^{t} W_{|L_{V_k \oplus \cdots \oplus V_r}}(\tilde{c_k}) \ .$$

Comme le polynôme minimal de l'application linéaire $L_{|V_k \oplus \cdots \oplus V_r}$ vaut Q_k pour tout $k \leq r$, on obtient finalement que :

$$\dim W_L(c_1, \ldots, c_t) \leq \dim W_L(\bar{c_1}, \ldots, \bar{c_t}) = \dim \sum_{k=1}^{t} W_{L_{|V_k \oplus \cdots \oplus V_r}}(\tilde{c_k})$$

$$\leq \sum_{k=1}^{t} \deg Q_k \ .$$

□

3.4.2.1 Exemples

Afin d'illustrer ce résultat, nous expliquons ce qui se passe spécifiquement pour certains SPN dont les facteurs invariants sont relativement nombreux.

Prince. La couche linéaire du chiffrement Prince admet 8 facteurs invariants :

$$\begin{aligned} Q_1(X) &= Q_2(X) = \mathsf{Min}_L(X) \\ &= X^{20} + X^{18} + X^{16} + X^{14} + X^{12} + X^8 + X^6 + X^4 + X^2 + 1 \\ Q_3(X) &= Q_4(X) = X^8 + X^6 + X^2 + 1 = (X+1)^4(X^2 + X + 1)^2 \\ Q_5(X) &= Q_6(X) = Q_7(X) = Q_8(X) = (X+1)^2 \end{aligned}$$

D'après notre théorème fondamental (théorème 3.31), on sait donc que pour un ensemble D de cardinal 5, la dimension maximale que l'on peut avoir est $20 + 20 + 8 + 8 + 2 = 58$, alors que l'on obtient une dimension 56 pour les constantes de tour choisies par les concepteurs. On constate ici que le choix des constantes de tour dans Prince n'est donc pas optimal, mais aussi que peu importe le choix de ces constantes, il n'est pas possible d'atteindre la dimension 64 avec le nombre de tours utilisé dans ce chiffrement. Il faudrait en effet un ensemble de cardinal 8 pour cela.

Mantis et Midori-64. Pour ces chiffrements, la couche linéaire admet, elle, 16 facteurs invariants :

$$Q_1(X) = \cdots = Q_8(X) = (X+1)^6 \text{ et } Q_9(X) = \cdots = Q_{16}(X) = (X+1)^2 .$$

D'après l'ensemble D de cardinal 7 (respectivement 8) obtenu par le choix des constantes de tour fait par les concepteurs de Mantis_7 (respectivement Mantis_8), on engendre un espace de dimension 42 (respectivement 48) ce qui est optimal : on ne peut pas faire mieux avec le même nombre de tours et la même couche linéaire. De plus, pour ces chiffrements, on sait qu'il faut au moins 16 éléments dans D pour atteindre la dimension maximale 64 (c'est-à-dire 17 constantes de tour) et ainsi couvrir tout l'espace.

Dans le cas de Midori, les constantes de tour ont été particulièrement mal choisies, puisque celles-ci sont non-nulles uniquement sur les bits de poids faible, impliquant que l'espace $W_L(D)$ ne peut avoir une dimension plus grande que 16. Ce phénomène est la principale faiblesse de Midori au regard des attaques par invariant, et c'est donc ici un mauvais choix de constantes qui rend le chiffrement faible, puisque Mantis utilise la même couche linéaire que Midori.

Skinny. Le cas de Skinny est relativement simple : la couche linéaire est une matrice binaire de taille 16 et son polynôme minimal est $X^{16} + 1 = (X+1)^{16}$. Dans ces conditions, si on regarde l'application de la matrice sur $\mathbb{F}_2^{64}$, on peut assurer que Skinny admet 4 facteurs invariants tous égaux au polynôme minimal. Ainsi, il est nécessaire d'avoir 4 éléments dans l'espace D pour atteindre la dimension 64.

Afin d'avoir une explication plus visuelle, nous avons calculé la dimension maximale que l'on peut atteindre avec t constantes de tour pour les couches linéaires des chiffrements Skinny, Mantis et Midori. Ces résultats illustrant le théorème 3.31 sont décrits à la figure 3.4. On constate assez rapidement que les couches linéaires ont une grande influence sur ce comportement, puisque selon les applications linéaires utilisées, le nombre de tour nécessaires pour couvrir tout l'espace varie énormément. Il est aussi important de remarquer que ce phénomène n'est en rien lié aux critères classiques employés pour choisir la couche linéaire.

3.4.3 Choisir des constantes de tour aléatoires

Les attaques par invariant étant un type d'attaque récent, leur compréhension était assez limité. Avec les résultats précédents, nous avons pu donner une explication à l'existence de ces invariants. Plus intéressant encore, nous donnons des critères sur le choix des constantes de tour. Avant notre travail, il n'existait pas de critère sur le choix des constantes de tour, excepté le fait qu'elles doivent être distinctes pour rendre les tours différents les uns des autres et résister par exemple aux "slide attacks" [BW99, BW00]. Comme les constantes de tour étaient souvent choisies arbitrairement, nous nous intéressons dans cette section à déterminer la probabilité qu'un ensemble de constantes uniformément aléatoires

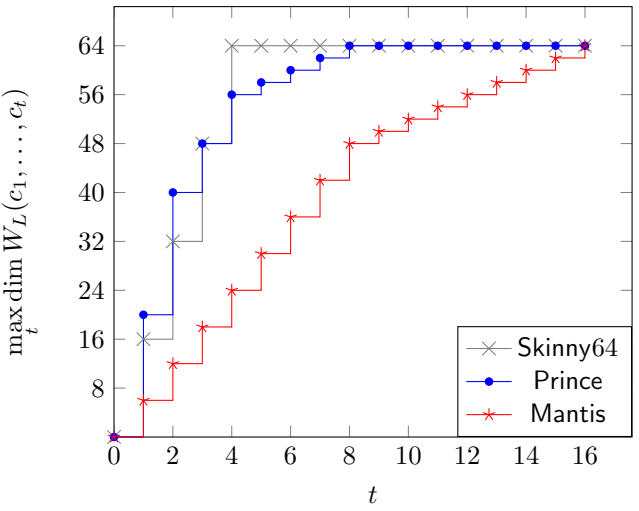

Figure 3.4 – Dimension maximale possible pour $W_L(c_1, \ldots, c_t)$ pour t valeurs $c_1, \ldots, c_t$ pour les chiffrements Skinny-64, Prince et Mantis.

D engendre un espace $W_L(D)$ de dimension maximale, *i.e.* égale à la taille de bloc.

Pour commencer, on s'intéresse au cas particulier où D est de cardinal 1 : $D = \{c\}$.

Proposition 3.35. *Soit L une permutation linéaire de $\mathbb{F}_2^n$ et Min_L son polynôme minimal factorisé de la manière suivante :*

$$\mathsf{Min}_L(X) = M_1(X)^{e_1} M_2(X)^{e_2} \cdots M_k(X)^{e_k}$$

où $M_1, \ldots, M_k$ sont des polynômes distincts et irréductibles sur $\mathbb{F}_2$. Alors la probabilité pour un élément choisi aléatoirement de manière uniforme $c \in \mathbb{F}_2^n$ d'obtenir $\dim W_L(c) = \deg \mathsf{Min}_L$ est

$$\Pr_{c \xleftarrow{\$} \mathbb{F}_2^n}[\dim W_L(c) = \deg \mathsf{Min}_L] = \prod_{i=1}^{k} \left(1 - \frac{1}{2^{\mu_i \deg M_i}}\right) ,$$

où μ_i est le nombre de facteurs invariants de L multiples de $M_i^{e_i}$.

Afin de prouver cette proposition, nous aurons besoin à la fois de la forme canonique rationnelle, définie par le théorème 3.30 page 46, mais aussi de la décomposition en diviseurs élémentaires, définie par le corollaire 3.29 page 42. Ces deux décompositions sont uniques et décrivent la même application linéaire mais ne captent pas les mêmes propriétés, comme le.a lecteur.rice a pu s'en rendre compte au travers de l'exemple de la couche linéaire de Prince.

Démonstration. On utilise la décomposition en fonction des diviseurs élémentaires définie au corollaire 3.29 page 42 décrite dans [Her75, Page 308]. $\mathbb{F}_2^n$ peut être décomposé de la manière suivante :

$$\mathbb{F}_2^n = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{r_i} V_{i,j}$$

de telle sorte que la matrice induite par la transformation linéaire L et appliquée sur $V_{i,j}$ soit la matrice compagnon associée au polynôme $M_i(X)^{\ell_{i,j}}$ où, pour chaque i, les valeurs $\ell_{i,j}$ pour $1 \leq j \leq r_i$ forment une suite décroissante d'entiers, avec $\ell_{i,1} = e_i$.

Donc, le polynôme minimal relativement à L de n'importe quel élément u appartenant à $V_{i,j}$ est nécessairement un diviseur de $M_i(X)^{\ell_{i,j}}$. On décompose ensuite n'importe quel élément de $\mathbb{F}_2^n$, selon la somme directe fondée sur les diviseurs élémentaires :

$$c = \sum_{i=1}^{k} \sum_{j=1}^{r_i} u_{i,j}$$

où $u_{i,j} \in V_{i,j}$. Finalement, on déduit que $\mathsf{ord}_L(c) = \mathsf{Min}_L$ si et seulement si, pour tout $1 \leq i \leq k$, il existe un indice j tel que $\mathsf{ord}_L(u_{i,j}) = M_i^{e_i}$. Ceci ne peut évidemment arriver que si $\ell_{i,j} = e_i$. On définit alors $\mu_i = \max\{j : \ell_{i,j} = e_i\}$. Donc on se restreint seulement aux sous-espaces vectoriels $V_{i,j}$ tels que $j \leq \mu_i$.

De plus, en utilisant cette décomposition de L en fonction des diviseurs élémentaires, on sait que les facteurs invariants de la décomposition de L sous sa forme canonique rationnelle sont donnés par :

$$Q_v = \prod_{i=1}^{k} M_i^{\ell_{i,v}}$$

où $\ell_{i,v} = 0$ si $v > r_i$. Ainsi, on observe que la valeur μ_i définie précédemment peut aussi être vue comme le nombre de facteurs invariants (obtenus dans la forme canonique rationnelle) Q_v qui sont multiples du polynôme $M_i^{e_i}$.

On définit maintenant l'événement suivant :

$$E_{i,j} : \quad \mathsf{ord}_L(u_{i,j}) = M_i^{\ell_{i,j}}.$$

Alors,

$$\Pr_{c \xleftarrow{\$} \mathbb{F}_2^n}[\dim W_L(c) = \deg \mathsf{Min}_L] = \prod_{i=1}^{k} \Pr\left[\bigcup_{j=1}^{\mu_i} E_{i,j}\right].$$

Pour un i donné, l'application de L restreinte à $V_{i,j}$ correspond à la matrice compagnon associée au polynôme $M_i^{e_i}$ pour tout $j \leq \mu_i$. Ainsi, pour un i fixé, la probabilité de l'événement $E_{i,j}$ est donc la même pour tout $j \leq \mu_i$. Cette probabilité correspond à la proportion de polynômes de degré plus petit que $\deg(M_i^{\ell_{i,j}})$. En effet, comme nous l'avons déjà expliqué dans la preuve de

la proposition 3.26, il y a une correspondance entre les éléments appartenant aux sous-espaces $V_{i,j}$ et les états initiaux possibles d'un LFSR de polynôme caractéristique $M_i^{\ell_{i,j}}$. De plus, le nombre de polynômes qui sont premiers avec un certain polynôme P est donné par la fonction indicatrice d'Euler ϕ :

$$\phi(P) := |\{f \in \mathbb{F}_2[X] | \deg(f) < \deg(P), \mathsf{pgcd}(f, P) = 1\}| .$$

Si de plus P est irréductible, alors pour n'importe quelle puissance de P, on a $\phi(P^k) = 2^{(k-1)\deg P}(2^{\deg P} - 1)$. On en déduit que

$$\Pr[E_{i,j}] = \frac{\phi(M_i^{\ell_{i,j}})}{2^{\ell_{i,j}\deg M_i}}$$

$$\Pr[E_{i,j}] = \frac{2^{(\ell_{i,j}-1)\deg M_i}(2^{\deg M_i} - 1)}{2^{\ell_{i,j}\cdot\deg M_i}}$$

$$\Pr[E_{i,j}] = 1 - \frac{1}{2^{\deg M_i}}.$$

Finalement, en utilisant le principe d'inclusion-exclusion, on obtient que

$$\Pr\left[\bigcup_{j=1}^{\mu_i} E_{i,j}\right] = \sum_{j=1}^{\mu_i}(-1)^{j-1}\binom{\mu_i}{j}\left(1 - \frac{1}{2^{\deg M_i}}\right)^j$$

$$\Pr\left[\bigcup_{j=1}^{\mu_i} E_{i,j}\right] = \left(1 - \frac{1}{2^{\mu_i \deg M_i}}\right).$$

□

LED. On a vu que la couche linéaire de LED a un polynôme minimal de degré 64 (et que donc une seule constante suffit à engendrer tout l'espace). Le polynôme minimal de LED étant le polynôme suivant :

$$\mathsf{Min}_L(X) = (X^8 + X^7 + X^5 + X^3 + 1)^4(X^8 + X^7 + X^6 + X^5 + X^2 + 1)^4 ,$$

il admet donc 2 facteurs irréductibles, chacun de degré 8. D'après la proposition 3.35, on obtient que la probabilité qu'une constante prise aléatoirement engendre par L tout l'espace $\mathbb{F}_2^{64}$ est

$$\Pr[W_L(c) = \mathbb{F}_2^{64}] = (1 - 2^{-8})^2 \approx 0.9922.$$

3.4.3.1 Avec plusieurs constantes

Finalement, nous pouvons généraliser la proposition 3.35 lorsque nous prenons t valeurs aléatoires.

Théorème 3.36. *Soit L une permutation linéaire sur $\mathbb{F}_2^n$ et Min_L son polynôme minimal factorisé de la manière suivante :*

$$\mathsf{Min}_L(X) = M_1(X)^{e_1} M_2(X)^{e_2} \cdots M_k(X)^{e_k}$$

où $M_1, \ldots, M_k$ sont des polynômes distincts et irréductibles sur $\mathbb{F}_2$. Alors, la probabilité que $W_L(c_1, \ldots, c_t)$ soit égal à $\mathbb{F}_2^n$ vaut

$$\Pr_{c_1,\ldots,c_t \xleftarrow{\$} \mathbb{F}_2^n}[W_L(c_1, \cdots, c_t) = \mathbb{F}_2^n] = \prod_{j=1}^{k} \prod_{i_j=0}^{r_j-1} \left(1 - \frac{1}{2^{(t-i_j)\deg(M_j)}}\right) ,$$

où r_j est le nombre de facteurs invariants de L multiples de M_j.

Lorsque $t < r$ avec r le nombre de facteurs invariants, le produit devient nul, ce qui correspond évidemment à notre théorème 3.31. Par souci de clarté de la preuve de ce théorème, nous prouvons au préalable la proposition et le lemme qui suivent. Nous utilisons encore la décomposition de L en diviseurs élémentaires et non sa décomposition en facteurs invariants. Nous commençons donc par prouver la propriété lorsque le polynôme minimal de l'application linéaire est une puissance d'un polynôme irréductible.

Lorsque le polynôme minimal de l'application linéaire est une puissance d'un polynôme irréductible, les diviseurs élémentaires sont les mêmes que les facteurs invariants, ce qui nous permet de démontrer la proposition suivante.

Proposition 3.37. *Soit V un espace vectoriel sur $\mathbb{F}_2$. Soit L une application linéaire de V dans lui-même avec exactement r facteurs invariants, tels que le polynôme minimal de L est de la forme P^e où P est un polynôme irréductible. Alors, la probabilité que $W_L(c_1, \ldots, c_t)$ soit égal à V est égale à*

$$\Pr_{c_1,\ldots,c_t \xleftarrow{\$} V}[W_L(c_1, \cdots, c_t) = V] = \prod_{i=0}^{r-1} \left(1 - \frac{1}{2^{(t-i)\deg(P)}}\right)$$

Démonstration. Soit $P^{e_1}, \ldots, P^{e_r}$ avec $e = e_1 \geq e_2 \geq \cdots \geq e_r$ les facteurs invariants de L. Alors on décompose V en la somme directe suivante : $V = V_1 \oplus V_2 \oplus \cdots \oplus V_r$ où $L_{|V_i}$ est représentée par la matrice compagnon $C(P^{e_i})$. Donc, pour chaque c_i, il existe $(u_{i,1}, u_{i,2}, \ldots, u_{i,r}) \in V_1 \times V_2 \times \cdots \times V_r$ tel que $c_i = u_{i,1} + u_{i,2} + \cdots + u_{i,r}$.

Dans un premier temps, nous montrons que si $W_L(c_1, \ldots, c_t) = V$, alors il existe une constante c_i pour un certain i entre 1 et t, telle que $W_L(u_{i,1}) = V_1$. Tout d'abord, par construction de la somme directe, les éléments $u_{i,j}$ pour $j \geq 2$ n'appartiennent pas au sous-espace V_1. Donc, si $W_L(c_1, \ldots, c_t) = V$, alors nécessairement $W_L((u_{i,1})_{1 \leq i \leq t})$ doit couvrir tout l'espace V_1. De plus, si $u_{i,1}$ et $u_{j,1}$ sont tels que $W_L(u_{i,1}) \subsetneq V_1$ et $W_L(u_{j,1}) \subsetneq V_1$, alors on peut assurer que $W_L(u_{i,1}, u_{j,1}) \subsetneq V_1$. En effet, comme $W_L(u_{i,1}) \subsetneq V_1$, le polynôme minimal de $u_{i,1}$ relativement à L est égal à P^a, pour un certain a strictement plus petit que e_1. De manière équivalente, le polynôme minimal de $u_{j,1}$ relativement à L est

aussi de la forme P^b pour un certain $b < e_1$. On suppose alors, sans perdre de généralité que $a \leq b$. Rappelons que, pour n'importe quelle application linéaire L et n'importe quel entier ℓ, $\ker(M^\ell) \subseteq \ker(M^{\ell+1})$. Ici, $u_{i,1}$ et $u_{j,1}$ sont tels que $W_L(u_{i,1}) \subseteq \ker(P^a(L))$ et $W_L(u_{j,1}) \subseteq \ker(P^b(L))$. On obtient alors que

$$W_L(u_{i,1}) \subseteq \ker(P^a(L)) \subseteq \ker(P^b(L)) \subsetneq V_1$$

et donc

$$W_L(u_{i,1}, u_{j,1}) \subseteq \ker(P^b(L)) \subsetneq V_1 \ ,$$

ce qui implique qu'au moins un des ensembles $W_L(u_{i,1})$ doit être égal au sous-espace V_1.

Pour conclure la preuve de cette proposition, on montre le résultat par récurrence sur le nombre de facteurs invariants r.

— $\boldsymbol{r = 1}$. D'après l'observation faite précédemment, $W_L(c_1, \ldots, c_t) = V$ si et seulement si au moins l'un des éléments parmi $c_1, \ldots, c_t$ admet P^e comme polynôme minimal relativement à L. De plus, d'après la proposition 3.35, on sait que la probabilité qu'un élément aléatoire $c \in V$ ait pour polynôme minimal P^e vaut

$$1 - \frac{1}{2^{\deg P}} \ .$$

Ainsi, comme les t constantes sont choisies indépendamment les unes des autres, la probabilité qu'aucune de ces t constantes n'admette P^e comme polynôme minimal vaut donc $(2^{-\deg P})^t$, ce qui implique que

$$\Pr_{c_1,\ldots,c_t \xleftarrow{\$} V}[W_L(c_1, \cdots, c_t) = V] = 1 - \frac{1}{2^{t \deg(P)}} \ .$$

— **Récurrence.** Maintenant, on suppose que la propriété que l'on cherche à montrer est vraie pour n'importe quelle permutation linéaire ayant $r - 1$ facteurs invariants et dont le polynôme minimal est une puissance d'un polynôme irréductible. On considère alors une application linéaire L ayant r facteurs invariants. Si $W_L(c_1, \ldots, c_t) = V$, alors au moins une des t constantes vérifie $W_L(u_{i,1}) = V_1$, cet événement apparaît avec la probabilité $1 - \frac{1}{2^{\deg P}}$. Quitte à réordonner les éléments $c_1, \ldots, c_t$, on suppose que c_1 permet de couvrir V_1, *i.e.* $W_L(u_{1,1}) = V_1$.
On considère maintenant l'application L' définie comme l'application de L sur l'espace quotienté par $V_1 = W_L(u_{1,1})$: $L' = L_{|V/W_L(c_1)}$. Comme c_1 admet Min_L comme polynôme minimal relativement à L, alors on sait que les facteurs invariants de L' sont $P^{e_2}, \ldots, P^{e_r}$ (voir par exemple dans [Gie95, Fact 2.2]).
Finalement, la probabilité de couvrir tout l'espace V est la probabilité que l'une des constantes (par exemple c_1) engendre V_1 (le premier facteur invariant) multipliée par la probabilité que l'application définie sur l'espace quotient engendre tout l'espace restant avec les autres constantes. Plus précisément, on a

$$\Pr[W_L(c_1, \cdots, c_t) = \mathbb{F}_2^n] = \left(1 - \frac{1}{2^{t \deg(P)}}\right) \Pr[W_{L'}(c'_2, \cdots, c'_t) = V/V_1]$$

où $c'_i = (c_i)_{V/V_1}$. L'application L' ayant exactement $r-1$ facteurs invariants, on peut appliquer l'hypothèse et conclure la preuve.

□

Cependant, nous n'avons montré notre théorème que pour le cas particulier où le polynôme minimal est une puissance d'un polynôme irréductible. Pour prouver le cas général, on utilise le lemme suivant.

Lemme 3.38. *Soit L une permutation linéaire de $\mathbb{F}_2^n$. On suppose qu'il existe deux sous-espaces de $\mathbb{F}_2^n$ notés V_1 et V_2, en somme directe et invariants par L et tels que $V_1 \oplus V_2 = \mathbb{F}_2^n$ et tels que les polynômes minimaux de $L_{|V_1}$ et $L_{|V_2}$ soient premiers entre eux. Alors, pour n'importe quelles constantes $c_1, \ldots, c_t \in \mathbb{F}_2^n$,*

$$W_L(c_1, \ldots, c_t) = W_L(a_1, \ldots, a_t) \oplus W_L(b_1, \ldots, b_t)$$

où (a_i, b_i) est l'unique paire de $V_1 \times V_2$ telle que $c_i = a_i + b_i$.

Démonstration. Tout d'abord, comme V_1 et V_2 sont invariants par L, il est clair que $W_L(a_1, \ldots, a_t) \cap W_L(b_1, \ldots, b_t) = \{0\}$.

Ensuite, on vérifie que $W_L(c_1, \ldots, c_t) \subseteq W_L(a_1, \ldots, a_t) \oplus W_L(b_1, \ldots, b_t)$. Soit $x \in W_L(c_1, \ldots, c_t)$, alors

$$\begin{aligned} x &= \sum_{\ell \in \mathbb{N}} \sum_{i=1}^{t} \lambda_{i,\ell} L^\ell(c_i) \\ &= \left(\sum_{\ell \in \mathbb{N}} \sum_{i=1}^{t} \lambda_{i,\ell} L^\ell(a_i) \right) + \left(\sum_{\ell \in \mathbb{N}} \sum_{i=1}^{t} \lambda_{i,\ell} L^\ell(b_i) \right) . \end{aligned}$$

Il nous reste alors à montrer que

$$W_L(a_1, \ldots, a_t) \oplus W_L(b_1, \ldots, b_t) \subseteq W_L(c_1, \ldots, c_t) .$$

Soit P_1 et P_2 les polynômes minimaux des applications $L_1 = L_{|V_1}$ et $L_2 = L_{|V_2}$ et d_1 et d_2 leurs degrés respectifs. On considère le sous-espace $W \subseteq \mathbb{F}_2^n$ défini par

$$\begin{aligned} W = &\Big\langle P_2(L^j)(c_i), 1 \leq i \leq t, 0 \leq j < d_1 \Big\rangle \\ &+ \Big\langle P_1(L^j)(c_i), 1 \leq i \leq t, 0 \leq j < d_2 \Big\rangle . \end{aligned}$$

Comme chaque $P_1(L^j)(c_i)$ (respectivement $P_2(L^j)(c_i)$) est une combinaison linéaire de vecteurs de la forme $L^\ell(c_i)$, on a bien l'inclusion $W \subseteq W_L(c_1, \ldots, c_t)$. Par ailleurs, comme $b_i \in V_2$ et que $P_2(X^j)$ est un multiple du polynôme minimal de L_2 (par définition de P_2), on a

$$P_2(L^j)(c_i) = P_2(L^j)(a_i + b_i) = P_2(L^j)(a_i) + P_2(L^j)(b_i) = P_2(L^j)(a_i) ,$$

et de manière équivalente

$$P_1(L^j)(c_i) = P_1(L^j)(b_i) .$$

Ainsi,

$$W = \Big\langle P_2(L^j)(a_i),\ 1 \le i \le t,\ 0 \le j < d_1 \Big\rangle + \Big\langle P_1(L^j)(b_i),\ 1 \le i \le t,\ 0 \le j < d_2 \Big\rangle .$$

On considère maintenant l'application définie par

$$\phi_1 : \begin{array}{ccc} V_1 & \to & V_1 \\ x & \mapsto & P_2(L)(x) \end{array} .$$

Cette application est linéaire par construction et est bijective. En effet, on montre que son noyau est réduit à $\{0\}$: soit $x \in \ker \phi_1$, alors $P_2(L)(x) = 0$, donc le polynôme minimal de x relativement à L, $\mathsf{ord}_L(x)$, est un diviseur de P_2. Or x appartient à V_1 puisque l'application ϕ_1 est définie de V_1 dans lui-même, donc $\mathsf{ord}_L(x)$ est aussi un diviseur de P_1. Comme P_1 et P_2 sont, par hypothèse du lemme, premiers entre eux, on en déduit que $x = 0$.

De plus, $W_L(a_1, \ldots, a_t)$ est, par définition, invariant par L, et donc invariant par ϕ_1. Dans ces conditions, on obtient que

$$\begin{aligned} &\Big\langle P_2(L^j)(a_i), 1 \le i \le t, 0 \le j < d_1 \Big\rangle \\ &= P_2(L)\left(\Big\langle L^j(a_i), 1 \le i \le t, 0 \le j < d_1 \Big\rangle\right) \\ &= \phi_1(W_L(a_1, \ldots, a_t)) = W_L(a_1, \ldots, a_t) . \end{aligned}$$

Et, de manière équivalente,

$$\Big\langle P_1(L^j)(b_i), 1 \le i \le t, 0 \le j < d_2 \Big\rangle = W_L(b_1, \ldots, b_t) .$$

Finalement, notre espace W défini précédemment est donc égal à

$$W = W_L(a_1, \ldots, a_t) \oplus W_L(b_1, \ldots, b_t) .$$

Comme $W \subseteq W_L(c_1, \ldots, c_t)$, on en déduit que

$$W_L(a_1, \ldots, a_t) \oplus W_L(b_1, \ldots, b_t) \subseteq W_L(c_1, \ldots, c_t) .$$

□

Maintenant, nous sommes en mesure de prouver le théorème 3.36.

Démonstration. (Du théorème 3.36, page 55) Soit L une permutation linéaire quelconque sur $\mathbb{F}_2^n$. On décompose alors son polynôme minimal en produit de facteurs premiers :

$$\mathsf{Min}_L(X) = M_1(X)^{e_1} M_2(X)^{e_2} \cdots M_k(X)^{e_k}$$

où tous les polynômes M_i sont irréductibles. Alors, d'après la décomposition basée sur les diviseurs élémentaires [Her75, Page 308], on sait qu'il existe k

sous-espaces notés $U_1, \ldots, U_k$ invariants par L tels que $\mathbb{F}_2^n = U_1 \oplus U_2 \oplus \cdots \oplus U_k$ et tels que le polynôme minimal de chaque application linéaire induite par L sur U_i soit égal à $M_i^{e_i}$. On considère maintenant t constantes $c_1, \ldots, c_t$ choisies aléatoirement dans $\mathbb{F}_2^n$. Alors, d'après le lemme 3.38, on sait que

$$W_L(c_1, \ldots c_t) = \bigoplus_{i=1}^{k} W_L(u_{i,1}, \ldots, u_{i,t})$$

où $(u_{1,j}, \ldots, u_{k,j})$ est l'unique k-uplet appartenant au produit cartésien $U_1 \times \cdots \times U_k$ tel que $c_j = \sum_{i=1}^{k} u_{i,j}$ pour tout $1 \leq j \leq t$. On en déduit alors que

$$\begin{aligned} &\Pr_{c_1,\ldots,c_t \xleftarrow{\$} \mathbb{F}_2^n}[W_L(c_1, \ldots, c_t) = \mathbb{F}_2^n] \\ &= \prod_{i=1}^{k} \Pr_{u_{i,1},\ldots,u_{i,t} \xleftarrow{\$} U_i}[W_{L_i}(u_{i,1}, \ldots, u_{i,t}) = U_i] \ . \end{aligned}$$

D'après la proposition 3.37, on sait que pour tout $1 \leq i \leq k$,

$$\Pr_{u_{i,1},\ldots,u_{i,t} \xleftarrow{\$} V_i}[W_{L_i}(u_{i,1}, \ldots, u_{i,t}) = V_i] = \prod_{j=0}^{r_i - 1} \left(1 - \frac{1}{2^{(t-j)\deg(M_i)}}\right) \ ,$$

où r_i est le nombre de facteurs invariants de L qui sont multiples de M_i. Ce qui implique le résultat énoncé. □

Comme nous l'avons fait pour les précédentes observations, nous appliquons ce résultat à certains SPNs.

Prince. Le polynôme minimal de la couche linéaire de Prince est

$$\begin{aligned} \mathsf{Min}_L(X) &= X^{20} + X^{18} + X^{16} + X^{14} + X^{12} + X^8 + X^6 + X^4 + X^2 + 1 \\ &= (X^4 + X^3 + X^2 + X + 1)^2 (X^2 + X + 1)^4 (X + 1)^4 \ . \end{aligned}$$

Ce polynôme admet donc trois facteurs invariants irréductibles :

$$\begin{aligned} M_1(X) &= X^4 + X^3 + X^2 + X + 1 \\ M_2(X) &= X^2 + X + 1 \\ M_3(X) &= (X + 1) \ . \end{aligned}$$

De plus, les 8 facteurs invariants de L sont

$$\begin{aligned} Q_1(X) &= Q_2(X) = \mathsf{Min}_L(X) \\ Q_3(X) &= Q_4(X) = (X + 1)^4 (X^2 + X + 1)^2 \\ Q_5(X) &= Q_6(X) = Q_7(X) = Q_8(X) = (X + 1)^2 \end{aligned}$$

On rappelle que μ_i est défini comme le nombre de facteurs invariants de L multiples de $M_i^{e_i}$ dans la décomposition en facteurs irréductibles du polynôme minimal de L.

Ici, on a donc $\mu_1 = 2$, $\mu_2 = 2$ et que $\mu_3 = 4$. La proposition 3.35 implique directement que pour une unique constante c, $\dim W_L(c) \leq 20$ et que

$$\Pr[\dim W_L(c) = 20] = (1 - 2^{-8})(1 - 2^{-4})^2 \approx 0.8755$$

pour une constante c choisie de manière uniforme. Comme L admet huit facteurs invariants, au moins $t = 8$ éléments $c_1, \ldots, c_8$ sont nécessaires pour engendrer l'espace tout entier ($W_L(c_1, \ldots, c_t) = \mathbb{F}_2^{64}$). Le nombre de facteurs invariants dans lesquels M_i apparaît est donné par $r_1 = 2$, $r_2 = 4$ et $r_3 = 8$. D'après le théorème 3.36, on obtient que, pour 8 éléments $c_1, \ldots, c_8$ pris aléatoirement, la probabilité d'engendrer tout l'espace vaut

$$\prod_{i=0}^{1} \left(1 - 2^{-(8-i)\cdot 4}\right) \times \prod_{i=0}^{3} \left(1 - 2^{-(8-i)\cdot 2}\right) \prod_{i=0}^{7} \left(1 - 2^{-(8-i)}\right) \simeq 0.2895.$$

Mantis et Midori. Le polynôme minimal de la couche linéaire de ces chiffrements admet un unique facteur irréductible : $X + 1$. Cette application linéaire admet 16 facteurs invariants. Les huit premiers facteurs invariants sont égaux au polynôme minimal qui est de degré 6. D'après la proposition 3.35, la probabilité qu'une constante aléatoire engendre un espace de dimension 6 est donc

$$\Pr[\dim W_L(c) = 6] = (1 - 2^{-8}) \approx 0.9961 \ .$$

Vu le nombre de facteurs invariants dans la décomposition de L dans sa forme canonique rationnelle, on sait qu'il est nécessaire d'avoir au moins 16 constantes de tour pour couvrir tout l'espace. De plus, la probabilité d'engendrer tout l'espace avec seulement 16 valeurs choisies aléatoirement est donnée par l'expression

$$\prod_{j=1}^{16} \left(1 - \frac{1}{2^j}\right) \simeq 0.28879 \ .$$

Cette probabilité est relativement faible, cependant, augmenter le nombre de constantes de 16 à 20 augmente la probabilité d'engendrer tout l'espace de 0.28879 à 0.93879. De manière générale, augmenter sensiblement le nombre de constantes de tour augmente rapidement la probabilité d'engendrer tout l'espace.

La figure 3.5 décrit comment la probabilité d'engendrer tout l'espace augmente en fonction du nombre de constantes choisies aléatoirement pour les chiffrements LED, Skinny-64, Prince et Mantis.

Le fait que la forme des courbes de la figure 3.5 soit la même vient du fait que chacune des couches linéaires de ces chiffrements ont un polynôme minimal multiple de $(X + 1)$, et ce facteur apparaît dans chacun des facteurs invariants. Ainsi, pour chacune de ces applications linéaires, le terme qui correspond à ce polynôme de degré 1 vaut

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

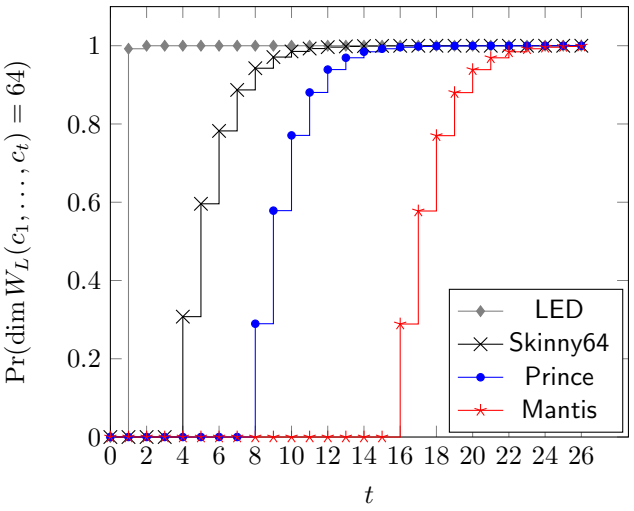

Figure 3.5 – Probabilité que $W_L(c_1,\ldots,c_t) = \mathbb{F}_2^n$ pour des constantes uniformément aléatoires.

et il s'avère que ce terme est le terme dominant dans l'expression de la probabilité calculée grâce au théorème 3.36. Principalement, lorsque $r = t$, ce terme est au moins plus petit que $(1-2^{-1})(1-2^{-2})(1-2^{-3})(1-2^{-4}) \simeq 0.3$, ce qui nous empêche d'avoir une grande probabilité d'engendrer tout l'espace lorsque $(X+1)$ apparaît dans la décomposition en facteurs invariants.

3.4.4 Conclusion

Dans ce chapitre, nous avons analysé les chiffrements à bas coût construits selon un réseau de substitution-permutation où le cadencement de clef est réduit à l'addition d'une constante de tour. Suite aux premières attaques par invariant, nous nous sommes demandé pourquoi ce phénomène se produisait sur certains chiffrements et non sur tous. Dans un premier temps, une approche algorithmique nous a permis de prouver l'absence d'invariants à la fois par la couche linéaire et la couche de substitution. Cependant, cette approche algorithmique n'est efficiente que si la dimension du plus petit espace engendré par les constantes de tour et la couche linéaire est grande comparativement à la taille du bloc utilisé.

Dans un second temps, nous avons expliqué pourquoi la dimension de $W_L(D)$ pouvait considérablement varier d'un chiffrement à l'autre. Les dimensions maximales possibles pour $W_L(D)$ sont liées à la forme canonique rationnelle de la couche linéaire du chiffrement. Principalement, le nombre de facteurs invariants de l'étage linéaire donne le nombre minimal de tours nécessaires pour s'assurer que $W_L(D)$ couvre tout l'espace, ce qui permet de prouver l'absence d'invariants à la fois par la couche linéaire et la couche de substitution.

De plus, comme les constantes de tour sont souvent choisies de manière aléatoire, nous avons étudié le comportement de $W_L(D)$ lorsque l'ensemble D est pris aléatoirement et de manière uniforme. Ici, le nombre de facteurs invariants dans la forme canonique rationnelle de L intervient dans nos résultats, ainsi que le degré des diviseurs élémentaires de son polynôme minimal de L qui interviennent. Principalement, lorsque $X + 1$ est un diviseur du polynôme minimal, il y a un risque que le chiffrement se comporte assez mal relativement aux invariants.

En prenant en compte tout ce qui a été dit dans ce chapitre, il convient donc de faire extrêmement attention aux composantes involutives. En effet, si une matrice est une involution, alors son polynôme minimal est $(X + 1)^2$, et donc le nombre de tours nécessaires pour éviter l'existence d'invariants est très élevé.

Nous avons donc montré l'origine de l'existence d'invariants sur les chiffrements de type SPN lorsque le cadencement de clef est réduit à l'addition d'une constante de tour. Surtout, nous fournissons aux futur.e.s concepteur.rice.s un critère leur permettant de choisir leurs constantes de tour et leur étage linéaire pour se prémunir contre ce type d'attaques. En revanche, nous nous devons de préciser que nous ne prouvons que l'absence d'ensembles qui soient invariants à la fois par l'étage linéaire et l'étage de substitution. Il serait pourtant imaginable de monter une attaque par invariant pour laquelle l'étage de substitution transforme un sous-ensemble A en un sous-ensemble A', et que l'étage linéaire transforme A' en A. C'est d'ailleurs le cas pour certaines attaques par sous-espace invariant où A et A' sont deux translatés d'un même espace vectoriel [LMR15]. Pour prendre en compte ce type de situation, il faudrait considérer des invariants pour la fonction de tour complète $(S \circ L)$, ce qui est algorithmiquement difficile vu les tailles de bloc considérés. Cela reste donc encore une question ouverte et compliquée.

Chapitre 4

Attaques sur les registres filtrés exploitant la structure de corps fini

Dans ce chapitre, nous décrivons une cryptanalyse des registres filtrés en exploitant la structure de corps fini. Les résultats décrits dans ce chapitre ont été publiés à *Fast Software Encryption* en 2016 dans [CR16].

4.1 Les chiffrements à flot

4.1.1 Définitions

À la différence des chiffrements par bloc, les chiffrements à flot (ou chiffrements de flux) sont spécifiquement conçus pour imiter le plus fidèlement possible le chiffrement de Vernam [1]. Il existe aussi des chiffrements à flot auto-synchronisant dans lesquels la suite chiffrante dépend des messages chiffrés précédents, mais leurs taux d'utilisation étant relativement faible, nous nous intéressons aux chiffrements à flot synchrones. Un chiffrement à flot synchrone consiste à additionner bit à bit ("ou" exclusif) le texte clair à une suite pseudo-aléatoire pour produire le texte chiffré [2]. Autrement dit, un algorithme de chiffrement à flot est défini génériquement au moyen d'un générateur pseudo-aléatoire (PRNG [3]) et d'une opération combinant la sortie de ce générateur avec le texte clair pour produire le texte chiffré.

Un PRNG peut être dérivé d'un chiffrement par bloc au moyen d'un mode opératoire (par exemple le mode compteur), mais peut aussi reposer sur une

1. Voir introduction
2. L'utilisation du "ou" exclusif n'est pas le seul choix possible, il suffit que l'opération soit inversible mais c'est la plus simple.
3. Pseudo Random Number Generator

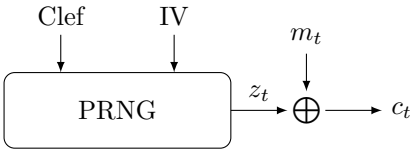

Figure 4.1 – Schéma générique d'un chiffrement à flot, z_t désigne la suite chiffrante, m_t le message clair et c_t le message chiffré.

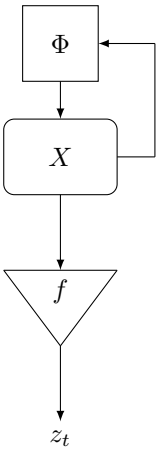

Figure 4.2 – Schéma générique d'un générateur pseudo-aléatoire, X désigne l'état interne, Φ la fonction de transition et f la fonction de filtrage.

construction *ad hoc*. De manière générale, un générateur pseudo-aléatoire prend en entrée une source d'entropie (graine) de petite taille, et produit une suite de taille plus grande. Évidemment, la sortie d'un PRNG n'est pas aléatoire puisque la suite produite est de taille plus grande que la source d'entropie. En revanche, il est possible de produire une suite de grande taille, de manière à ce qu'il soit algorithmiquement difficile de la distinguer d'une suite aléatoire. En utilisant de bons générateurs pseudo-aléatoires, on peut alors imiter le principe du chiffrement de Vernam en utilisant comme suite chiffrante une suite pseudo-aléatoire, ce qui nous permet d'obtenir des algorithmes de chiffrement symétrique efficaces. Classiquement, on utilise un PRNG que l'on initialise avec la clef secrète et un vecteur d'initialisation (IV) qui est public, afin de générer une suite sortante $(z_t)_{t\geq 0}$ (voir figure 4.1).

Ainsi, la sécurité du chiffrement repose sur les caractéristiques du générateur pseudo-aléatoire utilisé. Génériquement, un générateur pseudo-aléatoire est modélisé à l'aide d'une fonction de transition et d'une fonction dite de filtrage comme décrit à la figure 4.2.

4.1.2 Attaques sur les chiffrements à flot

Types d'attaques. En se plaçant dans un modèle favorable pour l'attaquant.e, c'est-à-dire dans le modèle à clair connu, on peut supposer que ce.tte dernier.ère connaît la suite chiffrante $(z_t)_{0\leq t<N}$ produite par le générateur pseudo-aléatoire, où N désigne ici la quantité de données à laquelle l'attaquant.e a accès. En effet, la connaissance du message initial et du texte chiffré permet de retrouver la suite chiffrante en calculant simplement $c_t \oplus m_t = z_t$. Dans ces conditions, on dira qu'un chiffrement à flot est sûr si, connaissant la suite chiffrante, il est algorithmiquement difficile de réaliser une des actions suivantes :

— retrouver la clef secrète ;

— retrouver l'état initial ;

— prédire un ou plusieurs bits non connus de la suite chiffrante ;

— distinguer la suite connue d'une suite aléatoire.

Un chiffrement à flot comporte souvent une phase d'initialisation qui calcule l'état initial à partir de la clef et du vecteur d'initialisation. C'est la raison pour laquelle une attaque par recouvrement de clef est plus puissante qu'une attaque par recouvrement de l'état initial.

Recherche exhaustive. Tout comme pour les chiffrements par bloc, si la clef secrète utilisée est de taille κ, alors le coût de la recherche exhaustive est en 2^κ.

Attaque générique en compromis temps/mémoire/données. Le principe du compromis temps-mémoire a été introduit par Hellman en 1980 [Hel80] pour inverser de manière générique une fonction à sens unique [4]. L'idée générale est de réunir des couples entrée/sortie de la fonction en les stockant dans une table (ce qui coûte de la mémoire) pour ensuite faire coïncider ces informations avec les données observées. Pour des chiffrements par bloc, cette attaque est très coûteuse en mémoire. Cependant, dans le cas particulier des chiffrements à flot, le coût en mémoire est bien plus faible [Bab95, Gol97]. Sans rentrer dans les détails, cette attaque permet de retrouver l'état initial d'un chiffrement à flot avec une complexité en temps, en mémoire et en données de l'ordre de $2^{n/2}$ où n est la taille de l'état interne du générateur pseudo-aléatoire. La complexité de la recherche exhaustive de la clef étant en 2^κ, on prendra toujours un état interne de taille au moins 2 fois la taille de la clef (*i.e.* $2\kappa \leq n$).

4.2 Chiffrements à flot et LFSR

Afin de minimiser le coût d'implémentation des algorithmes de chiffrement, plusieurs algorithmes de chiffrement à flot utilisent des fonctions de transition linéaires. Pour pouvoir chiffrer une quantité de données importante au regard

4. Une fonction à sens unique (One-way function) est une fonction qui peut être aisément calculée mais qui est difficile à inverser.

de la taille de l'état interne, le.a cryptographe s'attelle à choisir une fonction de transition telle que la période de la suite des états internes soit la plus grande possible quel que soit l'état initial. Dans le cas des fonctions de transition linéaire, cela signifie que le polynôme minimal de l'application linéaire doit être primitif. Parmi toutes les possibilités, la plus intéressante est d'utiliser des registres à décalage à rétroaction linéaire (LFSR - Linear Feedback Shift Register) en raison de leur faible coût d'implémentation, mais aussi des propriétés statistiques bien connues des suites engendrées [LN83, Hel11], ce qui permet de justifier facilement leur caractère pseudo-aléatoire. Les algorithmes de chiffrement comme E0, utilisé pour la communication Bluetooth, SNOW 2.0 (ISO/IEC 18033-4 standard) et A5/1 (GSM) sont des algorithmes fondés sur des LFSR et sont largement employés en pratique. Ainsi, l'étude des LFSR mérite notre attention, afin d'évaluer plus en détail la sécurité de ces chiffrements. Évidemment, les LFSR ne sont jamais utilisés seuls, car ils sont linéaires et l'algorithme de Berlekamp-Massey [Mas69] permet de retrouver l'état initial en un temps quadratique en la taille du registre. Ils sont donc utilisés avec d'autres composantes non-linéaires : par exemple, la famille de chiffrement Grain [HJM05] utilise un LFSR combiné avec un registre mis à jour de manière non-linéaire.

4.2.1 Registres à décalage à rétroaction linéaire

Définition 4.1 (Registre à décalage à rétroaction linéaire)**.** *Un LFSR de taille n sur le corps $\mathbb{F}_q$ est un automate fini qui produit une suite d'éléments $\mathbf{s} = (s_t)_{t\geq 0}$ du corps $\mathbb{F}_q$ satisfaisant une relation de récurrence de degré n sur $\mathbb{F}_q$.*

$$s_{t+n} = \sum_{i=1}^{n} c_i s_{t+n-i},\ \forall t \geq 0$$

Les n coefficients $c_1, \ldots, c_n$ sont les coefficients de rétroaction du LFSR.
Le registre du LFSR contient L cellules, chacune étant constituée d'un élément de $\mathbb{F}_q$. Le contenu de ces cellules forme l'état du LFSR à un instant t. De plus le LFSR est initialisé par $s_0, \ldots, s_{n-1}$: c'est l'état initial du LFSR.

L'état initial d'un LFSR doit nécessairement être non-nul, sinon la suite engendrée par le LFSR est la suite constante à la valeur 0.

La suite produite par un LFSR est donc déterminée de manière unique par ses coefficients de rétroaction et son état initial. Les coefficients de rétroaction définissent le *polynôme de rétroaction* et le *polynôme caractéristique* du LFSR.

Définition 4.2 (Polynôme de rétroaction)**.** *Le polynôme de rétroaction d'un LFSR de taille n dont les coefficients de rétroaction sont $c_1, \ldots, c_n$ est défini par*

$$P_R(X) = 1 - \sum_{i=1}^{n} c_i X^i$$

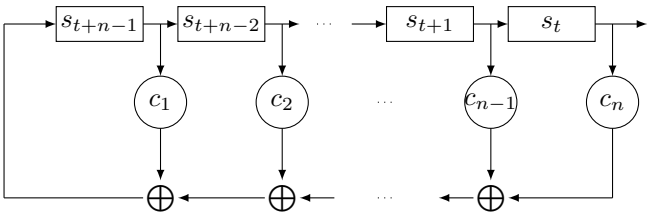

Figure 4.3 – Représentation de Fibonacci d'un LFSR.

Définition 4.3 (Polynôme caractéristique)**.** *Le polynôme caractéristique d'un LFSR de taille n dont le polynôme de rétroaction est P_R est le polynôme défini par*

$$P(X) = X^n P_R(1/X)$$

Exemple 1. Soit un LFSR de taille 6 sur $\mathbb{F}_2$, dont les coefficients de rétroaction sont $(c_1, c_2, c_3, c_4, c_5, c_6) = (1, 1, 0, 0, 0, 1)$, alors le polynôme de rétroaction vaut

$$P_R(X) = 1 + X + X^2 + X^6$$

et le polynôme caractéristique vaut

$$P(X) = X^6 + X^5 + X^4 + 1 \,.$$

De plus, si l'état initial est par exemple $s_0 s_1 s_2 s_3 s_4 = 101100$, alors la suite engendrée commencera par 101100011001...

La caractérisation des suites engendrées par les LFSR [Zie59] nous permet de restreindre notre étude aux registres à décalage dont le polynôme caractéristique est un polynôme primitif. Par exemple, si le polynôme de rétroaction du LFSR n'est pas irréductible, certains états initiaux produisent des suites qui peuvent être engendrées par un LFSR de plus petite taille. Plus généralement, si le polynôme caractéristique n'est pas primitif, alors la période engendrée ne sera pas maximale, puisque *l'ordre du polynôme caractéristique du LFSR détermine la période de la suite produite.*

4.2.2 LFSR et corps finis

En identifiant dans une certaine base les éléments de $\mathbb{F}_q^n$ avec le corps fini $\mathbb{F}_{q^n}$, il apparaît que l'opération induite par la mise à jour de l'état interne d'un LFSR correspond à la multiplication dans le corps fini par un élément défini par le polynôme caractéristique.

Proposition 4.4. *[McE87] Soit P un polynôme irréductible dans $\mathbb{F}_q[X]$ de degré n. Soit $\alpha \in \mathbb{F}_{q^n}$ une racine de P et $\beta_0, \ldots, \beta_{n-1}$ la base duale de $\{1, \alpha, \ldots, \alpha^{n-1}\}$, i.e.,*

$$\mathsf{Tr}(\alpha^i \beta_j) = \begin{cases} 0 & \text{si } i = j \\ 1 & \text{si } i \neq j \end{cases}$$

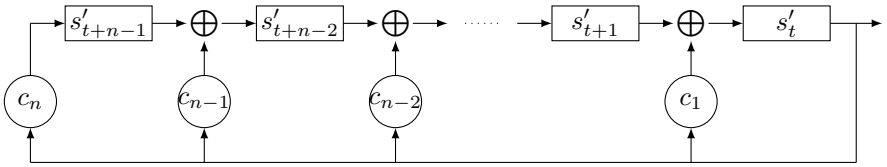

Figure 4.4 – Représentation de Galois d'un LFSR.

où Tr *est la fonction trace du corps* $\mathbb{F}_{q^n}$ *dans* $\mathbb{F}_q$*. Alors, l'état interne du LFSR de polynôme caractéristique* P *à l'instant* $(t+1)$ *est égal à l'état interne du LFSR à l'instant* t *multiplié par* α*, où les vecteurs sont identifiés aux éléments du corps fini* $\mathbb{F}_{q^n}$ *décomposés dans la base* $\{\beta_0, \ldots, \beta_{n-1}\}$*.*

Représentation de Galois. Comme un LFSR implémente la multiplication par un certain élément dans le corps fini, nous pouvons le représenter d'une manière différente de la représentation classique de Fibonacci (figure 4.3), sans pour autant modifier la fonction de transition : c'est la représentation de Galois du LFSR (figure 4.4).

4.2.3 Complexité linéaire

Définition 4.5 (Complexité linéaire)**.** *La complexité linéaire d'une suite* $\mathbf{s} = (s_t)_{t\geq 0}$ *d'éléments du corps fini* $\mathbb{F}_q$*, notée* $\Lambda(s)$*, est le plus petit entier* Λ *tel que* $\mathbf{s}$ *soit engendrée par un LFSR de taille* Λ *sur* $\mathbb{F}_q$*. S'il n'existe pas de tel LFSR, on pose* $\Lambda(s) = \infty$*. Par convention, la complexité linéaire de la suite constante est* 0*. La complexité linéaire d'une suite récurrente correspond au degré du polynôme minimal de cette suite.*

Dans toute la suite on considère que l'on travaille sur des corps de caractéristique 2.

L'algorithme de Berlekamp-Massey [Mas69]. Cet algorithme permet de déterminer la complexité linéaire d'une suite donnée, ainsi que le polynôme de rétroaction du plus petit LFSR qui engendre cette suite. La complexité de cet algorithme est quadratique en la longueur de la suite. Par conséquent, la complexité linéaire d'une suite produite par un générateur pseudo-aléatoire est un critère de sécurité à prendre en compte. Nous verrons même que cette quantité est bien plus pertinente que d'autres critères cryptographiques jusqu'alors considérés comme pertinents.

Algorithme 2 Algorithme de Berlekamp-Massey.

Entrée: $\mathbf{s}^n = s_0 s_1 ... s_{n-1}$, une suite de longueur n.
Sortie: $\Lambda(s)$, la complexité linéaire de $\mathbf{s}^n$, ainsi que le polynôme de rétroaction P correspondant.

$P(X) \leftarrow 1$, $P'(X) \leftarrow 1$, $\Lambda \leftarrow 0$, $m \leftarrow -1$, $d' \leftarrow 1$.
pour t allant de 0 à $n-1$ **faire**
  $d \leftarrow s_t + \sum_{i=1}^{\Lambda} p_i s_{t-i}$
  si $d \neq 0$ **alors**
    $T(X) \leftarrow P(X)$
    $P(X) \leftarrow P(X) - d(d')^{-1} P'(X) X^{t-m}$
    si $2\Lambda \leq t$ **alors**
      $\Lambda \leftarrow t + 1 - \Lambda$
      $m \leftarrow t$
      $P'(X) \leftarrow T(X)$
      $d' \leftarrow d$
    fin si
  fin si
fin pour
  Renvoyer Λ et P

4.2.4 Construction de chiffrements à flot basés sur des LFSR

Comme nous l'avons dit précédemment, nous n'utilisons pas de registres à décalage à rétroaction linéaire seuls pour construire un algorithme de chiffrement. Cependant, les propriétés statistiques des LFSR étant suffisamment étudiées et leur coût d'implémentation étant faible, leur intérêt reste majeur pour la construction d'algorithmes de chiffrement symétriques. Dans cette partie, nous décrivons deux utilisations génériques de ces registres : la combinaison de registres et les LFSR filtrés.

La combinaison de registres. L'idée de ce type de générateurs pseudo-aléatoires est de combiner plusieurs LFSR avec une fonction booléenne afin de produire la suite chiffrante. Plus précisément, nous prenons ℓ LFSR différents, potentiellement de tailles différentes. On note $(u_t^i)_{t\geq 0}$ la suite produite par chacun de ces LFSR pour tout $1 \leq i \leq \ell$ et on choisit une fonction booléenne à ℓ variables. Alors, la suite chiffrante $(s_t)_{t\geq 0}$ engendrée par ce générateur combiné est donnée par la relation

$$s_t = f(u_t^1, u_t^2, \ldots, u_t^\ell), \ \forall t \geq 0 \ .$$

Le schéma général d'un générateur combiné est décrit à la figure 4.5. La complexité linéaire de ce type de générateurs est liée au degré algébrique de la fonction booléenne utilisée [Her86, RS87]. Il faudra donc prendre en compte plusieurs critères lors du choix de la fonction booléenne pour éviter certaines attaques sur ce type de chiffrement.

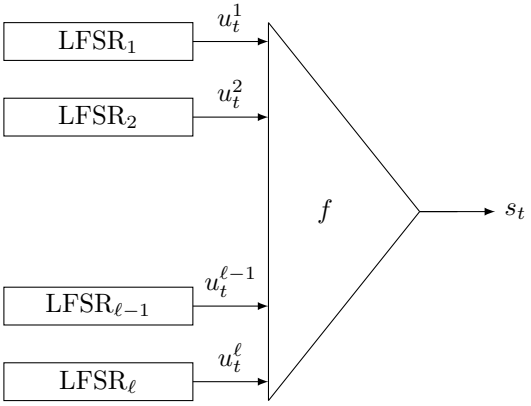

Figure 4.5 – Combinaison de LFSR.

Les registres filtrés.

Définition 4.6 (Registre filtré)**.** *Un registre filtré (ou générateur filtré) est un générateur pseudo-aléatoire composé d'un unique LFSR de taille n, dont le contenu est filtré par une fonction booléenne f (voir figure 4.6). La suite pseudo-aléatoire générée ainsi satisfait la relation suivante :*

$$s_t = f(u_{t+\gamma_1}, \ldots, u_{t+\gamma_\ell})$$

où $(u_t)_{t\leq 0}$ est la suite produite par le LFSR et $(\gamma_i)_{1\leq i\leq \ell}$ une suite décroissante d'entiers de $\{0, \ldots, n-1\}$.

L'algorithme de chiffrement à flot Sfinks [BLM+05] soumis à la compétition eSTREAM [ECR05] suit exactement ce schéma spécifique.

Il est aussi important de garder en mémoire que dans la pratique, la fonction booléenne utilisée prend en entrée un petit nombre de variables (10-20), alors que la taille du registre est souvent de taille 128 ou 256 bits.

4.3 Cryptanalyse de chiffrements à flot basés sur les LFSR

Dans cette section, nous nous intéressons à évaluer la sécurité des algorithmes de chiffrement basés sur les LFSR. De manière générique, nous décrivons des attaques existantes sur les combinaisons de générateurs et les registres filtrés. Les attaques par corrélation (rapides) et les attaques algébriques sont les deux principales attaques sur les chiffrements à flot en général. Les schémas que nous avons décrits sont très simples, cependant, un nombre non-négligeable de chiffrements à flot utilisent ces constructions [ECR05] mais dans une forme

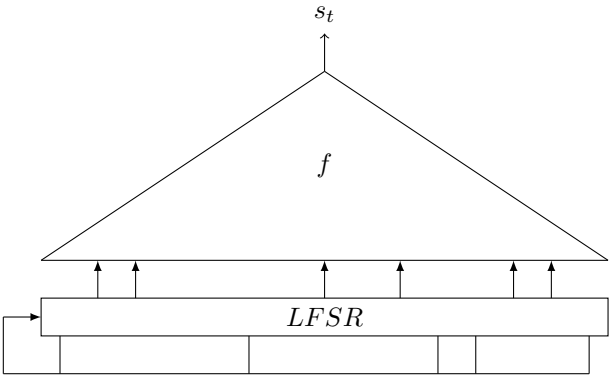

Figure 4.6 – Registre filtré.

légèrement modifiée. Ainsi, les attaques que nous décrivons peuvent dans certains cas être adaptées à des chiffrements spécifiques [LV04, MJB04, Nay07].

4.3.1 Attaques algébriques

Dans le cas des registres filtrés décrits à la section 4.2.4, la fonction de mise à jour de l'état interne est une fonction linéaire. Ainsi, le degré des équations liant la suite chiffrante (connue) et l'état initial est constant et est égal au degré de la fonction booléenne utilisée (noté d). L'attaque algébrique de base consiste à écrire toutes les équations obtenues entre l'état initial et la suite chiffrante. En linéarisant tous les monômes [5], on obtient un système d'équations linéaires dont le nombre d'inconnues est de l'ordre de

$$D = \sum_{i=1}^{d} \binom{n}{i}.$$

où n est la taille de l'état interne et $\binom{n}{i}$ correspond au nombre de monômes de degré i. En collectant au moins D valeurs de la suite chiffrante, nous pouvons résoudre le système correspondant avec une complexité de l'ordre de $\mathcal{O}(D^3)$ avec une simple élimination de Gauss. Cette complexité peut être améliorée par l'utilisation d'algorithmes de résolution de systèmes polynomiaux comme par exemple ceux utilisant des bases de Gröbner. De plus, plusieurs variantes de ces attaques ont également été développées afin d'en améliorer sensiblement la complexité [CM03, Cou03, RH07, GRHH11], principalement en effectuant une partie des calculs dans une phase de pré-calcul (indépendante de l'état initial).

L'attaque de Courtois et Meier en 2003 [CM03] n'utilise pas le degré de la fonction, mais exploite l'existence de relations de petit degré induites par l'équation existante. Le principe de cette attaque est d'exploiter l'existence d'une

5. c'est-à-dire en considérant tous les monômes comme des nouvelles variables indépendantes

fonction de petit degré qui annule la fonction. Ainsi le critère cryptographique associé à ce type d'attaques n'est pas le degré algébrique, mais l'*immunité algébrique*, c'est-à-dire le plus petit degré d'un annulateur non-nul de la fonction booléenne (définition 2.20 du chapitre 2). Formulé autrement, cette attaque met en évidence le fait que, dans certains cas, une équation de haut degré peut impliquer l'existence d'une autre équation de degré plus faible : une équation peut en cacher une autre !

4.3.2 Attaques par corrélation

Initialement proposé par Siegenthaler en 1985 [Sie85] sur la combinaison de registres (figure 4.2.4) et amélioré en 2002 dans le cas linéaire [CJM02] puis en 2012 dans un cadre plus général [CN12] au moyen d'une transformée de Walsh rapide, ce type d'attaque exploite l'existence d'un biais entre la sortie du générateur et la sortie d'un autre générateur dont l'état interne forme une partie de l'état total.

Prenons le générateur combiné décrit à la figure 4.5. En supposant que la sortie de ce générateur est corrélée à la sortie de chaque registre le constituant, l'attaque de Siegenthaler permet de retrouver l'état initial en faisant une recherche exhaustive sur chaque sous-partie de l'état, pour finalement retrouver l'état complet. La technique de Siegenthaler est une technique de type *diviser pour mieux régner*, c'est-à-dire que l'on arrive à retrouver plusieurs parties du secret (l'état initial) indépendamment les unes des autres.

Tandis que le coût de la recherche exhaustive sur ce type de générateurs est de l'ordre de

$$\prod_{i=1}^{\ell}(2^{n_i}-1)$$

opérations, la technique de Siegenthaler a un coût de

$$\sum_{i=1}^{\ell}(2^{n_i}-1)$$

opérations, où ℓ est le nombre de registres et n_i désigne la longueur de ces registres. Afin d'éviter ce type d'attaque, il est nécessaire de choisir une fonction booléenne ayant une forte *résilience* (définition 2.18 du chapitre 2). En revanche, il n'est pas possible d'avoir à la fois une forte résilience et un haut degré algébrique pour la fonction booléenne (proposition 2.19 du chapitre 2). En d'autres termes, on ne peut avoir, sur ce type de générateurs, à la fois une grande complexité linéaire et être résistant à l'attaque de Siegenthaler sauf à augmenter considérablement la longueur des LFSR utilisés.

4.3.3 Attaques par corrélation rapides

Dans le cas des générateurs filtrés, l'attaque de Siegenthaler ne s'applique pas puisque qu'il n'y a qu'un seul registre. Cependant, il est possible de réaliser

des attaques lorsque la fonction booléenne est suffisamment proche au sens de la distance de Hamming d'une fonction linéaire. Il est à noter que ces attaques ne sont pas une amélioration de l'attaque de Siegenthaler, mais sont, à mon avis fondamentalement différentes dans la manière d'exploiter les propriétés des générateurs. Les attaques par corrélation rapides [MS89] ont été largement étudiées [JJ99, JJ00, CT00, CJS01, CJM02, LV04] et sont encore un sujet d'actualité [ZGM17, TIM+18]. Le critère cryptographique pour la fonction booléenne associé à cette attaque est la *non-linéarité*, c'est-à-dire la distance à toutes les fonctions affines (définition 2.17 du chapitre 2).

De manière générale, si l'on considère un LFSR filtré par f, d'état initial X_0, générant une suite $\mathbf{s}$, où $\mathbf{s}$ est fortement corrélée avec σ, suite sortant d'un LFSR non filtré d'état initial X'_0 de longueur L, alors les complexités en données (N) nécessaires de la suite chiffrante, en précalcul (P) et en temps (T) sont les suivantes.

$$N = 2^{\frac{L}{d-1}} \left(\frac{1}{2\varepsilon}\right)^{\frac{2(d-2)}{d-1}} \qquad P = \frac{N^{d-2}}{(d-2)!} \qquad T = 2^{\frac{L}{d-1}} \left(\frac{1}{2\epsilon}\right)^{\frac{2d(d-2)}{d-1}} \tag{4.1}$$

où d est un entier $d \geq 3$ qui correspond au nombre de termes des équations de parité utilisées dans l'attaque [CT00] et ε est le biais entre $\mathbf{s}$ et σ, c'est-à-dire $\Pr[s_t = \sigma_t] = \frac{1}{2} - \varepsilon$.

4.4 Équivalence monomiale de registres filtrés

À partir de maintenant, nous nous intéressons à un unique LFSR filtré, et nous décrivons une équivalence non-linéaire entre plusieurs registres filtrés. Cette équivalence que nous appellerons équivalence monomiale a été introduite par Sondre Rønjom et Carlos Cid à Fast Software Encryption en 2010 [RC10]. Comme nous venons de le rappeler, les registres filtrés sont vulnérables à plusieurs types d'attaques et, pour s'en prémunir, la fonction booléenne doit avoir une haute immunité algébrique et une haute non-linéarité. Dans cette partie, nous expliquons en quoi l'équivalence non-linéaire affecte également la sécurité des registres filtrés.

Définition 4.7 (Équivalence)**.** *On dira qu'un générateur pseudo-aléatoire E' est équivalent à un autre système E si E' peut produire la même suite que le système E et réciproquement, et qu'il existe une bijection entre les états initiaux produisant les mêmes suites.*

On considère un LFSR de taille n, de polynôme de caractéristique primitif P (pour assurer la période maximale), filtré par une fonction booléenne f que l'on regarde comme une fonction à n variables[6]. On note $(u_t)_{t\geq 0}$ la suite engendrée

6. même si en pratique, celle-ci prend un bien plus petit nombre de variables que la taille du registre, ce qui signifie que la sortie de la fonction ne dépend pas de toutes ses variables d'entrée, *i.e.* la fonction a des structures linéaires.

par ce LFSR. La suite sortante $\mathbf{s}$ est donc définie par

$$s_t = f(u_t, u_{t+1}, \ldots, u_{t+n-1}).$$

On note X_0 l'état initial du LFSR. Ainsi, ce générateur filtré est défini par la donnée de $P \in \mathbb{F}_2[X]$ et de $f \in \mathcal{B}_n$. Le couple (P, f) est appelé la *représentation multivariée* du LFSR filtré.

4.4.1 Représentation univariée

Comme nous l'avons expliqué à la section précédente, nous pouvons identifier l'état interne d'un LFSR de taille n à un élément du corps $\mathbb{F}_{2^n}$ au sens de la proposition 4.4. De la même manière, la fonction de filtrage est vue comme une fonction de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_2$ (c'est-à-dire vue sous sa représentation univariée). Avec ces notations, on considère l'isomorphisme ϕ de $\mathbb{F}_2^n$ dans le corps fini $\mathbb{F}_{2^n}$, défini par la base appropriée $\{\beta_0, \ldots, \beta_{n-1}\}$. Alors, l'état interne à l'instant t du LFSR ($X_t \in \mathbb{F}_{2^n}$) initialisé par $X_0 = \Phi(u_0, \ldots, u_{n-1})$ correspond exactement à

$$X_t = \alpha^t X_0$$

où α est une racine (primitive) du polynôme caractéristique du LFSR. Dans ces conditions, la valeur de la suite chiffrante à l'instant t est donnée par la relation

$$s_t = f \circ \Phi^{-1}(X_0 \alpha^t) \ .$$

Ainsi, tout registre filtré admet une *représentation équivalente* définie par une racine $\alpha \in \mathbb{F}_{2^n}$ du polynôme caractéristique du LFSR, et une fonction F de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_2$, $F = f \circ \Phi^{-1}$.

Notation 4.8. *On notera les fonctions booléennes en majuscule lorsqu'elles sont vues sous forme polynomiale univariée et minuscule lorsqu'elles sont vues sous forme algébrique normale (multivariée).*

De cette représentation univariée définie par le couple (α, F), il est possible de retrouver la représentation multivariée (P, f) : comme P est irréductible, ce dernier correspond au polynôme minimal de α et f est égal à $F \circ \Phi$ où Φ est l'isomorphisme de $\mathbb{F}_2^n$ dans $\mathbb{F}_{2^n}$ associé à la base duale de $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$. En revanche, pour une représentation (P, f) il existe plusieurs valeurs de α possibles : toutes les racines conjuguées de P, *i.e.* $\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}$. Il y a donc n fonctions de filtrage F possibles en représentation univariée associées à chaque choix de racine de P. Cependant, tous ces choix sont linéairement équivalents puisqu'ils diffèrent seulement par la composition avec l'application de Frobenius. Or, la composition de F par une application linéaire ne change pas les propriétés cryptographiques classiquement considérés, notamment l'immunité algébrique et la non-linéarité.

La représentation de F sous sa forme univariée, c'est-à-dire sous la forme d'un polynôme à coefficients dans $\mathbb{F}_{2^n}$ peut être calculée à l'aide d'une transformée de Fourier discrète [MS77, Bla83, GG04].

Proposition 4.9 (Transformée de Fourier discrète d'une fonction)**.** *Soit F une fonction de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_{2^n}$. Alors, il existe un unique polynôme dans $\mathbb{F}_{2^n}[X]/(X^{2^n}+X)$ tel que*

$$F(X) = \sum_{i=0}^{2^n-1} A_i X^i \ .$$

De plus, $A_{2^n-1} = \sum_{x\in\mathbb{F}_{2^n}} F(x)$ et les coefficients A_i, $0 \leq i \leq 2^n-2$ sont donnés par la transformée de Fourier discrète des valeurs de F prises sur les (2^n-1) éléments non-nuls de $\mathbb{F}_2^n$:

$$A_i = \sum_{k=0}^{2^n-2} F(\gamma^k)\gamma^{-ki},\ 0 \leq i \leq 2^n-2$$

où γ est un élément primitif de $\mathbb{F}_{2^n}$.

De manière évidente, la valeur de $F(0)$ n'impacte en rien les propriétés cryptographiques : cette valeur apparaît seulement lorsque l'état interne du registre s'annule, ce qui est évidemment à éviter, puisque la suite engendrée serait constante.

Cette représentation univariée des fonctions vectorielles (de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_{2^n}$) peut aussi être vue comme un polynôme interpolateur.

Dans le cas de notre étude, la fonction de filtrage est une fonction booléenne, c'est-à-dire que la fonction est à valeurs dans $\mathbb{F}_2$, ce qui implique les relations $A_{2i} = A_i^2$ pour tout $0 \leq i \leq 2^n-2$. En d'autres termes, cela signifie que l'on peut regrouper les coefficients en fonction des classes cyclotomiques modulo (2^n-1), afin de retrouver la représentation trace des fonctions booléennes (définition 2.12 du chapitre 2) :

$$F(X) = \sum_{i\in\Gamma} \mathsf{Tr}^{n_i}(A_i X^i) \ ,$$

où Γ est l'ensemble de tous les représentants des classes cyclotomiques modulo (2^n-1) et n_i correspond à la taille de la classe cyclotomique de i et $A_i \in \mathbb{F}_{2^{n_i}}$.

4.4.2 Équivalence monomiale

En utilisant la représentation univariée, il est facile de remarquer que pour tout $\lambda \in \mathbb{F}_{2^n}$ non nul, la suite engendrée par le registre filtré de polynôme caractéristique P et de fonction de filtrage F, initialisé avec $X_0 \in \mathbb{F}_{2^n}$ est exactement la même que la suite obtenue par le même LFSR, mais filtré par la fonction booléenne $G(x) = F(\lambda x)$ pour tout $x \in \mathbb{F}_{2^n}$ et initialisé par l'état initial $Y_0 = \lambda^{-1}X_0$. Cependant, cette équivalence entre ces registres est une équivalence linéaire. Ainsi, son intérêt cryptographique est limité, puisque toutes les fonctions de la forme $F(\lambda x)$ possèdent les mêmes propriétés cryptographiques classiques que la fonction initiale F.

Cependant, Rønjom et Cid [RC10] ont exhibé une autre équivalence, non-linéaire, entre les registres filtrés. L'idée de cette équivalence est de modifier la

racine primitive (*i.e.* le polynôme caractéristique du LFSR) utilisée pour la mise à jour de l'état interne.

Équivalence non-linéaire. Soit α et β deux éléments primitifs du corps fini $\mathbb{F}_{2^n}$. Comme ce sont tous deux des éléments primitifs, cela signifie qu'il existe un entier k, premier avec $(2^n - 1)$, tel que $\beta = \alpha^k$. On note P_α et P_β les polynômes minimaux respectifs de α et β. Alors, il est clair que pour tout $t \geq 0$, l'état interne X_t du LFSR de polynôme caractéristique P_α initialisé par un élément arbitraire non nul X_0 et l'état interne Y_t du LFSR de polynôme caractéristique P_β intialisé par l'état interne $Y_0 = X_0^k$ vérifient la relation

$$Y_t = Y_0\beta^t = X_0^k\alpha^{kt} = (X_0\alpha^t)^k = X_t^k \ \forall t \geq 0 \ .$$

Finalement, en notant F la fonction de filtrage du LFSR de polynôme caractéristique P_α, la suite $\mathbf{s}$ produite par ce générateur filtré est donnée par la relation

$$s_t = F(X_0\alpha^t) \ \forall t \geq 0 \ .$$

Soit G la fonction booléenne à n variables définie par la relation $G(X) = F(X^r)$, où r est défini comme l'inverse de k modulo $(2^n - 1)$, *i.e.* $kr \equiv 1 \mod (2^n - 1)$. Alors la suite sortante $\mathbf{s}'$ du LFSR de polynôme caractéristique P_β, filtré par la fonction G et initialisé par l'élément $Y_0 = X_0^k$ satisfait

$$s'_t = G(X_t^k) = F((X_t^k)^r) = F(X_t) = s_t \ \forall t \geq 0 \ .$$

Le générateur filtré défini par (α, F) est donc équivalent (au sens de la définition 4.7) au générateur défini par $(\beta = \alpha^k, F(X^r))$ où r est l'inverse de k modulo (2^n-1). Cette équivalence entre générateurs filtrés est appelée *équivalence monomiale*.

Sans faire d'hypothèse particulière sur la fonction de filtrage, cette équivalence monomiale donne

$$\frac{\phi(2^n - 1)}{n}$$

différents générateurs filtrés équivalents, où ϕ est l'indicatrice d'Euler.

Il est à noter que seulement les permutations induisent une équivalence, *i.e.* k doit être premier avec $(2^n - 1)$.

Nous avons vu que les racines conjuguées impliquent une équivalence linéaire, ce qui permet de regrouper les générateurs équivalents en groupes de n générateurs, correspondant aux classes cyclotomiques de k. Comme $\mathsf{pgcd}(k, 2^n - 1) = 1$, la taille des classes cyclotomiques est nécessairement n. Certains cas dégénérés de fonctions booléennes donnent moins de générateurs *différents* dans la classe d'équivalence : il se pourrait qu'il existe un entier k, $\mathsf{pgcd}(k, 2^n - 1) = 1$, tel que pour tout $X \in \mathbb{F}_{2^n}$, $F(X^k) = F(X)$.

Comme ces générateurs sont équivalents, une attaque par recouvrement de l'état initial du générateur induit par P_β permet de retrouver l'état initial X_0 du

registre défini par P_α, en utilisant la relation $X_0 = Y_0^r$. Finalement, on déduit de ces observations la propriété suivante.

Proposition 4.10 (Générateurs équivalents)**.** *Le niveau de sécurité d'un générateur filtré est le niveau* minimal *de sécurité de l'ensemble des générateurs dans sa classe d'équivalence.*

Sondre Rønjom et Carlos Cid ont exhibé cette équivalence en se demandant s'il était possible d'attaquer un générateur équivalent dont le polynôme de rétroaction serait creux, afin d'améliorer les attaques par corrélation rapides, puisque ceci permettrait d'obtenir des équations de petit poids [RC10, page 12]. Cette question reste ouverte.

Exemple 2. Considérons une fonction booléenne monomiale à n variables, par exemple $\mathsf{Tr}(X^r)$, pour un certain r premier avec $(2^n - 1)$. Une telle fonction peut avoir une haute non-linéarité et une haute immunité algébrique. On pourrait donc penser qu'il pourrait d'agir d'une bonne fonction de filtrage (pour certains choix de r). Or, d'après ce qui précède, il est possible de considérer le générateur équivalent, induit par la permutation $X \mapsto X^k$ avec $kr \equiv 1 \mod (2^n - 1)$. Dans ces conditions, le générateur initial filtré par $\mathsf{Tr}(X^r)$ est équivalent à un unique LFSR, dont l'état initial peut être retrouvé avec une complexité quadratique en la taille du registre ($\mathcal{O}(n^2)$) à l'aide de l'algorithme de Berlekamp-Massey.

Étant donné que les critères cryptographiques sur la fonction de filtrage ne sont pas préservés par la transformation $X \mapsto X^k$, il est tout naturel de se demander quel est l'impact de cette équivalence non-linéaire sur les attaques classiques des générateurs filtrés, notamment les attaques algébriques et les attaques par corrélation rapides. Les résultats obtenus avec Anne Canteaut sont décrits dans la suite de ce chapitre et ont été publiés à Fast Software Encryption en 2016 [CR16].

4.5 Attaques algébriques "univariées"

Déterminer les propriétés cryptographiques d'une fonction booléenne en considérant tous les changements possibles de racine primitive du corps fini $\mathbb{F}_{2^n}$ semble a priori difficile, puisque les critères de non-linéarité et d'immunité algébrique ne sont pas invariants par cette transformation [RC10, Appendix A].

Pour ce qui est du degré et de l'immunité algébrique, les travaux de Massey et Serconek [MS94], puis plus récemment ceux de Gong, Helleseth et Rønjom [RGH07, RH07, Gon11, HR11] ont montré que la représentation univariée est bien plus pertinente que la représentation multivariée pour évaluer correctement la sécurité des registres filtrés. Pour le cas des attaques algébriques, c'est cette représentation univariée qui donne exactement la valeur de la complexité linéaire de la suite produite, ce qui détermine précisément la sécurité de ce type de générateur. En effet, l'action de l'équivalence monomiale sur la fonction de filtrage est bien plus simple à décrire lorsque l'on choisit la représentation univariée que la représentation multivariée.

Soit F une fonction booléenne donnée sous sa forme univariée, *i.e.*

$$F(X) = \sum_{i=0}^{2^n-2} A_i X^i \ ,$$

alors l'ensemble des fonctions booléennes dans la classe d'équivalence de F est défini par les fonctions G dont la représentation univariée

$$G(X) = \sum_{i=0}^{2^n-2} B_i X^i$$

est obtenue en décimant la représentation univariée de F par un entier k premier avec $(2^n - 1)$, *i.e.*,

$$B_i = A_{ik \mod (2^n-1)} \ .$$

4.5.1 Complexité linéaire

Comme nous l'avons décrit à la section 4.3, l'attaque algébrique la plus simple consiste à collecter des équations (multivariées) liant l'état initial et la suite chiffrante. Dans ces conditions, la complexité de l'attaque dépend du degré de la fonction de filtrage. Or, plutôt que de linéariser le système, nous pouvons adopter une stratégie complètement différente et beaucoup plus performante : nous pouvons considérer la sortie du générateur filtré comme la sortie d'un unique LFSR (de taille bien plus grande). La taille de ce LFSR est la complexité linéaire Λ de la suite chiffrante (définition 4.5). Cette valeur détermine exactement la complexité de résolution du plus petit système linéaire qui lie les bits de la suite chiffrante et l'état initial. Pour le cas particulier de la combinaison de registres, nous savons que la complexité linéaire croit suffisamment vite avec le degré de la fonction [Key76, Rue86]. En revanche, dans le cas des générateurs filtrés, le théorème de Blahut [Bla83, MS94] implique que la valeur de la complexité linéaire Λ est entièrement déterminée par le nombre de coefficients non nuls dans la représentation univariée de la fonction de filtrage. Plus précisément, soit $A_i \in \mathbb{F}_{2^n}$ pour $0 \leq i \leq 2^n - 2$ les coefficients de la représentation univariée de F, alors la complexité linéaire de la suite sortante d'un LFSR de polynôme caractéristique primitif et filtré par F est donnée par

$$\Lambda = \#\{0 \leq i \leq 2^n - 2 \ : A_i \neq 0\} \ .$$

L'équivalence monomiale correspond à la décimation par un certain entier k premier avec $(2^n - 1)$ des coefficients dans la représentation univariée. La valeur des coefficients reste donc inchangée par cette transformation, ce qui nous permet d'assurer la proposition suivante.

Proposition 4.11 (Invariance de la complexité linéaire)**.** *La complexité linéaire d'une suite engendrée par un LFSR filtré est un invariant pour l'équivalence monomiale.*

Une observation importante faite par Rønjom et Helleseth dans [RH07] est que la complexité linéaire est toujours plus petite que le nombre de monômes présents dans l'attaque algébrique classique. En effet, pour l'attaque algébrique classique, le système linéarisé admet approximativement

$$\sum_{i=1}^{\deg f} \binom{n}{i}$$

inconnues. Or, on sait que le degré de la fonction monomiale $\mathsf{Tr}(\lambda x^k)$ est égal au poids de Hamming de k, c'est-à-dire le nombre de 1 dans sa représentation binaire. Ainsi, si l'on a une fonction booléenne de degré d, on sait que tous les coefficients A_k dans la représentation univariée où $\mathsf{w_H}(k) > \deg f$ doivent nécessairement être nuls. Ainsi, la complexité linéaire Λ du générateur, *i.e.*, le nombre de coefficients non nuls dans la représentation univariée, est au plus le nombre d'entiers k tels que $\mathsf{w_H}(k) \leq \deg(f)$, ce qui correspond au nombre d'inconnues dans le système multivarié linéarisé. Dans ces conditions, pour n'importe quel générateur filtré obtenu par équivalence monomiale, la meilleure attaque algébrique a une complexité en données de $\mathcal{O}(\Lambda)$. La connaissance de Λ bits de suite chiffrante permet de déterminer l'état initial du registre en un temps de calcul linéaire en Λ. Cependant, la phase de précalcul consiste à calculer la complexité linéaire engendrée par le générateur, ainsi que le polynôme minimal de la suite chiffrante. Ceci peut être réalisé en appliquant l'algorithme de Berlekamp-Massey (algorithme 2, page 69) au générateur filtré, initialisé par une valeur arbitraire avec un coût quadratique en la complexité linéaire, *i.e.* en $\mathcal{O}(\Lambda^2)$. Cette complexité peut être améliorée en utilisant des techniques similaires à celles employées dans les attaques algébriques rapides afin d'obtenir une complexité de l'ordre de $\mathcal{O}(\Lambda \log^2 \Lambda)$ comme expliqué dans [RH07]. Comme ces attaques exploitent le spectre de Fourier de la fonction de filtrage, il est clair que l'équivalence monomiale n'affecte en rien la complexité de ces attaques.

4.5.2 Attaques algébriques

D'après les observations précédentes, Rønjom et Cid ont défini dans leur article l'immunité algébrique généralisée [RC10, Définition 6], c'est-à-dire la plus petite immunité algébrique pour toutes les fonctions booléennes dans la classe d'équivalence induite par le changement de racine primitive dans le corps fini $\mathbb{F}_{2^n}$. Comme expliqué au chapitre 2 mais aussi à la section 4.3, les attaques algébriques permettent à l'attaquant.e de diminuer le degré des équations liant l'état initial et les bits de la suite chiffrante en utilisant l'existence d'un annulateur de petit degré de la fonction de filtrage [CM03]. La même idée peut être utilisée afin d'améliorer les attaques algébriques utilisant la représentation univariée de la fonction. En d'autres termes, si le polynôme représentant la fonction de filtrage n'est pas creux mais que la fonction admet un annulateur creux, alors l'attaque peut être réalisée sur cet annulateur plutôt que sur la fonction elle-même (comme pour les attaques algébriques classiques). Ainsi, la complexité de la meilleure

attaque est déterminée par la plus petite complexité linéaire d'un annulateur de la fonction de filtrage. Cette quantité est définie comme l'*immunité spectrale* de la suite sortante [GRHH11, Définition 1].

Définition 4.12 (Immunité Spectrale)**.** *Soit* $\mathbf{s}$ *une suite binaire de période* N*, alors l'immunité spectrale* $\mathsf{SI}(\mathbf{s})$ *est définie par*

$$\mathsf{SI}(\mathbf{s}) = \min_{\mathbf{b}\in\mathbb{F}_2^N} \{\Lambda(\mathbf{b}) | \mathbf{s}\cdot\mathbf{b} = \mathbf{0} \text{ ou } (\mathbf{s}+\mathbf{1})\cdot\mathbf{b} = \mathbf{0}, \mathbf{b} \neq 0\}$$

Exemple 3. C'est cette propriété particulière qui a permis à Sondre Rønjom [Røn15] de casser la famille de chiffrements à flot Welch-Gong [NG05] dont les suites chiffrantes ont une faible immunité spectrale.

En utilisant la correspondance entre suites et fonctions booléennes [GG04], cette définition s'étend tout naturellement aux fonctions booléennes en considérant le nombre de coefficients non nuls dans la représentation univariée. Ainsi, nous pourrons parler de complexité linéaire d'une fonction booléenne comme la complexité linéaire d'une suite engendrée par un générateur filtré par cette fonction. Nous discuterons de ce nouveau critère cryptographique en prenant un peu de recul à la conclusion. Or, pour tout annulateur G de F, nous avons toujours l'inégalité

$$\Lambda(G) \leq \sum_{i=0}^{\deg G} \binom{n}{i},$$

impliquant directement que l'attaque exploitant la représentation univariée est toujours plus performante que l'attaque classique.

Il nous reste à montrer que l'équivalence monomiale n'affecte pas non plus l'immunité spectrale de F.

Soit k un entier premier avec $(2^n - 1)$ et soit $F'(x) = F(x^k)$. Soit G' un annulateur non-nul de F'. Par définition G' est un annulateur non-nul de F' si et seulement si $G(x) = G'(x^r)$ est un annulateur non-nul de F, avec $rk \equiv 1 \mod (2^n - 1)$. La complexité linéaire de G' est donc égale à la complexité linéaire de G, annulateur de F. Dans ces conditions, une attaque algébrique sur le générateur filtré par F a exactement le même coût que sur le générateur filtré par F'. Plus généralement, l'immunité spectrale des fonctions booléennes est invariante par l'équivalence monomiale.

Proposition 4.13. *L'équivalence monomiale n'affecte pas la complexité des meilleures attaques algébriques sur les générateurs filtrés.*

4.6 Attaques par corrélation "univariées"

Comme expliqué à la section 4.3, le principe de ces attaques est, pour un générateur filtré, d'approximer la fonction de filtrage par une fonction linéaire [7].

7. Contrairement aux registres combinés, le critère de *résilience* n'est pas pertinent ici.

Le critère cryptographique associé aux attaques par corrélation rapides est la *non-linéarité* des fonctions booléennes (définition 2.17 du chapitre 2). Rønjom et Cid [RC10, Section 6.2] ont expliqué dans leur article qu'il est nécessaire d'étendre le critère de non-linéarité. Dans cette section, nous montrons dans un premier temps que le critère pertinent pour évaluer la sécurité des registres filtrés est en fait la *non-linéarité généralisée* introduite en 2001 par Gong et Youssef [YG01]. Dans un second temps, nous expliquons comment appliquer une attaque de type *diviser pour mieux régner*, similaire à l'attaque de Siegenthaler, sur un générateur filtré, ce qui peut paraître étrange car nous n'avons accès qu'à un unique registre. Cette attaque utilise les sous-groupes multiplicatifs du corps fini $\mathbb{F}_{2^n}$,

4.6.1 Non-linéarité généralisée

Soit un LFSR de taille n, défini par une racine primitive α et initialisé par un élément non-nul arbitraire $X_0 \in \mathbb{F}_{2^n}$, filtré par une fonction booléenne F. On suppose qu'il existe $\lambda \in \mathbb{F}_{2^n} \backslash \{0\}$ et un entier k, premier avec $(2^n - 1)$ tels que la fonction F est corrélée avec la fonction G définie par $G(x) = \mathsf{Tr}^n(\lambda x^k)$, *i.e.*

$$\sum_{x \in \mathbb{F}_{2^n}} (-1)^{F(x)+G(x)} \neq 0 \, .$$

On note s_t la sortie du générateur filtré par F et σ_t la sortie du générateur filtré par G. Comme $\mathsf{pgcd}(k, 2^n - 1) = 1$, nous pouvons appliquer l'équivalence monomiale induite par k sur le LFSR filtré par G, comme décrit à la figure 4.7.

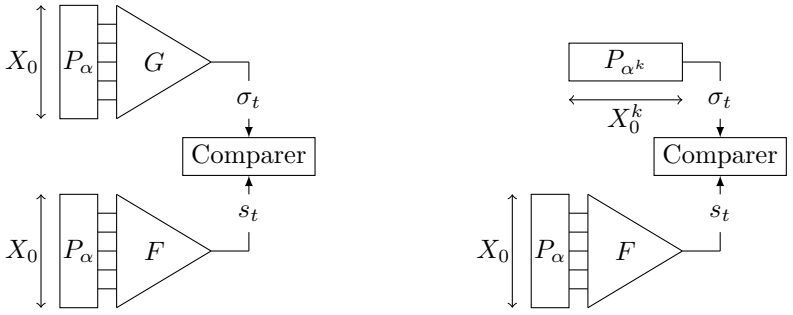

Figure 4.7 – Attaque par corrélation généralisée, avec $\mathsf{pgcd}(k, 2^n - 1) = 1$ et $G(x) = \mathsf{Tr}^n(\lambda x^k)$.

Dans ces conditions, le nouveau polynôme caractéristique du LFSR initialement filtré par G correspond au polynôme minimal de l'élément primitif α^k, et l'état initial de ce LFSR devient X_0^k. La sortie σ_t est obtenue linéairement en fonction de X_0^k en appliquant $x \mapsto \lambda x$, ce qui signifie qu'il est possible de réaliser une attaque par corrélation rapide afin de retrouver l'état initial du LFSR de polynôme caractéristique P_{α^k}, c'est-à-dire X_0^k. Comme k est premier avec $(2^n - 1)$, on retrouve X_0 en appliquant la permutation inverse.

Plus généralement, il est possible de réaliser une attaque par corrélation rapide sur les registres filtrés dès lors que la fonction de filtrage est fortement corrélée avec n'importe quelle fonction composante d'une permutation monomiale, *i.e.* une fonction de la forme

$$x \mapsto \mathsf{Tr}^n(\lambda x^k) \text{ avec } \mathsf{pgcd}(k, 2^n - 1) = 1 \ .$$

Ce qui est très amusant et surprenant, c'est que ce critère cryptographique sur les fonctions booléennes a été introduit par Youssef et Gong en 2001 [YG01], mais n'était alors pas motivé par une attaque. Ici, nous retombons exactement sur le critère de *non-linéarité généralisée*, mais il est motivé par une attaque réelle. Nous pouvons même aller plus loin : le critère classique de *non-linéarité* n'est pas le critère pertinent à prendre en compte pour les générateurs filtrés afin d'évaluer la sécurité réelle de ce type de générateurs au regard des attaques par corrélation.

Définition 4.14 (Transformée de Walsh étendue [YG01])**.** *Soit F une fonction de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_2$, alors sa transformée de Walsh étendue est définie par*

$$\widehat{F}(\lambda, k) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{F(x)+\mathsf{Tr}(\lambda x^k)}$$

où $\lambda \in \mathbb{F}_{2^n}$ et $\mathsf{pgcd}(k, 2^n - 1) = 1$. Alors la non-linéarité généralisée *de F est définie par*

$$\mathsf{NLG}(F) = 2^{n-1} - \frac{1}{2} \max_{\substack{\lambda \in \mathbb{F}_{2^n} \\ k:\gcd(k,2^n-1)=1}} |\widehat{F}(\lambda, k)|$$

est la distance de F à toutes les fonctions composantes des permutations puissances sur $\mathbb{F}_{2^n}$.

4.6.2 Retrouver une partie de l'état

Pour l'instant, l'équivalence monomiale nous permet uniquement de considérer les exposants k premier avec $(2^n - 1)$. Dans ce qui suit, on suppose désormais que $\mathsf{pgcd}(k, 2^n - 1) > 1$. Évidemment, il n'y a plus de notion d'équivalence, puisque $x \mapsto x^k$ n'est plus une permutation.

Nous considérons à nouveau un LFSR de taille n, de polynôme caractéristique primitif P_α, filtré par une fonction booléenne F à n variables. On suppose maintenant qu'il existe une fonction booléenne H de $\mathbb{F}_{2^n}$ dans $\mathbb{F}_2$, telle que F est corrélée à la fonction définie par $G : x \mapsto H(x^k)$. Dans ces conditions, nous pouvons appliquer une transformation (non-bijective) sur le LFSR défini par α et filtré par G. En effet, l'état interne du LFSR défini par α et initialisé par X_0 à l'instant t est $X_0\alpha^t$, donc la suite $\sigma_\mathbf{t}$ produite par ce LFSR filtré par G vérifie

$$\sigma_t = G(X_0\alpha^t) = H(X_0^k\alpha^{kt}) \text{ pour tout } t \geq 0 \ .$$

Par ailleurs, si l'on considère un LFSR de polynôme caractéristique P_{α^k}, le polynôme minimal de α^k, alors les états internes successifs de ce registre sont

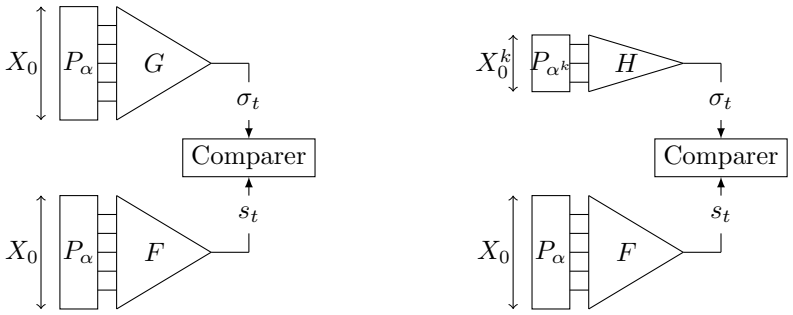

Figure 4.8 – Attaque par corrélation généralisée lorsque $\mathsf{pgcd}(k, 2^n - 1) > 1$.

donnés par $(Y_0\alpha^{kt})_{t\geq 0}$ où Y_0 désigne l'état initial. En choisissant $Y_0 = X_0^k$, il apparaît que le registre filtré par G de polynôme caractéristique P_α engendre la même suite que le registre filtré par H et de polynôme caractéristique P_{α^k} (voir figure 4.8).

Comme α est un élément primitif et que $\mathsf{pgcd}(k, 2^n - 1) > 1$, le polynôme P_{α^k} n'est plus primitif (la suite σ n'est pas de période maximale). Plus précisément, la période de σ est égale à l'ordre multiplicatif de α^k et est alors notée τ_k :

$$\tau_k = \mathsf{ord}(\alpha^k) = \frac{2^n - 1}{\mathsf{pgcd}(k, 2^n - 1)} \ .$$

Cette quantité est aussi le nombre de valeurs possibles de X_0^k. Elle joue un rôle majeur dans la complexité de notre attaque.

Comme nous l'avons dit précédemment, l'attaque de Siegenthaler sur la combinaison de registre ne peut s'appliquer en l'état, puisqu'elle consiste à réaliser une recherche exhaustive sur les différentes parties de l'état initial, ce qui, dans le cas des registres filtrés rend la complexité de l'attaque plus grande que la recherche exhaustive. Cependant, dans notre cas, nous avons $\tau_k < 2^n$, ce qui signifie que la recherche exhaustive sur le registre défini par α^k coûte moins cher que la recherche exhaustive sur l'état tout entier.

Soit ε le biais entre $\mathbf{s}$ et σ. Alors la quantité de données nécessaire pour détecter ce biais (lemme de Neyman-Pearson) est de l'ordre de

$$\mathsf{Données} = \mathcal{O}\left(\frac{2\log(\tau_k)}{\varepsilon^2}\right)$$

où ε peut aussi être défini comme la corrélation entre la fonction F et la fonction G [HJB09]. La complexité en temps de l'algorithme de Siegenthaler est dans notre cas de l'ordre de

$$\mathsf{Temps} = \mathcal{O}\left(\frac{\tau_k \log(\tau_k)}{\varepsilon^2}\right) \ .$$

Cet algorithme nous permet de retrouver l'état initial du LFSR défini par α^k, c'est-à-dire X_0^k. Or, comme k n'est pas premier avec $(2^n - 1)$, la connaissance de

X_0^k ne nous permet pas de retrouver l'état initial complet X_0. Mais nous avons cependant accès à une partie de l'information.

Lemme 4.15. *La connaissance de X_0^k fournit exactement $\log_2(\tau_k)$ bits d'information sur X_0, où $\tau_k = (2^n - 1)/\mathsf{pgcd}(k, 2^n - 1)$.*

Démonstration. Soit X_0 un élément non-nul du corps fini $\mathbb{F}_{2^n}$ et α un élément primitif de ce même corps. Alors il existe un unique entier i compris entre 0 et $(2^n - 2)$ tel que $X_0 = \alpha^i$. Soit maintenant r le reste et q le quotient de la division euclidienne de i par τ_k, *i.e.* $r \equiv i \mod \tau_k$, alors

$$X_0^k = \alpha^{qk\tau_k}\alpha^{rk} = \alpha^{rk}$$

par définition de τ_k.

De plus, r est l'unique entier compris entre 0 et $\tau_k - 1$ tel que $X_0^k = \alpha^{rk}$. En effet, si l'on prend r_1 et r_2, $r_1 > r_2$, tels que $\alpha^{r_1 k} = \alpha^{r_2 k}$, alors $\alpha^{(r_1 - r_2)k} = 1$. Donc, par définition de τ_k, $(r_1 - r_2)$ est nécessairement un multiple de τ_k (l'ordre multiplicatif de α^k), donc r est bien l'unique entier appartenant à $[0, \tau_k - 1]$ tel que $X_0^k = \alpha^{kr}$. Ainsi, pour $X_0 = \alpha^i$, la connaissance de X_0^k nous donne exactement la valeur du reste dans la division euclidienne de i par τ_k, ce qui donne exactement $\log_2(\tau_k)$ bits d'information sur l'état initial X_0. □

4.6.3 Diviser pour mieux régner

D'après ce qui précède, nous pouvons retrouver $\log_2(\tau_k)$ bits d'information sur X_0. Il reste néanmoins $(n - \log_2(\tau_k))$ bits à retrouver pour avoir la connaissance complète de X_0. La manière la plus simple est de faire une recherche exhaustive sur les bits inconnus restant, ce qui a un coût proportionnel aux nombre de solutions, *i.e.*

$$\frac{2^n - 1}{\tau_k} = \mathsf{pgcd}(k, 2^n - 1) \ .$$

En revanche, si l'on examine le problème de plus près, on voit qu'il peut être intéressant de combiner en parallèle plusieurs de ces attaques, mais pour différentes valeurs de k. Cette méthode semble être la plus performante, puisqu'elle permet de "couper" l'état en plusieurs parties afin d'appliquer avec succès une attaque de type *diviser pour mieux régner*.

On choisit k_1 et k_2 deux entiers, tous deux non premiers avec $2^n - 1$, et on suppose que l'on peut réaliser avec succès deux attaques par corrélation en appliquant l'idée de Siegenthaler comme décrit à la section 4.6.2 afin de retrouver les éléments $X_0^{k_1}$ et $X_0^{k_2}$. En ayant choisi au préalable un élément primitif α et en identifiant les éléments non nuls de $\mathbb{F}_{2^n}$ avec les puissances successives de α, nous retrouvons exactement les restes dans les divisions euclidiennes de l'unique entier i définissant $X_0 = \alpha^i$ par τ_{k_1} et τ_{k_2}. En d'autres termes, nous connaissons 2 entiers r_1 et r_2 tels que

$$r_1 \equiv i \mod \tau_{k_1} \text{ et } r_2 \equiv i \mod \tau_{k_2} \ .$$

De manière toute naturelle, grâce au théorème des restes chinois, ce système d'équations nous permet de retrouver le reste dans la division euclidienne de i par le plus petit commun multiple des entiers τ_{k_1} et τ_{k_2} ($\mathsf{ppcm}(\tau_{k_1}, \tau_{k_2})$). Lorsque les entiers τ_{k_1} et τ_{k_2} ne sont pas premiers entre eux, cela signifie qu'une partie de l'information est redondante dans la connaissance de $X_0^{k_1}$ et $X_0^{k_2}$. Ainsi, la meilleure situation pour l'attaquant.e est le cas où les ordres multiplicatifs respectifs de α^{k_1} et α^{k_2} sont premiers entre eux.

4.6.4 Amélioration lorsque H est linéaire

Dans le cas particulier où la fonction H par laquelle on approxime F est linéaire, c'est-à-dire $G(x) = \mathsf{Tr}(\lambda x^k)$ pour un k non premier avec $(2^n - 1)$, la sortie σ définie précédemment devient linéaire en l'état initial X_0^k. Dans ces conditions, l'ensemble des attaques par corrélation rapides peut s'appliquer afin de retrouver l'état initial en un temps plus rapide que la recherche exhaustive.

Or, comme le problème se ramène à un problème de décodage d'un code linéaire [MS89], le paramètre qui joue le plus grand rôle dans la complexité de l'algorithme est la dimension du code linéaire associé. La dimension de ce code est égale au degré du polynôme minimal de α^k, c'est-à-dire à la taille du LFSR défini par α^k. De manière équivalente, c'est aussi la taille de la classe cyclotomique de k, *i.e.* le plus petit entier m tel que $2^m \equiv 1 \mod \tau_k$. En d'autres termes, si l'élément α^k appartient au sous-corps $\mathbb{F}_{2^m}$ de $\mathbb{F}_{2^n}$, alors l'attaque par corrélation rapide s'applique sur un code de dimension m, et non sur un code de dimension n, où nécessairement m est un diviseur de n. Ainsi, l'attaquant.e est capable de retrouver $\log_2(\tau_k)$ bits d'information sur l'état initial, avec une complexité optimale au regard des attaques par corrélation. Évidemment, le cas le plus favorable pour l'attaquant.e est lorsque $\tau_k = 2^m - 1$, afin de retrouver le maximum d'information et non une sous-partie. Évaluer la complexité des attaques par corrélation rapides est chose ardue, puisqu'il existe plusieurs techniques et plusieurs compromis entre la quantité de données et le temps de calcul [MS89, JJ99, JJ00, CT00, CJS01, MFI01, CJM02]. Nous ne nous attarderons pas sur ce point, mais il est cependant important de se souvenir que l'attaque est plus performante si la fonction H est linéaire.

Exemple 4. Soit un LFSR de taille 10, de polynôme caractéristique

$$P(X) = X^{10} + X^9 + X^7 + X^6 + X^5 + X^4 + X^3 + X^2 + 1 \ .$$

Nous choisissons une fonction booléenne F à 10 variables équilibrée ayant une bonne non-linéarité suivant la construction de Dobbertin [8] [Dob95]. Nous choisissons donc une fonction booléenne courbe, constante sur un espace de dimension $\frac{n}{2}$, sur lequel nous remplaçons la moitié des valeurs de la fonction pour la rendre équilibrée. Par exemple, nous prenons la fonction $\mathsf{Tr}^{10}(\alpha x^{33})$ où α est une racine

8. Les fonctions courbes n'étant jamais équilibrées, nous ne les utilisons pas en pratique, la construction de Dobbertin permet d'obtenir des fonctions booléennes équilibrées dont la non-linéarité est optimale en considérant cette restriction.

(primitive) de P, et nous la modifions selon cette construction. La modification de cette fonction qui consiste à changer quelques valeurs dans la table de vérité augmente drastiquement le nombre de monômes dans sa représentation univariée, ce qui augmente d'autant la complexité linéaire de la suite chiffrante. Sur 10 variables, cette fonction équilibrée ainsi obtenue a une non-linéarité de 481 et une immunité algébrique de 3. Nous avons calculé la complexité linéaire de la suite chiffrante, et elle est égale à 992. Ainsi, cette fonction de filtrage répond à tous les critères classiques d'immunité algébrique, de complexité linéaire et de non-linéarité. Cependant, c'est un excellent exemple pour montrer à quel point l'attaque décrite précédemment peut fonctionner avec succès. En effet, par construction, cette fonction est évidemment très proche (en distance de Hamming) de la fonction boolénne $G(x) = \mathsf{Tr}^{10}(\alpha x^{33})$, ce qui signifie que la suite chiffrante est fortement corrélée à la sortie du LFSR de polynôme caractéristique $P_{\alpha^{33}}$. Plus précisément, le biais entre ces deux suites vaut $\varepsilon = 1 - 2^{-9}\mathsf{d}_H(F, G) = 0.96$. Nous pouvons donc appliquer une attaque par corrélation rapide sur un LFSR de taille 5, puis retrouver 5 bits de l'état initial du registre [9].

L'attaque est nettement plus rapide que l'attaque par corrélation rapide classique puisque le biais considéré vaut $\varepsilon = 0.96$ et que la dimension du code vaut 5. Pour l'attaque classique, ces valeurs sont respectivement $\varepsilon' = 1 - 481 \times 2^{-9} = 0.06$ et 10 pour la dimension du code linéaire associé. Les 33 dernières possibilités restantes peuvent être examinées par une recherche exhaustive.

Analyse de l'avantage. L'exemple utilisé précédemment peut être vu comme un exemple trop particulier et trop dégénéré puisque nous l'avons construit en utilisant une fonction monomiale, ce qui n'est jamais le cas dans la pratique : dans la pratique, les fonctions de filtrage utilisées sont construites en utilisant la forme algébrique normale pour des raisons évidentes d'implémentation. Cependant, la situation précédente peut aussi apparaître dans des cas moins dégénérés où la fonction de filtrage n'a pas de structure spécifique. Afin de quantifier avec précision l'avantage de l'attaquant.e au regard de notre nouvelle attaque, il est tout d'abord nécessaire d'analyser précisément la complexité des meilleures attaques par corrélation rapides.

La complexité des attaques par corrélation rapides a déjà été décrite par la formule 4.1 section 4.3.3 page 73. Par exemple, la complexité de l'attaque originale qui utilise des équations de poids 3 a un coût estimé de

$$\mathsf{Donnees} = \mathcal{O}\left(\frac{1}{\varepsilon} \times 2^{\frac{n}{2}}\right) \text{ and } \mathsf{Temps} = \mathcal{O}\left(\left(\frac{1}{\varepsilon}\right)^3 \times 2^{\frac{n}{2}}\right)$$

où n est ici la taille du registre. Utiliser des équations de parité ayant un plus grand poids w permet de diminuer l'influence de la taille du LFSR en replaçant $2^{n/2}$ par $2^{n/(w-1)}$. Cependant, ceci augmente l'influence du biais, où le terme en ε de la complexité en données est remplacé par $\varepsilon^{2(w-2)/(w-1)}$. Sans rentrer dans les détails car ce n'est pas le sujet ici, la complexité en temps peut être améliorée

9. "presque" à cause du $2^5 - 1$.

par plusieurs techniques, mais la complexité en données de la majorité de ces algorithmes a approximativement un comportement similaire.

Par ailleurs, et de manière indépendante de notre objet d'étude, la recherche de polynôme multiple d'un polynôme donné ayant un petit poids et un degré suffisamment petit est un problème algorithmiquement difficile. Dans la littérature, il est noté LWPM (Low Weight Polynomial Multiple). Dans le cadre des LFSR, la recherche d'un tel polynôme coûte en précalcul. Cependant, il se peut que la recherche d'un tel polynôme coûte cher, et ce coût doit quand même être pris en compte dans la complexité totale des attaques par corrélation rapides. Ce problème étant difficile, il est à l'origine du chiffrement asymétrique TCHo [AFMV07], qui utilise comme clef privée un polynôme de petit poids, multiple d'un polynôme dense définissant la clef publique du chiffrement. Il est donc possible de concevoir un algorithme de chiffrement à flot à l'aide d'un LFSR dont le polynôme caractéristique possède un multiple creux, ce qui constituerait un chiffrement à flot avec une trappe, puisqu'il est difficile de retrouver ledit polynôme creux multiple.

Exemple 5. On considère dans cet exemple le même polynôme de degré 10 que celui décrit dans l'exemple 4, mais nous choisissons maintenant une fonction avec seulement 6 variables dont la forme algébrique normale est donnée par $f(x_0,x_1,x_2,x_3,x_4,x_5) = x_0x_1x_2x_3x_4 + x_0x_1x_2x_3x_5 + x_0x_1x_2x_4x_5 + x_0x_1x_2x_4 + x_0x_1x_2 + x_0x_1x_3x_4 + x_0x_1x_3 + x_0x_1x_4 + x_0x_1x_5 + x_0x_1 + x_0x_2x_3x_4 + x_0x_2x_3x_5 + x_0x_2x_4x_5 + x_0x_2x_4 + x_0x_2 + x_0x_3x_4 + x_0x_4 + x_0 + x_1x_2x_3x_4x_5 + x_1x_2x_3x_4 + x_1x_2x_3x_5 + x_1x_2x_3 + x_1x_2x_4 + x_1x_2 + x_1x_3x_5 + x_1x_3 + x_1x_4 + x_1x_5 + x_1 + x_2x_3x_4x_5 + x_2x_3x_4 + x_2x_3x_5 + x_2x_3 + x_2 + x_3x_4 + x_4x_5 + x_4$.

De plus, les entrées de f sur le registre de taille 10 sont données par les valeurs $(\gamma_1,\ldots,\gamma_6) = (9,8,6,3,1,0)$. La *non-linéarité* de f considérée comme une fonction booléenne à 10 variables vaut 352, son immunité algébrique vaut 3 (ce qui est optimal pour une fonction à 6 variables), et la complexité linéaire de la sortie de ce générateur vaut 637. Cependant, il existe une fonction G de la forme $G(x) = \mathsf{Tr}(\lambda x^{33})$, qui est à distance 456 de F (selon la distance de Hamming). La corrélation entre la suite chiffrante et la sortie du registre non-filtré de taille 5 est donc dans ce cas égal à $\varepsilon = 0.11$. Ainsi, notre nouvelle attaque par corrélation est plus efficace qu'une attaque par corrélation rapide classique ($n = 10$ et $\varepsilon' = 0.31$). Plus précisément, si l'on prend l'algorithme itératif de décodage qui utilise des équations de parité de poids 3, alors le ratio entre les complexités en données des 2 attaques vaut

$$\frac{\mathsf{Donnees}}{\mathsf{Donnees}'} = \left(\frac{\varepsilon'}{\varepsilon}\right) \times 2^{\frac{n_{33}-n}{2}} = 0.498 \ .$$

4.6.5 Amélioration à l'aide d'une transformée de Fourier

Dans le cas général, c'est-à-dire dans le cas où la fonction booléenne H est non-linéaire, l'attaque par corrélation classique [Sie85] consiste à réaliser une recherche exhaustive sur tous les états initiaux possibles ($Y_0 = X_0^k$), avec

$\mathsf{pgcd}(k, 2^n - 1) = 1$. Pour chaque Y_0 possible, les N premiers bits de la suite correspondante σ sont engendrés, puis l'attaquant.e calcule la corrélation entre la suite chiffrante $\mathbf{s}$ observée, c'est-à-dire la quantité

$$\sum_{t=0}^{N-1} (-1)^{s_t+\sigma_t} \tag{4.2}$$

où N est le nombre de bits de la suite chiffrante nécessaire à la détection du biais, *i.e.*,

$$N = \frac{2\ln(\tau_k)}{\varepsilon^2}$$

où ε est la corrélation attendue entre la sortie $\mathbf{s}$ et σ. La complexité en temps de cet algorithme est donc proportionnelle à

$$\tau_k \times N = \frac{2\tau_k \ln(\tau_k)}{\varepsilon^2} \ .$$

Dans ce qui suit, nous montrons que nous pouvons appliquer la même technique que celle utilisée pour la combinaison de générateurs dans [Nay07, CN12] dans notre contexte, quand on approxime la sortie du générateur par la sortie d'un autre générateur dont le nombre d'états internes possibles est réduit.

Soit k un entier non premier avec $(2^n - 1)$, on note toujours τ_k l'ordre multiplicatif de l'élément α^k où α est un élément primitif du corps fini $\mathbb{F}_{2^n}$. On note $\langle \alpha^k \rangle$ le sous-groupe multiplicatif de $\mathbb{F}_{2^n}^*$ engendré par α^k, *i.e.*, l'ensemble de cardinal τ_k : $\langle \alpha^k \rangle = \{1, \alpha^k, \alpha^{2k}, \ldots, \alpha^{(\tau_k-1)k}\}$. Cet ensemble est composé de tous les états internes possibles ($Y_0 = X_0^k$) du registre de polynôme caractéristique P_{α^k}. Le but de l'attaquant.e ici est donc de retrouver l'état initial $Y_0 \in \langle \alpha^k \rangle$ qui maximise la quantité (4.2) avec $\sigma_t = H(Y_0\alpha^{kt})$.

Ainsi, pour tout $Y_0 \in \langle \alpha^k \rangle$, l'attaquant.e calcule la quantité

$$\mathcal{Z}(Y_0) = \sum_{t=0}^{N-1} (s_t \oplus \sigma_t) = \sum_{r=0}^{\tau_k-1} \sum_{q=0}^{\lceil \frac{N-r}{\tau_k} \rceil - 1} (s_{q\tau_k+r} \oplus \sigma_{q\tau_k+r}) \ .$$

Comme la suite σ est de période τ_k, on a $\sigma_t = \sigma_{t+\tau_k}$ pour tout $t \geq 0$. On obtient donc

$$\mathcal{Z}(Y_0) = \sum_{r=0}^{\tau_k-1} (\sigma_r \oplus 1) \left(\sum_{q=0}^{\lceil \frac{N-r}{\tau_k} \rceil - 1} s_{q\tau_k+r} \right) + \sum_{r=0}^{\tau_k-1} \sigma_r \left(\left\lceil \frac{N-r}{\tau_k} \right\rceil - \sum_{q=0}^{\lceil \frac{N-r}{\tau_k} \rceil - 1} s_{q\tau_k+r} \right)$$

Pour tout r compris entre 0 et τ_k, on définit la quantité $\Sigma_{\mathbf{s}}(r)$:

$$\Sigma_{\mathbf{s}}(r) = \sum_{q=0}^{\lceil \frac{N-r}{\tau_k} \rceil - 1} s_{q\tau_k+r} \ .$$

Ainsi, on obtient

$$\mathcal{Z}(Y_0) = \sum_{r=0}^{\tau_k-1} (\sigma_r \oplus 1)\Sigma_{\mathbf{s}}(r) + \sum_{r=0}^{\tau_k-1} \sigma_r \left(\left\lceil \frac{N-r}{\tau_k} \right\rceil - \Sigma_{\mathbf{s}}(r) \right) .$$

Comme σ est une suite binaire, on utilise le fait que $(-1)^{\sigma_r} = 1 - 2\sigma_r$ et $-(-1)^{\sigma_r} = 1 - 2(\sigma_r \oplus 1)$ pour tout $r \geq 0$, donc

$$\begin{aligned}\mathcal{Z}(Y_0) =& \frac{1}{2} \sum_{r=0}^{\tau_k-1} (1 + (-1)^{\sigma_r})\Sigma_{\mathbf{s}}(r) + (1 + (-1)^{\sigma_r}) \left\lceil \frac{N-r}{\tau_k} \right\rceil - (1 - (-1)^{\sigma_r})\Sigma_{\mathbf{s}}(r) \\ =& \frac{1}{2} \sum_{r=0}^{\tau_k-1} 2(-1)^{\sigma_r}\Sigma_{\mathbf{s}}(r) + (-1)^{\sigma_r} \left(\left\lceil \frac{N-r}{\tau_k} \right\rceil \right) + \left\lceil \frac{N-r}{\tau_k} \right\rceil .\end{aligned}$$

Finalement, on obtient

$$\mathcal{Z}(Y_0) = \frac{N}{2} + \sum_{r=0}^{\tau_k-1} (-1)^{\sigma_r} \left(\Sigma_{\mathbf{s}}(r) - \frac{1}{2} \left\lceil \frac{N-r}{\tau_k} \right\rceil \right) .$$

Ainsi, la corrélation entre σ et $\mathbf{s}$ s'exprime de la manière suivante :

$$\sum_{t=0}^{N-1} (-1)^{s_t + \sigma_t}(Y_0) = \sum_{t=0}^{N-1} 1 - 2(s_t \oplus \sigma_t) = N - 2\mathcal{Z}(Y_0) ,$$

donc

$$\sum_{t=0}^{N-1} (-1)^{s_t + \sigma_t(Y_0)} = \sum_{r=0}^{\tau_k-1} (-1)^{\sigma_r(Y_0)} \left(\left\lceil \frac{N-r}{\tau_k} \right\rceil - 2\Sigma_{\mathbf{s}}(r) \right) .$$

Nous cherchons à calculer cette valeur pour tous les éléments Y_0 de la forme α^{ik}, pour i compris entre 0 et τ_k, cependant, les différentes suites σ engendrées par les Y_0 sont seulement décalées les unes par rapport aux autres. Plus précisément, pour tout $t \geq 0$, on a

$$\sigma_t(\alpha^{ik}) = H(\alpha^{ik}\alpha^{tk}) = H(\alpha^{(t+i)k}) = \sigma_{t+i}(\alpha^0) = \sigma_{t+i}(1) .$$

Dans ces conditions, la recherche de Y_0 se ramène à la recherche de l'entier i, compris entre 0 et τ_k, qui maximise la quantité

$$\sum_{r=0}^{\tau_k-1} (-1)^{\sigma_{r+i \bmod \tau_k}(1)} \left(\left\lceil \frac{N-r}{\tau_k} \right\rceil - 2\Sigma_{\mathbf{s}}(r) \right) ,$$

ce qui correspond finalement au produit de convolution de deux vecteurs de taille τ_k : $(\sigma_t(1))_{0 \leq t < \tau_k}$ et $(\Sigma_{\mathbf{s}}(t))_{0 \leq t < \tau_k}$. Trouver l'entier i qui nous intéresse peut se faire à l'aide d'une transformée de Fourier rapide, dont la complexité en temps est de l'ordre de $\mathcal{O}(\tau_k \log(\tau_k))$ opérations [Bla85, Jou09, page 299]. La complexité en mémoire de l'attaque est en $\mathcal{O}(\tau_k)$, et la complexité totale de notre attaque est donc de l'ordre de

$$\mathsf{Temps} = \tau_k \log(\tau_k) + \frac{2\ln(\tau_k)}{\varepsilon^2} .$$

Exemple 6. On considère maintenant, pour étayer notre propos, un LFSR de taille 12 et de polynôme caractéristique

$$P(X) = X^{12} + X^{10} + X^9 + X^8 + X^7 + X^5 + X^4 + X^3 + X^2 + X + 1\ ,$$

et filtré par la même fonction booléenne que celle utilisée dans l'exemple 5, mais en choisissant les entrées de la fonction selon la suite $(\gamma_1, \ldots, \gamma_6) = (11, 10, 7, 5, 2, 0)$. Alors, la corrélation entre F et toute fonction de la forme $G = \mathsf{Tr}(\lambda x^k)$, avec $k = \ell\frac{2^n-1}{2^m-1}$ où $\mathsf{pgcd}(\ell, 2^n-1) = 1$ est toujours trop petite pour réaliser de manière efficace une attaque par corrélation rapide sur un registre de taille plus petite. En revanche, il est possible d'utiliser l'exposant $k = 45$. L'ordre multiplicatif de α^{45} pour un α primitif vaut 91 : $\mathsf{ord}(\alpha^{45}) = 91$. Dans ce cas, il est possible d'augmenter le biais, puisque l'on s'autorise toutes les fonctions H possibles et pas uniquement les fonctions linéaires. La meilleure approximation par une fonction de la forme $G(x) = H(x^k)$ donne une corrélation de 0.125. En utilisant une transformée de Fourier rapide comme décrit précédemment, notre attaque requiert approximativement $(592 + 574) = 1166$ opérations, et nécessite 574 bits de suite chiffrante. Le reste de l'état initial est ensuite retrouvé par recherche exhaustive.

4.6.6 Approximations de F par une fonction de type $H(x^k)$

Toutes les attaques précédentes exploitent l'existence d'une fonction G de la forme $G(x) = H(x^k)$, pour k non premier avec $(2^n - 1)$, dont la corrélation avec F est élevée. En particulier, les attaques par corrélation rapides décrites à la section 4.6.4 faisant intervenir un registre de taille plus petite que le registre initial nous imposent de définir un critère plus large que le critère de *non-linéarité généralisée* [YG01]. Il est donc nécessaire de prendre en considération la distance à toutes les fonctions de la forme $\mathsf{Tr}(\lambda x^k)$, où $k = \ell\frac{2^n-1}{2^m-1}$, quand m est un diviseur de n et $\mathsf{pgcd}(\ell, 2^m - 1) = 1$ (c'est-à-dire quand x^k décrit le sous-corps $\mathbb{F}_{2^m}$ de $\mathbb{F}_{2^n}$). Ce critère est bien plus pertinent que de considérer uniquement la distance à toutes les fonctions composantes d'une *permutation* monomiale.

De plus, ce critère n'est pas le seul à prendre en compte : si n est premier (*i.e.* le seul sous-corps est $\mathbb{F}_2$), mais que $(2^n - 1)$ n'est pas un nombre premier, alors nous mettons ici en avant la possibilité de réaliser malgré tout une attaque, de type *diviser pour mieux régner*, en appliquant la technique de Siegenthaler [Sie85]. Nous discuterons de ces nouveaux critères à la fin de ce document.

Dans ce qui suit nous nous intéressons à calculer la meilleure approximation de la fonction de filtrage F par une fonction de la forme $G(x) = H(x^k)$. Sans perdre de généralité et par souci de lisibilité, nous prenons pour k un diviseur de $(2^n - 1)$, quitte à remplacer k par $\mathsf{pgcd}(k, 2^n - 1)$. De plus, on note $\tau = \tau_k = \mathsf{ord}(\alpha^k) = (2^n - 1)/k$. Soit $\langle\alpha^\tau\rangle$ le sous-groupe cyclique d'ordre k de $\mathbb{F}_{2^n}^*$ engendré par α^τ. Nous définissons les ensembles $(E_i)_{0\leq i<\tau}$ de $\mathbb{F}_{2^n}$ de la manière suivante :

$$E_i = \alpha^i\langle\alpha^\tau\rangle = \left\{\alpha^i, \alpha^{i+\tau}, \alpha^{i+2\tau}, \ldots, \alpha^{i+(k-1)\tau}\right\}\ .$$

Par construction, les E_i pour i compris entre 0 et $\tau - 1$ sont disjoints et forment une partition du groupe multiplicatif $\mathbb{F}_{2^n}^*$:

$$\mathbb{F}_{2^n}^* = \bigcup_{i=0}^{\tau-1} E_i \ .$$

Comme G est de la forme $H(x^k)$, G est constante sur chacun des ensembles E_i, $0 \leq i < \tau$; en effet, pour $x = \alpha^i \times \alpha^{j\tau}$, on a

$$G(x) = H((\alpha^i \alpha^{j\tau})^k) = H(\alpha^{ik}) \ .$$

Ainsi, nous pouvons exprimer la corrélation entre F et G de la manière suivante :

$$\begin{aligned} \sum_{x \in \mathbb{F}_{2^n}} (-1)^{F(x)+H(x^k)} &= 1 + \sum_{x \in \mathbb{F}_{2^n}^*} (-1)^{F(x)+H(x^k)} \\ &= 1 + \sum_{i=0}^{\tau-1} (-1)^{H(\alpha^{ik})} \left(\sum_{y \in E_i} (-1)^{F(y)} \right) \end{aligned} \tag{4.3}$$

en ayant pris soin de choisir $H(0) = F(0)$.

Cas pgcd$(k, \tau) = 1$. Si k et τ sont premiers entre eux, alors il apparaît que tous les éléments α^{ik}, pour $0 \leq i < \tau$ se trouvent tous dans des ensembles E_j différents. Dans ce cas, la fonction H qui maximise la corrélation entre G et F est définie par

$$H(\alpha^{ik}) = \begin{cases} 0 & \text{si } \sum_{y \in E_i} (-1)^{F(y)} > 0 \\ 1 & \text{si } \sum_{y \in E_i} (-1)^{F(y)} < 0 \end{cases} \ .$$

En d'autres termes, $H(\alpha^{ik}) = 1$ si et seulement si le poids de Hamming de la table de vérité de F quand les entrées sont restreintes au sous-ensemble E_i est strictement plus grand que $k/2$. De plus, comme k est un diviseur de $(2^n - 1)$, c'est un entier nécessairement impair, ce qui implique l'unicité de la fonction H qui maximise la corrélation entre F et G. Pour ce choix optimal de la fonction H, on a

$$\sum_{x \in \mathbb{F}_{2^n}} (-1)^{F(x)+H(x^k)} = 1 + \sum_{i=0}^{\tau-1} \left| \sum_{y \in E_i} (-1)^{F(y)} \right| \geq 1 + \tau$$

puisque le nombre de termes dans chaque E_i est impair, impliquant que

$$\sum_{i=0}^{\tau-1} \left| \sum_{y \in E_i} (-1)^{F(y)} \right| \geq \tau \ .$$

Finalement, pour n'importe quelle fonction F, il existe toujours une (unique) fonction H telle que la corrélation entre F et G de la forme $G(x) = H(x^k)$ soit au moins de $(1 + \tau)2^{-n} \backsimeq k^{-1}$.

Cas $\mathsf{pgcd}(k, \tau) > 1$. Dans le cas où l'exposant k n'est pas premier avec l'ordre multiplicatif de α^k, un phénomène particulier apparaît : les éléments α^{ik} et $\alpha^{(i+\frac{\tau}{d})k}$ se retrouvent dans le même ensemble E_j, où $d = \mathsf{pgcd}(k, \tau) > 1$. En effet, $\alpha^{\frac{k\tau}{d}}$ appartient à $\langle \alpha^\tau \rangle$. Dans ces conditions, nous réécrivons l'équation (4.3) de la manière suivante :

$$\sum_{x \in \mathbb{F}_{2^n}} (-1)^{F(x)+H(x^k)} = 1 + \sum_{i=0}^{\frac{\tau}{d}-1} (-1)^{H(\alpha^{ik})} \left(\sum_{j=0}^{d-1} \left(\sum_{y \in E_{i+j\frac{\tau}{d}}} (-1)^{F(y)} \right) \right) .$$

Alors, la valeur de la fonction H au point α^{ik} qui maximise la corrélation est maintenant définie par le poids de Hamming de F restreint à l'ensemble $\bigcup_{j=0}^{d-1} E_{i+j\frac{\tau}{d}}$. Cet ensemble est aussi de cardinalité impaire, ce qui nous permet de conclure de la même manière que précédemment que, pour n'importe quelle fonction F, la corrélation entre F et G de la forme $H(x^k)$ est au moins $(1+\frac{\tau}{d})2^{-n}$.

Il est important de noter que ce critère est fondamentalement nouveau et n'est a priori pas lié aux critères habituels. Pour ne donner qu'un exemple, il existe des fonctions courbes qui sont constantes sur les ensembles $\lambda\langle\alpha^\tau\rangle$ pour $\tau = 2^{n/2} + 1$ [Dil74], ce qui montre bien la nécessité de considérer l'ensemble des nouveaux critères sur les fonctions de filtrage (ce que nous essaierons de faire dans les perspectives à la fin de ce document) afin d'évaluer plus en détail la sécurité des registres filtrés.

4.7 Ce qu'il faut retenir

Dans ce chapitre, nous avons investigué la sécurité des registres filtrés, au regard de l'équivalence monomiale introduite par Rønjom et Cid en 2011 qui consiste à changer la racine primitive qui définit la fonction de mise à jour de l'état interne. Tandis que ce type de registre était considéré sûr à partir du moment où la fonction de filtrage était choisie avec une bonne *immunité algébrique* et une bonne *non-linéarité*, nous avons montré que des phénomènes plus étranges pouvaient apparaître.

Tout d'abord, nous avons vu que modifier la racine primitive ne modifie pas la complexité des meilleures attaques algébriques. De plus, il est apparu que le critère pertinent à prendre en compte pour évaluer la sécurité au regard des attaques algébriques n'est pas l'*immunité algébrique*, mais l'*immunité spectrale*, pour laquelle on ne regarde plus les fonctions sous leur forme multivariée, mais sous leur forme univariée.

Ensuite, comme l'équivalence monomiale peut modifier la *non-linéarité*, nous avons aussi expliqué que le critère pertinent pour assurer la résistance aux attaques par corrélation n'était pas la non-linéarité, mais la *non-linéarité généralisée*, c'est-à-dire la distance à toutes les fonctions composantes de permutations monomiales, et non uniquement aux fonctions affines.

De plus, ce dernier critère n'est pas le plus important, puisque nous avons réussi à décomposer l'état interne du registre en utilisant la structure particulière

des corps finis, en exploitant les différents sous-groupes multiplicatifs de $\mathbb{F}_{2^n}^*$, afin de retrouver l'état initial par une attaque de type *diviser pour mieux régner*. C'est cette attaque qu'il faut principalement retenir, puisqu'elle est généralement la plus performante et met en évidence de potentielles failles de ce type de générateurs, sans aucun lien avec les attaques précédentes connues.

En revanche, tout n'est pas si rose pour l'attaquant.e. Commme nous l'avons dit plusieurs fois, en pratique, la fonction de filtrage est une fonction ne prenant qu'une ou quelques dizaines de variables en entrée. Quand on examine la représentation univariée, nous devons considérer la fonction comme ayant n variables où n est la taille du registre, typiquement 128 ou 256. Les critères classiques d'immunité algébrique et de non-linéarité peuvent être naturellement évalués par un calcul sur la dizaine de variables et non en considérant la taille totale du registre, ce qui n'est plus le cas pour la non-linéarité généralisée et pour la distance aux fonctions de la forme $H(x^k)$. Ceci rend donc tout calcul exact de ces nouveaux critères hors de portée pour les tailles utilisées en pratique. Le simple calcul de la représentation univariée de F comme polynôme sur $\mathbb{F}_{2^n}$ coûterait beaucoup trop cher : c'est d'ailleurs la raison pour laquelle nous n'avons pu donner que des exemples sur des registres de petite taille.

Finalement, la faisabilité de notre attaque est principalement due à la forte structure induite par la fonction de rétroaction particulière des LFSR, qui correspond à la multiplication par un élément primitif du corps. Paradoxalement, cette fonction de rétroaction est utilisée pour assurer les propriétés statistiques de la suite chiffrante. Ainsi, d'un côté, nous avons besoin d'utiliser une structure mathématique forte afin de pouvoir garantir certaines propriétés, cependant, cette même structure amène des vulnérabilités inattendues. Ceci nous interroge donc sur le réel intérêt de l'utilisation de registres à décalage à rétroaction linéaire. À moins d'utiliser par exemple un registre de taille 521, puisque le nombre de Mersenne $2^{521} - 1$ est un nombre premier.

Chapitre 5

Cryptanalyse de FLIP

Dans le chapitre précédent, nous étudiions les LFSR filtrés. Ceux-ci ne sont généralement pas utilisés directement, mais de nombreux algorithmes pratiques peuvent être vus comme des variantes du LFSR filtré. Ceci implique que les attaques génériques décrites au début du chapitre précédent peuvent être adaptées à des cas pratiques. Un exemple est l'algorithme de chiffrement FLIP.

Dans ce chapitre, nous décrivons une attaque sur la famille de chiffrements à flot nommée FLIP [MJSC16]. C'est lors des journées nationales *Codage et Cryptographie* de 2015 que Pierrick Méaux présenta le système de chiffrement FLIP, adapté aux besoins spécifiques du chiffrement complètement homomorphe hybride. En collaboration avec Sébastien Duval et Virginie Lallemand, nous avons cryptanalysé cette famille de chiffrement puis communiqué aux auteurs nos résultats, ce qui leur a permis de modifier leur algorithme, qui sera publié *in fine* à la conférence *EUROCRYPT* en 2016 par P. Méaux, A. Journault, F.X. Standaert et C. Carlet. Notre travail donna lieu à l'article intitulé *Cryptanalysis of the FLIP Family of Stream Ciphers* [DLR16] à la conférence *CRYPTO* en 2016 [DLR16]. Ainsi, les paramètres attaqués correspondent aux paramètres initiaux présentés aux journées *Codage et Cryptographie* et non aux paramètres de la version finale publiée à *EUROCRYPT*.

5.1 Le chiffrement complètement homomorphe

5.1.1 Définitions

Définition 5.1. *Un système de chiffrement complètement homomorphe (FHE - Fully Homomorphic Encryption) est un système de chiffrement asymétrique pour lequel pour toute fonction f de $\mathcal{C} \times \mathcal{C}$ dans $\mathcal{C}$, la propriété suivante est vérifiée :*

$$\forall C_1, C_2 \in \mathcal{C},\ D(f(C_1, C_2)) = f(D(C_1), \mathcal{D}(C_2))$$

où D désigne la fonction de déchiffrement et $\mathcal{C}$ désigne l'espace des chiffrés.

En d'autres termes, la fonction de déchiffrement d'un système de chiffrement complètement homomorphe doit commuter avec deux opérations (la multiplication et l'addition) sur l'espace des chiffrés et l'espace des clairs. Cette propriété permet de déléguer des calculs à un serveur, sans que celui-ci n'ait accès aux données en entrées de la fonction d'évaluation, ni au résultat de celle-ci.

C'est en 2009 que Craig Gentry proposa le premier système de chiffrement asymétrique complètement homomorphe [Gen09] basé sur le problème difficile de recherche de plus court vecteur dans un réseau euclidien. Sans rentrer dans les détails, ce type de cryptosystème produit le chiffré à partir d'éléments d'un réseau euclidien auxquels on ajoute une erreur selon une certaine distribution. La clef privée correspond à une "bonne" base définissant ledit réseau et la clef publique à une "mauvaise" base. Alors, retrouver le point du réseau auquel on a ajouté l'erreur est facile avec la bonne base est difficile avec la mauvaise base, ces deux bases définissant le même réseau euclidien. Alors que l'addition n'augmente que très peu le niveau d'erreur dans les chiffrés, il n'en est pas de même pour la multiplication. Ainsi, à partir d'un certain nombre de multiplications le niveau d'erreur augmente trop, rendant impossible le déchiffrement. Ce problème est résolu par l'utilisation d'une technique appelée *bootstrapping*, qui permet de limiter le niveau d'erreur. Un des problèmes du chiffrement complètement homomorphe est son taux d'expansion : pour chiffrer 1 bit, la taille du chiffré correspondant est de l'ordre d'une centaine de kilo-octets. Il est donc impossible de transmettre un message chiffré avec un système de chiffrement asymétrique complètement homomorphe.

5.1.2 Le chiffrement hybride

Cependant, ce problème peut être résolu en utilisant une technique hybride, qui combine un système de chiffrement complètement homomorphe asymétrique avec un cryptosystème symétrique [NLV11].

Considérons un algorithme de chiffrement asymétrique complètement homomorphe donné par HE_{pk} la fonction de chiffrement associée à la clef publique d'Alice, ainsi qu'un algorithme de chiffrement symétrique défini par E_K, la fonction de chiffrement associée à la clef secrète *symétrique* d'Alice. La clef secrète d'Alice K est évidemment indépendante de sa clef privée sk utilisée pour le chiffrement asymétrique complètement homomorphe.

Alors le fonctionnement du chiffrement homomorphe hybride est le suivant (voir figure 5.1) :

— Alice chiffre sa clef secrète *symétrique*, avec le chiffrement asymétrique complètement homomorphe : $HE_{pk}(K)$, et envoie cette valeur au serveur.

— Alice chiffre ses données m à l'aide du chiffrement symétrique : $E_K(m)$, et envoie cette valeur (accompagnée de la fonction d'évaluation) au serveur.

— Le serveur n'a pas accès à K, mais seulement à $HE_{pk}(K)$. Il ne peut donc pas retrouver les données m, cependant, il peut *évaluer homomorphiquement*, en utilisant les propriétés de l'algorithme asymétrique, le système de

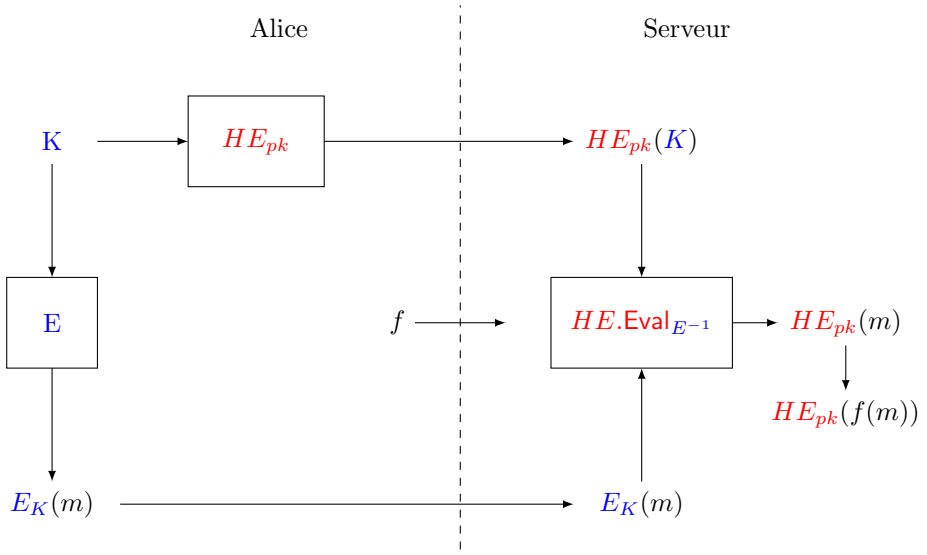

Figure 5.1 – Schéma générique du chiffrement homomorphe hybride.

déchiffrement symétrique : E_K^{-1}. Ainsi, le serveur est capable de calculer $HE_{pk}(m)$, ce qui lui permettra ensuite d'appliquer la fonction d'évaluation demandée par Alice.

— Alice reçoit le résultat de l'évaluation de f, chiffré homomorphiquement. Il ne lui reste plus qu'à déchiffrer à l'aide de sa clef privée sk.

5.1.3 Les chiffrements symétriques adaptés

Sans rentrer dans les détails, la profondeur multiplicative du circuit du chiffrement symétrique utilisé dans un chiffrement hybride joue un rôle fondamental dans le coût total du chiffrement. Par exemple, utiliser le standard AES dans le cadre du FHE est extrêmement coûteux [GHS12, CCK+13, DHS16]. Les chiffrements par bloc à bas coût proposés récemment ne sont pas non plus une bonne solution car ils comportent trop d'itérations [DSES14, LN14] et peuvent même amener à de bien moins bonnes performances que l'AES. En revanche, plusieurs constructions existantes sont mieux adaptées. Nous pouvons citer par exemple le chiffrement par bloc appelé LowMC [ARS+15] (en raison justement de la faible profondeur multiplicative du circuit correspondant), qui fut la première solution apportée pour un chiffrement symétrique adapté à la fois au chiffrement homomorphe, mais aussi à la problématique de MPC (Multi-Parti Computation).

Cependant, les chiffrements par bloc ne sont pas les seules options possibles, et il apparaît que le fonctionnement intrinsèque des chiffrements à flot permet de bien mieux contrôler la profondeur multiplicative, ce qui est très intéressant

dans le cadre du FHE. Ainsi, dans un article de 2016 [CCF+16], A. Canteaut et ses coauteur.e.s proposèrent d'utiliser une version modifiée (Kreyvium) d'un finaliste de la compétition eSTREAM : Trivium [Can06].

Finalement, à EUROCRYPT 2016, Pierrick Méaux et ses coauteurs ont proposé la famille de chiffrement à flot FLIP, dont ils avaient présenté une première version aux journées *Codage et Cryptographie* en 2015. Cependant, cette proposition initiale souffre principalement de l'existence d'un système d'équations trop spécifique, introduisant des faiblesses que nous décrivons dans ce qui suit.

5.2 Le chiffrement FLIP

5.2.1 "Filter Permutator"

Par analogie aux générateurs filtrés dont nous avons réalisé l'étude au chapitre 4, les auteurs de FLIP ont appelé la structure générique de leur chiffrement *filter permutator*. Le schéma générique de FLIP, décrit à la figure 5.2, combine trois composantes cryptographiques :

— un registre de taille N (jamais mis à jour) dans lequel est stockée la clef secrète ;

— un algorithme, initialisé uniquement par l'IV, qui engendre de manière publique des permutations de $\{1, \ldots, N\}$ n'ayant pas de propriétés distinguantes ;

— une fonction booléenne de filtrage notée $F \in \mathcal{B}_N$.

Ces trois composantes sont combinées de la manière suivante : une fois que l'algorithme de génération de permutations est initialisé par le vecteur d'initialisation, la clef secrète est stockée dans le registre. À chaque instant t, le générateur de permutations engendre une permutation P_t de l'ensemble $\{1, \ldots, N\}$, qui mélange les bits de la clef, avant d'appliquer la fonction de filtrage F qui produit un bit z_t, générant ainsi la suite chiffrante $\mathbf{z}$. Suivant le principe des chiffrements à flot additifs, le texte chiffré est ensuite obtenu en XORant bit à bit le message clair et la suite chiffrante.

Le générateur pseudo-aléatoire utilisé est basé sur AES-128, cependant, notre attaque n'exploite pas la génération de permutations, mais uniquement la fonction de filtrage utilisée. Nous supposons donc que le PRNG utilisé est suffisamment bon, et qu'aucune faiblesse n'est exploitable sur cette partie du chiffrement.

5.2.2 La fonction de filtrage

Naturellement, la fonction de filtrage ne peut pas être choisie n'importe comment : celle-ci doit avoir de bonnes propriétés cryptographiques afin d'éviter les attaques connues sur les chiffrements à flot comme les *attaques algébriques* ou les *attaques par corrélation*. Ainsi, les auteurs de FLIP ont choisi une fonction

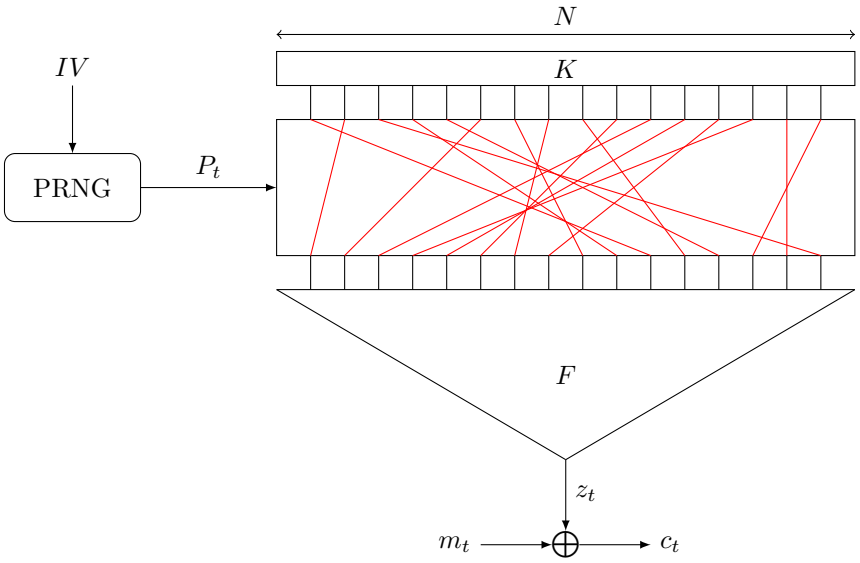

Figure 5.2 – Le chiffrement FLIP.

booléenne qui suit une construction particulière, afin d'assurer une certaine sécurité au regard des attaques classiques.

La fonction de filtrage F est construite à l'aide de la somme de 3 fonctions f_1, f_2 et f_3 dont les variables sont indépendantes. Ces fonctions sont définies selon les trois familles spécifiques qui suivent, pour lesquelles nous reprenons les notations des auteurs de FLIP.

Définition 5.2 (Fonctions de type L)**.** *Pour $n \in \mathbb{N}^*$, la n-ième fonction booléenne de type L, notée L_n, est la première fonction symétrique élémentaire à n variables, c'est-à-dire la somme de tous les monômes de degré 1 :*

$$L_n(x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} x_i \ .$$

Par exemple, $L_4(x_0, x_1, x_2, x_3) = x_0 + x_1 + x_2 + x_3$.

Définition 5.3 (Fonctions de type Q)**.** *Pour $n \in \mathbb{N}^*$, la n-ième fonction de type Q, notée Q_n, est une fonction booléenne quadratique à $2n$ variables définie par*

$$Q_n(x_0, \ldots, x_{2n-1}) = \sum_{i=0}^{n-1} x_{2i}x_{2i+1} \ .$$

Par exemple, $Q_3(x_0, x_1, x_2, x_3, x_4, x_5) = x_0x_1 + x_2x_3 + x_4x_5$.

Définition 5.4 (Fonctions de type T)**.** *Pour $n \in \mathbb{N}^*$, la n-ième fonction de type T (fonction dite "triangulaire") est la fonction booléenne de degré k à $\frac{k(k+1)}{2}$ variables définie par*

$$T_k(x_0, \ldots, x_{\frac{k(k+1)}{2}-1}) = \sum_{i=1}^{k} \prod_{j=0}^{i-1} x_{j+\sum_{\ell=0}^{i-1} \ell} \, .$$

Par exemple, $T_3(x_0, x_1, x_2, x_3, x_4, x_5) = x_0 + x_1x_2 + x_3x_4x_5$.

L'intérêt de ces trois types de fonctions booléennes est que chacune d'elle possède une propriété cryptographique particulière : les fonctions de type L apportent une bonne *résilience*, les fonctions de type Q ont une bonne *non-linéarité*, alors que les fonction triangulaires possèdent une bonne *immunité algébrique*. L'idée des auteurs de FLIP pour choisir la fonction de filtrage est d'utiliser une somme de ces trois familles, dont les variables sont prises indépendamment, afin que la fonction de filtrage globale hérite des trois propriétés cryptographiques classiques.

Plus précisément, soit trois entiers n_1, n_2 pair et n_3 de la forme $\frac{k(k+1)}{2}$. On définit f_1, f_2 et f_3 trois fonctions booléennes à n_1 (respectivement n_2 et n_3 variables) telles que

— $f_1(x_0, \ldots, x_{n_1-1}) = L_{n_1}$;

— $f_2(x_{n_1}, \ldots, x_{n_1+n_2-1}) = Q_{n_2/2}$;

— $f_3(x_{n_1+n_2}, \ldots, x_{n_1+n_2+n_3-1}) = T_k$ où $n_3 = \frac{k(k+1)}{2}$.

Alors la fonction de filtrage utilisée dans FLIP est définie par

$$F(x_0, \ldots, x_{n_1+n_2+n_3-1}) = L_{n_1} + Q_{n_2/2} + T_k$$

où $n_1 + n_2 + n_3 = N$. En utilisant cette construction particulière, les auteurs de FLIP ont alors pu déterminer exactement la valeur de la non-linéarité, de l'immunité algébrique et de la résilience de F. Cependant, l'analyse cryptographique de ce *permutateur filtré* [MJSC16] a été réalisée par les auteurs en transposant les arguments classiques, notamment les arguments utilisés pour analyser la sécurité des registres filtrés. Or, étant donné le caractère très particulier de la construction de FLIP, ces arguments ne peuvent plus être utilisés, puisque le poids de Hamming en entrée de la fonction de filtrage est constant. Comme les critères de non-linéarité et d'immunité algébrique sont étudiés sur tout l'espace $\mathbb{F}_2^n$, il convient d'affiner ces critères. Nous avons communiqué ces remarques aux auteurs de FLIP, ce qui nous a conduit à une collaboration avec Pierrick Méaux et Claude Carlet, où nous justifions *rigoureusement* le choix de conception de FLIP. Ces travaux seront décrits dans le chapitre 6 et ont été publiés dans un article aux *IACR Transactions on Symmetric Cryptology* [CMR17a].

Afin d'éviter l'existence de clefs faibles, et d'assurer un espace des clefs plus grand que 2^{80} ou 2^{128}, les auteurs de FLIP ont suggéré que la clef soit choisie avec un poids de Hamming de $\frac{N}{2}$. Toujours au chapitre 6 nous verrons pourquoi il est important d'avoir une clef équilibrée. Au regard de leur analyse initiale, les

auteurs de FLIP ont à l'origine proposé les paramètres décrits dans la table 5.1, en prétendant une sécurité de 80 bits (respectivement 128 bits) pour une taille de clef de 192 bits (respectivement 400 bits).

FLIP(n_1, n_2, n_3)	Taille de clef (N)	Sécurité (en bits)
FLIP(47,40,105)	192	80
FLIP(87,82,231)	400	128

Table 5.1 – Paramètres des premières versions de FLIP et le niveau de sécurité revendiquée par les auteurs.

5.2.3 Caractéristiques générales de FLIP

Rappelons tout d'abord les principales caractéristiques de FLIP que nous allons exploiter pour mettre à mal sa sécurité.

— Le registre dans lequel la clef est stockée initialement n'est jamais mis à jour ;

— la clef est de poids de Hamming $\frac{N}{2}$ afin d'éviter les clefs faibles ;

— une permutation aléatoire est engendrée à chaque instant (notée P_t pour $t \geq 0$), impliquant que les équations décrivant la suite chiffrante sont toutes de la forme

$$z_t = F(K_{P_t(0)}, K_{P_t(1)}, \ldots, K_{P_t(N-1)})$$

où K_i pour $0 \leq i \leq N-1$ désigne le i-ième bit de la clef K ;

— F est une fonction booléenne ayant de bonnes propriétés cryptographiques.

5.3 Remarques générales

5.3.1 Modèle d'attaque

Étant donné que nous étudions un chiffrement à flot, nous nous plaçons dans le modèle le plus courant des attaques contre ces derniers, c'est-à-dire le modèle à clair connu. Ainsi, nous supposons que nous avons accès à une partie de la suite chiffrante $\mathbf{z}$. Notre attaque n'est pas un distingueur sur cette suite, mais une attaque par recouvrement de clef, ce qui, dans le cas particulier de FLIP est équivalent à retrouver tout l'état, puisqu'il n'y a pas de phase d'initialisation comme c'est habituellement le cas. Afin de quantifier avec précision l'avantage de l'attaquant.e, nous utilisons classiquement la complexité en données notée C_D et la complexité en temps notée C_T [1].

1. La complexité en mémoire n'étant pas un facteur limitant dans notre attaque.

5.3.2 Supposer et déterminer

Les attaques sur les chiffrements à flot de type *supposer et déterminer* [2] ont été décrites pour la première fois dans [EJ00, HR00] et sont relativement bien connues en cryptographie symétrique. L'idée de ces attaques est d'émettre des hypothèses sur une partie secrète de l'état d'un système, puis de corroborer cette hypothèse avec l'information connue. En mettant en évidence suffisamment rapidement des contradictions à partir de ladite hypothèse, l'attaquant.e est capable de *déterminer* si son hypothèse est bonne ou pas, pour finalement retrouver une partie de ce qui lui est inconnu.

Pour le cas des chiffrements à flot, lorsque la fonction de rétroaction est suffisamment compliquée, il est a priori difficile de suivre les bits de l'état initial dont les valeurs ont été devinées dans le but de trouver des contradictions. Mais dans notre cas particulier de la famille de chiffrements à flot FLIP, le registre dans lequel la clef est stockée n'est jamais mis à jour, ce qui implique que l'effet d'un bit deviné sur la clef sera le même tout au long du processus de génération de la suite chiffrante. Dit autrement : la *diffusion* est très mauvaise dans FLIP.

De plus, une particularité de la fonction de filtrage utilisée dans FLIP est qu'elle possède très peu de monômes de degré élevé, ce qui est souvent exploitable en cryptanalyse, comme nous l'avons vu dans les attaques algébriques au chapitre 4 et comme nous allons le voir aux chapitre 7 et 8.

5.3.3 Observations sur la fonction de filtrage F

Les fonctions de type L et Q n'apportent, dans la forme algébrique normale de la fonction de filtrage que des monômes de degré 1 ou 2. Ainsi, tous les monômes de degré supérieur ou égal à 3 apparaissent uniquement dans la fonction dite triangulaire, dont nous rappelons que la forme algébrique normale est donnée par

$$\begin{aligned} f_3(x_{n_1+n_2}, \ldots, x_{n_1+n_2+n_3-1}) &= T_k(x_{n_1+n_2}, \ldots, x_{n_1+n_2+n_3-1}) \\ &= \sum_{i=1}^{k} \prod_{j=n_1+n_2}^{n_1+n_2+i-1} x_{j+\sum_{\ell=0}^{i-1}\ell} \end{aligned}$$

où k est le degré algébrique de f_3 qui vérifie la relation $n_3 = \frac{k(k+1)}{2}$.

D'après l'expression de la forme algébrique normale de f_3, on voit qu'il y a exactement $k-2$ monômes de degré supérieur ou égal à 3 dans la forme algébrique normale de la fonction de filtrage F, couvrant exactement $n_3 - 3$ variables qui n'apparaissent dans aucun autre monôme de degré 1 ou 2. Étant données les contraintes imposées par FLIP dans un chiffrement complètement homomorphe hybride, k doit être pris suffisamment petit [3], ce qui constitue une faiblesse que nous exploitons dans notre attaque.

2. Guess and Determine en anglais

3. Pour donner une idée de l'ordre de grandeur de k, les instances concrètes de FLIP considèrent $k = 14$ et $k = 21$ pour des sécurités respectives de 80 et 128 bits.

5.3.4 Idée principale

Notre cryptanalyse a pour objectif l'annulation de tous les monômes de degré plus grand que 3 en supposant certains bits de la clef à 0. En effet, en supposant suffisament de bits de clefs nuls, il apparaît que la probabilité d'annuler tous les monômes de haut degré devient suffisamment non-négligeable pour être exploitée par l'attaquant.e. Ainsi, il nous est possible de trouver des équations de degré 2 qui lient les bits de clef restant (ceux qui n'ont pas été supposés nuls) et les bits de la suite chiffrante. Les hypothèses faites nous permettent donc de diminuer drastiquement le degré du système à résoudre, et par conséquent la complexité de l'attaque.

Dans les attaques de type *supposer et déterminer* classiques, c'est un certain sous-ensemble de bits qui est choisi, puis l'ensemble des valeurs possibles pour ces bits spécifiques est testé. Ici, nous réalisons une variante, puisque nous ne testons pas la valeur des bits, mais la position des bits nuls [4]. En supposant que certains bits sont à valeur nulle, nous récupérons alors des équations de degré 2. En utilisant suffisamment de bits connus de la suite chiffrante pour inverser le système correspondant, nous pouvons décider si l'hypothèse initiale est correcte ou pas suivant que le système linéarisé correspondant admet ou non des contradictions. Étant donné que le système obtenu est de degré 2, une technique de linéarisation classique est suffisante pour résoudre le système non-linéaire en prenant soin de collecter suffisamment d'équations.

5.3.5 Un exemple sur une version réduite

Afin de comprendre en détail le principe de l'attaque, nous la décrivons sur un exemple extrêmement réduit, où une clef de taille 22 bits est utilisée comme décrit à la figure 5.3, avec une fonction de filtrage de degré 5 qui suit la construction de FLIP, *i.e.*

$$\begin{aligned} F(x_0, ..., x_{21}) =& x_0 + x_1 + x_2 + x_3x_4 + x_5x_6 + x_7 + x_8x_9 \\ &+ x_{10}x_{11}x_{12} + x_{13}x_{14}x_{15}x_{16} + x_{17}x_{18}x_{19}x_{20}x_{21} \ . \end{aligned}$$

Ensuite, nous prenons 3 positions au hasard, par exemple les positions 9, 13 et 20.

Nous faisons alors l'hypothèse que ces bits sont nuls comme décrit à la figure 5.4 pour pouvoir annuler les 3 monômes de degré 3, 4 et 5.

Finalement, pour chaque permutation engendrée, on vérifie si l'équation est de degré 2. Par exemple, l'équation correspondante à la permutation décrite à la figure 5.5 est

$$z_1 = k_7 + k_2 + k_3k_1 + k_{11}k_{17} + k_8k_6k_{18} + k_{21}k_{15} + k_{21}k_{15}k_4k_{16} + k_{12}k_{19}k_0k_{14}k_{10}$$

où z_1 est le premier bit de suite chiffrante connu. Cette équation est de degré 5, donc nous n'en tenons pas compte.

4. La même variante s'applique au cas de la construction du générateur pseudo-aléatoire de Goldreich, instancié avec certains types de prédicats, comme nous l'expliquerons en détail au chapitre 7.

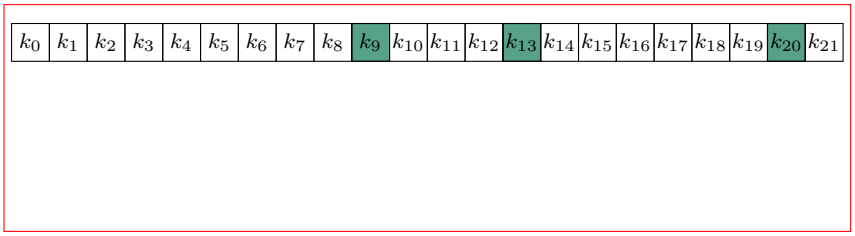

Figure 5.3 – Exemple sur une version réduite.

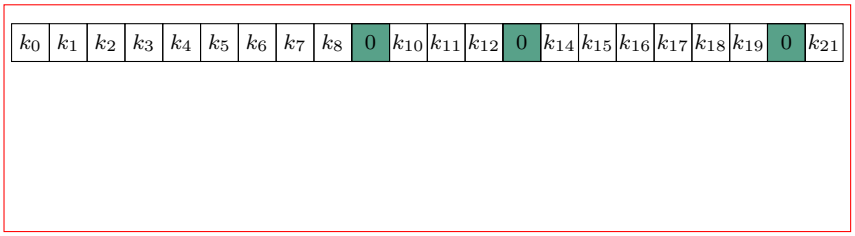

Figure 5.4 – Exemple sur une version réduite.

En revanche, l'équation correspondante à la permutation décrite à la figure 5.6 est

$$z_2 = k_{21} + k_4 + k_0 + k_{12}k_{17} + k_8k_6 + k_7 + k_{16}k_{10}$$

où z_2 est le deuxième bit de suite chiffrante connu. Cette équation est de degré 2, donc nous la conservons en mémoire.

5.3.6 La génération de permutations

Comme expliqué précédemment, à chaque instant, une permutation pseudo-aléatoire est engendrée, et appliquée aux bits de la clef. Dans tout ce qui suit, nous n'utilisons pas les propriétés de ce générateur, et nous supposons que les permutations engendrées se comportent comme des permutations aléatoires. En revanche, certaines permutations peuvent envoyer les bits de clef que nous supposons nuls vers la partie linéaire, laissant des monômes de haut degré dans l'expression de la forme algébrique normale de nos équations. L'idée de l'attaque est donc de considérer uniquement les bits de suite chiffrante pour lesquels la permutation envoie les bits supposés nuls dans les monômes de haut degré, ce qui les annule.

5.3.7 Probabilité d'annuler tous les monômes de haut degré

Alors que la complexité en temps (C_T) de notre attaque dépend principalement du nombre moyen d'hypothèses dont nous aurons besoin de faire avant d'examiner

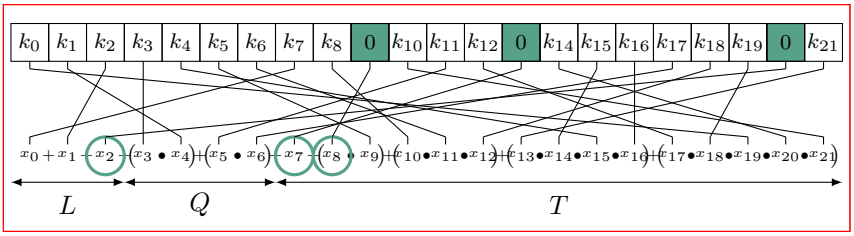

Figure 5.5 – Exemple sur une version réduite.

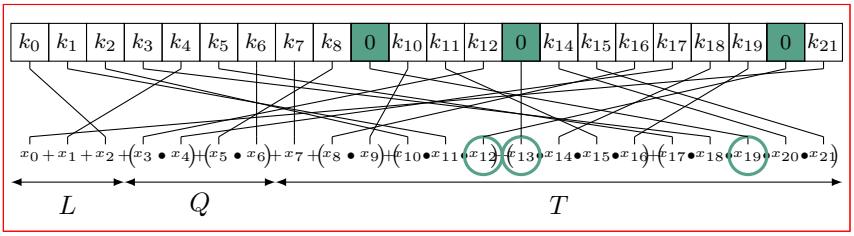

Figure 5.6 – Exemple sur une version réduite.

une hypothèse correcte, la complexité en données (C_D) dépend elle directement de la probabilité d'annuler tous les monômes de degré élevé de la fonction de filtrage. Ainsi, on obtiendra un compromis possible entre la complexité en données et la complexité en temps, puisque C_D diminue avec le nombre de bits supposés à 0 [5], alors que nécessairement C_T augmente avec le nombre d'hypothèses effectuées [6]. Dans ce qui suit, nous déterminons exactement les compromis possibles et nous montrons en quoi notre attaque est réalisable en pratique.

Soit ℓ le nombre de bits supposés nuls. Nous cherchons à calculer la probabilité qu'une permutation aléatoire à un instant t, P_t soit telle que l'expression du bit de la suite chiffrante z_t soit quadratique en les bits de la clef. On note $\Pr_\ell$ cette probabilité. En d'autres termes, chaque monôme de haut degré doit contenir au moins un bit supposé nul.

Tout d'abord, le nombre de monômes de degré plus grand que 3 est exactement $k-2$. Ainsi, il est nécessaire de supposer au moins $k-2$ bits à 0, *i.e.*,

$$\Pr_{\ell<k-2} = 0 \ .$$

Cas particulier : $\ell = k-2$. Afin de comprendre plus en détail le cas général, il est plus simple d'analyser ce qu'il se passe dans le cas particulier où le nombre de

5. La probabilité de tout annuler est d'autant plus élevée que le nombre de bits supposés à 0 est grand.

6. Plus le nombre de positions sur lesquelles on fait une hypothèse est grand, moins on a de chances que cette hypothèse soit vraie

bits supposés nuls est égal au plus petit nombre possible, c'est-à-dire au nombre de monômes de degré supérieur ou égal à 3. Dénombrons donc le nombre de cas favorables, c'est-à-dire les différentes façons possibles de placer 1 bit supposé nul dans chacun de ces monômes. Par construction de la fonction booléenne, nous avons exactement un monôme de degré 3, un monôme de degré 4,..., un monôme de degré k, et les variables impliquées dans chacun de ces monômes sont indépendantes. Dans ces conditions, pour chacun de ces monômes de degré d, pour d compris entre 3 et k, nous avons d choix possibles pour choisir une variable à annuler dans ce monôme. Ainsi, nous avons

$$3 \times 4 \times 5 \times \cdots \times k = \frac{k!}{2}$$

façons différentes d'annuler l'ensemble des monômes. Afin de calculer $\Pr_\ell$, nous divisons par le nombre de cas total, qui correspond au nombre de façons de choisir ℓ bits dans l'ensemble des positions possibles, c'est-à-dire $\binom{N}{\ell}$. On obtient donc

$$\Pr_{\ell=k-2} = \frac{k!/2}{\binom{N}{\ell}} \, .$$

Cas général : $\ell > k-2$. Même si nous verrons que le cas précédent est suffisant pour monter une attaque sur les instances concrètes de FLIP, il est indispensable de considérer le cas général où l'on s'autorise un ℓ plus grand, afin d'obtenir un compromis entre la complexité en temps et la complexité en données [7].

Dénombrons les cas favorables. On compte, pour chaque monôme de degré d, pour d compris entre 3 et k, le nombre de façons de choisir i_d bits intervenant dans ce monôme. Comme ℓ correspond au nombre de bits supposés nuls, et que i_d correspond aux nombres de bits supposés nuls, pris dans le monôme de degré d, nous avons nécessairement la condition suivante :

$$\sum_{d=3}^{k} i_d \leq \ell \, .$$

Finalement, il reste à placer les $\ell - \sum_{d=3}^{k} i_d$ dans les autres variables, c'est-à-dire celles qui interviennent dans des monômes de degré un ou deux (il y en a $N - n_3 + 3$). Ainsi, la probabilité recherchée est égale à

$$\Pr_{\ell \geq k-2} = \frac{1}{\binom{N}{\ell}} \times \left(\sum_{i_3+i_4+\cdots+i_k \leq \ell} \binom{3}{i_3} \binom{4}{i_4} \cdots \binom{k}{i_k} \binom{N-n_3+3}{\ell - \sum_{j=3}^{k} i_j} \right)$$

où $i_d \geq 1$ pour d compris entre 3 et k. Naturellement, nous pouvons remarquer que, lorsque $\ell = k-2$, alors nous avons nécessairement $i_d = 1$ pour tout d

7. Une telle analyse est toujours satisfaisante, puisqu'elle permet de montrer que la limitation des données n'est pas nécessairement un frein à l'attaque.

compris entre 3 et k, ce qui donne

$$\Pr_{\ell=k-2} = \frac{\binom{3}{1}\binom{4}{1}\cdots\binom{k}{1}\binom{N-n_3+3}{\ell-\ell}}{\binom{N}{\ell}} = \frac{3\times 4\times 5\times\cdots\times k}{\binom{N}{\ell}} .$$

Une autre manière de calculer cette probabilité est de s'intéresser à l'événement complémentaire, qui correspond à tous les cas où au moins un monôme de degré supérieur ou égal à 3 n'est pas annulé.

Soit D un ensemble d'entiers représentant les degrés des monômes, et $\mathcal{A}_D$ l'événement *La permutation n'envoie aucun des bits supposés nuls dans les monômes de degré inclus dans* D, c'est-à-dire

$$\mathcal{A}_D : \{\forall d \in D, m_d \neq 0\}$$

où m_d est le monôme de degré d (induit par la fonction triangulaire). La probabilité de l'événement $\mathcal{A}_D$ est facilement calculable en fonction de l'ensemble D. En effet, on a

$$\Pr\left[\mathcal{A}_D\right] = \frac{\binom{N-\sum_{i\notin D} i}{\ell}}{\binom{N}{\ell}} .$$

Finalement, en utilisant le principe d'*inclusion-exclusion*, on obtient

$$\Pr\left[\bigcup_{d\in\{3,\cdots,k\}} \mathcal{A}_{\{d\}}\right] = \sum_{s=1}^{k-2}\left((-1)^s \sum_{\substack{D\subseteq\{3,\cdots,k\}\\ |D|=s}} \Pr(\mathcal{A}_D)\right) ,$$

ce qui nous donne

$$\Pr\left[\bigcup_{d\in\{3,\cdots,k\}} \mathcal{A}_{\{d\}}\right] = \frac{\sum_{s=1}^{k-2}\left((-1)^s \sum_{\substack{D\subseteq\{3,\cdots,k\}\\ |D|=s}} \binom{N-\sum_{j\notin D} j}{\ell}\right)}{\binom{N}{\ell}} .$$

Dans ces conditions, on obtient la probabilité recherchée initialement :

$$\Pr_\ell = \Pr\left[\bigcap_{d\in\{3,\cdots,k\}} \overline{\mathcal{A}_{\{d\}}}\right] = 1 - \Pr\left[\bigcup_{d\in\{3,\cdots,k\}} \mathcal{A}_{\{d\}}\right] .$$

L'évaluation des cette formule appliquée aux instances concrètes de FLIP pour des valeurs de ℓ raisonnables est donnée par les tables 5.2 et 5.3 afin de dégager un compromis possible. Cependant, il est important de noter que, même pour le plus petit choix de ℓ possible, les probabilités correspondantes sont suffisamment élevées pour mener l'attaque avec succès.

Pour la version de FLIP à 80 bits de sécurité, avec $\ell = 12 = k-2$, la probabilité qu'un bit de suite chiffrante amène une équation de degré 2 vaut $\Pr_{\ell=12} = 2^{-26.335}$. Pour la version à 128 bits de sécurité, il est nécessaire d'avoir $\ell \geq 19$, qui nous donne pour la valeur extrême $\Pr_{\ell=19} = 2^{-42.382}$.

5.4 Notre attaque

5.4.1 Description

Données initiales. Étant dans le modèle à clair connu, on suppose que l'on a accès à exactement C_D valeurs consécutives de la suite chiffrante (la complexité en données), notées z_t, pour t allant de 0 à $C_D - 1$. Les permutations sont engendrées par le vecteur d'initialisation (public), et elles sont connues, et on suppose ledit générateur pseudo-aléatoire parfait, impliquant que les permutations correspondant à chaque bit de suite chiffrante P_t, pour t allant de 0 à $C_D - 1$ sont choisies aléatoirement et de manière uniformément aléatoires. Ainsi, nous nous intéressons à résoudre un système d'équations non-linéaires, de la forme

$$z_t = F \circ P_t(K)$$

pour t allant de 0 à $C_D - 1$, où K est la seule inconnue, de poids de Hamming $\frac{N}{2}$, et de taille N bits.

Première étape : hypothèse sur les bits de clef. La première étape de notre attaque consiste à supposer que ℓ bits pris aléatoirement parmi les N bits de clef sont tous nuls, avec $\ell \geq k - 2$. Ayant fait une telle hypothèse, les bits de la suite chiffrante s'expriment alors en fonction de seulement $N - \ell$ variables. De plus, la probabilité qu'une telle hypothèse soit correcte, c'est-à-dire que ces bits-là soient effectivement nuls, notée $\Pr_{rg}$, vaut

$$\Pr_{rg} = \frac{\binom{\frac{N}{2}}{\ell}}{\binom{N}{\ell}} .$$

Si la clef n'était pas équilibrée, cette probabilité serait égale à $2^{-\ell}$: une hypothèse sur 1 bits serait vraie avec probabilité $\frac{1}{2}$. Or, la condition d'équilibre sur la clef change cette probabilité quand on considère plusieurs bits : il y a $\frac{N}{2}$ bits nuls dans la clef, et les cas favorables sont ceux où les ℓ bits supposés nuls sont choisis dans les $\frac{N}{2}$ positions, ce qui fait

$$\binom{\frac{N}{2}}{\ell}$$

possibilités. En revanche, comme $\ell \ll N$, la probabilité $\Pr_{rg}$ est assez proche de $2^{-\ell}$ tant que ℓ est petit.

Ainsi, cela signifie que nous allons devoir réitérer nos hypothèses en moyenne $\Pr_{rg}^{-1}$ fois, avant de formuler une hypothèse correcte.

Deuxième étape : collecter les équations de degré 2. Une fois qu'une hypothèse a été émise, nous nous attelons à collecter suffisamment d'équations de

degré deux. Le nombre d'équations minimum à collecter, pour pouvoir résoudre le système correspondant par linéarisation vaut

$$v_\ell = N - \ell + \binom{N-\ell}{2},$$

ce qui correspond au nombre de monômes de degré un ou deux sur $N-\ell$ variables (les ℓ autres étant supposées nulles). Comme nous l'avons expliqué à la section précédente, cette partie nous coûte en données, puisqu'il faudra attendre en moyenne Pr_ℓ^{-1} bits de suite chiffrante avant de pouvoir collecter une équation de degré 2.

Troisième étape : résoudre le système correspondant. Une fois que les équations de degré 2 correspondant à l'hypothèse ont été déterminées, il ne reste plus qu'à résoudre le système obtenu. Pour cela, nous utilisons une simple technique de linéarisation, c'est-à-dire que nous considérons chaque nouveau monôme de degré deux comme une nouvelle variable, puis nous inversons le système linéaire induit par une élimination de Gauss, dont la complexité est de l'ordre de $\mathcal{O}(v_\ell^3)$ opérations [8].

5.4.2 Complexité et compromis

5.4.2.1 Complexité

En supposant que les équations obtenues se comportent comme des équations aléatoires, le nombre d'équations nécessaires pour obtenir un système inversible est du même ordre de grandeur que le nombre d'inconnues. En effet, pour des équations linéaires aléatoires, la probabilité d'avoir des équations linéairement dépendantes est très faible [Lan93]. Nous verrons avec la vérification expérimentale que cette hypothèse est cohérente. Par définition de Pr_ℓ, nous pouvons donc assurer que le nombre de bits de suite chiffrante nécessaires (C_D) est bien de l'ordre de

$$C_D = v_\ell \times \frac{1}{\mathrm{Pr}_\ell}.$$

Finalement, la complexité en temps de notre attaque correspond au coût de résolution du système linéaire v_ℓ^3, multiplié par le nombre moyen d'hypothèses à faire avant de formuler une hypothèse correcte, qui est déterminé par Pr_{rg}, ce qui donne une complexité en temps qui est de l'ordre de

$$C_T = v_\ell^3 \times \frac{1}{\mathrm{Pr}_{rg}}.$$

Pour ce qui est de la complexité en mémoire de l'attaque, elle ne provient que du stockage du système linéaire à résoudre et est donc environ de l'ordre de

$$C_M = v_\ell^2.$$

8. Cette technique de linéarisation est classique en cryptographie pour le cas des attaques algébriques sur les chiffrements à flot comme déjà expliqué à la section 4.3 page 70.

Ici, nous voyons bien en quoi il y a un compromis possible entre la complexité en temps et la complexité en données. En effet, $\Pr_\ell$ est croissant avec ℓ, donc plus ℓ est grand, plus la complexité en données est faible, mais faire croître ℓ augmente dans le même temps la complexité en temps. Ceci est assez intéressant pour l'attaquant.e, puisque cela permet de réaliser notre attaque, même si la quantité de données est limitée.

5.4.2.2 Compromis

En appliquant notre attaque sur les deux versions de FLIP initialement proposées dans [MJSC16] avec une taille de clef de 192 bits (respectivement 400 bits) pour une sécurité revendiquée par les auteurs de 80 bits (respectivement 128 bits), nous obtenons les résultats décrits dans les tables 5.2 et 5.3, où sont données les différentes valeurs de $\Pr_\ell$, $\Pr_{rg}$, v_ℓ ainsi que les complexités en temps et en données correspondantes. Nous constatons que le coût de notre attaque est largement plus faible que la sécurité revendiquée par les auteurs de FLIP.

ℓ	$\Pr_\ell$	v_ℓ	$\Pr_{rg}$	C_D	C_T	C_M
12	-26.335	13.992	-12.528	40.326	54.503	27.983
13	-23.049	13.976	-13.627	37.025	55.554	27.951
14	-20.653	13.960	-14.736	34.613	56.615	27.919
15	-18.738	13.943	-15.854	32.682	57.684	27.887
16	-17.141	13.927	-16.982	31.069	58.763	27.854
17	-15.775	13.911	-18.120	29.686	59.852	27.821
18	-14.585	13.894	-19.267	28.480	60.950	27.788
19	-13.536	13.878	-20.425	27.414	62.057	27.755
20	-12.601	13.861	-21.592	26.462	63.175	27.722
21	-11.762	13.844	-22.771	25.606	64.303	27.688
22	-11.004	13.827	-23.960	24.831	65.442	27.654
23	-10.315	13.810	-25.160	24.125	66.591	27.621
24	-9.686	13.793	-26.371	23.479	67.750	27.586
25	-9.110	13.776	-27.593	22.886	68.921	27.552
26	-8.580	13.759	-28.827	22.339	70.103	27.517
27	-8.092	13.741	-30.073	21.833	71.297	27.483
28	-7.640	13.724	-31.331	21.364	72.502	27.448
29	-7.221	13.706	-32.601	20.927	73.720	27.413
30	-6.832	13.689	-33.883	20.520	74.949	27.377
31	-6.469	13.671	-35.179	20.140	76.191	27.342
32	-6.131	13.653	-36.487	19.784	77.446	27.306
33	-5.816	13.635	-37.809	19.450	78.714	27.270
34	**-5.520**	**13.617**	**-39.145**	**19.137**	**79.995**	**27.233**
> 34	...	...	...	...	> 80	...

Table 5.2 – $\log_2$ des complexités en temps, en données et en mémoire en fonction du nombre de bits devinés, ℓ, pour l'instance FLIP(47,40,105).

ℓ	$\Pr_\ell$	v_ℓ	$\Pr_{rg}$	C_D	C_T	C_M
19	-42.382	16.151	-19.647	58.533	68.100	32.302
20	-38.522	16.144	-20.721	54.666	69.151	32.287
21	-35.589	16.136	-21.799	51.725	70.206	32.272
22	-33.169	16.128	-22.881	49.298	71.266	32.257
23	-31.097	16.121	-23.967	47.218	72.329	32.241
24	-29.282	16.113	-25.058	45.395	73.397	32.226
25	-27.667	16.105	-26.153	43.772	74.469	32.211
26	-26.214	16.098	-27.253	42.311	75.546	32.195
27	-24.895	16.090	-28.357	40.985	76.627	32.180
28	-23.691	16.082	-29.465	39.773	77.712	32.164
29	-22.584	16.074	-30.578	38.658	78.802	32.149
30	-21.562	16.067	-31.696	37.629	79.896	32.133
31	-20.615	16.059	-32.818	36.674	80.994	32.118
32	-19.734	16.051	-33.944	35.785	82.097	32.102
33	-18.912	16.043	-35.075	34.955	83.205	32.086
34	-18.142	16.035	-36.211	34.178	84.317	32.071
35	-17.421	16.027	-37.352	33.448	85.434	32.055
36	-16.743	16.020	-38.497	32.762	86.556	32.039
37	-16.104	16.012	-39.648	32.116	87.683	32.023
38	-15.502	16.004	-40.803	31.505	88.814	32.007
39	-14.932	15.996	-41.963	30.928	89.950	31.991
40	-14.393	15.988	-43.128	30.381	91.091	31.975
41	-13.883	15.980	-44.298	29.862	92.237	31.959
42	-13.398	15.972	-45.473	29.370	93.388	31.943
43	-12.937	15.964	-46.653	28.901	94.543	31.927
44	-12.499	15.956	-47.838	28.455	95.704	31.911
45	-12.082	15.947	-49.028	28.029	96.870	31.895
46	-11.684	15.939	-50.224	27.624	98.042	31.879
47	-11.305	15.931	-51.425	27.236	99.218	31.862
48	-10.942	15.923	-52.631	26.865	100.400	31.846
49	-10.596	15.915	-53.842	26.511	101.586	31.830
50	-10.265	15.907	-55.059	26.171	102.779	31.813
51	-9.948	15.898	-56.282	25.846	103.976	31.797
52	-9.644	15.890	-57.509	25.534	105.180	31.780
53	-9.353	15.882	-58.743	25.235	106.388	31.763
54	-9.074	15.873	-59.982	24.947	107.602	31.747
55	-8.806	15.865	-61.227	24.671	108.822	31.730
56	-8.548	15.857	-62.477	24.405	110.048	31.713
57	-8.301	15.848	-63.734	24.149	111.279	31.697
58	-8.063	15.840	-64.996	23.903	112.516	31.680
59	-7.835	15.831	-66.264	23.666	113.758	31.663
60	-7.614	15.823	-67.538	23.437	115.007	31.646

61	-7.402	15.815	-68.818	23.217	116.262	31.629
62	-7.198	15.806	-70.104	23.004	117.522	31.612
63	-7.001	15.797	-71.397	22.799	118.789	31.595
64	-6.812	15.789	-72.695	22.601	120.062	31.578
65	-6.629	15.780	-74.000	22.409	121.341	31.561
66	-6.452	15.772	-75.311	22.224	122.627	31.543
67	-6.281	15.763	-76.629	22.044	123.918	31.526
68	-6.116	15.754	-77.953	21.871	125.216	31.509
69	-5.957	15.746	-79.284	21.703	126.521	31.491
70	**-5.803**	**15.737**	**-80.621**	**21.540**	**127.832**	**31.474**
> 70	...	...	...	...	> 128	...

Table 5.3 – $\log_2$ des complexités en temps, en données et en mémoire en fonction du nombre de bits devinés, ℓ, pour l'instance FLIP(87,82,231).

5.4.3 Vérification expérimentale

En cryptographie, les instances proposées ont souvent une taille suffisante pour que beaucoup de cryptanalyses soient impossibles à vérifier en pratique car elles nécessitent un trop grand nombre d'opérations. Ici, sur la version de FLIP à 80 bits de sécurité, notre attaque a une complexité de l'ordre de $2^{54.5}$ opérations, ce qui est encore assez élevé. Or, nous avons mené une analyse en faisant plusieurs hypothèses : par exemple que les équations que l'on récupère se comportent comme des équations aléatoires. Or, c'est loin d'être le cas puisqu'elles ont toutes le même nombre de termes. Il est donc indispensable de vérifier que la cryptanalyse est valide en appliquant expérimentalement l'algorithme sur une version réduite (lorsque une telle version a un sens). Dans le cas de FLIP, il est assez facile de considérer une version réduite qui garde les mêmes proportions que les versions initiales.

La version réduite que nous considérons a une taille de clef de 64 bits, et nous choisissons les paramètres n_1, n_2 et n_3 qui définissent la fonction de filtrage de manière à conserver les proportions : $n_1 = 14$, $n_2 = 14$ et $n_3 = 36$. Ainsi, nous travaillons sur FLIP$(14, 14, 36)$.

La fonction de filtrage ainsi obtenue est donc définie par

$$f(x_0, \ldots, x_{63}) = f_1(x_0, \ldots, x_{13}) + f_2(x_{14}, \ldots, x_{27}) + f_3(x_{28}, \ldots, x_{63})$$

où l'on définit f_1, f_2 et f_3 de la même manière que les auteurs de FLIP :

$$\begin{aligned} f_1(x_0, \ldots, x_{13}) = L_{14}(x_0, \ldots, x_{13}) &= x_0 + x_1 + \cdots + x_{13} \\ f_2(x_{14}, \ldots, x_{27}) = Q_7(x_{14}, \ldots, x_{27}) &= x_{14}x_{15} + x_{16}x_{17} + \cdots + x_{26}x_{27} \\ f_3(x28, \ldots, x_{63}) = T_8(x_{28}, \ldots, x_{63}) &= x_{28} + x_{29}x_{30} + \cdots + x_{56}x_{57}\cdots x_{63} \ . \end{aligned}$$

Le degré algébrique de f est donc égal à 8, et elle est aussi d'immunité algébrique 8.

Nous avons implémenté notre attaque sur cette version réduite de FLIP, en choisissant ℓ égal à 8. D'après nos résultats théoriques, la complexité en temps pour ces paramètres doit être d'environ $C_T = 2^{40.638}$ opérations élémentaires. De plus, on a pour $\Pr_{rg}$ et $\Pr_\ell$ les valeurs suivantes :

$$\Pr_{rg} = 2^{-8.717}$$
$$\Pr_\ell = 2^{-7.814} .$$

Le nombre v_ℓ d'inconnues dans le système à résoudre vaut, pour $\ell = 8$, 1596. Ainsi, nous pouvons prévoir une complexité en données de l'ordre de $2^{18.454}$ bits de suite chiffrante pour mener à bien notre attaque sur cette version réduite.

Dans la table 5.4, nous résumons les résultats théoriques et pratiques pour cette version réduite. Nous remarquons que les résultats pratiques corroborent parfaitement nos résultats obtenus par l'analyse théorique, ce qui confirme la validité des hypothèses que nous avons faites lors de notre analyse.

	$\Pr_{rg}^{-1}$	C_D	$\Pr_\ell$	Nb. op.	Temps (s)
Pratique	437.1	$2^{18.455}$	$2^{-7.813}$	$2^{38.588}$	280.93
Théorie	420.8	$2^{18.454}$	$2^{-7.814}$	$2^{40.638}$	–

Table 5.4 – Comparaison des résultats théoriques et expérimentaux : attaque sur la version réduite FLIP (14,14,36) avec $\ell = 8$ (moyenne réalisée sur 1000 tests, utilisant Intel(R) Xeon(R) CPU W3670 à 3.20GHz (12MB cache), avec 8GB de RAM).

5.4.4 Améliorations potentielles

5.4.4.1 Complexité en données

Comme la complexité en données dépend principalement des différentes permutations P_t, il est plus astucieux de formuler les hypothèses en tenant compte desdites permutations. Cela revient à choisir pour les bits supposés à 0 ceux qui apparaissent le plus souvent dans la partie correspondant à la fonction triangulaire. De plus, les permutations engendrées sont pré-calculables, car elles ne dépendent que de l'IV. Dans ces conditions, le nombre de bits de suite chiffrant nécessaires sera plus faible. Cette idée sera développée plus en détail au chapitre 7, où le contexte est différent et où la complexité en données est plus limitante que dans le cas de FLIP.

5.4.4.2 Pré-calculs

Afin de quantifier exactement l'avantage de l'attaquant.e, il est toujours utile de séparer les calculs qui sont indépendants du secret des autres. En revenant au principe de Kerckhoffs, le coût en pré-calcul de l'attaque correspond à l'analyse du système, indépendamment de la clef, ce qui doit être considéré comme négligeable,

puisque le système cryptographique doit être pérenne. Par ailleurs, si le coût en pré-calcul est aberrant [9], on est aussi en droit de remettre en cause la faisabilité de l'attaque, même si celle-ci nécessite une immense part de pré-calcul et un faible coût *on line*. Ici, étant donné que la forme des équations ne dépend que des permutations engendrées et de la fonction de filtrage, l'attaquant.e peut réaliser l'inversion de plusieurs systèmes de degré 2 (correspondant à plusieurs hypothèses), sans avoir accès aux bits de la suite chiffrante qui dépendent des données secrètes. Cependant, cela nécessite de stocker en mémoire chacune des matrices correspondant à ces inversions, impliquant une complexité en mémoire de l'ordre de

$$C_M = v_\ell^2 \times \mathrm{Pr}_{rg}^{-1} \ .$$

5.4.4.3 Modèle à IV choisi

Un autre modèle d'attaque, plus fort que celui décrit précédemment, est de considérer que l'attaquant.e a la possibilité de choisir le vecteur d'initialisation. Dans ces conditions, l'attaquant.e pourrait aussi pré-calculer des suites produites par plusieurs valeurs d'IV, afin de choisir une valeur d'IV particulière pour laquelle les permutations seraient plus favorables, par exemple laisseraient invariante une partie des bits de clef, ce qui diminuerait encore la complexité en données de l'attaque.

5.4.4.4 Changement d'IV

Alors que dans la majorité des chiffrements à flot, une phase d'initialisation "mélange" les bits de clef et les bits de l'IV, ce n'est pas du tout le cas dans FLIP. Cette propriété singulière implique qu'une re-synchronisation fréquente, qui constituerait à changer l'IV très souvent tout en gardant la même clef ne permet pas de se prémunir de notre attaque. En effet, des équations obtenues pour plusieurs IVs et la même clef auront toutes les mêmes caractéristiques, puisque le contenu du registre dans lequel la clef est stockée est indépendant des bits de l'IV. Ainsi, les équations obtenues avec des IVs différentes sont toujours exprimées en fonction des mêmes variables, les bits de la clef.

5.4.4.5 Sécurité de la construction générique

La famille de chiffrement FLIP passe facilement à l'échelle : on peut facilement augmenter ou diminuer la taille du registre en respectant les proportions initiales. Au vu de notre analyse, nous pouvons assurer que pour ces proportions le niveau de sécurité de FLIP est au plus de l'ordre de $\sqrt{N}$, où N est la taille de la clef.

Démonstration. La complexité en temps de notre attaque vaut

$$C_T = v_\ell^3 \times \frac{1}{\mathrm{Pr}_{rg}} \ .$$

9. Comme pour l'attaque présentée au chapitre 4.

Comme nous l'avons dit précédemment, le nombre d'hypothèses est très petit devant N, ce qui nous permet d'approximer $\Pr_{rg} \simeq 2^{-\ell}$. Comme $v_\ell = N - \ell + \binom{N-\ell}{2}$, on peut approximer la complexité en temps de notre attaque par

$$C_T = N^6 \times 2^\ell \ .$$

De plus, nous savons que le nombre d'hypothèses ℓ à réaliser est supérieur ou égal au nombre de monômes de degré supérieur ou égal à 3, ce qui implique que $n_3 = (\ell+2)(\ell+3)/2$. Donc, $\ell \simeq \sqrt{n_3}$. Finalement, on obtient

$$\log C_T \simeq \alpha\sqrt{N} \ .$$

□

5.4.4.6 Faire moins d'hypothèses

Nous voyons que le principal coût de notre attaque provient du nombre d'hypothèses réalisées. Nous pouvons alors penser à une variante de l'attaque où l'on choisit $\ell < k - 2$, mais où l'on s'autorise à résoudre un système dont les équations sont de degré plus grand que 2. Il s'avère que ces variantes sont moins performantes que notre attaque d'origine, pour les instances de FLIP et les paramètres proposés.

5.5 Le nouveau FLIP

Notre cryptanalyse de FLIP a été réalisée sur les instances présentées dans une version soumise à EUROCRYPT 2016, et dont nous avons eu connaissance lors des journées *Codage et Cryptographie* de 2015. Nous avons alors averti les auteurs de FLIP des vulnérabilités de leur système de chiffrement. Ils ont donc modifié les paramètres de FLIP en prenant en compte notre cryptanalyse, assurant ainsi une plus grande sécurité.

Principalement, les auteurs ont augmenté la taille des clefs d'un facteur 3 pour les mêmes objectifs de sécurité, et ont rajouté plusieurs fonctions de type triangulaire, afin de garder la même profondeur multiplicative, pour rester performant vis-à-vis du chiffrement complètement homomorphe, tout en rajoutant des monômes dans la forme algébrique normale. La forme des équations reste très particulière, car l'ANF de la fonction de filtrage reste très creuse par rapport au nombre de variables pris en entrée.

Plus précisément, la construction générique reste la même : la fonction de filtrage est toujours une fonction booléenne ayant autant de variables qu'il y a de bits de clef. F est définie de la manière suivante : c'est toujours la *somme directe* de trois fonctions booléennes f_1, f_2 et f_3 à n_1 (respectivement n_2 et n_3) variables, où f_1 et f_2 sont toujours de type L et Q, mais f_3 est définie comme la somme de plusieurs fonctions de type T. La notation utilisée par les auteurs de FLIP est Δ^k_{nb} qui définit une fonction booléenne comme la somme directe de nb fonctions triangulaires de degré k.

Par exemple, on a

$$\Delta_2^3 = x_0 + x_1x_2 + x_3x_4x_5 + x_6 + x_7x_8 + x_9x_{10}x_{11} \ .$$

La fonction booléenne Δ_{nb}^k est donc une fonction booléenne à $nb \times \frac{k(k+1)}{2}$ variables. Finalement, dans la version finale de FLIP [MJSC16], les auteurs ont proposé quatre nouvelles instances, notées naturellement de la même manière que précédemment : $\mathsf{FLIP}(n_1, n_2, \Delta_{nb}^k)$, où n_1 correspond au nombre de variables de la partie linéaire de F, n_2 à la partie de type Q et Δ est définie ci-dessus.

Pour une sécurité revendiquée de 80 bits, les auteurs ont alors proposé deux versions concrètes :

— $\mathsf{FLIP}(42, 128, \Delta_8^9)$, où $N = 530$;

— $\mathsf{FLIP}(46, 136, \Delta_4^{15})$, où $N = 662$.

Ces valeurs de 530 et 662 sont à comparer à l'ancienne valeur de 192, pour une même sécurité de 80 bits.

Pour une sécurité revendiquée de 128 bits, les instances sont les suivantes :

— $\mathsf{FLIP}(82, 224, \Delta_8^{16})$, où $N = 1384$;

— $\mathsf{FLIP}(86, 238, \Delta_4^{23})$, où $N = 1704$.

Les tailles de clef considérées sont donc bien plus grandes que dans les versions précédentes. Évidemment, comme ces paramètres ont été choisis pour résister à notre attaque, cette dernière ne s'applique plus, puisque le nombre d'hypothèses à effectuer devient trop important, et la taille du système augmente aussi considérablement.

Les tailles de clef utilisées peuvent paraître aberrantes en comparaison de la majorité des systèmes symétriques, puisque l'on essaye le plus souvent de garantir un niveau de sécurité égal à la taille de la clef (ce qui est loin d'être le cas ici). Cependant, il est à noter que ce système de chiffrement n'est pas fait pour être utilisé classiquement, mais seulement dans le cadre d'un chiffrement hybride, combiné avec un chiffrement complètement homomorphe, contexte dans lequel la transmission de la clef secrète symétrique (que l'on ne fait qu'une seule fois) coûte beaucoup moins cher qu'une profondeur multiplicative importante.

FLIP est donc extrêmement performant en terme de profondeur multiplicative. Cependant, une autre métrique est à prendre en compte quand on combine un chiffrement symétrique avec un chiffrement complètement homomorphe : le nombre de multiplications réalisées pour 1 bit de chiffré, ce qui dans le cas de FLIP est assez important (environ une centaine de portes AND par bit). Le pari des auteur.e.s de Rasta [DEG+18] est donc de proposer un système de chiffrement symétrique, dont la profondeur multiplicative est faible ainsi que le nombre de multiplications par bit de chiffré. Ces travaux reflètent donc une des directions récentes investiguées en cryptographie symétrique : concevoir des chiffrements adaptés à des contraintes extrêmes de performance et à des applications particulières (MPC, FHE).

5.6 Conclusion

Notre travail exploite une structure très particulière des équations décrivant la suite générée par FLIP : la présence d'un très faible nombre de monômes. C'est exactement la même caractéristique que nous utilisons sur le PRG de Goldreich au chapitre 7, et dans une moindre mesure dans notre cryptanalyse sur Ketje au chapitre 8. Une question sous-jacente est la capacité à identifier les systèmes non-linéaires qui sont faciles à résoudre. En effet, d'un point de vue plus général, notre attaque peut être considérée comme une manière non classique de résoudre un certain type de systèmes d'équations non-linéaires, largement plus efficace que la recherche exhaustive ou qu'une simple linéarisation. Ce n'est sûrement pas la méthode la plus efficace, mais notre attaque permet une nouvelle fois de constater qu'une structure suffisamment forte sur les objets que l'on étudie a beaucoup de chances de pouvoir être exploitée d'une manière ou d'une autre dans un sens favorable à l'attaquant.e. Il semble difficile en cryptographie d'avoir à la fois le beurre et l'argent du beurre...

Chapitre 6

Fonctions booléennes à entrées restreintes

6.1 Introduction

6.1.1 Contexte

Suite à la cryptanalyse de FLIP réalisée avec Sébastien Duval et Virginie Lallemand, nous nous sommes rendu.e.s compte d'une particularité propre à cette famille de chiffrements. En effet, comme nous l'avons vu au chapitre précédent, FLIP possède une spécificité inhabituelle qui découle de la non mise à jour du registre : le poids de Hamming de l'entrée de la fonction de filtrage est invariant tout au long du chiffrement. En effet, comme une permutation des bits de la clef est appliquée à chaque cycle d'horloge, cela ne modifie pas le poids de la clef "permutée", rendant constant le poids de Hamming en entrée de la fonction de filtrage tout au long du chiffrement.

Alors que l'utilisation d'un LFSR permet d'assurer diverses propriétés statistiques sur les états internes successifs [1], ce n'est donc plus du tout le cas pour le chiffrement FLIP, où nous devons analyser le chiffrement en prenant en compte cette particularité. En supposant que le générateur pseudo-aléatoire est de bonne qualité, on peut supposer que toutes les permutations ont la même probabilité d'occurrence, impliquant que tout mot de poids de Hamming égal à celui de la clef a la même probabilité de constituer l'entrée de la fonction de filtrage, les autres n'arrivant jamais.

Ce phénomène peut sembler anodin, cependant, il implique que les arguments de sécurité qui reposent sur les bonnes propriétés cryptographiques de la fonction booléenne n'ont plus aucun sens. L'analyse classique de sécurité d'un chiffrement à flot consiste à étudier l'équilibre de la sortie du générateur, afin d'éviter l'existence d'un distingueur très simple, ainsi que la résistance aux attaques algébriques et

1. Voir chapitre 4

aux attaques par corrélation (rapides). L'argument classique consiste à dire que les états internes successifs du registre se comportent de manière aléatoire (raison pour laquelle on emploie des constructions connues telles les LFSR), puis de conclure en utilisant les propriétés de la fonction de filtrage (équilibre, immunité algébrique, résilience et non-linéarité). Cependant, ceci ne fonctionne que si tous les états internes sont équiprobables.

Exemple 7. Examinons un cas complètement dégénéré qui suit la construction de FLIP, avec une fonction de filtrage linéaire à N variables (où N est la taille de clef dans FLIP) définie par $F(x_0, \ldots, x_{N-1}) = x_0 + x_1 + x_2 + \cdots + x_{N-1}$, c'est-à-dire la première fonction symétrique élémentaire. Cette fonction booléenne est équilibrée, et sa valeur est égale à la parité du poids de Hamming de $(x_0, \ldots, x_{N-1}) \in \{0,1\}^N$. Alors appliquer cette fonction de filtrage au contenu d'un LFSR engendrerait une suite équilibrée, l'appliquer dans la construction de FLIP engendre une suite constante puisque le poids de Hamming de son entrée est constant.

Cet exemple, certes complètement dégénéré, montre cependant le fait que la fonction de filtrage soit équilibrée dans FLIP n'assure en aucun cas que la suite chiffrante n'est pas biaisée. Donc, il est absolument nécessaire de trouver les critères pertinents pour assurer une certaine sécurité sur le chiffrement FLIP, c'est-à-dire affiner les critères cryptographiques classiques afin que ceux-ci aient un réel sens dans notre contexte.

C'est ce que nous faisons dans ce chapitre : nous analysons les fonctions booléennes lorsque l'entrée est réduite à un sous-ensemble donné, et plus particulièrement lorsque le poids de Hamming de l'entrée est fixé, de manière à analyser *rigoureusement* la sécurité de FLIP. Nous montrerons à quel point restreindre l'entrée peut dégrader les qualités cryptographiques des fonctions booléennes : l'équilibre, l'immunité algébrique, et la non-linéarité. Les résultats de ce chapitre sont le fruit d'une collaboration avec une partie des auteurs de FLIP : Claude Carlet et Pierrick Méaux et ont été publiés dans un article aux *IACR Transactions on Symmetric Cryptology* en 2017 [CMR17a].

6.1.2 Aperçu des critères

6.1.2.1 La résilience

Dans le cas de FLIP, l'état interne est stocké dans un unique registre. Ainsi, la décomposition de l'état en plusieurs parties ne peut se faire qu'en exploitant les permutations. Un.e attaquant.e qui chercherait à isoler dans une attaque de type *diviser pour mieux régner* une partie des variables de manière à obtenir un système d'équations probabiliste en une partie des bits de clef devrait nécessairement exploiter une régularité des permutations P_t. De manière générale, le nombre de bits pris dans la partie linéaire, ainsi que les calculs que nous ferons sur le biais indiquent qu'une telle action est suffisamment coûteuse pour ne pas nous en préoccuper. Il semble difficile à première vue de généraliser l'attaque de Siegenthaler sur le chiffrement FLIP, même si nous verrons dans le chapitre 7 que la résilience peut jouer un rôle quand même, mais dans des proportions bien

plus faibles qu'attendu. C'est la raison pour laquelle nous ne nous attarderons pas sur le critère de la résilience dans FLIP, car il est peu pertinent.

6.1.2.2 L'équilibre

Comme expliqué au chapitre 2, le premier critère que l'on demande aux fonctions booléennes utilisées en cryptographie est l'équilibre, afin d'éviter l'existence de distingueurs triviaux. Alors qu'il est assez facile de construire des fonctions booléennes équilibrées sur tout l'espace en rajoutant simplement une variable linéaire indépendante des autres, cela ne fonctionne plus lorsque l'entrée est restreinte à un sous-ensemble quelconque. Nous nous intéresserons donc à l'étude de fonctions booléennes dont toutes les restrictions à une famille d'ensemble disjoints sont équilibrées. Pour le cas particulier de la restriction aux sous-ensembles des mots de poids de Hamming fixé, nous donnerons des constructions de fonctions (presque) équilibrées au moyen de leur forme multivariée, *i.e.* de leur forme algébrique normale.

6.1.2.3 La non-linéarité

La non-linéarité est une quantité importante pour quantifier la résistance aux attaques par corrélation. Ainsi, nous étudierons la non-linéarité restreinte à des sous-ensembles de $\mathbb{F}_2^n$, et nous verrons que se restreindre à des sous-ensembles d'entrées peut dégrader fortement la non-linéarité. Nous construirons notamment des fonctions courbes sur tout l'espace qui deviennent de non-linéarité restreinte nulle.

6.1.2.4 Immunité algébrique

L'immunité algébrique est un troisième paramètre qui joue un rôle important pour quantifier la résistance aux attaques de type algébrique puisqu'elle définit le degré minimal du système algébrique à résoudre. Nous dériverons alors des résultats sur l'immunité algébrique restreinte, en nous focalisant sur les sous-ensembles des mots de poids fixé.

6.2 L'équilibre à poids fixé

6.2.1 Dégradation

Comme nous l'avons déjà mis en évidence, la première fonction symétrique élémentaire est équilibrée, mais est constante lorsque le poids de son entrée est fixé. Les fonctions symétriques sont naturellement les "pires" fonctions possibles pour l'équilibre lorsque le poids de Hamming est fixé.

Définition 6.1. *Pour $i \in \{1, \ldots, n\}$, nous notons σ_i la i-ième fonction symétrique élémentaire, c'est-à-dire*

$$\sigma_i(x) = \sum_{1 \leq j_1 < \cdots < j_i \leq n} \prod_{\ell=1}^{i} x_{j_\ell} \ .$$

Les fonctions symétriques élémentaires définissent une base de l'espace vectoriel de toutes les fonctions qui sont constantes lorsque le poids de Hamming de l'entrée est fixé. La fonction majorité appartient à cet espace de fonctions. Il est aussi connu que pour tout $k \in \{1, \ldots, n\}$, $\mathsf{w_H}(x) = k$ si et seulement si $\sigma_i(x) = \binom{k}{i} \mod 2$ pour tout $0 \leq i \leq k$.

Dans ces conditions, on observe bien qu'une fonction équilibrée sur tout l'espace peut, en fixant le poids de l'entrée devenir complètement dégénérée au regard de ce critère. Nous verrons le même type de dégradation possible sur les critères de non-linéarité et d'immunité algébrique.

6.2.2 Définitions

Notation 6.2. *Pour k compris entre 0 et n, nous notons $S_{n,k}$ le sous-ensemble de $\mathbb{F}_2^n$ des éléments de poids de Hamming k :*

$$S_{n,k} = \{x \in \mathbb{F}_2^n, \mathsf{w_H}(x) = k\} \ .$$

Tout d'abord, nous élargissons la notion du poids de Hamming d'une fonction booléenne comme étant le nombre de 1 dans sa table de vérité par la notation suivante.

Notation 6.3. *Soit f une fonction booléenne à n variables. Pour k compris entre 0 et n, nous notons $\mathsf{w_H}(f)_k$ le poids de Hamming du vecteur des valeurs de f restreint à l'ensemble des éléments de $\mathbb{F}_2^n$ de poids de Hamming égal à k :*

$$\mathsf{w_H}(f)_k = |\{x \in \mathbb{F}_2^n, \mathsf{w_H}(x) = k, f(x) = 1\}| \ .$$

De plus, nous notons de la même manière $\overline{\mathsf{w_H}(f)_k}$ le nombre de 0 dans la table de vérité de f restreinte à $S_{n,k}$:

$$\overline{\mathsf{w_H}(f)_k} = |\{x \in \mathbb{F}_2^n, \mathsf{w_H}(x) = k, f(x) = 0\}| = \binom{n}{k} - \mathsf{w_H}(f)_k \ .$$

Définition 6.4 (Fonctions parfaitement équilibrées à poids fixé)**.** *Soit f une fonction booléenne à n variables. Alors f est dite* parfaitement équilibrée à poids fixée (WPB[2]) *si, pour tout $k \in \{1, \ldots, n-1\}$, la restriction de f à $S_{n,k}$ est équilibrée. En d'autres termes, pour tout k compris entre 1 et $n-1$,*

$$\mathsf{w_H}(f)_k = \frac{\binom{n}{k}}{2} \ .$$

2. Weightwise Perfectly Balanced en anglais

Afin de rendre la fonction équilibrée sur $\mathbb{F}_2^n$, nous imposons de plus la condition suivante :

$$f(0,\ldots,0) = 0 \text{ et } f(1,\ldots,1) = 1\ .$$

Cette dernière condition semble restreindre notre étude, mais ce n'est pas le cas, puisqu'il est nécessaire d'avoir $f(0,\ldots,0) \neq f(1,\ldots,1)$, afin que la fonction soit équilibrée sur tout l'espace. En remplaçant f par $1+f$, nous obtenons toujours une fonction booléenne WPB, ce qui couvre l'ensemble des cas. Ainsi, la condition $f(0,\ldots,0) = 0$ et $f(1,\ldots,1) = 1$ ne restreint en aucun cas notre étude mais s'avère très utile pour la construction de telles fonctions, comme nous le verrons plus tard.

Naturellement, $\mathsf{w_H}(f)_k$ est toujours un entier compris entre 0 et $\binom{n}{k}$, ce qui implique que les fonctions parfaitement équilibrées à poids fixé n'existent que si pour tout k compris entre 1 et $n-1$, $\binom{n}{k}$ est pair. Cette propriété étant satisfaite si et seulement si n est une puissance de 2, les fonctions booléenne WPB n'existent donc que s'il existe un entier ℓ tel que $n = 2^\ell$ (implication directe du théorème de Lucas, publié en 1878 dans [Luc78]).

Afin de généraliser notre étude à un nombre arbitraire de variables, nous définissons donc les fonctions *presque* parfaitement équilibrées à poids fixé (WAPB [3]).

Définition 6.5 (Fonctions presque parfaitement équilibrées à poids fixé)**.** *Soit f une fonction booléenne à n variables. Alors f est une fonction presque parfaitement équilibrée à poids fixé (WAPB) si, pour tout $k \in \{1,\ldots,n-1\}$, $\mathsf{w_H}(f)_k = \frac{\binom{n}{k}}{2}$ si $\binom{n}{k}$ est pair et $\mathsf{w_H}(f)_k = \frac{\binom{n}{k}\pm 1}{2}$ si $\binom{n}{k}$ est impair.*

Évidemment, ces définitions sont appliquées aux sous-ensembles des mots de poids fixé, et plus exactement à la partition de $\mathbb{F}_2^n$ en ces sous-ensembles : $\{S_{n,k}, k \in [0,n]\}$. Cependant, il est clair que ces définitions peuvent être adaptées à d'autres partition. Notre travail se concentre principalement sur les sous-ensembles des mots de poids fixé, puisque c'est le critère pertinent à prendre en compte dans le cas de FLIP. Dans d'autres contextes comme nous le verrons à la conclusion avec les fonctions augmentées, les sous-ensembles peuvent être différents des sous-ensembles $S_{n,k}$.

6.2.3 Relation avec la forme algébrique normale

Le coût de stockage d'une fonction à n variables quelconque par sa table de vérité est de 2^n bits. En pratique, les fonctions booléennes sont donc rarement définies de cette manière, et on préfère souvent les représenter par leur forme algébrique normale. Ainsi, nos définitions précédentes sont peu utiles en pratique. Il faut donc étudier le lien de ces définitions avec la manière naturelle de définir les fonctions booléennes : la forme algébrique normale.

3. Weightwise Almost Perfectly Balanced

Afin de bien comprendre ce qui se passe, nous commençons simplement par considérer les mots dont le poids est impair. Pour tout n pair, la fonction booléenne linéaire f à n variables définie par

$$f(x_1, \ldots, x_n) = x_1 + x_2 + \cdots + x_{\frac{n}{2}}$$

est une fonction équilibrée sur l'ensemble des mots de poids de Hamming impair. En effet, soit $x \in \mathbb{F}_2^n$, tel que $\mathsf{w_H}(x)$ est impair. Alors si $\mathsf{w_H}(x_1, \ldots, x_{\frac{n}{2}})$ est impair, auquel cas $f(x_1, \ldots, x_n) = 1$, sinon $f(x_1, \ldots, x_n) = 0$. Les mots tels que $\mathsf{w_H}(x_1, \ldots, x_n) = k$, k impair et $\mathsf{w_H}(x_1, \ldots, x_{\frac{n}{2}})$ est impair sont naturellement de même cardinalité que les mots dont $\mathsf{w_H}(x_1, \ldots, x_n) = k$ et $\mathsf{w_H}(x_1, \ldots, x_{\frac{n}{2}})$ est pair. En effet, comme k est impair, $\mathsf{w_H}(x_{\frac{n}{2}+1}, \ldots, x_n)$ est pair, et le résultat vient en échangeant les $\frac{n}{2}$ premières variables avec les autres.

Finalement, nous pouvons donc assurer que toute fonction booléenne WPB doit nécessairement avoir la moitié des monômes de degré 1 dans sa forme algébrique normale, et plus précisément :

Proposition 6.6. *Soit f une fonction booléenne (presque) parfaitement équilibrée à poids fixé à n variables. Alors la forme algébrique normale de f contient exactement $\lceil n/2 \rfloor$ monômes de degré 1 et au moins $\lfloor n/4 \rfloor$ monômes de degré 2, où $\lceil n/2 \rfloor$ est égal à $n/2$ si n est pair et $(n \pm 1)/2$ si n est impair.*

Démonstration. Tout d'abord, dans le cas particulier où l'on a une fonction linéaire, $\mathsf{w_H}(f)_k$ est égal au nombre d'entrées de poids exactement k pour lesquels un nombre impair de monômes de f ont la valeur 1, c'est-à-dire le nombre de manières de choisir i variables, pour i impair compris entre 1 et k, de telle façon que ces i variables soient dans les variables apparaissant dans l'ANF [4] de la fonction linéaire. Ainsi, pour une fonction linéaire ℓ à n variables faisant intervenir d variables dans son ANF, nous avons

$$\mathsf{w_H}(\ell)_k = \sum_{i \text{ impair}} \binom{d}{i} \binom{n-d}{k-i} .$$

Soit maintenant $f \in \mathcal{B}_n$ que nous décomposons de la manière suivante :

$$f = \ell_f + q_f + f' ,$$

où $\ell_f \in \mathcal{B}_n$ est une fonction linéaire, $q_f \in \mathcal{B}_n$ est une fonction n'ayant que des monômes de degré 2 dans son ANF, et $f' \in \mathcal{B}_n$ n'ayant que des monômes de degré strictement supérieur à 2 dans son ANF. On suppose, sans perdre de généralité, que le coefficient constant de l'ANF est nul.

Pour les entrées de poids 1, chaque monôme de degré supérieur ou égal à 2 est nul. Ainsi, pour tout $x \in \mathbb{F}_2^n$, tel que $\mathsf{w_H}(x) = 1$, nous avons $f(x) = \ell_f(x)$. Donc,

$$\mathsf{w_H}(f)_1 = \mathsf{w_H}(\ell_f)_1 = \binom{|\ell_f|}{1} = |\ell_f| ,$$

4. Algebraic Normal Form en anglais

où $|\ell_f|$ désigne le nombre de monômes dans la forme algébrique normale de ℓ_f. Il est donc nécessaire que $|\ell_f|$ soit égal à $\frac{n}{2}$ si n est pair et $\frac{n\pm1}{2}$ sinon.

Maintenant nous nous intéressons aux entrées de poids 2. Naturellement, nous avons la relation suivante :

$$\mathsf{w_H}(f)_2 = \mathsf{w_H}(\ell_f + q_f)_2 = \mathsf{w_H}(\ell_f)_2 + \mathsf{w_H}(q_f)_2 - 2\mathsf{w_H}(q_f\ell_f)_2 \ .$$

Comme on a toujours $\mathsf{w_H}(q_f\ell_f) \leq \mathsf{w_H}(q_f)$, on peut assurer l'inégalité

$$\mathsf{w_H}(f)_2 - \mathsf{w_H}(\ell_f)_2 \geq \mathsf{w_H}(q_f) - 2\mathsf{w_H}(q_f) = -\mathsf{w_H}(q_f) \ .$$

Comme nous voulons que f soit équilibrée sur les mots de poids 2 et que par définition de f', $\mathsf{w_H}(f')_2 = 0$, on a

$$\mathsf{w_H}(f)_2 = \left\lceil \frac{n(n-1)}{4} \right\rfloor ,$$

et $\mathsf{w_H}(\ell_f) - \mathsf{w_H}(f)_2 \leq \mathsf{w_H}(q_f)$. Si n est pair, alors

$$\begin{aligned}
\mathsf{w_H}(\ell_f)_2 &= |\{X \in \mathbb{F}_2^n, \ell_f(X) = 1, \mathsf{w_H}(X) = 2\}| \\
&= |\{(x,y), x, y \in \mathbb{F}_2^{\frac{n}{2}}, \mathsf{w_H}(x) = 1, \mathsf{w_H}(y) = 1\}| \\
&= \frac{n}{2} \times \frac{n}{2} \ .
\end{aligned}$$

Donc, $\mathsf{w_H}(q_f) \geq \lfloor \frac{n}{4} \rfloor$. Si en revanche n est impair, alors peu importe la valeur de $|\ell_f|$, nous avons par symétrie (en échangeant x et y dans ce qui suit) que

$$\begin{aligned}
\mathsf{w_H}(\ell_f)_2 &= |\{X \in \mathbb{F}_2^n, \ell_f(X) = 1, \mathsf{w_H}(X) = 2\}| \\
&= |\{(x,y), x \in \mathbb{F}_2^{\frac{n+1}{2}}, y \in \mathbb{F}_2^{\frac{n-1}{2}} \mathsf{w_H}(x) = 1, \mathsf{w_H}(y) = 1\}| \\
&= \binom{\frac{n+1}{2}}{1}\binom{\frac{n-1}{2}}{1} = \frac{n^2-1}{4}
\end{aligned}$$

En dissociant les cas où $n \equiv 1 \mod 4$ et $n \equiv 3 \mod 4$, nous obtenons le même résultat que pour le cas pair.

— Si $n \equiv 1 \mod 4$, alors $\binom{n}{2} = \frac{n(n-1)}{2}$ et $n-1 \equiv 0 \mod 4$, donc $\binom{n}{2}$ est pair, impliquant un unique choix pour $\mathsf{w_H}(f)_2 = \frac{n(n-1)}{4}$. Donc $\mathsf{w_H}(\ell_f)_2 - \mathsf{w_H}(f)_2 = \frac{n-1}{4} = \lfloor \frac{n}{4} \rfloor$.

— Si $n \equiv 3 \mod 4$, alors $\binom{n}{2}$ n'est pas pair, mais nous savons que $\mathsf{w_H}(f)_2 = \frac{\binom{n}{2}\pm1}{2}$, donc $\mathsf{w_H}(q_f) \geq \frac{n^2-1}{4} - \frac{\binom{n}{2}+1}{2} = \frac{n-3}{4} = \lfloor \frac{n}{4} \rfloor$.

□

De plus, il est aussi nécessaire d'avoir au moins un monôme de degré $n/2$ dans la forme algébrique normale des fonctions parfaitement équilibrées à poids fixé.

Proposition 6.7. *Soit n une puissance de 2 et $f \in \mathcal{B}_n$ parfaitement équilibrée à poids fixé, alors la forme algébrique normale de f contient au moins un monôme de degré $n/2$.*

La preuve réside principalement dans la parité de $\mathsf{w_H}(f)_k$, dans le cas où f est parfaitement équilibrée (et donc que $n = 2^\ell$), *i.e.* nous regardons la parité de $\frac{1}{2}\binom{n}{k}$. Nous avons donc besoin du lemme suivant.

Lemme 6.8. *Pour tout $n = 2^\ell \geq 4$ et pour tout k strictement supérieur à 1 et strictement inférieur à n, et différent de $2^{\ell-1} = \frac{n}{2}$, nous avons*

$$\binom{n}{k} \equiv 0 \mod 4 ,$$

avec $\frac{1}{2}\binom{2^\ell}{2^{\ell-1}}$ impair.

Démonstration. En regardant en détail l'expression des coefficients binomiaux, on a

$$\begin{aligned} &2^\ell(2^\ell-1)(2^\ell-2)(2^\ell-3)(2^\ell-4)\cdots(2^\ell-k+2)(2^\ell-k+1) \\ &= 2^\ell p_1 2^0 p_2 2^1 p_3 2^0 p_4 2^4 \cdots p_{k-2} 2^{i_{k-2}} p_{k-1} 2^{i_{k-1}} , \end{aligned}$$

ainsi que

$$\begin{aligned} &k(k-1)(k-2)(k-3)(k-4)\cdots(2)(1) \\ &= k p'_{k-1} 2^{i_{k-1}} p'_{k-2} 2^{i_{k-2}} p'_{k-3} 2^{i_{k-3}} \cdots p'_2 2^1 p'_1 2^0 , \end{aligned}$$

avec p_j, p'_j impairs pour j allant de 1 à $k-1$. En réalisant la division du premier produit par le second, ce qui correspond au coefficient binomial $\binom{2^\ell}{k}$, on obtient alors que tous les 2^{i_j} se simplifient deux à deux. Donc, le coefficient binomial $\binom{2^\ell}{k}$ peut s'écrire de la manière suivante :

$$\binom{2^\ell}{k} = p\frac{2^\ell}{k} ,$$

où p est impair, ce qui conclue la preuve du lemme. □

Démonstration. (Proposition 6.7) Dans la suite on note $n = 2^\ell$. Soit m_d un monôme de degré d, on s'intéresse à la parité de $\mathsf{w_H}(m_d)_k$; pour tout k compris entre 1 et $n-1$, et $1 \leq d \leq k$, on a

$$\mathsf{w_H}(m_d)_k = \binom{n-d}{k-d} .$$

En particulier lorsque $k = d$, $\mathsf{w_H}(m_k)_k = 1$. D'après le lemme 6.8, on a $\mathsf{w_H}(f)_k \equiv 0 \mod 2$ pour tout k différent de 0, n ou $\frac{n}{2}$ et $\mathsf{w_H}(f)_{\frac{n}{2}} \equiv 1 \mod 2$ sinon.

Nous montrons maintenant par récurrence sur k que si f est WPB, alors le nombre de monômes de degré d dans son ANF, pour d strictement inférieur à $n/2$ est toujours pair : Soit f une fonction à $n = 2^\ell$ variables parfaitement équilibrée à poids fixé. Sans perdre de généralité, on considère le coefficient constant nul afin de suivre la définition des fonctions booléennes WPB.

Base. Pour $k = 1$, le poids de la fonction est déterminé par les monômes de degré 1, et nous savons qu'il est nécessaire d'avoir exactement $n/2 = 2^{\ell-1}$ monômes de degré 1.

Récurrence. Soit $k \geq 1$. On note $f = f_1 + f_2 + f_3$, avec f_1 contenant les monômes de degré inférieur ou égal à k, f_2 les monômes de degré exactement $k+1$ et f_3 les monômes de degré strictement supérieur à $k+1$. Alors, $\mathsf{w_H}(f)_{k+1} = \mathsf{w_H}(f_1 + f_2)_{k+1} = \mathsf{w_H}(f_1)_{k+1} + \mathsf{w_H}(f_2)_{k+1} - 2\mathsf{w_H}(f_1 f_2)_{k+1}$. En ne considérant maintenant que la parité de $\mathsf{w_H}(f)_{k+1}$, nous pouvons écrire la relation suivante :

$$\mathsf{w_H}(f)_{k+1} = \sum_{d=1}^{k} \sum_{m_d \in \mathsf{ANF}(f_1)} \mathsf{w_H}(m_d)_{k+1} + \sum_{m_{k+1} \in \mathsf{ANF}(f_2)} \mathsf{w_H}(m_{k+1})_{k+1} \mod 2 .$$

Par hypothèse de récurrence, pour tout d compris entre 1 et k, le nombre de monômes de degré d est pair. De plus, $\mathsf{w_H}(m_{k+1})_{k+1} = 1$ impliquant directement que

$$\mathsf{w_H}(f)_{k+1} = \sum_{m_{k+1} \in \mathsf{ANF}(f_2)} 1 \mod 2 .$$

Finalement, tant que $k + 1 < \frac{n}{2}$, on a $\mathsf{w_H}(f)_{k+1} = 0 \mod 2$, et $\mathsf{w_H}(f)_{n/2} = 1 \mod 2$, ce qui termine la preuve. □

D'après ce qui précède, nous avons vu qu'il est nécessaire pour une fonction WPB à 2^ℓ variables d'avoir exactement $2^{\ell-1}$ monômes de degré 1, au moins $2^{\ell-2}$ monômes de degré 2 et au moins 1 monôme de degré $2^{\ell-1}$. On pourrait se demander s'il est nécessaire d'avoir des monômes de degré 2^i pour des valeurs intermédiaires, notamment s'il est nécessaire d'avoir des monômes de degré 4 par exemple, mais il n'en est rien. Par exemple, la fonction booléenne à 16 variables définie par

$$f = \sum_{i=1}^{8} x_i + \sum_{i=1}^{4} x_i x_{i+8} + x_1 x_2 x_5 + x_1 x_4 x_{16}$$

est une fonction booléenne équilibrée sur les mots de poids 1, 2, 3 et 4, et peut donc être complétée par des monômes de degré strictement supérieur à 4 pour obtenir une fonction WPB, ces monômes de plus haut degré pouvant à eux seuls rééquilibrer la fonction sur les mots de poids 5, 6,...

6.2.4 Constructions de fonctions WPB et WAPB

Naturellement, comme nous avons défini un nouveau critère, il est nécessaire d'apporter des constructions génériques de telles fonctions, afin d'aider principalement des concepteur.rice.s de primitives cryptographiques s'ils.elles se placent dans le contexte que nous avons évoqué. Dans cette section, nous décrivons des familles de fonctions WPB et WAPB.

Une manière assez classique de construire des familles de fonctions booléennes est d'utiliser la *somme directe* de deux fonctions, c'est-à-dire de prendre la somme

de deux fonctions dont les variables sont indépendantes entre elles (chapitre 2). Alors que dans le contexte classique, si une seule des deux fonctions est équilibrée, la fonction résultante est aussi équilibrée, cela est complètement différent et plus complexe pour le cas où l'on opère sur des mots de poids fixé.

Nous allons tout d'abord nous focaliser sur les fonctions à 2^ℓ variables définies comme la somme directe de 2 fonctions WPB ou WAPB à $2^{\ell-1}$ variables, et montrer qu'elles ne peuvent jamais être WPB.

Lemme 6.9. *Soit f une fonction booléenne à $n = 2^\ell$ variables, de la forme : $f(x_1, \ldots, x_n) = g_1(x_1, \ldots, x_{\frac{n}{2}}) + g_2(x_{\frac{n}{2}+1}, \ldots, x_n)$ où g_1 et g_2 sont deux fonctions booléennes à $\frac{n}{2}$ variables telles que $g_1(0...0) + g_1(1...1) + g_2(0...0) + g_2(1...1) \equiv 0 \mod 2$. Alors f ne peut pas être parfaitement équilibrée à poids fixé.*

Démonstration. Soit f une fonction booléenne définie par la somme directe de deux fonctions booléennes g_1 et g_2 à $\frac{n}{2}$ variables. Alors nous pouvons lier $\mathsf{w_H}(f)_k$ à $\mathsf{w_H}(g_1)_i$ et $\mathsf{w_H}(g_2)_i$ pour $i \leq k$:

$$\mathsf{w_H}(f)_k = \sum_{i=0}^{k} \mathsf{w_H}(g_1)_i \left(\binom{\frac{n}{2}}{k-i} - \mathsf{w_H}(g_2)_{k-i} \right) + \mathsf{w_H}(g_2)_{k-i} \left(\binom{\frac{n}{2}}{i} - \mathsf{w_H}(g_2)_i \right)$$

Maintenant, on suppose que f est parfaitement équilibrée à poids fixé, et nous savons par le lemme 6.8 que $\mathsf{w_H}(f)_{\frac{n}{2}} = \frac{1}{2}\binom{n}{\frac{n}{2}} \equiv 1 \mod 2$. Alors, nous avons

$$\mathsf{w_H}(f)_{\frac{n}{2}} = \sum_{i=0}^{\frac{n}{2}} \binom{\frac{n}{2}}{\frac{n}{2}-i} \mathsf{w_H}(g_1)_i + \binom{\frac{n}{2}}{i} \mathsf{w_H}(g_2)_{\frac{n}{2}-i} - 2\mathsf{w_H}(g_1)_i \mathsf{w_H}(g_2)_{\frac{n}{2}-i} \, .$$

De plus, nous savons que, comme n est une puissance de 2, alors pour chaque $i \in [1, \frac{n}{2} - 1]$, $\binom{\frac{n}{2}}{i}$ est pair. En passant l'équation précédente modulo 2, on trouve la condition suivante :

$$1 \equiv \mathsf{w_H}(g_1)_0 + \mathsf{w_H}(g_1)_{\frac{n}{2}} + \mathsf{w_H}(g_2)_0 + \mathsf{w_H}(g_2)_{\frac{n}{2}} \mod 2 \, .$$

□

Ce lemme implique donc que l'on ne peut construire de fonction parfaitement équilibrée à poids fixé avec plus de variables en faisant la *somme directe* de deux fonctions parfaitement équilibrées à poids fixé

Corollaire 6.10. *Soit g_1 et g_2 deux fonctions à n variables parfaitement équilibrées à poids fixé à n variables, alors la fonction booléenne définie par la somme directe de ces deux fonctions n'est pas WPB.*

Tout ceci pourrait nous sembler à première vue un peu inutile, cependant cela montre que passer du contexte classique aux restrictions des fonctions booléennes rend l'étude nettement plus compliquée. Alors que construire une fonction équilibrée est très simple, il semble bien plus ardu de définir des fonctions WPB ou WAPB. Pour de telles fonctions, il s'avère qu'une bonne construction

à utiliser est la suivante : si f et g sont deux fonctions booléennes WPB à n variables, alors la fonction définie par

$$h(x,y) = f(x) + \prod_{i=1}^{n} x_i + g(y)$$

est une fonction WPB à $2n$ variables. Cette construction s'inspire de la *somme indirecte* :

$$h(x,y) = f(x) + g(y) + (f(x) + f'(x))(g(y) + g'(y)) \ ,$$

qui permet la construction de fonctions courbes [Car07] dans le contexte classique. De manière générale, nous pouvons construire un grand nombre de fonctions WPB en utilisant cette construction comme le montre le théorème qui suit.

Théorème 6.11. *Soit f, f' et g trois fonctions booléennes à n variables parfaitement équilibrées à poids fixé et g' une fonction booléenne quelconque à n variables. Alors*

$$h(x,y) = f(x) + \prod_{i=1}^{n} x_i + g(y) + (f(x) + f'(x))g'(y)$$

où $x, y \in \mathbb{F}_2^n$, est une fonction booléenne à $2n$ variables parfaitement équilibrée à poids fixé.

Démonstration. Par souci de lisibilité, on note dans cette preuve $\mathbf{0} = (0, \ldots, 0) \in \mathbb{F}_2^n$ et $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_2^n$. Comme f, f' et g sont WPB, on a $f(\mathbf{0}) = f'(\mathbf{0}) = g(\mathbf{0}) = 0$ et $f(\mathbf{1}) = f'(\mathbf{1}) = g(\mathbf{1}) = 1$. Soit k le poids de Hamming de $(x,y) \in \mathbb{F}_2^{2n}$, compris entre 0 et $2n$.

- Si $k = 0$, alors nécessairement $x = y = \mathbf{0}$ et $h(x,y) = f(\mathbf{0}) + g(\mathbf{0}) = 0$.
- Si k est compris entre 1 et $n-1$, alors l'ensemble $\{(x,y) \in \mathbb{F}_2^{2n} | \mathsf{w_H}(x,y) = k\}$ est la réunion disjointe des ensembles suivants :

  $$\{x \in \mathbb{F}_2^n | \mathsf{w_H}(x) = i\} \times \{y\} \ ,$$

  pour i compris entre 0 et $n-1$, avec y fixé et de poids $k-i$. Si $i = 0$, alors $h(x,y) = f(\mathbf{0}) + g(y)$. Comme g est WPB, $h(x,y)$ est équilibrée sur $\{\mathbf{0}\} \times \{y \in \mathbb{F}_2^n | \mathsf{w_H}(y) = k\}$. Lorsque $i \geq 0$, $h(x,y)$ est égal à $f(x) + g(y)$ ou à $f'(x) + g(y)$, selon la valeur de $g'(y)$. Comme f et f' sont WPB, h est équilibrée sur chacun de ces sous-ensembles.
- Si $k = n$, $\{(x,y) \in \mathbb{F}_2^{2n} | \mathsf{w_H}(x,y) = k\}$ est la réunion disjointe de l'ensemble $\{(\mathbf{0},\mathbf{1}), (\mathbf{1},\mathbf{0})\}$ avec les ensembles $\{x \in \mathbb{F}_2^n | \mathsf{w_H}(x) = i\} \times \{y\}$ pour i compris entre 1 et $n-1$ et y de poids $n-i$. Or, $h(\mathbf{0},\mathbf{1}) = 1$ et $h(\mathbf{1},\mathbf{0}) = 0$. Pour la même raison que précédemment, lorsque $i \geq 0$, $h(x,y)$ est égal à $f(x) + g(y)$ ou à $f'(x) + g(y)$, selon la valeur de $g'(y)$. Comme f et f' sont WPB, h est équilibrée sur chacun de ces sous-ensembles.

— Si $k \in \{n+1, \ldots, 2n-1\}$, alors l'ensemble $\{(x,y) \in \mathbb{F}_2^{2n} | \mathsf{w_H}(x,y) = k\}$ est la réunion disjointe de l'ensemble $\{1\} \times \{y \in \mathbb{F}_2^n | \mathsf{w_H}(y) = k-n\}$ avec les ensembles $\{x \in \mathbb{F}_2^n | \mathsf{w_H}(x) = i\} \times \{y\}$ où i est compris entre $k-n+1$ et $n-1$ avec y de poids $k-i$. Sur le premier ensemble, $h(x,y) = h(\mathbf{1},y) = g(y)$, comme g est WPB, h est équilibrée sur ce sous-ensemble, et pour la même raison que précédemment, h est aussi équilibrée sur chacun des ensembles $\{x \in \mathbb{F}_2^n | \mathsf{w_H}(x) = i\} \times \{y\}$ où i est compris entre $k-n+1$ et $n-1$ avec y de poids $k-i$.

— Finalement, si $k = 2n$, alors $x = y = \mathbf{1}$ et $h(\mathbf{1},\mathbf{1}) = 1+1+1 = 1$.

□

L'application la plus simple de ce théorème est obtenue lorsque $f = f'$ (ou de manière équivalente $g' = 0$). Notamment en choisissant $f(x_1,x_2) = x_1$ comme fonction de base WPB à 2 variables, et la même fonction pour g, on peut définir une famille de fonctions booléennes WPB pour tout $n = 2^\ell$:

Corollaire 6.12. *Soit $n = 2^\ell$, avec $\ell \geq 1$, alors la fonction booléenne à n variables définie par*

$$f(x_1,x_2,\ldots,x_n) = \sum_{a=1}^{\ell} \sum_{i=1}^{2^{\ell-a}} \prod_{j=0}^{2^{a-1}-1} x_{i+j2^{\ell-a+1}}$$

est une fonction booléenne parfaitement équilibrée à poids fixé.

Exemple 8. La fonction booléenne à 16 variables définie par

$$\begin{aligned} f(x_1,\ldots,x_{16}) &= x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8 \\ &\quad +x_1x_9+x_2x_{10}+x_3x_{11}+x_4x_{12} \\ &\quad +x_1x_5x_9x_{13}+x_2x_6x_{10}x_{14} \\ &\quad +x_1x_3x_5x_7x_9x_{11}x_{13}x_{15} \end{aligned}$$

est WPB.

Nous remarquons que cette forme algébrique normale comporte bien 8 monômes de degré 1, 4 monômes de degré 2 (proposition 6.6) et 1 monôme de degré 8 (proposition 6.7). De plus, les fonctions définies par le corollaire 6.12 ont toutes $n-1$ monômes dans leur forme algébrique normale. Nous conjecturons que $n-1$ est le nombre minimal de monômes que l'on puisse obtenir dans la forme algébrique normale d'une fonction WPB à $n = 2^\ell$.

Conjecture 6.13. *Soit n une puissance de 2, il n'existe pas de fonctions parfaitement équilibrée à poids fixé dont le nombre de monômes dans la forme algébrique normale est strictement plus petit que $n-1$.*

Il est clair que le théorème 6.11 nous donne beaucoup plus de fonctions WPB que celle décrite au corollaire 6.12. De plus, il est possible d'étendre la famille de fonctions particulières définies au corollaire 6.12 à toute valeur de n afin de définir une famille de fonctions presque parfaitement équilibrées à poids fixé pour un nombre quelconque de variables.

Proposition 6.14. *Soit $n \geq 2$ et soit $f_2(x_1, x_2) = x_1$. La fonction booléenne f_n à n variables définie récursivement par*

$$f_n(x_1, \ldots, x_n) = \begin{cases} f_{n-1}(x_1, \ldots, x_{n-1}) & \text{si } n \text{ est impair} \\ f_{n-1}(x_1, \ldots, x_{n-1}) + x_{n-2} + \prod_{i=1}^{2^{d-1}} x_{n-i} & \text{si } n = 2^d \\ f_{n-1}(x_1, \ldots, x_{n-1}) + x_{n-2} + \prod_{i=1}^{2^d} x_{n-i} & \text{si } n = p2^d \end{cases}$$

où $p > 1$ est impair, est presque parfaitement équilibrée à poids fixé de degré 2^{d-1}, où $d = \lfloor \log_2(n) \rfloor$. De plus, son ANF contient $n-1$ monômes si n est pair et $n-2$ monômes si n est impair.

La preuve de cette construction étant relativement complexe, voici les premières fonctions dérivées de cette construction pour des nombres pairs de variables (le cas impair étant simple).

$$\begin{aligned} f_2 &= x_1 \\ f_4 &= x_1 + x_2 + x_2x_3 \\ f_6 &= x_1 + x_2 + x_2x_3 + x_4 + x_4x_5 \\ f_8 &= x_1 + x_2 + x_2x_3 + x_4 + x_4x_5 + x_6 + x_4x_5x_6x_7 \\ f_{10} &= x_1 + x_2 + x_2x_3 + x_4 + x_4x_5 + x_6 + x_4x_5x_6x_7 + x_8 + x_9x_8 \\ f_{12} &= x_1 + x_2 + x_2x_3 + x_4 + x_4x_5 + x_6 + x_4x_5x_6x_7 + x_8 + x_9x_8 \\ &\quad + x_{10} + x_8x_9x_{10}x_{11} \end{aligned}$$

Le but de cette construction est de rajouter, à chaque étape, selon la divisibilité de n par 2^d, les monômes de degré 2^d nécessaires à la construction des fonctions WAPB et WPB. Il est de plus important de remarquer que, peu importe le cas, la variable x_n n'apparaît jamais dans la forme algébrique normale des fonctions f_n définies par la proposition 6.14.

Démonstration. Nous montrons tout d'abord le caractère équilibré de la fonction à poids fixé par récurrence sur n. Pour $n = 2$, $f_2(x_1, x_2) = x_1$ est WPB. On suppose maintenant que $n \geq 3$, et on fait l'hypothèse que pour tout $2 \leq i \leq n-1$, f_i est WAPB.

Cas où n est impair.

— Si $k = 0$, alors $\mathsf{w_H}(f_n)_0 = \mathsf{w_H}(f_{n-1})_0 = 0$.

— Si k est compris entre 1 et $n-1$, alors $\mathsf{w_H}(f_n)_k = \mathsf{w_H}(f_{n-1})_k + \mathsf{w_H}(f_{n-1})_{k-1}$. Comme n est impair, $n-1$ est pair. D'après le théorème de Lucas [Luc78], on déduit qu'au moins un des coefficients $\binom{n-1}{k}$, $\binom{n-1}{k-1}$ est pair. Ainsi, si les deux coefficients sont pairs,

$$\mathsf{w_H}(f_{n-1})_k + \mathsf{w_H}(f_{n-1})_{k-1} = \frac{\binom{n-1}{k} + \binom{n-1}{k-1}}{2} = \frac{\binom{n}{k}}{2} ,$$

et naturellement

$$\mathsf{w_H}(f_{n-1})_k + \mathsf{w_H}(f_{n-1})_{k-1} = \frac{\binom{n-1}{k} + \binom{n-1}{k-1} \pm 1}{2} = \frac{\binom{n}{k} \pm 1}{2}$$

sinon.

— Si maintenant $k = n$, alors $\mathsf{w_H}(f_n)_n = \mathsf{w_H}(f_{n-1})_{n-1} = 1$.

Dans les deux cas qui suivent, il est assez ardu de comprendre exactement comment fonctionne la construction. L'astuce consiste ici à appliquer plusieurs fois la récurrence (typiquement 2^{d-1} ou 2^d fois), de manière à rajouter à chaque étape les monômes de degré 2^i, pour i variant de 0 à d. En effet, la construction récursive appliquée dans les deux cas qui vont suivre 2^d ou 2^{d-1} fois permet de retrouver la fonction définie au corollaire 6.12, mais avec un ordonnancement des variables différents. La preuve est assez difficile à suivre, cependant, c'est la raison pour laquelle nous avons pu avoir un énoncé aussi simple : nous sommes passés d'une construction où l'on "coupe" le nombre de variables en deux, à une construction où l'on passe de n à $n+1$ variables. C'est celà qui rend la preuve relativement compliquée. Un exemple sur un petit nombre de variables sera donné dans les deux cas afin de mieux comprendre ce qui se passe.

Cas où $n = 2^d$, $d > 1$. Prenons comme exemple la construction de f_{16}. $16 = 2^4$, on applique la récurrence 2^3 fois, ainsi, on exprime f_{16} en fonction de f_8 et des monômes obtenus par la récurrence. Plus précisément, en continuant la construction décrite en exemple, on a

$$\begin{aligned}
f_{13} &= f_{12} \\
f_{14} &= f_{12} + x_{12} + x_{12}x_{13} \\
f_{15} &= f_{14} \\
f_{16} &= f_{12} + x_{12} + x_{12}x_{13} + x_{14} + x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}\ .
\end{aligned}$$

Dans ces conditions, si l'on exprime f_{16} en fonction de f_8, on obtient

$$\begin{aligned}
f_{16} = f_8 &\\
&+ x_8 + x_{10} + x_{12} + x_{14} \\
&+ x_8x_9 + x_{12}x_{13} \\
&+ x_8x_9x_{10}x_{11} \\
&+ x_8x_9x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}
\end{aligned}$$

Comme x_8 n'intervient pas dans l'ANF de f_8 par construction, on peut échanger les variables x_8 et x_{16}. Nous retrouvons donc bien, à un réordonnancement des variables près, la fonction f_8, ajoutée une fonction à 8 variables qui est, à réordonnancement près des variables, définie au corollaire 6.12. L'ajout du monôme de degré 8 rend alors l'ensemble WPB, comme montré au théorème 6.11.

De manière générale, on applique 2^{d-1} fois la récurrence. Comme n varie de 2^d à 2^{d-1}, on est donc toujours dans le cas 1 ou 3 sauf quand $n = 2^d$, on a

$$\begin{aligned} f_n &= \prod_{i=1}^{2^{d-1}} x_{n-i} + f_{2^{d-1}}(x_1, \ldots, x_{2^{d-1}}) \\ &\quad + x_{2^d-2} + x_{2^d-2} + \cdots + x_{2^{d-1}} \\ &\quad + x_{2^d-3}x_{2^d-4} + x_{2^d-7}x_{2^d-8} + \cdots + x_{2^{d-1}+1}x_{2^{d-1}} \\ &\quad + x_{2^d-5}x_{2^d-6}x_{2^d-7}x_{2^d-8} + \cdots + x_{2^{d-1}+3}x_{2^{d-1}+2}x_{2^{d-1}+1}x_{2^{d-1}} \\ &\quad \cdots \\ &\quad + x_{2^{d-1}+2^{d-2}-1}x_{2^{d-1}+2^{d-2}-2}\cdots x_{2^{d-1}} \end{aligned}$$

On remarque de plus que la fonction additionnée à f_{2^d} et $\prod_{i=1}^{2^{d-1}} x_{n-i}$ est exactement la fonction WPB définie à la proposition 6.12 après réordonnancement des variables, comme le montre l'exemple à 16 variables. De plus, x_{2^d} n'intervient pas dans la forme algébrique normale de f_{2^d}, et $x_{2^{d-1}}$ n'intervient pas dans la forme algébrique normale de $f_{2^{d-1}}$. Donc, on peut échanger ces variables, et finalement réécrire f_{2^d} de la manière suivante :

$$f_{2^d} = f_{2^{d-1}}(x_1, \ldots, x_{2^{d-1}-1}, x_{2^d}) + f_{2^{d-1}}(x_{2^{d-1}}, \ldots, x_{2^d-1}) + \prod_{i=1}^{2^{d-1}} x_{n-i}$$

ce qui, en appliquant le théorème 6.11 implique que la fonction booléenne f_{2^d} est WPB.

Cas où $n = p2^d$, $p > 1$ impair. Prenons comme exemple la construction de f_{12}. $12 = 3 \times 2^2$, donc nous appliquons la récurrence 4 fois, c'est-à-dire que l'on exprime f_{12} en fonction de f_8.

$$f_{12} = f_8 + x_8 + x_{10} + x_8x_9 + x_8x_9x_{10}x_{11} \ .$$

Les monômes colorés en bleu correspondent bien à une fonction à WPB à 4 variables, f_8 est supposée WPB et le monôme de degré 4 rajouté va nous permettre de conclure.

Plus généralement, aucun j dans l'intervalle $[n - 2^d + 1, n - 1]$, n'est une puissance de 2. On est donc toujours dans le cas 1 ou 3. En faisant la même chose que pour le cas $n = 2^d$, on obtient que

$$\begin{aligned} f_n &= \prod_{i=1}^{2^d} x_{n-i} + f_{n-2^d}(x_1, \ldots, x_{n-2^d}) \\ &\quad + x_{n-2} + x_{n-4} + \cdots + x_{n-2^d} \\ &\quad + x_{n-3}x_{n-4} + x_{n-7}x_{n-8} + \cdots + x_{n-2^d+1}x_{n-2^d} \\ &\quad + x_{n-5}x_{n-6}x_{n-7}x_{n-8} + \cdots + x_{n-2^d+3}x_{n-2^d+2}x_{n-2^d+1}x_{n-2^d} \\ &\quad \cdots \\ &\quad + x_{n-2^{d-1}-1}x_{n-2^{d-1}-2}\cdots x_{n-2^d} \end{aligned}$$

Comme x_n n'intervient pas dans la forme algébrique normale de f_n ainsi que x_{n-2^d} dans f_{n-2^d}, on peut réécrire f_n de la manière suivante en réordonnant ces variables :

$$f_n = f_{n-2^d}(x_1, \ldots, x_{n-2^d-1}, x_n) + g(x_{n-2^d}, \ldots, x_{n-1}) + \prod_{i=1}^{2^d} x_{n-i}$$

où g est WPB puisque l'on retrouve, à réordonnancement près des variables, la fonction à 2^d variables définie au corollaire 6.12, et f_{n-2^d} est WAPB par hypothèse de récurrence. Pour pouvoir conclure, on note k le poids de Hamming de $(x_1, \ldots, x_n)$.

— Si $k = 0$, alors $\mathsf{w_H}(f_n)_0 = 0$.

— Si $k \in [1, 2^d - 1]$, alors

$$\mathsf{w_H}(f_n)_k = \sum_{i=0}^{k} \mathsf{w_H}(g)_i \overline{\mathsf{w_H}(f_{n-2^d})_{k-i}} + \overline{\mathsf{w_H}(g)_i} \mathsf{w_H}(f_{n-2^d})_{k-i} \ ,$$

g étant WPB à 2^d variables, cela signifie que $\mathsf{w_H}(g)_i = \overline{\mathsf{w_H}(g)_i}$ (sauf pour $i = 0$ où $\mathsf{w_H}(g)_0 = 0$). Donc, on obtient

$$\mathsf{w_H}(f_n)_k = \mathsf{w_H}(f_{n-2^d})_k + \sum_{i=1}^{k} \mathsf{w_H}(g)_i (\overline{\mathsf{w_H}(f_{n-2^d})_{k-i}} + \mathsf{w_H}(f_{n-2^d})_{k-i}) \ .$$

Donc, comme $\overline{\mathsf{w_H}(f_{n-2^d})_{k-i}} + \mathsf{w_H}(f_{n-2^d})_{k-i}$ est, par définition la cardinalité des mots de poids $k - i$ et de taille $n - 2^d$.

$$\mathsf{w_H}(f_n)_k = \mathsf{w_H}(f_{n-2^d})_k + \frac{1}{2} \sum_{i=1}^{k} \binom{2^d}{i} \binom{n-2^d}{k-i} \ .$$

Les propriétés des coefficients binomiaux nous donnent alors la relation suivante :

$$\mathsf{w_H}(f_n)_k = \mathsf{w_H}(f_{n-2^d})_k + \frac{1}{2} \left(\binom{n}{k} - \binom{n-2^d}{k} \right) \ .$$

f_{n-2^d} étant WAPB à $n - 2^d$ variables, il en découle que $\mathsf{w_H}(f_n)_k = \frac{1}{2}\binom{n}{k}$ si $\binom{n-2^d}{k}$ est pair (*i.e.* $\binom{n}{k}$ est pair) et $\mathsf{w_H}(f_n)_k = \frac{1}{2}\left(\binom{n}{k} \pm 1\right)$ sinon.

— Si $k \in [2^d, n-1]$, en faisant une disjonction sur les poids possibles des

variables en entrées de f_{n-2^d} et des variables en entrée de g, on a

$$\begin{aligned}
\mathsf{w_H}(f_n)_k = &\sum_{i=1}^{2^d-1} \mathsf{w_H}(g)_i\overline{\mathsf{w_H}(f_{n-2^d})_{k-i}} + \overline{\mathsf{w_H}(g)_i}\mathsf{w_H}(f_{n-2^d})_{k-i} \\
&+ \mathsf{w_H}(g)_0\overline{\mathsf{w_H}(f_{n-2^d})_k} + \overline{\mathsf{w_H}(g)_0}\mathsf{w_H}(f_{n-2^d})_k \\
&+ \overline{\mathsf{w_H}(g)_{2^d}\mathsf{w_H}(f_{n-2^d})_{k-2^d}} + \mathsf{w_H}(g)_{2^d}\mathsf{w_H}(f_{n-2^d})_{k-2^d} \\
=&\sum_{i=1}^{2^d-1} \frac{1}{2}\binom{2^d}{i}\binom{n-2^d}{k-i} + \mathsf{w_H}(f_{n-2^d})_k + \mathsf{w_H}(f_{n-2^d})_{k-2^d} \\
=&\frac{1}{2}\left(\binom{n}{k} - \binom{n-2^d}{k} - \binom{n-2^d}{k-2^d}\right) \\
&+ \mathsf{w_H}(f_{n-2^d})_k + \mathsf{w_H}(f_{n-2^d})_{k-2^d}
\end{aligned}$$

Comme $n - 2^d \equiv 0 \mod 2^{d+1}$, nous pouvons assurer qu'au moins un des coefficients binomiaux $\binom{n-2^d}{k}$, $\binom{n-2^d}{k-2^d}$ est pair, ce qui nous permet de conclure sur le caractère (presque) équilibré de f_n.

— Si $k = n$, alors

$$\mathsf{w_H}(f_n)_n = \mathsf{w_H}(f_{n-2^d})_{n-2^d}\mathsf{w_H}(g)_{2^d} + \overline{\mathsf{w_H}(f)_{n-2^d}\mathsf{w_H}(g)_{2^d}} = 1$$

.

Finalement, nous pouvons donc conclure que f_n est une fonction WAPB pour tout $n \geq 2$.

Soit $n \geq 2$, montrons que f_n est de degré 2^{d-1} où $d = \lfloor \log_2(n) \rfloor$ et possède $n-1$ monômes si n est pair et $n-2$ si n est impair.

La propriété est vraie au rang 2 et 3 : $f_2(x_1, x_2) = x_1$ et $f_3(x_1, x_2, x_3) = x_1$. Soit $n \geq 2$, on suppose la propriété vraie pour f_n. Soit d tel que $d = \lfloor \log_2(n) \rfloor$.

— Si $n+1$ est impair (cas 1), alors $f_{n+1}(x_1, \ldots, x_{n+1}) = f_n(x_1, \ldots, x_n)$ et $n+1 < 2^{d+1}$. Par hypothèse de récurrence, f_n est de degré 2^{d-1} et possède $n-1$ monômes, donc f_{n+1} est également de degré 2^{d-1} et possède $n-1$ monômes.

— Si $n+1 = 2^k$ (cas 2), alors nécessairement $k = d+1$, on rajoute deux monômes dont un de degré 2^d. Par hypothèse de récurrence, comme $n+1$ est pair, n est impair, donc f_n possède $n-2$ monômes et est de degré 2^{d-1}, donc f_{n+1} est de degré 2^d et possède n monômes.

— Si $n+1 = p2^k$, p impair et $p > 1$ (cas 3), alors, comme $n+1 < 2^{d+1}$, donc $2^k < 2^{d+1}/p$, avec $p \geq 3$, donc $k < d-1$. Par hypothèse de récurrence, on a bien f_{n+1} de même degré que f_n et son nombre de monômes est n,

ce qui conclut la preuve. □

6.3 La non-linéarité

Alors qu'il est assez simple de montrer à quel point le critère de l'équilibre peut dégénérer, il semble plus ardu d'exhiber des exemples de fonctions booléennes qui ont une bonne non-linéarité sur $\mathbb{F}_2^n$, mais pour lesquelles ce critère est complètement dégénéré lorsque l'entrée est fixée. Cependant, de telles fonctions existent, et même plus, il existe des fonctions *courbes*, *i.e.* des fonctions qui ont la meilleure non-linéarité possible, mais qui sont linéaires à poids fixé.

Or, en y réfléchissant un peu, il suffit de reprendre nos fonctions symétriques élémentaires (définition 6.1), et plus particulièrement celle de degré 2 à n variables :

$$\sigma_2(x) = \binom{\mathsf{wH}(x)}{2} = \sum_{1 \le i < j \le n} x_i x_j \ .$$

Cette fonction booléenne est connue, à l'addition de la première fonction symétrique élémentaire près, comme la seule fonction courbe symétrique [Sav94]. Comme cette fonction est symétrique, elle est donc constante à poids fixé, donc elle est aussi de non-linéarité nulle. Dans cette section, nous définissons la non-linéarité à poids fixé, puis nous donnons quelques résultats sur ce nouveau critère.

6.3.1 Définitions et bornes

Définition 6.15. *Soit S un sous-ensemble de $\mathbb{F}_2^n$ et f n'importe quelle fonction booléenne à n variables. Nous définissons la non-linéarité de f sur S notée $\mathsf{NL}_S(f)$ la plus petite distance de Hamming de f à toutes les fonctions affines, restreintes à S.*

Soit S n'importe quel sous-ensemble de $\mathbb{F}_2^n$ et f une fonction de S dans $\mathbb{F}_2$ (ou f n'importe quelle fonction booléenne à n variables, restreinte à S). Soit $\ell(x) = a \cdot x + c$ une fonction affine quelconque. On note $f_a(x)$ la fonction définie sur S par $f_a(x) = f(x) + a \cdot x$. Alors, on a

$$\sum_{x \in S} (-1)^{f(x)+a\cdot x} = \sum_{x \in S} (1 - 2f_a(x)) \ .$$

Donc, le poids de Hamming de $f_a(x)$ restreint à S peut être écrit comme

$$\sum_{x \in S} f_a(x) = \frac{|S|}{2} - \frac{1}{2} \sum_{x \in S} (-1)^{f(x)+a\cdot x} \ ,$$

où la somme est réalisée sur $\mathbb{Z}$. Dans ces conditions, la distance entre f et ℓ vaut

$$\frac{|S|}{2} - \frac{(-1)^c}{2} \sum_{x \in S} (-1)^{f(x)+a\cdot x} \ .$$

On déduit alors de ces observations la proposition suivante.

Proposition 6.16. *Pour toute fonction booléenne à n variables et tout sous-ensemble S de $\mathbb{F}_2^n$,*

$$\mathsf{NL}_S(f) = \frac{|S|}{2} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} \left| \sum_{x \in S} (-1)^{f(x)+a\cdot x} \right| .$$

Alors que la plupart des critères classiques sont invariants par composition d'une application linéaire inversible, il n'en est rien pour les nouveaux critères que l'on décrit dans ce chapitre. En effet, comme l'on s'intéresse à des *sous-ensembles* arbitraires n'ayant aucune structure spécifique, les transformations linéaires ne préservent a priori pas les sous-ensembles que l'on considère (et particulièrement lorsque le poids est fixé).

Cependant, des résultats similaires au cas classique existent : nous pouvons tout d'abord donner une borne supérieure sur la meilleure non-linéarité possible en fonction de la taille de S, qui correspond à la relation de Parseval.

Proposition 6.17. *Pour tout sous-ensemble S de $\mathbb{F}_2^n$, et toute fonction booléenne à n variables,*

$$\mathsf{NL}_S(f) \leq \frac{|S|}{2} - \frac{\sqrt{|S|}}{2} .$$

Démonstration.

$$\sum_{a \in \mathbb{F}_2^n} \left(\sum_{x \in S} (-1)^{f(x)+a\cdot x} \right)^2 = \sum_{x,y \in S} (-1)^{f(x)+f(y)} \sum_{a \in \mathbb{F}_2^n} (-1)^{a\cdot(x+y)} .$$

Or, $\sum_{a \in \mathbb{F}_2^n} (-1)^{a\cdot(x+y)}$ vaut 0 si $x \neq y$ et 2^n sinon. Ainsi, on a

$$\sum_{a \in \mathbb{F}_2^n} \left(\sum_{x \in S} (-1)^{f(x)+a\cdot x} \right)^2 = 2^n |S| .$$

Finalement, on déduit qu'il existe au moins un $a \in \mathbb{F}_2^n$ tel que

$$\left(\sum_{x \in S} (-1)^{f(x)+a\cdot x} \right)^2 \geq |S| ,$$

ce qui conclut la preuve. □

Évidemment, lorsque $S = \mathbb{F}_2^n$, on retrouve la non-linéarité classique, où la borne est atteinte quand la fonction est une fonction *courbe*.

De plus, cette borne peut-être améliorée sensiblement, et nous appliquerons notre amélioration au cas où $S = S_{n,k}$.

Proposition 6.18. *Soit S un sous-ensemble de $\mathbb{F}_2^n$, f une fonction booléenne à n variables et $\mathcal{F}$ une famille d'espaces vectoriels F pour lesquels il existe $v \in \mathbb{F}_2^n$ tel que $v \cdot (x+y) = 1$ pour tout $(x,y) \in S^2$, $x \neq y$, $x+y \in F^\perp$. Alors,*

$$\mathsf{NL}_S(f) \leq \frac{|S|}{2} - \frac{1}{2}\sqrt{|S| + \lambda},$$

où

$$\lambda = \max_{F \in \mathcal{F}} \Big| \sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)} \Big|.$$

Démonstration. Soit F un sous-espace vectoriel de $\mathbb{F}_2^n$. Alors, on a

$$\begin{aligned}
\sum_{a \in F} \left(\sum_{x \in S} (-1)^{f(x)+a\cdot x} \right)^2 &= \sum_{(x,y) \in S^2} (-1)^{f(x)+f(y)} \sum_{a \in F} (-1)^{a\cdot(x+y)} \\
&= |F| \sum_{\substack{(x,y) \in S^2 \\ x+y \in F^\perp}} (-1)^{f(x)+f(y)} \\
&= |F| \left(|S| + \sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)} \right),
\end{aligned}$$

ce qui implique

$$\max_{a \in \mathbb{F}_2^n} \Big| \sum_{x \in S} (-1)^{f(x)+a\cdot x} \Big| \geq \sqrt{|S| + \sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)}},$$

et que

$$\mathsf{NL}_S(f) \leq \frac{|S|}{2} - \frac{1}{2}\sqrt{|S| + \sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)}}.$$

Supposons maintenant qu'il existe v appartenant à $\mathbb{F}_2^n$ tel que, pour tout $(x,y) \in S^2$, $x \neq y$ tel que $x+y \in F^\perp$, on a $v \cdot (x+y) = 1$. Supposons de plus que

$$\sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)} \neq 0$$

et on note λ sa valeur. Alors, sans perdre de généralité, on peut considérer que λ est positif. En effet, si λ est négatif, alors en considérant $f'(x) = f(x) + v \cdot x$; nous avons

$$\sum_{\substack{(x,y) \in S^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f'(x)+f'(y)} = \sum_{\substack{(x,y) \in E^2 \\ 0 \neq x+y \in F^\perp}} (-1)^{f(x)+f(y)+v\cdot(x+y)} = -\lambda > 0$$

. □

En choisissant pour $\mathcal{F}$ la famille de tous les hyperplans de $\mathbb{F}_2^n$, on obtient le corollaire suivant :

Corollaire 6.19. *Soit S un sous-ensemble de $\mathbb{F}_2^n$ et f une fonction booléenne à n variables, alors :*

$$\mathsf{NL}_S(f) \leq \frac{|S|}{2} - \frac{1}{2}\sqrt{|S| + \lambda},$$

où

$$\lambda = \max_{a \in \mathbb{F}_2^n, a \neq 0} \Big| \sum_{\substack{(x,y) \in S^2 \\ x+y=a}} (-1)^{f(x)+f(y)} \Big|.$$

Nous remarquons que ce corollaire fournit aussi une autre manière de montrer que toutes les dérivées des fonctions courbes sont équilibrées. En effet, comme les fonctions courbes atteignent la borne maximale, cela implique que $\lambda = 0$, ce qui signifie exactement que toute dérivée est équilibrée.

6.3.2 Cas du poids fixé

On applique les résultats précédents aux sous-ensembles $S_{n,k}$. Par souci de lisibilité, on note $\mathsf{NL}_k := \mathsf{NL}_{S_{n,k}}$ pour k allant de 0 à n. Ainsi, on obtient pour toute fonction booléenne à n variables que

$$\mathsf{NL}_k(f) \leq \frac{\binom{n}{k}}{2} - \frac{1}{2}\sqrt{\binom{n}{k}}. \tag{6.1}$$

Cette borne ne peut évidemment être atteinte que si $\binom{n}{k}$ est un carré. Nous montrons maintenant que même dans ce cas la borne n'est pas atteinte.

Corollaire 6.20. *Pour tout n et $k \in \{1, \ldots, n\}$, la borne (6.1) n'est jamais atteinte, sauf peut-être pour $(n, k) = (50, 3)$ ou $(50, 47)$.*

Démonstration. Tout d'abord, Erdös a montré [AZ04] que les coefficients binomiaux ne sont jamais des carrés, sauf lorsque $k = 0$, 1 ou 2 (et naturellement $k = n$, $n - 1$ ou $n - 2$) ou $n = 50$ et $k \in \{3, 47\}$.

$k = 1$ Lorsque $k = 1$, chaque monôme de degré strictement plus grand que 1 est nul, donc la fonction considérée coïncide avec une fonction affine, définie par tous les monômes de degré 1 et le coefficient constant dans la forme algébrique normale de f.

$k = 2$ On utilise le corollaire 6.19 et on calcule la valeur de λ. Soit $a \in \mathbb{F}_2^n$, $a \neq 0$. On note i le poids de Hamming de a. Tout d'abord, si i est impair, alors $\{(x, y) \in S_{n,k}^2 | x + y = a\}$ est l'ensemble vide. Si en revanche i est pair, alors $|\{(x, y) \in S_{n,k}^2 | x + y = a\}|$ est égal au nombre de choix de $i/2$ indices dans le support de a et $k - i/2$ indices en dehors du support de a. Ainsi, on a

$$|\{(x, y) \in S_{n,k}^2 | x + y = a\}| = \binom{i}{\frac{i}{2}} \binom{n-i}{k-\frac{i}{2}}.$$

Or, comme la somme

$$\sum_{\substack{(x,y)\in S^2\\ x+y=a}} (-1)^{f(x)+f(y)}$$

est invariante si x et y sont échangés, cela signifie que si le nombre d'éléments dans $\{(x,y) \in S_{n,k}^2 | x+y=a\}$ n'est pas un multiple de 4, alors λ est égal à deux fois la somme d'un nombre impair d'entiers valant ± 1, où λ est défini par le corollaire 6.19, il est donc strictement positif. Pour $n \geq 4$, en prenant a de poids égal à 4, on obtient $|\{(x,y) \in S_{n,k}^2 | x+y=a\}| = 6$. Donc la borne ne peut être atteinte, ce qui conclut la preuve.

Naturellement, le cas $k = n$, $k = n-1$ et $k = n-2$ découle naturellement du raisonnement précédent en considérant la fonction booléenne suivante :

$$f'(x_1, x_2, \ldots, x_3) = f(\neg x_1, \neg x_2, \ldots, \neg x_n)$$

□

Suite à notre travail, Sihem Mesnager a amélioré notre borne sur la non-linéarité, en élevant la somme à une puissance plus grande que 2 [MZD18].

6.3.3 Lien avec les codes de Reed et Muller

Nous utilisons maintenant le lien entre les fonctions booléennes et les codes de Reed et Muller, car ce sont deux approches différentes des objets que l'on regarde. Les codes de Reed et Muller $\mathcal{RM}(r,n)$ sont des codes binaires de longueur 2^n dont les mots de code sont les évaluations sur l'ensemble des entrées possibles ($\mathbb{F}_2^n$) des fonctions booléennes à n variables dont le degré est au plus r. Fixer le poids de Hamming en entrée revient à poinçonner le code de Reed et Muller, sur toutes les entrées dont le poids de Hamming est différent de k.

Définition 6.21. *Pour tout $n \in \mathbb{N}^*$, $r, k \in [0,n]$, on note $\mathcal{RM}(r,n)_k$ le code de Reed et Muller poinçonné de longueur $\binom{n}{k}$ obtenu en poinçonnant $\mathcal{RM}(r,n)$ sur chaque entrée de poids de Hamming différent de k.*

$\mathcal{RM}(1,n)_k$ correspond donc à l'évaluation de toutes les fonctions affines à n variables sur les entrées de poids de Hamming égal à k ; ainsi, pour toute fonction booléenne f, $\mathsf{NL}_k(f)$ est la distance entre la table de vérité de f restreinte aux entrées de poids k et $\mathcal{RM}(1,n)_k$. Naturellement, la valeur maximale de $\mathsf{NL}_k(f)$ est, par définition égale au *rayon de recouvrement* de $\mathcal{RM}(1,n)_k$, lui même minoré par la distance minimale divisée par deux.

Proposition 6.22. *$\mathcal{RM}(1,n)_k$ est un code linéaire de paramètres $[\binom{n}{k}, n, d]$, avec*

$$d = \frac{1}{2}\left(\binom{n}{k} - \max_{0<j\leq n/2} \left|\sum_{i\in\mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i}\right|\right) .$$

Démonstration. Soit $\ell_I(x) = \sum_{i \in I} x_i$ une fonction linéaire quelconque dont la restriction aux entrées de poids fixé à k est non constante, et soit j le cardinal de I. Alors, $j \in \{1, \ldots, n-1\}$. Le nombre d'éléments x de poids de Hamming égal à k tels que $|\mathsf{supp}(x) \cap I| = i$ vaut exactement $\binom{j}{i}\binom{n-j}{k-i}$. Le poids de ℓ_I est donc égal à

$$\sum_{i \text{ impair}} \binom{j}{i}\binom{n-j}{k-i}$$

alors que le poids de la fonction affine $(\ell_I + 1)$ vaut

$$\sum_{i \text{ pair}} \binom{j}{i}\binom{n-j}{k-i} .$$

On en déduit alors que la distance minimale de $\mathcal{RM}(1,n)_k$ vaut

$$d = \min_{0<j<n} \left(\min \left(\sum_{i \text{ impair}} \binom{j}{i}\binom{n-j}{k-i}, \sum_{i \text{ pair}} \binom{j}{i}\binom{n-j}{k-i} \right) \right) .$$

Or,

$$\sum_{i \text{ impair}} \binom{j}{i}\binom{n-j}{k-i} = \frac{1}{2} \left(\sum_{i \in \mathbb{Z}} \binom{j}{i}\binom{n-j}{k-i} - \sum_{i \in \mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i} \right)$$

et

$$\sum_{i \text{ pair}} \binom{j}{i}\binom{n-j}{k-i} = \frac{1}{2} \left(\sum_{i \in \mathbb{Z}} \binom{j}{i}\binom{n-j}{k-i} + \sum_{i \in \mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i} \right) ,$$

donc

$$d = \frac{1}{2} \min_{0<j<n} \left(\sum_{i \in \mathbb{Z}} \binom{j}{i}\binom{n-j}{k-i} - \left| \sum_{i \in \mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i} \right| \right)$$

et

$$d = \frac{1}{2} \left(\binom{n}{k} - \max_{(0<j<n)} \left| \sum_{i \in \mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i} \right| \right) .$$

Comme

$$\left| \sum_{i \in \mathbb{Z}} (-1)^i \binom{j}{i}\binom{n-j}{k-i} \right|$$

est invariant par la modification $j \mapsto n-j$ (en modifiant i en $k-i$) ; nous pouvons donc remplacer $\max_{0<j<n}$ par $\max_{0<j\leq n/2}$. □

Dumer et Kapralova ont aussi étudié en 2013 [DK13] ce code poinçonné particulier et ont plus récemment généralisé leur étude à $\mathcal{RM}(r,n)_k$ dans [DK17]. Son rayon de recouvrement n'est toujours pas connu, mais la distance minimale

nous donne cependant une borne inférieure $(d/2)$. Ainsi, c'est une approche qui nous permettrait d'affiner notre connaissance sur la non-linéarité lorsque l'entrée est restreinte à un certain sous-ensemble.

Déterminer la valeur maximale de la non-linéarité restreinte semble être un problème très ardu [5], surtout sans considérer de structure particulière sur les éléments poinçonnés. En effet, ce qui rend beaucoup de techniques classiques impossibles à utiliser dans notre contexte est l'absence de structure dans l'ensemble considéré. Principalement, nos ensembles n'étant pas stables par application d'une fonction linéaire, cela empêche beaucoup de manipulations sur les fonctions que l'on considère.

6.3.4 Dégradation en considérant le poids de Hamming fixé

Comme nous l'avons déjà remarqué précédemment, fixer le poids de Hamming peut drastiquement détériorer la non-linéarité d'une fonction booléenne ; il suffit de prendre le cas de la fonction symétrique élémentaire de degré 2.

Proposition 6.23 (Caractérisation des fonctions de NL_k nulle, pour tout k)**.** *Soit $f \in \mathcal{B}_n$, alors $\mathsf{NL}_k(f) = 0$ pour tout $k \in \{0, \dots, n\}$ si et seulement s'il existe $n+1$ fonctions affines notées $a_0, \dots, a_n \in \mathcal{B}_n$ telles que*

$$f(x) = a_0 + \sum_{i=1}^{n} \sigma_i(x) a_i(x) \ .$$

Définition 6.24 (La fonction Σ)**.** *On définit la fonction vectorielle $\Sigma : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ par ses n fonction coordonnées égales aux n fonctions symétriques.*

Démonstration. Soit f de non-linéarité nulle pour tout k, ou, de manière équivalente, chaque restriction de f à $S_{n,k}$ est affine. Cela est équivalent à l'existence de fonctions symétriques $\phi_0, \dots, \phi_n$ telles que

$$f(x) = \phi_0(x) + \sum_{i=1}^{n} \phi_i(x) x_i \ .$$

Or, toute fonction booléenne symétrique peut s'écrire comme composition d'une fonction affine avec la fonction vectorielle Σ ci-dessus. Ainsi, nous pouvons assurer que $\mathsf{NL}_k(f) = 0$ pour tout k si et seulement si f est de la forme

$$f(x) = \ell_0 \circ \Sigma(x) + \sum_{i=1}^{n} \ell_i \circ \Sigma(x) x_i \ ,$$

où les ℓ_i sont affines. En regroupant les termes par leur expression entre les fonctions symétriques *élémentaires*, on obtient le résultat voulu. □

5. On ne sait déjà pas très bien le faire quand on ne restreint pas les entrées.

Cette caractérisation n'est valide que lorsque l'on considère tous les poids possibles. Il s'agit d'une condition particulière restrictive. En effet, dans la pratique, il ne suffit parfois que d'un seul poids pour lequel la fonction a des critères corrects (c'est exactement le cas de FLIP). Contraindre toutes les restrictions, pour l'ensemble des poids possibles ne doit être fait que si le contexte l'oblige.

De plus, cette caractérisation nous permet de nous rendre compte que l'intersection entre les fonctions courbes et les fonctions de non-linéarité nulle n'est pas vide, et nous pouvons alors décrire cet espace, lorsque l'on se restreint aux fonctions quadratiques.

Proposition 6.25. *Pour tout $n \geq 4$, les fonctions courbes et quadratiques qui satisfont $\mathsf{NL}_k(f) = 0$ pour tout k compris entre 0 et n sont de la forme*

$$f(x) = \sigma_1(x)\ell(x) + \sigma_2(x)$$

où ℓ est affine et $\ell(1, \ldots, 1) = 0$.

Démonstration. D'après la proposition 6.23, une fonction quadratique satisfait $\mathsf{NL}_k(f) = 0$ pour tout k si et seulement si, à l'addition d'une fonction affine près, elle est de la forme

$$f(x) = \sigma_1(x)\ell(x) + c\sigma_2(x)$$

où $c = 0$ ou 1 et ℓ est linéaire. La forme symplectique associée [Car07] $(x, y) \mapsto f(x + y) + f(x) + f(y) + f(0)$ est égale dans notre cas à

$$\sigma_1(y)\ell(x) + \sigma_1(x)\ell(y) + c \sum_{1 \leq j \neq i \leq n} x_i y_j \ .$$

En notant $\ell(x) = \sum_{i=1}^{n} l_i x_i$, le noyau de cette forme symplectique, *i.e.*

$$K = \{x \in \mathbb{F}_2^n | \forall y \in \mathbb{F}_2^n, f(x + y) + f(x) + f(y) + f(0) = 0\}$$

est donné par l'espace vectoriel des équations

$$(L_i) : \ell(x) + l_i \sum_{j=1}^{n} x_j + c \sum_{j \neq i} x_j = 0 \ ,$$

où i parcourt $\{1, \ldots, n\}$. Étudier le noyau K défini ci-dessus va nous permettre de caractériser les fonctions courbes, puisque les fonctions courbes quadratiques sont caractérisées par un noyau réduit à $\{0\}$ [Car07]. La somme de deux équations $(L_i) + (L_{i'})$ vaut

$$(L_i + L_{i'}) : (l_i + l_{i'}) \sum_{j=1}^{n} x_j + c(x_i + x_{i'}) = 0 \ .$$

Si $l_i = l_{i'}$, on obtient que, pour tout $x \in K$, $x_i = x_{i'}$ si $c = 1$ et aucune condition sinon. Si $l_i \neq l_{i'}$, on obtient que, pour tout $x \in K$, $\sum_{j=1}^{n} x_j = x_i + x_{i'}$ si $c = 1$ et pour tout $x \in K$, $\sum_{j=1}^{n} x_j = 0$ sinon.

Ainsi, en notant

$$I = \{i = 1, \ldots, n | l_i = 0\} ,$$

on a que, si $c = 1$, alors toutes les coordonnées d'indice $i \in I$ d'un élément de K doivent être égales à un certain bit b, et celles dans le complémentaire de I doivent être égales à $b + \sum_{j=1}^{n} x_j$. Si $c = 0$, alors il n'y a aucune condition sur $x \in K$ si $I = \emptyset$ ou $I = \{1, \ldots, n\}$ et si $I \neq \emptyset$ ou $\{1, \ldots, n\}$, la condition devient $\sum_{j=1}^{n} x_j = 0$.

Déterminons maintenant si K contient un élément x non nul.

— Supposons $x \in K$ tel que $\sum_{j=1}^{n} x_j = 0$, alors :

  — si $c = 1$, alors ou tous les x_i sont nuls, auquel cas (L_i) est satisfaite, ou tous les x_i sont égaux à 1. Dans ce cas n est nécessairement pair, et donc l'équation (L_i) implique la condition $\ell(1, \ldots, 1) = 1$. Cette dernière condition est équivalente au fait que I est de cardinalité impaire. Ainsi, $\ell(1, \ldots, 1) = 1$, si et seulement si $(1, \ldots, 1) \in K$.

  — Si $c = 0$, alors toutes les équations (L_i) sont de la forme $\ell(x) = 0$. Dans ces conditions, $K \neq \{0\}$, sauf si l'intersection de l'hyperplan défini par $\sum_{j=1}^{n} x_j = 0$ avec $\ell(x) = 0$ est égal à $\{0\}$, ce qui ne peut être le cas que pour $n = 2$. Donc, le cas $c = 0$ ne peut être compatible avec le caractère courbe de f que pour $n = 2$, ce qui nous permet d'écarter ce cas.

— Supposons $x \in K$ tel que $\sum_{j=1}^{n} x_j = 1$. Alors $c = 1$. D'après les observations ci-dessus, tous les x_i tels que $i \in I$ sont égaux à b, et ceux dont l'indice est dans le complémentaire de I sont égaux à $b + 1$. Comme les fonctions courbes n'existent que pour un nombre pair de variables, et que $\sum_{j=1}^{n} x_j = 1 = b|I| + (1 + b)(n - |I|) \mod 2$, alors $|I| \equiv 1 \mod 2$, ou, de manière équivalente, le nombre de monômes dans l'ANF de ℓ est impair, et dans ce cas $K \neq \{0\}$.

Finalement, il ne reste qu'un seul cas où $K = \{0\}$: pour $n \geq 4$, $c = 1$ et $\ell(1, \cdots, 1) = 0$. Nous remarquons que cette preuve revient à déterminer les cas où le système d'équations donné par les (L_i) est inversible, c'est-à-dire quand le noyau est réduit à $\{0\}$. □

6.3.5 Somme directe et non-linéarité restreinte à poids fixé

L'utilisation de la somme directe de deux fonctions booléennes comme définie au chapitre 2 à la définition 2.21 est utile pour la construction de fonctions cryptographiques, notamment car cela assure de transférer les propriétés cryptographiques classiques des fonctions constituantes vers la fonction ayant plus de variables. Pour le cas du poids de Hamming fixé, les relations sont plus compliquées. Pour le cas de la non-linéarité nous pouvons seulement dire la chose suivante.

Proposition 6.26 (Somme directe et $\mathsf{NL}_{S_{n,k}}$)**.** *Soit $f \in \mathcal{B}_n$ et $g \in \mathcal{B}_m$ et $h \in \mathcal{B}_{n+m}$ définie comme la somme directe de f et g, alors*

$$\mathsf{NL}_{S_{n+m,k}}(h) \geq \sum_{i=0}^{k} \binom{n}{i} \mathsf{NL}_{S_{m,k-i}}(g) + \mathsf{NL}_{S_{n,i}}(f) \left(\binom{m}{k-i} - 2\mathsf{NL}_{S_{m,k-i}}(g) \right) .$$

Démonstration.

$$\begin{aligned}
& \mathsf{NL}_k(h) \\
= \; & \frac{\binom{n+m}{k}}{2} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m} \left| \sum_{(x,y) \in S_{N,k}} (-1)^{h(x,y)+a\cdot x+b\cdot y} \right| \\
\geq \; & \frac{\binom{n+m}{k}}{2} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m} \sum_{i=0}^{k} \left| \sum_{x \in S_{n,i}, y \in S_{m,k-i}} (-1)^{f(x)+g(y)+a\cdot x+b\cdot y} \right| \\
= \; & \frac{\binom{n+m}{k}}{2} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m} \sum_{i=0}^{k} \left| \sum_{x \in S_{n,i}} (-1)^{f(x)+a\cdot x} \right| \left| \sum_{y \in S_{m,k-i}} (-1)^{g(y)+b\cdot y} \right| \\
\geq \; & \frac{\binom{n+m}{k}}{2} - \frac{1}{2} \sum_{i=0}^{k} \max_{a \in \mathbb{F}_2^n} \left| \sum_{x \in S_{n,i}} (-1)^{f(x)+a\cdot x} \right| \max_{b \in \mathbb{F}_2^m} \left| \sum_{y \in S_{m,k-i}} (-1)^{g(y)+b\cdot y} \right| \\
= \; & \frac{\binom{n+m}{k}}{2} - \frac{1}{2} \sum_{i=0}^{k} \left(\binom{n}{i} - 2\mathsf{NL}_i(f) \right) \left(\binom{m}{k-i} - 2\mathsf{NL}_{k-i}(g) \right) \\
= \; & \sum_{i=0}^{k} \binom{n}{i} \mathsf{NL}_{k-i}(g) + \sum_{i=0}^{k} \binom{m}{k-i} \mathsf{NL}_i(f) - 2 \sum_{i=0}^{k} \mathsf{NL}_i(f) \mathsf{NL}_{k-i}(g) \\
= \; & \sum_{i=0}^{k} \binom{n}{i} \mathsf{NL}_{k-i}(g) + \sum_{i=0}^{k} \mathsf{NL}_i(f) \left(\binom{m}{k-i} - 2\mathsf{NL}_{k-i}(g) \right) .
\end{aligned}$$

□

6.4 Immunité algébrique à poids fixé

Tout comme pour les deux précédents critères, nous pouvons exhiber des fonctions booléennes dont l'immunité algébrique est diminuée par le fait de restreindre le poids de Hamming des entrées. Comme l'immunité algébrique est moins intuitif que les deux critères précédents, nous proposons de décrire cette dégradation sur un exemple avec peu de variables.

Exemple 9. Soit f la fonction booléenne à 4 variables définie par

$$f(x_1, x_2, x_3, x_4) = 1 + x_1 + x_2 x_3 .$$

Cette fonction est d'immunité algébrique 2, valeur qui est optimale pour une fonction à 4 variables. La table de vérité de cette fonction est la suivante où ce qui est coloré en jaune correspond aux entrées de poids 2.

x_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
f	1	0	1	0	1	0	0	1	1	0	1	0	1	0	0	1

On se rend alors compte que, sur $S_{4,2}$, cette fonction est toujours de degré algébrique 2, mais son immunité algébrique restreinte vaut 1. En effet, $x_1(1 + x_1 + x_2x_3) = x_1x_2x_3$. Comme on est sur les mots de poids 2, $x_1x_2x_3$ est nul, donc $\mathsf{AI}_2(f) = 1$

Comme nous l'avons déjà expliqué aux chapitres précédents, les travaux de Courtois et Meier ont permis de montrer qu'une équation peut "en cacher une autre", ce qui implique que le critère pertinent n'est pas le degré algébrique des fonctions booléennes mais leur immunité algébrique. Dans le cas où l'on ne restreint pas l'entrée ($S = \mathbb{F}_2^n$), Courtois et Meier [CM03] ont montré que pour tous entiers d et e tels que $d + e \geq n$, il existe une fonction booléenne non nulle g, de degré au plus e, et une fonction booléenne h, de degré algébrique au plus d, telles que $h = gf$. Dans le contexte classique, la borne sur l'immunité algébrique est $\lceil n/2 \rceil$, lorsque n désigne le nombre de variables. Dans cette section, nous revisitons l'*immunité algébrique* en restreignant les entrées tout d'abord à des sous-ensembles arbitraires de $\mathbb{F}_2^n$ puis aux entrées de poids fixe.

6.4.1 Immunité algébrique restreinte

Soit S un sous-ensemble arbitraire de $\mathbb{F}_2^n$ et f n'importe quelle fonction définie sur S. On rappelle que dans le cas classique, l'immunité algébrique d'une fonction f est définie comme le plus petit degré algébrique possible pour un annulateur non-nul de f ou de $f + 1$. Dans ces conditions, nous pouvons généraliser la définition 2.20 du chapitre 2 lorsque l'on considère un sous-ensemble d'entrées.

Définition 6.27 (Immunité algébrique restreinte)**.** *Soit S un sous-ensemble de $\mathbb{F}_2^n$ et $f \in \mathcal{B}_n$. Alors l'immunité algébrique restreinte à S de f est définie par*

$$\mathsf{AI}_S(f) = \min\{\deg(g) | gf = 0 \text{ ou } g(f+1) = 0 \text{ et } g \neq 0 \text{ sur } S\}$$

Dans cette section, nous utilisons plusieurs fois une matrice particulière que nous définissons de la manière suivante.

Définition 6.28. *Soit d un entier et S un sous-ensemble non-vide de $\mathbb{F}_2^n$, alors nous définissons la matrice $M_{d,S}$, de taille*

$$\left(\sum_{i=0}^{d} \binom{n}{i}\right) \times |S|$$

dont les lignes sont indexées par les éléments u de $\mathbb{F}_2^n$ tels que $\mathsf{wH}(u) \leq d$, et les colonnes sont indexées par les éléments x de S, et où

$$M_{d,S}[u, x] = x^u = \prod_{i=1}^{n} x_i^{u_i}$$

Exemple 10. Sur $\mathbb{F}_2^4$, prenons par exemple

$$S = \{0001, 0101, 0111, 1111, 1101, 0100\}$$

et $d = 1$. Alors en indexant les lignes par les u tels que $\mathsf{w_H}(u) \leq 1$, *i.e.* $\{0000, 0001, 0010, 0100, 1000\}$ on a :

$$M_{1,S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Sans faire d'hypothèse particulière sur S, nous pouvons d'abord démontrer la proposition suivante qui généralise le travail de Courtois et Meier.

Proposition 6.29. *Soit S un sous-ensemble non-vide de $\mathbb{F}_2^n$ et f une fonction booléenne à n variables. Soit d et e deux entiers positifs. Soit $M_{d,S}$ et $M_{e,S}$ deux matrices définies par la définition 6.28. Si les rangs de ces matrices vérifient*

$$\mathsf{rang}(M_{d,S}) + \mathsf{rang}(M_{e,S}) > |S| \,,$$

alors il existe $g \in \mathcal{B}_n$ de degré plus petit que e, dont la restriction à S est non nulle, ainsi qu'une fonction booléenne h à n variables de degré algébrique plus petit que d, telles que la fonction gf coïncide avec h sur S.

Dans le contexte classique ($S = \mathbb{F}_2^n$), l'ensemble des monômes forme une famille libre et génératrice de l'espace vectoriel des fonctions booléennes, donc les matrices sont toujours de rang plein. Ce qui rend notre travail plus ardu ici, c'est la restriction à l'ensemble S, qui implique des dépendances entre les monômes : par exemple, les fonctions symétriques, lorsque S décrit un espace de poids fixe appartiennent au noyau de cette matrice, ce qui implique que les matrices ne sont pas de rang plein, et que la condition sur le rang est celle qui est pertinente, puisqu'il est nécessaire que la fonction g que l'on multiplie à f soit non identiquement nulle sur S.

Démonstration. Soit $\mathcal{F}_d$ (respectivement $\mathcal{F}_e$) une famille de monômes de degré plus petit que d (respectivement e), de taille maximale, et libre sur la restriction S. On note ces monômes de la manière standard : x^u, avec $\mathsf{w_H}(u) \leq d$. Par définition du rang de la matrice $M_{d,S}$ (respectivement $M_{e,S}$), on a $|\mathcal{F}_d| = \mathsf{rang}(M_{d,S})$ (respectivement $|\mathcal{F}_e| = \mathsf{rang}(M_{e,S})$).

Ensuite, nous considérons l'ensemble de fonctions $\mathcal{F}_e f$ composé des fonctions obtenues par le produit des éléments de $\mathcal{F}_e$ avec la fonction booléenne f (avec des répétitions possibles). Par hypothèse, $|\mathcal{F}_d| + |\mathcal{F}_e f|$ est strictement supérieur à la dimension de l'espace des fonctions booléennes définies uniquement sur la restriction S. Ainsi, il existe nécessairement une combinaison linéaire entre les éléments de $\mathcal{F}_d$ et $\mathcal{F}_e f$, qui induit une fonction nulle sur S. En séparant d'un côté les éléments venant de $\mathcal{F}_d$ et ceux de $\mathcal{F}_e f$, on obtient respectivement une

fonction h et une fonction g, telles que $fg = h$ sur S. De plus, les restrictions de g et de h à S sont nécessairement non-nulles, puisque les familles $\mathcal{F}_d$ et $\mathcal{F}_e$ ont été choisies libres sur S. □

En appliquant la proposition 6.29 à $e = 0$, nécessairement $\mathsf{rang}(M_{0,S}) = 1$, puisque la fonction constante à 1 ne s'annule jamais, on obtient le corollaire qui suit.

Corollaire 6.30. *Soit S un sous-ensemble non-vide de $\mathbb{F}_2^n$ et $f \in \mathcal{B}_n$. Soit d tel que $\mathsf{rang}(M_{d,S}) \geq |S|$, alors il existe une fonction booléenne à n variables de degré plus petit que d qui coïncide avec f sur S.*

Comme nous l'avons vu précédemment, les monômes de degré strictement plus grand que k sont tous nuls lorsque l'on considère une restriction à des mots de poids inférieur à k. Nous verrons dans la suite que ces monômes ne sont pas les seules fonctions qui sont dans le noyau de $M_{d,S}$ lorsque S correspond aux mots de poids fixé. Il est donc nécessaire d'avoir en tête que le degré algébrique peut décroître en restreignant l'entrée.

Par ailleurs, nous pouvons assurer l'existence d'un annulateur de degré plus petit que e en appliquant la proposition 6.29 à $d = 0$.

Corollaire 6.31. *Soit S un sous-ensemble non-vide de $\mathbb{F}_2^n$ et f une fonction booléenne à n variables, Soit e tel que $\mathsf{rang}(M_{e,S}) = |S|$, alors il existe un annulateur non-nul de f sur S et de degré algébrique au plus e.*

Finalement, le cas de l'immunité algébrique, qui considère simultanément les annulateurs de f et ceux de $f + 1$, revient à choisir $e = d$ (comme dans le contexte classique).

Corollaire 6.32. *Soit S un sous-ensemble non-vide de $\mathbb{F}_2^n$ et f une fonction booléenne à n variables, Soit e tel que $\mathsf{rang}(M_{e,S}) > \frac{|S|}{2}$, alors il existe g de degré algébrique au plus e, telle que gf ou $g(f + 1)$ est nulle sur S, et g n'est pas identiquement nulle sur S.*

Démonstration. En utilisant la même idée que dans [MPC04], nous reprenons les fonctions g et h obtenues dans la proposition 6.29. Ou bien g et h coïncident sur S, et dans ce cas, on a $gf + h = g(f + 1) = 0$ sur S, ou bien ces deux fonctions ne coïncident pas, et dans ce cas, en multipliant l'égalité $h = gf$ sur S par f, on obtient que $hf = gf$ sur S, et donc $(g + h)f = 0$. Comme g et h ne coïncident pas sur S et sont toutes deux de degré plus petit que $d = e$, on obtient bien une fonction annulatrice de f sur S et non-nulle sur S. □

Finalement, nous pouvons borner supérieurement la valeur de l'immunité algébrique lorsque l'on restreint nos entrées par le corollaire suivant.

Corollaire 6.33. *L'immunité algébrique de n'importe quelle fonction booléenne restreinte à S est bornée supérieurement par*

$$\min\left\{e;\ \mathsf{rang}(M_{e,S}) > \frac{|S|}{2}\right\}.$$

A priori, sans faire d'hypothèses sur la structure de S, on ne peut pas prévoir quelle sera la valeur de $\mathsf{rang}(M_{e,S})$. Ainsi, les bornes sur l'immunité algébrique ne peuvent être obtenues qu'en investiguant cette matrice, et ce pour tous les sous-ensembles que l'on devra considérer. Comme notre analyse s'axe autour du chiffrement FLIP, nous étudions cette matrice lorsque $S = S_{n,k}$.

6.4.2 Immunité algébrique pour des entrées de poids de Hamming fixé

Dans la suite, on note $M_{n,k,d}$ la matrice définie comme à la définition 6.28, où $S = S_{n,k}$, c'est-à-dire à l'ensemble des éléments de $\mathbb{F}_2^n$ de poids de Hamming égal à k. En utilisant les résultats précédents, il nous reste à étudier la matrice $M_{n,k,d}$, et plus particulièrement à calculer son rang. On constate par ailleurs que cette matrice est une matrice génératrice du code de Reed et Muller d'ordre d et poinçonné en enlevant les entrées de poids différent de k comme nous l'avons défini à la section 6.3.3.

Théorème 6.34. *Le rang de $M_{n,k,d}$ est égal à*

$$\binom{n}{\min(d, k, n-k)}.$$

Démonstration. Le principe de cette preuve consiste à trouver une relation de récurrence sur la valeur du rang. Afin d'y parvenir, nous utilisons une construction de cette matrice semblable à la construction $u||u+v$ des codes de Reed et Muller : toute fonction booléenne f à n variables et de degré algébrique au plus d peut s'écrire de la manière suivante :

$$f(x_1, \ldots, x_n) = x_n g(x_1, \ldots, x_{n-1}) + h(x_1, \ldots, x_{n-1}) \,,$$

où g est de degré algébrique au plus d et h est de degré algébrique au plus $d-1$.

Afin de simplifier la lecture de cette preuve, nous utilisons les notations suivantes :

$$N_k = \binom{n}{k} \quad \text{et} \quad D = \sum_{0 \le i \le d} \binom{n}{i}.$$

Soit $\psi_{n,k,d}$ l'application linéaire qui envoie la forme algébrique normale de toute fonction booléenne à n variables de degré au plus d sur la restriction de sa table de vérité à $S_{n,k}$:

$$\begin{array}{rccc} \psi_{n,k,d} & : & \mathbb{F}_2^D & \longrightarrow & \mathbb{F}_2^{N_k} \\ & & (a_u)_{u \in \mathbb{F}_2^n, \mathsf{wH}(u) \le d} & \longmapsto & \left(\sum_{u \preceq x} a_u\right)_{x \in \mathbb{F}_2^n, \mathsf{wH}(x) = k} \end{array}$$

où $u \preceq x$ signifie que le support de u est inclus dans le support de x ($\mathsf{supp}(u) \subseteq \mathsf{supp}(x)$). Cette application $\psi_{n,k,d}$ est évidemment linéaire par construction et le rang de la matrice $M_{n,k,d}$ est exactement le rang de cette application.

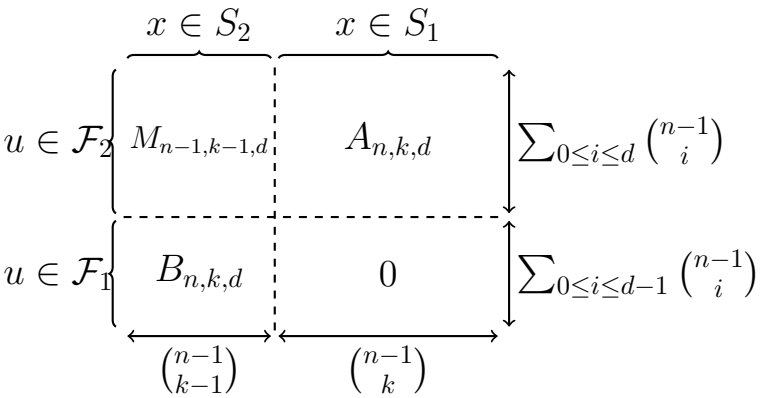

Figure 6.1 – Décomposition de la matrice $M_{n,k,d}$.

On note m_u le monôme x^u. Comme ces monômes forment une famille génératrice de l'ensemble des fonctions booléennes de degré plus petit que d (par définition), il convient de dire que le rang de l'application peut aussi être défini comme le rang de la famille de vecteurs images des $(m_u)_{\mathsf{w_H}(u) \le d}$ par $\psi_{n,k,d}$:

$$\{\psi_{n,k,d}(m_u)\}_{u \in \mathbb{F}_2^n, \mathsf{w_H}(u) \le d} .$$

Ensuite, nous partageons cette famille de vecteurs en deux familles $\mathcal{F}_1$ et $\mathcal{F}_2$ de la manière suivante :

$$\begin{aligned} \mathcal{F}_1 &= \{u \in \mathbb{F}_2^n, \mathsf{w_H}(u) \le d; u_n = 0\} \\ \mathcal{F}_2 &= \{u \in \mathbb{F}_2^n, \mathsf{w_H}(u) \le d; u_n = 1\} . \end{aligned}$$

D'une manière qui vient tout naturellement, nous découpons aussi notre ensemble $S_{n,k}$ en deux sous-ensembles S_1 et S_2 :

$$\begin{aligned} S_1 &= \{x \in \mathbb{F}_2^n, \mathsf{w_H}(x) = k; x_n = 0\} \\ S_2 &= \{x \in \mathbb{F}_2^n, \mathsf{w_H}(x) = k; x_n = 1\} . \end{aligned}$$

Avec ces notations, nous remarquons tout de suite que pour tout u dans la famille $\mathcal{F}_2$, et tout x appartenant à S_1, on a $m_u(x) = x^u = 0$. En ordonnant la matrice selon $\mathcal{F}_1$, $\mathcal{F}_2$, S_1 et S_2, la matrice $M_{n,k,d} \in \mathbb{F}_2^D \times \mathbb{F}_2^{N_k}$ qui représente l'application $\psi_{n,k,d}$ a donc la forme représentée à la figure 6.1

L'application linéaire induite par la matrice $A_{n,k,d}$ décrite à la figure 6.1 prend ses entrées dans l'ensemble des fonctions booléennes de degré plus petit que d, et dans lesquelles la variable x_n n'apparaît pas. L'espace d'arrivée de cette application est l'ensemble des tables de vérité restreintes aux entrées x dont le poids de Hamming est exactement k et où la valeur de la coordonnée x_n vaut 0. Donc, comme x_n n'apparaît pas les formes algébriques normales des fonctions en

entrée et que $x_n = 0$, $A_{n,k,d}$ définit donc exactement l'application linéaire allant de l'espace des fonctions booléennes à $n-1$ variables, de degré plus petit que d vers la table de vérité restreinte aux mots de poids k.

De manière similaire, $B_{n,k,d}$ représente l'application linéaire qui envoie les fonctions booléennes de degré au plus $d-1$, vers la table de vérité correspondante restreinte aux mots de poids $k-1$ (car $x_n = 1$ et non 0), et avec nécessairement $n-1$ variables. Ainsi, la matrice $A_{n,k,d}$ représente l'application $\psi_{n-1,k-1,d}$ et $B_{n,k,d}$ représente $\psi_{n-1,k-1,d-1}$, et il en est évidemment de même pour la matrice $M_{n-1,k-1,d}$ décrite à la figure 6.1.

Afin d'établir une relation de récurrence sur le rang de la matrice, nous allons montrer que $\mathsf{rang}(A_{n,k,d}) + \mathsf{rang}(B_{n,k,d}) = \mathsf{rang}(M_{n,k,d})$, c'est-à-dire que la matrice $M_{n-1,k-1,d}$ ne joue pas de rôle dans le rang de $M_{n,k,d}$. En effet, supposons que l'on ait un vecteur v de taille $\binom{n}{k}$ défini par une combinaison linéaire des lignes de notre matrice ($v \in \mathrm{Im}(\psi_{n,k,d})$) et dont les $\binom{n-1}{k}$ dernières coordonnées soient nulles, c'est-à-dire que cette dernière partie du vecteur appartient à $\ker(A_{n,k,d})$, alors ce vecteur v est linéairement dépendant des vecteurs définis par $B_{n,k,d}$ ($v \in \mathrm{Im}(B_{n,k,d})$). En effet, il suffit de voir ce problème en terme de fonctions booléennes : si f est une fonction booléenne appartenant à l'espace vectoriel engendré par $\mathcal{F}_1$ et telle que $\forall x \in S_1$, $f(x) = 0$ (f représente ici un tel vecteur v), alors f (donc v) est dans l'espace engendré par $\mathcal{F}_2$:

$$f(x_1, \ldots, x_n) = x_n g(x_1, \ldots, x_{n-1}) + h(x_1, \ldots, x_{n-1}) \; .$$

f est de degré plus petit que d, donc h est aussi de degré plus petit que d et g est de degré plus petit que $d-1$. Or, pour tout $x \in S_1$ ($x_n = 0$), $f(x) = 0$, donc $h(x_1, \ldots, x_{n-1}) = 0$ sur l'ensemble des mots de poids k, ce qui signifie que f est engendrée par $\mathcal{F}_2$. Finalement, il découle de cette observation que

$$\dim(\mathrm{Im}(\psi_{n,k,d})) = \dim(\mathrm{Im}(\psi_{n-1,k-1,d-1})) + \dim(\mathrm{Im}(\psi_{n-1,k,d})) \; .$$

De plus, si $d \geq k$, alors $\dim(\mathrm{Im}(\psi_{n,k,d})) = \binom{n}{k}$: plus précisément, l'ensemble des monômes de degré exactement k correspond directement à la base canonique des fonctions booléennes définies sur $S_{n,k}$ (en regardant la table de vérité comme l'espace vectoriel $\mathbb{F}_2^{N_k}$). Pour $d \geq n-k$, nous ne prenons pas les monômes, mais nous considérons les fonctions booléennes de la forme

$$f(x) = (1 + x_{i_1})(1 + x_{i_2}) \cdots (1 + x_{i_{n-k}})$$

qui sont de degré plus petit que d, et qui forment elles aussi la base canonique des fonctions booléennes définies sur $S_{n,n-k}$.

Donc, nous avons les résultats suivants :

— si $d \geq k$, alors $\dim(\mathrm{Im}(\psi_{n,k,d})) = \binom{n}{k}$;

— et si $d \geq n-k$, alors $\dim(\mathrm{Im}(\psi_{n,k,d})) = \binom{n}{n-k} = \binom{n}{k}$;

— naturellement, si $d = 0$, on a bien $\dim(\mathrm{Im}(\psi_{n,k,d})) = 1 = \binom{n}{0}$.

Finalement, le résultat se montre par récurrence sur n : pour $n = 1$, la propriété du théorème 6.34 que l'on cherche à montrer est bien vérifiée.

Soit $n \geq 1$, on suppose que pour tout d et k compris entre 0 et n la propriété est vérifiée. Soit $d \leq n+1$ et $k \leq n+1$, alors si $k = n+1$, la dimension est égale à 1 et la propriété reste vraie (la table de vérité est réduite à un seul élément), et si $d = n+1$, alors la dimension vaut $\binom{n}{k}$ comme indiqué précédemment et la propriété reste encore vérifiée. On peut donc supposer maintenant que $d \leq n$ et $k \leq n$, alors

$$\dim(\mathrm{Im}(\psi_{n+1,k,d})) = \dim(\mathrm{Im}(\psi_{n,k-1,d-1})) + \dim(\mathrm{Im}(\psi_{n,k,d}))\ ,$$

donc, par hypothèse de récurrence,

$$\dim(\mathrm{Im}(\psi_{n+1,k,d})) = \binom{n}{\min(d-1, k-1, n-k+1)} + \binom{n}{\min(d, k, n-k)}\ .$$

Deux cas se profilent :

- si $\min(d, k, n-k+1) = d$, alors $\min(d-1, k-1, n-k+1) = d-1$, donc $\dim(\mathrm{Im}(\psi_{n+1,k,d})) = \binom{n}{d-1} + \binom{n}{d} = \binom{n+1}{d}$;
- si $\min(d, k, n-k+1) = k$, alors $\min(d-1, k-1, n-k+1) = k-1$ et $\min(d, k, n-k) = k$ ou $n-k$. Dans tous les cas, $\dim(\mathrm{Im}(\psi_{n+1,k,d})) = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$;
- si $\min(d, k, n-k+1) = n-k+1$, alors $\min(d-1, k-1, n-k+1) = k-1$ ou $n-k+1$ et $\min(d, k, n-k) = n-k$. Dans tous les cas, $\dim(\mathrm{Im}(\psi_{n+1,k,d})) = \binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$.

Tout ceci nous prouve la propriété au rang $n+1$, ce qui conclut la preuve et nous permet d'assurer que

$$\dim(\mathrm{Im}(\psi_{n,k,d})) = \binom{n}{\min(d, k, n-k)}\ .$$

□

En appliquant le théorème 6.34 au corollaire 6.33, nous pouvons donc déduire le résultat suivant :

Corollaire 6.35. *Soit f une fonction booléenne à n variables et soit k un entier tel que $k \leq n/2$, alors l'immunité algébrique restreinte à $S_{n,k}$ est bornée supérieurement par*

$$\min\left\{e;\ 2\binom{n}{e} > \binom{n}{k}\right\}\ .$$

Ce résultat donne une borne supérieure sur l'immunité algébrique restreinte, mais elle est obtenue séparément pour chaque ensemble $S_{n,k}$. Savoir si cette borne est atteignable sur l'ensemble des sous-ensemble de poids fixé reste un problème ouvert : est-il possible d'avoir une fonction booléenne qui est d'immunité

algébrique restreinte optimale sur tous les sous-ensembles $S_{n,k}$, pour tout k allant de 1 à $n-1$?

De plus, pour $r > 0$, on a

$$2\binom{n}{k-r} = \binom{n}{k} \frac{2k(k-1)\cdots(k-r+1)}{(n-k+r)(n-k+r-1)\cdots(n-k+1)} ,$$

si de plus $\frac{k-r+1}{n-k+1} > 2^{-1/r}$, ce qui correspond à

$$k > \frac{2^{-1/r}(n+1)+r-1}{1+2^{-1/r}} = \frac{n+1+(r-1)2^{1/r}}{2^{1/r}+1} ,$$

donc *a fortiori*

$$\frac{k-r+2}{n-k+2} > 2^{-1/r}, \ldots, \frac{k}{n-k+r} > 2^{-1/r} .$$

Dans ce cas l'inégalité

$$2\binom{n}{k-r} > \binom{n}{k}$$

est satisfaite. Lorsque $k = n/2$, la condition $k > \frac{n+1+(r-1)2^{1/r}}{2^{1/r}+1}$ devient $n(2^{1/r}+1) > 2(n+1+(r-1)2^{1/r})$, ou de manière équivalente $n > \frac{2+2(r-1)2^{1/r}}{2^{1/r}-1}$.

Donc, pour le cas particulier où l'on se restreint aux entrées de poids fixé à $k = n/2$, la meilleure immunité algébrique restreinte à $S_{n,n/2}$ possible pour une fonction booléenne à n variables est strictement moins bonne que la meilleure immunité algébrique classique possible.

Pour l'immunité algébrique, c'est une structure particulière du sous-ensemble S considéré qui fait diminuer le rang de la matrice $M_{e,S}$, et qui induit une augmentation de la meilleure immunité algébrique possible restreinte à S. En effet, si l'on choisit S de manière uniforme et aléatoire, on peut s'attendre à un comportement suffisamment "aléatoire" de la matrice, de sorte que celle-ci soit de rang plein, ce qui diminuerait encore plus la borne sur l'immunité algébrique restreinte. Par ailleurs, si l'on suppose une structure encore plus forte sur le sous-ensemble S, alors certaines propriétés fortement indésirables pourraient apparaître.

Prenons un exemple extrême : on suppose que S est un hyperplan particulier de $\mathbb{F}_2^n$: $S = \{x \in \mathbb{F}_2^n | x_n = 0\}$, alors dans ce cas la matrice $M_{e,S}$ est égale à une matrice génératrice du code de Reed et Muller $\mathcal{RM}(e, n-1)$, et dans ce cas, on a $\mathsf{rang}(M_{e,S}) = \sum_{i=0}^{e} \binom{n-1}{i}$, valeur qui est plus petite que $\sum_{i=0}^{e} \binom{n}{e}$ et que l'on pourrait attendre pour un S pris aléatoirement et de cardinalité 2^{n-1} pour des petites valeurs de e. Dans ces conditions, l'immunité algébrique est optimale au regard de la cardinalité de S, puisqu'elle atteint la borne pour une fonction à $n-1$ variables.

Identification du noyau. A l'aide du théorème 6.34, nous avons accès à la dimension de l'image de l'application $\psi_{n,k,d}$, ainsi qu'à la dimension de son noyau. Nous pouvons décrire ce noyau, ce qui permet de comprendre quels types de fonctions booléennes il faut éviter dans notre contexte de poids de Hamming fixé en entrée.

Proposition 6.36. *Soit k, r et n tels que $k \leq \frac{n}{2}$ et soit $0 \leq i_1 < i_2 < \cdots < i_r \leq n$, alors la fonction booléenne définie par*

$$\begin{cases} x_{i_1}x_{i_2}\cdots x_{i_r}\left(\sum_{j\neq i_1,i_2,\cdots,i_r} x_j\right) & \text{si } k-r \equiv 0 \mod 2 \\ x_{i_1}x_{i_2}\cdots x_{i_r}\left(1+\sum_{j\neq i_1,i_2,\cdots,i_r} x_j\right) & \text{si } k-r \equiv 1 \mod 2 \end{cases}$$

est nulle sur les sous-ensembles $S_{n,k}$ de $\mathbb{F}_2^n$.

Plus généralement, pour tout $j < k$ et s, et pour tout u de poids de Hamming j, la fonction booléenne définie par :

$$\begin{cases} x^u \times \left(\sum_{\{i_1,\ldots,i_{s-j}\}\cap \mathsf{supp}(u)=\emptyset} \prod_{\ell=1}^{s-j} x_{i_\ell}\right) & \text{si } \binom{k-j}{s-j} \equiv 0 \mod 2 \\ x^u \times \left(1+\sum_{\{i_1,\ldots,i_{s-j}\}\cap \mathsf{supp}(u)=\emptyset} \prod_{\ell=1}^{s-j} x_{i_\ell}\right) & \text{si } \binom{k-j}{s-j} \equiv 1 \mod 2 \end{cases}$$

est nulle sur $E_{n,k}$.

Démonstration. En réordonnant les variables, nous pouvons supposer sans perdre de généralité que $x^u = x_1x_2\ldots x_j$. On note f la fonction définie par la proposition 6.36. Soit x de poids de Hamming égal à k, alors

— si $x^u = 0$, on a $f(x) = 0$,

— si $x^u = 1$, on a

$$f(x) = \sum_{\{i_1,\ldots,i_{s-j}\}\cap supp(u)=\emptyset} \prod_{\ell=1}^{s-j} x_{i_\ell} + \binom{k-j}{s-j} \mod 2\,.$$

Or, comme $x^u = 1$, nécessairement $x_1 = x_2 = \cdots = x_j = 1$. Comme $\mathsf{w_H}(x) = k$, le poids de Hamming du vecteur $(x_{j+1}, x_{j+2}, \cdots, x_n)$ est fixé à la valeur $k-j$. La fonction booléenne définie par

$$\sum_{\{i_1,\ldots,i_{s-j}\}\cap supp(u)=\emptyset} \prod_{\ell=1}^{s-j} x_{i_\ell}$$

étant une fonction symétrique élémentaire à $n-j$ variables de degré $s-j$, elle est donc constante lorsque le poids de Hamming de $(x_{j+1},\ldots,x_n)$ est fixe, et sa valeur est $\binom{k-j}{s-j} \mod 2$. Finalement, $f(x) = 0$ si x est de poids de Hamming égal à k.

□

De plus, nous pouvons décrire l'ensemble $\mathrm{Im}\,(M_{n,k,d})$:

Corollaire 6.37. *Si $d \geq k$, alors l'ensemble des monômes de degré égal à k forme une base de* $\mathrm{Im}\,(M_{n,k,d}) = \mathbb{F}_2^{\binom{n}{k}}$.

Corollaire 6.38. *Si en revanche $d \geq n-k$, alors l'ensemble des fonctions booléennes de la forme $(1+x_{i_1})(1+x_{i_2})\cdots(1+x_{i_{n-k}})$ forme une base de* $\mathrm{Im}\,(M_{n,k,d}) = \mathbb{F}_2^{\binom{n}{k}}$.

Démonstration. Voir la fin de la preuve du théorème 6.34. □

Comme nous l'avons déjà dit précédemment, Dumer et Kapralova ont étudié le code de Reed et Muller poinçonné sur les mots de poids de Hamming fixé d'ordre r, et ont publié leurs résultats dans [DK17], article dans lequel on retrouve notre résultat du théorème 6.34 prouvé de manière similaire. Nos travaux sont indépendants. La pré-publication comme rapport sur IACR eprint [CMR17b] est antérieure à [DK17]. Mais, à la suite d'une discussion personnelle avec Dumer, nous avons appris *a posteriori* que ce résultat était déjà mentionné dans la thèse de Kapralova de 2013. Finalement, quelques mois plus tard, nous nous sommes aussi rendus compte que ce résultat était connu depuis 1966 [Got66], et mentionné dans [Wil90] mais dans un tout autre contexte que celui des fonctions booléennes.

6.4.3 Somme directe et immunité algébrique restreinte

L'immunité algébrique classique d'une fonction booléenne définie comme la somme directe de deux fonctions f et g est toujours supérieure à l'immunité algébrique de f et à celle de g. Il n'en est rien lorsque l'on fixe le poids : ajouter une fonction booléenne en somme directe peut faire diminuer l'immunité algébrique.

La motivation initiale de notre travail étant le chiffrement à flot FLIP où la fonction de filtrage est définie par des sommes de monômes dont les variables sont indépendantes, il est donc utile d'étudier l'immunité algébrique restreinte pour ces fonctions particulières dans le but de prouver une certaine résistance de FLIP aux attaques algébriques.

Lemme 6.39 (Somme directe et AI_k)**.** *Soit h la fonction booléenne définie par la somme directe de la fonction f et g, deux fonctions booléennes à respectivement n et m variables, alors pour tout $k \leq \min(n,m)$, on a*

$$\mathsf{AI}_k(h) \geq \min_{0\leq\ell\leq k}(\max[\mathsf{AI}_\ell(f), \mathsf{AI}_{k-\ell}(g)])\ .$$

Démonstration. Soit a un annulateur non-nul de h sur $S_{n+m,k}$. Alors nous allons montrer que a est un annulateur non-nul de f ou de $f+1$ sur $S_{n,\ell}$ et de g ou $g+1$ sur $S_{m,k-\ell}$ pour un ℓ arbitraire.

Pour tout $(x,y) \in S_{n+m,k}$, $a(x,y)h(x,y) = 0$ (où $x \in \mathbb{F}_2^n$ et $y \in \mathbb{F}_2^m$). Comme a est non-nul sur $S_{n+m,k}$, il existe $(x_0,y_0) \in S_{n+m,k}$ tel que $a(x_0,y_0) = 1$. Soit

$\ell = \mathsf{w_H}(x_0)$, alors $\mathsf{w_H}(y_0) = k - \ell$. Dans ces conditions la fonction booléenne à m variables définie par $a'(y) = a(x_0, y)$ est un annulateur non-nul de g ou de $1 + g$ (selon la valeur de $f(x_0)$) par construction et est de degré algébrique inférieur ou égal à celui de a restreint à $S_{n+m,k}$ (car on a simplement restreint la table de vérité). De la même manière en fixant y à la valeur y_0, on construit un annulateur non-nul de f ou de $f + 1$ (selon la valeur de $g(y_0)$) et de degré algébrique inférieur ou égal à celui de a sur $S_{n+m,k}$.

En faisant la même chose pour un annulateur de $h + 1$ sur $S_{n+m,k}$, on obtient que $\deg_k(a) \geq \deg(a') \geq \mathsf{AI}_{k-\ell}(g)$ et $\deg_k(a) \geq \mathsf{AI}_\ell(f)$, ce qui conclut la preuve. □

Théorème 6.40 (Lien entre AI, AI_k et somme directe)**.** *Soit h la somme directe de f et g, deux fonctions booléennes à respectivement n et m variables. Soit k tel que $n \leq k \leq m$. Alors*

$$\mathsf{AI}_k(h) \geq \mathsf{AI}(f) - \deg(g) \; .$$

Démonstration. Soit $ann(x, y)$ un annulateur non nul de h sur $S_{n+m,k}$. Soit $(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$, tel que $\mathsf{w_H}(a, b) = k$ et tel que $ann(a, b) = 1$. Comme (a, b) est de poids de Hamming k, nous pouvons, sans perdre de généralité quitte à réordonner les variables, supposer que pour tout $j = 1, \ldots, n$, on a la propriété $b_j = a_j + 1$, et que pour tout j allant de $n + 1$ à k, on a $b_j = 1$ (et donc $b_j = 0$ pour $j \geq k + 1$).

On définit de plus l'application linéaire suivante :

$$\begin{array}{rccc} L & : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^m \\ & & (x_1, \ldots, x_n) & \longmapsto & (x_1 + 1, x_2 + 1, \ldots, x_n + 1, 1, \ldots, 1, 0, \ldots, 0) \end{array}$$

où la partie $1, \ldots, 1$ est de taille $k - n$. On a $L(a) = b$, ainsi, la fonction booléenne à n variables $A(x) = ann(x, L(x))$ est non-nulle et annule la fonction booléenne $f(x) + g \circ L(x)$ sur $\mathbb{F}_2^n$.

Si $g(b) = 0$, alors la fonction $ann(x, L(x))(g'(x) + 1)$ où $g'(x) = g \circ L(x)$ est un annulateur non-nul de f. En effet,

$$ann(a, L(a))(g'(a) + 1) = ann(a, b)(g(b) + 1) = 1(0 + 1) = 1$$

et pour tout $x \in \mathbb{F}_2^n$, on a

$$\begin{aligned} A(x)(g'(x) + 1)f(x) &= A(x)(g'(x) + 1)(f(x) + g'(x) + g'(x)) \\ &= (g'(x) + 1)\underbrace{A(x)(f(x) + g'(x))}_{0} + A(x)\underbrace{(g'(x) + 1)g'(x)}_{0} \\ &= 0 \; . \end{aligned}$$

La fonction $ann(x, L(x))(g'(x) + 1)$ est de degré au plus $\deg(h) + \deg(g)$, donc $\deg(h) + \deg(g) \geq \mathsf{AI}(f)$. Si en revanche $g(b) = 1$, alors en appliquant le même raisonnement à la fonction $f + 1$ au lieu de f et $g + 1$ au lieu de g, alors on a $\deg(h) + \deg(g) \geq \mathsf{AI}(f)$.

Finalement, si $h(x, y)$ est un annulateur non-nul de $f(x)+g(x)+1$ sur $S_{n+m,k}$, alors nous avons les mêmes conclusions en remplaçant f par $f+1$ ou g par $g+1$, ce qui complète et termine la preuve. □

Cette borne prouve en particulier que, si $k \geq n$, alors réaliser la somme directe avec une fonction constante (g identiquement nulle) permet de ne pas diminuer l'immunité algébrique à poids fixé par rapport à l'immunité algébrique classique de la fonction initiale. Ceci peut être très utile pour les concepteur.rice.s de chiffrements, puisque cela permet de prouver la résistance aux attaques algébriques en utilisant les résultats connus dans le contexte classique.

Cependant, alors que ce théorème permet de prouver des bornes sur l'immunité algébrique à poids fixé, il montre aussi que réaliser la somme directe de 2 fonctions peut faire décroître celle-ci, ce qui semble contre-intuitif lorsque l'on réalise le parallèle avec le cas classique (chapitre 2). Les deux exemples suivants illustrent parfaitement à quel point la somme directe peut faire diminuer l'immunité algébrique :

Exemple 11. Soit f une fonction booléenne à trois variables définie par

$$f(x_1, x_2, x_3) = x_1 + x_2 x_3$$

et g à 7 variables définie par

$$g(x_4, \ldots, x_{10}) = \sum_{i=4}^{10} x_i \ .$$

Alors $f(x_1, x_2, x_3)+g(x_4, \ldots, x_{10})$ est d'immunité algébrique restreinte aux mots de poids $k = 5$ égale à 1, alors que f est d'immunité algébrique 2 et que g est de degré 1. En effet, on remarque que

$$x_2(f(x_1, x_2, x_3) + g(x_4, \ldots, x_{10})) = x_2 \left(1 + \sum_{i=1}^{10}\right). \tag{6.2}$$

Le membre de droite de l'équation (6.2) correspond bien à une fonction booléenne définie à la proposition 6.36 et est donc nulle sur $S_{10,5}$.

Exemple 12. Plus généralement, prenons l'exemple de la fonction majorité à n variables, n impair, qui, rappelons le, est d'immunité algébrique optimale ($\frac{n+1}{2}$). La fonction à $2n$ variables définie par

$$h(x, y) = 1 + \mathsf{MAJ}_(x) + \mathsf{MAJ}_(y)$$

est nulle à poids de Hamming fixé à n. En effet, pour $\mathsf{w_H}(x) + \mathsf{w_H}(y) = n$, ou bien $\mathsf{w_H}(x) \leq \frac{n-1}{2}$ et alors $\mathsf{w_H}(y) \geq \frac{n+1}{2}$, ou bien $\mathsf{w_H}(x) \geq \frac{n+1}{2}$ et $\mathsf{w_H}(y) \leq \frac{n-1}{2}$. Dans tous les cas, la fonction h à $2n$ variables est nulle sur les mots de poids n.

6.5 Conclusion

6.5.1 Sur la sécurité de FLIP

L'application cryptographique directe de notre travail était, à l'origine, l'étude correcte et approfondie de la sécurité du chiffrement FLIP. Étant donné que la fonction de filtrage utilisée dans FLIP est décrite relativement simplement à l'aide de peu de monômes en somme directe, il nous a été possible de trouver des bornes inférieures sur les différents paramètres cryptographiques d'immunité algébrique, de non-linéarité, ainsi que de calculer le biais dans la sortie de FLIP. Ces résultats sont donnés dans les tables 6.1 et 6.2 et expliqués plus en détail dans notre article publié aux *IACR Transactions on Symmetric Cryptology* [CMR17a].

Table 6.1 – Résistance de FLIP aux attaques par corrélation, N désigne le nombre de variables, v_0 l'intervalle des poids k pour lesquels la borne inférieure sur la non-linéarité implique qu'une attaque par corrélation ne peut se faire en moins de 2^{80} (respectivement 2^{128}) en utilisant les techniques connues.

Instance	N	v_0	Sécurité
FLIP530	530	$[107, 464]$	80 bits
FLIP662	662	$[136, 556]$	80 bits
FLIP1394	1394	$[221, 1239]$	128 bits
FLIP1704	1704	$[266, 1492]$	128 bits

Table 6.2 – Bornes inférieures sur AI_k des instances de FLIP pour $k = \frac{N}{2}$, N désigne le nombre de variables, $\mathsf{AI}(f)$ est l'immunité algébrique classique de la fonction de filtrage f, n est le nombre de variables de f, et la borne sur l'immunité algébrique est obtenue en appliquant le théorème 6.40.

Instance	N	$\mathsf{AI}(f)$	Borne
FLIP530	530	9	4
FLIP662	662	15	6
FLIP1394	1394	16	6
FLIP1704	1704	23	8

Notre travail n'a pas mis en évidence d'attaque concrète sur le chiffrement FLIP. Ainsi, le biais en sortie est extrêmement faible ; et l'immunité algébrique et la non-linéarité à poids fixé restent suffisamment élevées, sous réserve de prendre une clef secrète ayant un poids de Hamming s'éloignant peu de $n/2$. Alors que le caractère "équilibré" de la clef secrète imposé par les concepteurs de FLIP n'était pas motivé par un mauvais comportement du chiffrement, nous avons pu

expliquer en quoi ce caractère est en réalité primordial. Finalement, nous avons pu déterminer avec précision quels sont les poids de Hamming "acceptables" pour une clef utilisée dans FLIP, qui sont les poids pour lesquels les critères exacts d'équilibre, de non-linéarité, ainsi que d'immunité algébrique assurent une bonne résistance à un grand panel d'attaques connues.

6.5.2 Ce qu'il faut retenir

Tout d'abord, il n'était pas évident, à première vue, de se rendre compte que l'analyse de sécurité initiale du chiffrement FLIP n'était pas pertinente. Ce travail met donc en garde tout concepteur.rice de chiffrement de réaliser des raccourcis trop rapides entre la sécurité du système et les critères classiques sur les fonctions booléennes, les boîtes-S, ou encore d'autres composantes cryptographiques. En effet, tous les critères cryptographiques que nous utilisons viennent dans un premier temps d'attaques concrètes sur des chiffrements particuliers. Dans ces conditions, lorsque l'on conçoit des chiffrements ayant des propriétés non-conventionnelles, il faut sans cesse se demander si les critères que l'on prend en considération sont pertinents, ce qui est a priori chose ardue.

Ce travail nous permet aussi de montrer l'importance, dans la recherche en cryptographie, de la définition de nouveaux critères pertinents pour évaluer la sécurité comme nous l'avons vu aussi au chapitre 4. En effet, au travers de nouvelles attaques, ainsi que de nouvelles observations, il est important de déterminer des critères fondamentaux sur les objets que l'on considère, toujours dans le but d'affiner l'évaluation de sécurité de nos chiffrements.

Par ailleurs, il s'avère que le contexte des restrictions aux entrées à poids fixé a rendu bien plus ardu les preuves sur les critères que l'on considère classiquement. Les sous-ensembles $S_{n,k}$ ne sont pas des sous-espaces vectoriels et ne sont que peu structurés, ce qui, lorsque l'on s'intéresse à la distance aux fonctions linéaires, rend naturellement tout plus compliqué.

Finalement, il est important de se rappeler qu'une étude approfondie doit être faite dès que les sous-ensembles considérés changent. En effet, un grand nombre de nos résultats sont spécifiques au cas du poids de Hamming fixé, ce qui signifie que si le contexte change un tant soit peu, alors de nouvelles études approfondies doivent être menées.

6.5.3 Questions ouvertes

Un grand nombre des bornes (supérieures et inférieures) que nous avons données dans notre travail semblent pouvoir être améliorées, afin de trouver des objets optimaux au regard de ces nouveaux critères et d'aider à la conception de futurs chiffrements.

La définition de nouveaux critères sur les fonctions booléennes ouvre à mon avis la porte à un grand nombre de questions. Principalement, la borne sur la non-linéarité restreinte est obtenue séparément sur chacun des sous-ensembles considérés, alors que la fonction utilisée est la même ; il est donc légitime de se

demander si cette borne ne pourrait pas être améliorée en considérant tous les sous-ensembles, vus comme partition de tout l'espace $\mathbb{F}_2^n$.

D'un point de vue bien plus large que celui de FLIP, nous avons mis en évidence certaines bornes sur les critères restreints à des sous-ensembles, ce qui permet d'imaginer des avancées dans ce sens, notamment en combinant ces observations avec les fonctions augmentées. Cette perspective mérite d'être plus investiguée, et nous en parlerons plus en détail en prenant du recul dans les perspectives, à la fin de ce document.

Chapitre 7

Cryptanalyse du PRG de Goldreich

Le générateur pseudo-aléatoire de Goldreich [Gol00] est une construction théorique qui possède une structure sensiblement similaire à celle de FLIP, mais dans un contexte largement différent. Cette construction, initialement destinée à décrire des fonctions à sens unique [1], définit une famille de générateurs pseudo-aléatoires, dont chaque bit de sortie ne dépend que d'un très petit nombre de bits de la source d'aléa utilisée en entrée (la graine).

Ce chapitre est consacré à l'analyse du PRG de Goldreich : nous appliquons la technique utilisée dans la cryptanalyse de FLIP, sur un objet utilisé en cryptographie asymétrique. Il décrit un travail réalisé en collaboration avec Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux et Mélissa Rossi [CDM+18]. L'intérêt principal de cette étude est, à mon sens le suivant : les cryptosystèmes utilisés en cryptographie asymétrique reposent sur des problèmes algorithmiques supposés difficiles, et sont donc "prouvés" sûrs, dans le sens où il ne peut y avoir d'attaques en temps polynomial en la taille des paramètres. Or, nous oublions, à mon avis, un peu trop souvent que la cryptographie est faite pour être utilisée, c'est-à-dire qu'à un certain moment, les systèmes sont voués à être implémentés. Dans ces conditions, les paramètres seront fixés et la notion de complexité asymptotique n'a plus de raison d'être. Il convient donc d'analyser plus en détail la sécurité des systèmes asymétriques, au regard des paramètres choisis. De la même manière, nous pourrions aussi avoir des algorithmes en temps polynomial qui cassent un cryptosystème, mais dont la complexité reste trop grande pour que l'attaque soit réalisable pour les paramètres utilisés. Les complexités sont en effet souvent des complexités asymptotiques, c'est-à-dire qu'on ne s'intéresse qu'au terme dominant, or il se pourrait aussi que pour des paramètres donnés, les autres termes ne soient pas du tout négligeables : il est difficile de savoir où commence l'asymptotique.

1. One-way functions

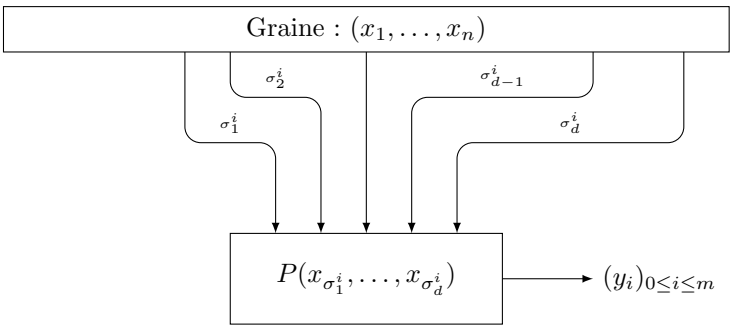

Figure 7.1 – Schéma de la construction du PRG de Goldreich.

Nous verrons ici en quoi les paramètres à utiliser pour le PRG de Goldreich doivent être pris suffisamment grands pour assurer une sécurité raisonnable, au regard de notre attaque dont la complexité est sous-exponentielle. Nous l'avons aussi implémentée en pratique et sa complexité est meilleure que celle attendue en théorie, et nous raffinons les critères d'(im)possibilités déjà connus de constructions de PRG ayant une faible localité.

7.1 Contexte et travaux précédents

7.1.1 Le PRG de Goldreich

La notion de fonction à sens unique est fondamentale en cryptographie asymétrique, étant donné qu'un grand nombre de preuves de sécurité se basent sur l'existence de telles fonctions, pour lesquelles il est facile de calculer $f(x)$, mais difficile de calculer $f^{-1}(x)$. Ainsi, il y a presque deux décennies de cela, Goldreich proposa une description simplifiée au maximum d'une fonction à sens unique, calculable efficacement, et extrêmement difficile à inverser, pouvant être généralisée en un générateur pseudo-aléatoire.

Le PRG de Goldreich [Gol00] est décrit comme suit. Soit n et m deux entiers, et $\sigma^1, \sigma^2, \ldots, \sigma^m$ une liste de m d-uplets de $\{1, 2, \ldots, n\}$, où d est une constante [2], appelée la *localité*. Maintenant, nous choisissons un prédicat $P : \{0,1\}^d \to \{0,1\}$, *i.e.* une fonction booléenne à d variables. Alors le PRG $f : \{0,1\}^n \to \{0,1\}^m$ de Goldreich est défini par

$$f(x) = P(x[\sigma^1])||P(x[\sigma^2])|| \cdots ||P(x[\sigma^m]) \ .$$

Dit autrement, f est calculée en appliquant le prédicat m fois à tous les bits de x dont les indices correspondent aux d-uplets $\sigma^1, \ldots, \sigma^m$. La construction est représentée à la figure 7.1.

2. d peut être logarithmique en la taille de n, mais ce qui est sûr, c'est que $d(n) \ll n$.

L'intérêt de la construction vient du fait que si la localité est vraiment petite (entre 5 et 10), alors calculer la fonction devient facile. Initialement prévue pour être une fonction à sens unique, cette construction a ensuite été étendue pour définir un PRG, c'est-à-dire que $m \geq n$, et la sortie est censée être difficile à distinguer d'une suite aléatoire. En particulier il est censé être difficile de retrouver la graine du PRG. Un des paramètres importants du PRG est l'*expansion* noté s et définie par la relation

$$m = n^{\mathsf{s}} .$$

7.1.2 Résultats précédents sur les PRG

Une des questions posées en cryptographie est l'existence de PRG, appartenant à la classe de complexité NC^0 (profondeur constante, et taille du circuit de la fonction polynomiale en la taille de l'entrée). Sans rentrer dans les détails, toujours dans un but de réduire au maximum le coût de description de la fonction, certain.e.s auteur.e.s se demandent quel est le nombre minimal de variables en entrée dont peut dépendre chaque bit de sortie de la fonction. Des résultats positifs et négatifs sur cette quantité, appelée la *localité* existent, dépendant du nombre de bits de sortie du PRG [CM01, AIK04, AIK08]. Il a été prouvé qu'il est impossible d'avoir une localité inférieure ou égale à 4, sous certaines conditions sur la taille de la sortie.

Pour ce qui est de leur utilisation, on retrouve les PRG ayant une petite localité dans des constructions de MPC [3] [GGM86], qui ont aussi motivé des constructions de chiffrement symétriques [ARS+15, MJSC16, CCF+16] dans lesquelles, comme pour le FHE, le nombre de portes AND ainsi que la profondeur du circuit sont des paramètres qui jouent un rôle essentiel.

7.1.3 Résultats précédents sur le PRG de Goldreich

7.1.3.1 Résultats positifs

Tout d'abord, le PRG de Goldreich est considéré résistant à un certain nombre d'attaques génériques basées sur des algorithmes de "retour en arrière" [4], formalisées dans [AHI04, CEMT14], à condition que le graphe induit par les sous-ensembles d'indices choisis ait certaines propriétés, que nous ne détaillerons pas ici. Pour ce qui est de la possibilité de distinguer la sortie du PRG d'une source d'aléa, il a été montré dans [OW14] que l'instance de localité 5 décrite précédemment résiste à tous les tests linéaires, *i.e.* on ne peut distinguer la sortie d'une suite aléatoire par simple combinaison linéaire des bits de sortie avec grande probabilité, tant que le nombre de bits de sortie est de l'ordre de $n^{1.5-\varepsilon}$, ε étant une constante non-nulle arbitrairement choisie.

3. Multi-party computation
4. Backtracking attacks

7.1.3.2 Résultats négatifs

Alors que l'attaque de Siegenthaler [Sie85] ne s'applique clairement pas ici, les résultats d'Applebaum et de O'Donnell et Witmer [OW14, App15] ont mis en évidence que le critère de *résilience* de la fonction de filtrage joue un rôle fondamental dans la sécurité de ce type de PRG. Cette observation est assez amusante, puisque c'est un critère connu depuis longtemps en cryptographie symétrique, il est donc redécouvert dans ce contexte, mais motivé par une attaque différente. Plus précisément, il existe une attaque sur le PRG de Goldreich dès que le prédicat n'est pas 2-résilient. De plus, il est bien connu que $\mathsf{res}(P) + \deg(P) \leq d - 1$, où d est la localité, *i.e.* le nombre de variables du prédicat. Comme $\deg(P)$ doit nécessairement être plus grand que 2, sinon il existe une inversion triviale, alors il est nécessaire que la localité soit au moins égale à 5.

Dans [BQ09], il est aussi mentionné une attaque sous-exponentielle, qui fonctionne à partir du moment où la taille de la sortie est polynomiale en n, *i.e.* $m = n^{\mathsf{s}}$, avec $\mathsf{s} > 1$, avec une complexité de l'ordre de

$$2^{\mathcal{O}(n^{1-(\mathsf{s}-1)/2d})} ,$$

où d est la localité du prédicat et s est l'*expansion* du générateur, avec $\mathsf{s} > 1$. En effet, le but d'un PRG est de transformer une source d'aléa de taille n en une suite de taille plus grande, indistinguable d'une suite aléatoire.

Les attaques algébriques [CM03] ont été décrites pour la première fois en 2003, et nécessitent de prendre en compte non pas le degré du prédicat mais son *immunité algébrique*. Les attaques fonctionnent évidemment de la même manière sur le PRG de Goldreich, impliquant que les deux critères à prendre en compte sont la *résilience* et l'*immunité algébrique*. Ce qui est relativement étonnant, c'est que ce n'est qu'en 2016 que Applebaum et Lovett [AL16] ont découvert les attaques algébriques, ce qui donna le résultat (relativement évident) que la taille de la sortie du PRG ne doit pas être plus grande que n^e, où e est l'immunité algébrique de la fonction de filtrage, sinon une simple attaque algébrique permet de retrouver la graine. Dans le papier d'Applebaum et Lovett à *STOC 2016*, l'immunité algébrique est appelée degré rationnel, mais cette notion est exactement la même que l'immunité algébrique introduite dans [CM03].

En 2003, dans [MST03], une instance concrète a été proposée, de localité $d = 5$, et utilisant le prédicat

$$P_5(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 x_5 .$$

Selon les résultats négatifs mentionnés précédemment, ce prédicat est le meilleur possible en terme de localité et il n'est pas considéré comme cassé par la communauté. Notre travail consiste donc à analyser en détail la sécurité du PRG de Goldreich, construit avec ce prédicat particulier.

7.2 Définitions

7.2.1 Prédicats

Nous définissons deux types de prédicats, largement considérés dans la construction de Goldreich, notamment dans [AL16].

Définition 7.1 ($\mathsf{XOR}_\ell\mathsf{M}_k$ prédicats)**.** *Nous appelons* $\mathsf{XOR}_\ell\mathsf{M}_k$ *les prédicats* P *à* $\ell + k$ *variables de la forme suivante*

$$P(x_1, \ldots, x_\ell, z_1, \ldots, z_k) = \sum_{i=1}^{\ell} x_i + M(z_1, \ldots, z_k) \ ,$$

où M *peut être n'importe quelle fonction booléenne (prédicat) à* k *variables.*

Définition 7.2 ($\mathsf{XOR}_\ell\mathsf{MAJ}_k$ prédicats)**.** *Nous appelons* $\mathsf{XOR}_\ell\mathsf{MAJ}_k$ *le prédicat* P *à* $\ell + k$ *variables de la forme suivante*

$$P(x_1, \ldots, x_\ell, z_1, \ldots, z_k) = \sum_{i=1}^{\ell} x_i + \mathsf{MAJ}(z_1, \ldots, z_k) \ ,$$

où MAJ *est la fonction majorité, i.e.*

$$\mathsf{MAJ}(z_1, \ldots, z_k) = 1 \Leftrightarrow \mathsf{w}_H(z_1, \ldots, z_k) \geq \left\lceil \frac{k}{2} \right\rceil \ ,$$

où $\mathsf{w_H}$ *désigne le poids de Hamming.*

7.2.2 Le PRG considéré

Nous reprenons la construction décrite ci-dessus. Nous avons donc m d-uplets de $\{1, \ldots, n\}$, de taille d notés $\sigma^1, \ldots, \sigma^m$. Pour tout $1 \leq i \leq m$, on a $\sigma^i = (\sigma^i_1, \sigma^i_2, \ldots, \sigma^i_d)$. Alors, la sortie du générateur pseudo-aléatoire que l'on considère est donnée par

$$(P(x_{\sigma^1_1}, \ldots, x_{\sigma^1_d}), \ldots, P(x_{\sigma^m_1}, \ldots, x_{\sigma^m_d})) \ .$$

Les d-uplets σ^i sont choisis aléatoirement et de manière uniforme. Dans cette partie nous considérons le prédicat P_5 défini par la forme algébrique normale suivante.

$$P_5(x_1, x_2, x_3, x_4, x_5) = x_1 + x_2 + x_3 + x_4 x_5$$

Cette fonction booléenne est de degré 2, d'immunité algébrique 2 et est 2-résiliente. Nous notons $(y_i)_{1 \leq i \leq m}$ la suite en sortie du PRG. Alors, nous nous intéressons à résoudre le système d'équations

$$\forall 1 \leq i \leq m, \ (E_i) \ \ x_{\sigma^i_1} + x_{\sigma^i_2} + x_{\sigma^i_3} + x_{\sigma^i_4} x_{\sigma^i_5} = y_i \ .$$

Inversion matricielle. Notre attaque n'est pas une attaque par distingueur, nous cherchons à résoudre le système, afin de retrouver la graine, *i.e.* la valeur de $x_1, \ldots, x_n$. Comme la complexité de la résolution des équations linéaires joue un rôle fondamental dans notre attaque, nous considérerons la complexité du meilleur algorithme actuel [Mac16] qui est $\mathcal{O}(n^\omega)$, où $\omega = \log_2(7)$.

7.3 Attaque "supposer et déterminer" sur P_5

7.3.1 Lien avec FLIP

Comme le.a lecteur.rice a pu le constater, la construction du PRG de Goldreich possède exactement la même structure que le chiffrement symétrique FLIP décrit au chapitre 5. En effet, il suffit de considérer un PRG construit comme précédemment, mais avec un prédicat prenant en entrée toutes les variables, *i.e.* avec une localité $d = n$, alors une telle construction suit exactement la construction de FLIP.

Donc, il est tout naturel d'appliquer une technique de type "supposer et déterminer", technique largement utilisée en cryptographie symétrique [HR00, EJ00], et que nous avons déjà exploitée avec succès sur le chiffrement FLIP avec Virginie Lallemand et Sébastien Duval.

Dans le cas de FLIP, nous étions contraints par le coût de résolution des équations algébriques et non par le nombre d'équations. En effet, au regard des paramètres et du degré des monômes, nous avions accès à bien plus de bits de sortie que de monômes, ce qui donnait dans ce cas un algorithme de résolution simple avec une complexité de l'ordre de $\mathcal{O}(n^{\omega d_f})$ où d_f était le degré de la fonction. Cependant, cette expression de la complexité n'a pas beaucoup de sens dans le cas de FLIP dont les paramètres sont fixés. Nous faisons ici cette remarque simplement pour la mettre en parallèle avec le travail sur le PRG de Goldreich. Pour les tailles des paramètres de FLIP, cette complexité dépasse largement les 2^{80} (respectivement 2^{128}) opérations, c'est la raison pour laquelle nous avons dû faire des hypothèses sur des bits de la clef pour justement faire diminuer le degré des équations, afin que le coût de résolution diminue.

Ici, nous sommes face à un tout autre problème : tout d'abord, les tailles de graine ne sont pas définies, et on s'intéresse alors à la complexité asymptotique. De plus, les équations sont certes de degré 2, mais le nombre de bits en sortie du PRG est largement plus faible que le nombre de monômes, impliquant que linéariser ne suffit pas et qu'il faut trouver plus d'équations. De manière générale, ce contexte est particulier, car il repose fondamentalement sur le fait que l'attaquant.e n'a pas assez d'information pour résoudre le système.

7.3.2 Fonctions booléennes à entrées restreintes

Tout comme dans FLIP, l'entrée de la fonction est de poids de Hamming fixé (celui de la graine utilisée). Cependant, au vu de la très faible localité, aucun biais n'est "détectable" au vu de la quantité d'équations, même s'il est clair que

la fonction ne peut pas être parfaitement équilibrée à poids fixé. De la même manière, le critère d'immunité algébrique ne change pas et reste le même puisque la fonction ne prend que très peu de variables en entrée. En effet, les fonctions à considérer et les fonctions exhibées au chapitre précédent pour lesquelles les critères sont dégénérés doivent prendre un grand nombre de variables. En effet, les fonctions qui s'annulent à poids fixé sont dérivées de fonctions symétriques, et dépendent donc naturellement d'un grand nombre de variables. Ainsi, tant que la localité reste faible il n'est pas utile de considérer les restrictions de la fonction aux entrées de poids constant. comme nous l'avons fait pour FLIP.

7.3.3 Observations

En suivant la construction du PRG de Goldreich, nos équations sont de la forme :

$$(E_i)\ \ x_{\sigma_1^i} + x_{\sigma_2^i} + x_{\sigma_3^i} + x_{\sigma_4^i} x_{\sigma_5^i} = y_i\ .$$

— Si dans nos équations quadratiques, ou $x_{\sigma_4^i}$ ou $x_{\sigma_5^i}$ sont connus, alors, peu importe leur valeur, l'équation correspondante devient linéaire en les bits de la graine x : si $x_{\sigma_4^i} = 1$, alors on a l'équation linéaire $x_{\sigma_1^i}+x_{\sigma_2^i}+x_{\sigma_3^i}+x_{\sigma_5^i} = y_i$ et si $x_{\sigma_4^i} = 0$, alors on a l'équation linéaire $x_{\sigma_1^i} + x_{\sigma_2^i} + x_{\sigma_3^i} = y_i$.

— Si deux équations partagent le même monôme quadratique, *i.e.* si $\sigma_4^i = \sigma_4^j$ et $\sigma_5^i = \sigma_5^j$ ou si $\sigma_4^i = \sigma_5^j$ et $\sigma_5^i = \sigma_4^j$, alors la somme de ces deux équations est une équation linéaire.

Le deuxième phénomène est appelé une collision, au sens de la définition suivante.

Définition 7.3 (Collision)**.** *Une collision est un couple* $(i, j) \in \{1, \ldots, m\}^2$*,* $i \neq j$ *tel que* $\{\sigma_4^i, \sigma_5^i\} = \{\sigma_4^j, \sigma_5^j\}$.

En utilisant ces deux observations, nous pouvons décrire le principe général de l'attaque.

7.3.4 Description de l'attaque

Étape 1 Trouver toutes les collisions entre les monômes de degré 2 et stocker les équations linéaires correspondantes. Leur nombre est noté c

Étape 2 Déterminer le plus petit ensemble de ℓ variables notées $x_{i_1}, \ldots, x_{i_\ell}$, intervenant dans le monômes quadratique d'au moins $n - c$ équations.

Étape 3 Pour les 2^ℓ valeurs de $(x_{i_1}, \ldots, x_{i_\ell})$, construire et résoudre les système linéaire correspondant. La plupart du temps, le système apportera une contradiction, sinon, il faudra vérifier si le résultat est correct.

7.3.5 Exemple

Afin de comprendre exactement les étapes de notre attaque, nous la décrivons sur un exemple avec $n = 10$ et $m = 15$. Supposons que l'on a les équations

suivantes :

$$
\begin{aligned}
y_1 &= x_2 + x_8 + x_9 + x_1x_5 && (7.1)\\
y_2 &= x_2 + x_5 + x_9 + x_4x_6 && (7.2)\\
y_3 &= x_1 + x_8 + x_9 + x_3x_4 && (7.3)\\
y_4 &= x_2 + x_5 + x_8 + x_7x_9 && (7.4)\\
y_5 &= x_1 + x_2 + x_5 + x_3x_{10} && (7.5)\\
y_6 &= x_5 + x_7 + x_9 + x_1x_3 && (7.6)\\
y_7 &= x_1 + x_4 + x_7 + x_9x_{10} && (7.7)\\
y_8 &= x_1 + x_4 + x_{10} + x_5x_7 && (7.8)\\
y_9 &= x_1 + x_4 + x_6 + x_8x_{10} && (7.9)\\
y_{10} &= x_2 + x_7 + x_9 + x_6x_8 && (7.10)\\
y_{11} &= x_1 + x_6 + x_7 + x_8x_{10} && (7.11)\\
y_{12} &= x_2 + x_4 + x_6 + x_5x_8 && (7.12)\\
y_{13} &= x_2 + x_4 + x_8 + x_5x_{10} && (7.13)\\
y_{14} &= x_2 + x_6 + x_9 + x_3x_{10} && (7.14)\\
y_{15} &= x_1 + x_5 + x_7 + x_6x_8\ . && (7.15)
\end{aligned}
$$

Alors, les collisions se situent aux équations (7.5), (7.9), (7.10), (7.11), (7.14) et (7.15), ce qui nous donne les trois équations linéaires suivantes.

$$
\begin{aligned}
y_5 + y_{14} &= x_1 + x_5 + x_6 + x_9\\
y_9 + y_{11} &= x_7 + x_4\\
y_{10} + y_{15} &= x_1 + x_2 + x_5 + x_9
\end{aligned}
$$

Ensuite, pour l'étape 2 de l'algorithme, nous comptons la fréquence d'apparition notée f_i des variables dans les monômes quadratiques en ayant pris soin d'enlever 3 des équations déjà utilisées pour éviter les redondances, par exemple (7.5), (7.9) et (7.10). Dans notre cas, $f_1 = 2$, $f_2 = 0$, $f_3 = 4 - 1 = 3$, $f_4 = 2$, $f_5 = 4$, $f_6 = 3 - 1 = 2$, $f_7 = 2$, $f_8 = 5 - 2 = 3$, $f_9 = 2$ et $f_{10} = 6 - 2 = 4$. Ainsi, en parcourant toutes les valeurs possibles des bits x_5 et x_{10}, nous retrouvons 4 systèmes d'équations différents. Par exemple, pour $(x_5, x_{10}) = (0, 0)$, on a le

système suivant.

$$\begin{aligned}
y_5 + y_{14} &= x_1 + x_6 + x_9 \\
y_9 + y_{11} &= x_7 + x_4 \\
y_{10} + y_{15} &= x_1 + x_2 + x_9 \\
y_1 &= x_2 + x_8 + x_9 \\
y_7 &= x_1 + x_4 + x_7 \\
y_8 &= x_1 + x_4 \\
y_{11} &= x_1 + x_6 + x_7 \\
y_{12} &= x_2 + x_4 + x_6 \\
y_{13} &= x_2 + x_4 + x_8 \\
y_{14} &= x_2 + x_6 + x_9
\end{aligned}$$

Ce système est de rang plein. Nous pouvons alors décider, en fonction de la valeur de la sortie, si l'hypothèse faite sur (x_5, x_{10}) est bonne ou pas, *i.e.* si $x_5 = x_{10} = 0$.

Dans ce qui suit, on considère que toutes les équations que l'on récupère sont généralement indépendantes. Cette hypothèse peut sembler hasardeuse, mais comme les d-uplets sont choisis de manière aléatoire en suivant une loi uniforme, on peut s'attendre à ce phénomène. De plus, nous l'avons vérifié expérimentalement.

7.3.6 Analyse de la complexité

Tout d'abord, nous devons calculer le nombre moyen de collisions que l'on obtient dans la première étape.

Proposition 7.4 (Nombre moyen de collisions)**.** *Soit n la taille de l'entrée du PRG, m le nombre d'équations et C la variable aléatoire qui compte le nombre de collisions sur les monômes de degré 2, alors le nombre moyen de collisions est*

$$\mathbb{E}(C) = m - \binom{n}{2} + \binom{n}{2}\left(\frac{\binom{n}{2} - 1}{\binom{n}{2}}\right)^m \in \mathcal{O}(n^{2(s-1)}) \ .$$

Démonstration. Nous considérons l'ensemble des monômes de degré 2, au nombre de $\binom{n}{2}$. Pour chaque équation, les deux variables de degré 2 sont choisies aléatoirement selon la distribution uniforme, donc la probabilité que le monôme $x_i x_j$ apparaisse est

$$p = \frac{1}{\binom{n}{2}} \ .$$

Nous considérons la variable $C_{i,j}$ qui vaut 0 si le monôme $x_i x_j$ apparaît 1 fois ou moins, et $k - 1$ sinon, où k est le nombre de fois où ce monôme apparaît,

i.e. le nombre de relations linéaires que nous récupérons. La valeur moyenne de $C_{i,j}$ est donnée par :

$$\mathbb{E}(C_{i,j}) = \sum_{k=2}^{m} P_{[B(m,p)=k]} \cdot (k-1),$$

où $P_{[B(m,p)=k]}$ est la probabilité d'avoir exactement k apparitions du monôme $x_i x_j$. Le nombre moyen de collisions sur les monômes quadratiques est donc ensuite obtenu en additionnant indépendamment chaque valeur moyenne de $C_{i,j}$.

$$\begin{aligned}
\mathbb{E}(C) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}(C_{i,j}) \\
&= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=2}^{m} P_{[B(m,p)=k]} \cdot (k-1) \\
&= \binom{n}{2} \sum_{k=2}^{m} P_{[B(m,p)=k]} \cdot (k-1) \\
&= \binom{n}{2} \sum_{k=2}^{m} \binom{m}{k} p^k (1-p)^{m-k} \cdot (k-1) \\
&= \left[\binom{n}{2} \sum_{k=0}^{m} \binom{m}{k} p^k (1-p)^{m-k} \cdot (k-1) \right] \\
&\quad - \left[\binom{n}{2} \sum_{k=0}^{1} \binom{m}{k} p^k (1-p)^{m-k} \cdot (k-1) \right] \\
&= \left[\binom{n}{2} \sum_{k=0}^{m} \binom{m}{k} p^k (1-p)^{m-k} \cdot (k-1) \right] + \binom{n}{2} p^0 (1-p)^m \\
&= \left[\binom{n}{2} \sum_{k=0}^{m} k \binom{m}{k} p^k (1-p)^{m-k} \right] - \binom{n}{2} + \binom{n}{2} (1-p)^m
\end{aligned}$$

En utilisant la valeur de p :

$$
\begin{aligned}
\mathbb{E}(C) = &\left[\sum_{k=0}^{m} k\binom{m}{k}\left(\frac{1}{\binom{n}{2}}\right)^{k-1}\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{m-k}\right] \\
&-\binom{n}{2}+\binom{n}{2}(1-p)^m \\
=&\left[\sum_{k'=0}^{m-1}(k'+1)\binom{m}{k'+1}\left(\frac{1}{\binom{n}{2}}\right)^{k'}\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{m-1-k'}\right] \\
&-\binom{n}{2}+\binom{n}{2}(1-p)^m \\
=&\left[\sum_{k'=0}^{m-1} m\binom{m-1}{k'}\left(\frac{1}{\binom{n}{2}}\right)^{k'}\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{m-1-k'}\right] \\
&-\binom{n}{2}+\binom{n}{2}\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{m} \\
=&m-\binom{n}{2}+\binom{n}{2}\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{m} \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}\left(1-\frac{1}{\binom{n}{2}}\right)^{n^{\mathsf{s}}} \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}e^{\ln\left(1-\frac{1}{\binom{n}{2}}\right)n^{\mathsf{s}}} \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}e^{\left(-\frac{1}{\binom{n}{2}}-\frac{1}{2\binom{n}{2}^2}+o\left[\frac{1}{\binom{n}{2}^3}\right]\right)n^{\mathsf{s}}} \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}e^{\left(-\frac{n^{\mathsf{s}}}{\binom{n}{2}}-\frac{n^{\mathsf{s}}}{2\binom{n}{2}^2}+o\left[\frac{n^{\mathsf{s}}}{\binom{n}{2}^3}\right]\right)} \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}\left(1-\frac{n^{\mathsf{s}}}{\binom{n}{2}}-\frac{n^{\mathsf{s}}}{2\binom{n}{2}^2}+o\left[\frac{n^{\mathsf{s}}}{\binom{n}{2}^3}\right]+\frac{n^{2\mathsf{s}}}{2\binom{n}{2}^2}+o\left[\frac{n^{2\mathsf{s}}}{\binom{n}{2}^2}\right]\right) \\
=&n^{\mathsf{s}}-\binom{n}{2}+\binom{n}{2}-n^{\mathsf{s}}-\frac{\binom{n}{2}n^{\mathsf{s}}}{2\binom{n}{2}^2}+\frac{\binom{n}{2}n^{2\mathsf{s}}}{2\binom{n}{2}^2}+o\left[\frac{\binom{n}{2}n^{2\mathsf{s}}}{\binom{n}{2}^2}\right] \\
=&-\frac{n^{\mathsf{s}}}{2\binom{n}{2}}+\frac{n^{2\mathsf{s}}}{2\binom{n}{2}}+o\left[\frac{n^{2\mathsf{s}}}{2\binom{n}{2}}\right] \\
\in&\mathcal{O}\left(n^{2(\mathsf{s}-1)}\right)\ .
\end{aligned}
$$

□

La table 7.1 donne ce nombre de collisions pour des valeurs usuelles de n et de s.

n	256	512	1024	2048	4096
$\mathsf{s} = 1.45$	142	269	506	946	1771
$\mathsf{s} = 1.4$	83	145	254	442	773
$\mathsf{s} = 1.3$	28	42	64	97	147

Table 7.1 – Nombre moyen de collisions.

7.3.6.1 Complexité des étapes 1 et 2

Étape 1. La complexité de la première étape de l'algorithme est en $\mathcal{O}(m \log(m))$, en triant les équations au regard des monômes quadratiques présents, rendant la complexité de cette étape négligeable devant la complexité de la dernière étape.

Étape 2. Le but de la deuxième étape est de déterminer, en fonction des équations, quels sont les bits sur lesquels il sera nécessaire de faire une hypothèse dans l'étape 3. La procédure que nous utilisons est la suivante.

Algorithme 3 Étape 2.

Entrée: Σ le système d'équations

$N \leftarrow 0$
Sortie $\leftarrow \{\}$
$S \leftarrow \{\}$
Répéter
    Trouver la variable qui apparaît le plus dans les monômes quadratiques
    Soit A l'ensemble de ces équations
    $S \leftarrow S \cup A$
    $N \leftarrow N + |A|$
tant que $N \geq n - c$
Renvoyer S

Le nombre d'itérations de la boucle dans cette procédure est ℓ par définition, sachant que nous devons parcourir l'ensemble des équations, le coût de cette étape est $\mathcal{O}(\ell m)$.

Comme ℓ joue un rôle fondamental, il est nécessaire de connaître sa valeur. Dans la proposition suivante, nous donnons le pire cas pour ℓ, *i.e.* quand ℓ est le plus grand possible.

Proposition 7.5. *Soit n le nombre de variables et m le nombre d'équations et c le nombre de collisions, alors le nombre de bits à supposer dans l'étape 2 est au plus égal à*

$$\ell = \left\lceil \frac{n(n-c)}{2(m-c)} + 1 \right\rceil .$$

Démonstration. Nous avons un système d'équations quadratiques à m équations dans lequel apparaissent exactement une fois par équation m monômes de degré 2, et nous nous intéressons à la fréquence d'apparition des n variables dans la partie quadratique. Or, en choisissant les variables les plus fréquentes indépendamment les unes des autres, il peut apparaître un cas dégénéré pour l'attaquant.e : lorsque ces variables interviennent trop souvent dans les mêmes monômes quadratiques.

Par exemple, si l'on a les monômes x_1x_2, x_1x_3, x_2x_3 et x_4x_5, alors la fréquence d'apparition de x_1, x_2 et x_3 vaut 2 et la fréquence d'apparition de x_4 et x_5 vaut 1. Ainsi, en choisissant x_1, x_2 et x_3, qui apparaissent le plus, on n'obtient que 3 équations (le fait de réaliser une hypothèse sur x_3 n'apporte pas d'équation linéaire supplémentaire). Ainsi, il est plus judicieux de choisir les variables sur lesquelles nous réalisons les hypothèses les unes après les autres.

Plus précisément, pour tout $1 \leq i \leq n$, nous notons N_i^1 la variable qui compte le nombre de fois où x_i apparaît effectivement dans un monôme de degré 2. Nous choisissons alors la variable qui apparaît le plus : sans perdre de généralité, nous pouvons considérer qu'il s'agit de x_1. Ainsi, comme nous avons $m-c$ équations, nous avons que $\sum_{i=1}^{n} N_i = 2(m-c)$. Donc, $N_1^1 \geq 2\frac{m-c}{n}$, et en fixant la valeur de x_1, nous obtenons exactement N_1^1 équations linéaires en plus des c premières obtenues grâce aux collisions.

Comme on a fixé la valeur de x_1, le système d'équations quadratiques qui reste totalise $m-c-N_1$ équations ainsi que $n-1$ inconnues $(x_2, \ldots, x_n)$. Pour tout $2 \leq i \leq n$, nous notons N_i^2 la variable qui compte le nombre de fois où x_i apparaît effectivement dans un monôme de degré 2 dans ce nouveau système d'équations. Nous choisissons alors la variable qui apparaît le plus dans les monômes de degré 2 dans ce système, et nous réappliquons la même chose que pour x_1. Sans perdre de généralité, on considère que la variable qui apparaît le plus est x_2. Alors, on a $N_2^2 \geq 2\frac{m-c-N_1^1}{n-1} \geq 2\frac{m-c}{n}$.

Nous continuons cette procédure tant que le nombre d'équations linéaires que l'on obtient est inférieur à n. Ainsi, au choix de la i-ème variable dont on fixe la valeur, on a un système d'équations quadratiques à $n-i+1$ variables et $m-c-N_1-N_2^2-\cdots-N_{i-1}^{i-1}$ équations. Toujours sans perdre de généralité nous pouvons considérer que x_i apparaît le plus souvent dans les monômes qudratiques du système à $n-i+1$ inconnues. Ainsi, pour tout $i \geq 1$,

$$N_i^i \geq 2\frac{m-c-N_1^1-N_2^2-\cdots-N_{i-1}^{i-1}}{n-i+1} \geq 2\frac{m-c}{n} \ ,$$

et nous obtenons à cette étape exactement $c+N_1+N_2+\cdots+N_i$ équations linéaires en fixant la valeur des variables $x_1, x_2, \ldots, x_i$.

Nous continuons à fixer la valeur des variables choisies de cette manière tant que $c+N_1^1+N_2^2+\cdots+N_i^i < n$. La valeur de i pour laquelle $c+N_1^1+N_2^2+\cdots+N_i^i \geq n$ correspond exactement à la valeur du nombre d'hypothèses à faire, *i.e.* ℓ. En particulier,

$$c+N_1^1+N_2^2+\cdots+N_{\ell-1}^{\ell-1} < n \ .$$

Comme pour tout $\ell \geq i \geq 1$, on a $N_i^i \geq 2\frac{m-c}{n}$, on obtient donc l'inégalité

$$c + 2(\ell - 1)\frac{m-c}{n} < n \ .$$

Donc, le nombre d'hypothèses à réaliser, *i.e.* ℓ est au plus de

$$\frac{n(n-c)}{2(m-c)} + 1 \ .$$

□

Finalement, la complexité de la deuxième étape est de l'ordre de $\mathcal{O}(n^2)$.

7.3.6.2 Complexité de l'algorithme

Comme nous pouvions l'imaginer, c'est bien la troisième étape qui nous coûte le plus cher dans notre algorithme. En effet, nous devons résoudre un système linéaire de taille n, mais 2^ℓ fois.

Pour chacune des 2^ℓ hypothèses réalisées, plusieurs cas peuvent se produire :

— le système renvoie une solution, et dans ce cas ces n bits sont testés et comparés avec les autres équations inutilisées ;

— le système est surdéterminé et aucune solution n'est trouvée et il faut passer à la valeur suivante des ℓ bits ;

— le système est sous-déterminé, et dans ce cas, il faut réaliser une hypothèse sur un bit de plus, mais comme nous l'avons remarqué en pratique, ce cas n'arrive que très rarement, voire jamais.

Finalement, le coût de cette troisième étape qui domine la complexité de notre algorithme dans le pire cas est donc, en injectant la valeur de ℓ obtenue précédemment,

$$\mathcal{O}\left(n^\omega 2^{\frac{n^{2-s}}{2}}\right) \ .$$

Sur la valeur de ℓ attendue. La valeur de ℓ que nous utilisons correspond au pire cas, *i.e.* lorsque toutes les variables apparaissent à la même fréquence. Or, nous réalisons nos hypothèses en fonction de cette fréquence d'apparition. Il convient donc d'étudier la valeur moyenne du nombre d'hypothèses que l'on cherche à mener. Cependant, cette fréquence d'apparition des variables peut être vue comme un problème de boules dans des urnes [5] largement étudié en combinatoire. Au second ordre, il est bien connu [JK77, KSC78] que la valeur de la fréquence maximale vaut

$$\Theta\left(\sqrt{\frac{m \log n}{n}} + \frac{m}{n}\right) \ ,$$

5. balls into bins

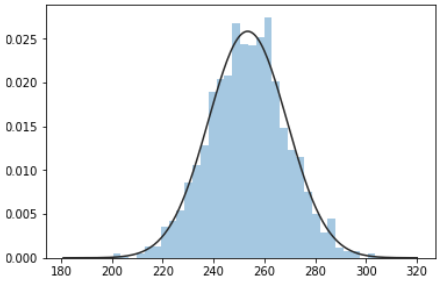

Figure 7.2 – Nombre de collisions pour $n = 1024$ et $\mathsf{s} = 1.4$ sur 2000 tests.

où m est le nombre de boules et n le nombre d'urnes. Dans notre contexte, cela est légèrement différent, puisqu'une variable ne peut apparaître 2 fois dans le même monôme. Cependant, nous pouvons borner inférieurement la fréquence en ne considérant qu'une variable (par exemple la première), et borner supérieurement en considérant qu'une variable peut être prise 2 fois dans le même monôme, ce qui change m en $2m$ dans la valeur donnée ci-dessus. Finalement, ceci nous montre que nous ne gagnons rien entre le pire cas et le cas moyen asymptotiquement. Cependant, nous verrons que pour des valeurs pratiques, la différence entre le cas moyen et le pire cas n'est pas négligeable.

7.3.7 Vérification expérimentale

L'avantage de l'attaque que nous décrivons ici est que l'on peut avoir une très bonne approximation de son coût sans la mener jusqu'au bout. En effet, en appliquant seulement les étapes 1 et 2, ce qui coûte très peu cher, nous pouvons connaître le coût de la troisième étape avec précision, puisque le nombre d'hypothèses qu'il faut réaliser est calculable facilement et détermine le coût de l'attaque.

Pour le nombre de collisions, les résultats théoriques donnés à la table 7.1 correspondent à ceux obtenus en pratique et sont donnés à la figure 7.2.

Comme nous l'avons vu, la valeur théorique de ℓ que nous avons utilisée jusqu'à maintenant correspond au pire cas pour l'attaquant.e. Même si le cas moyen est censé être asymptotiquement du même ordre de grandeur, il convient de vérifier expérimentalement à quel point le nombre d'hypothèses est plus petit que dans le pire cas. Les résultats expérimentaux sur le nombre d'hypothèses nécessaires sont décrits à la table 7.3 et sont à comparer avec les résultats décrits à la table 7.2 obtenus en appliquant la formule du pire cas donnée précédemment.

De plus, il apparaît que la variance du nombre d'hypothèses est extrêmement faible, comme le montre la figure 7.3. En effet, ℓ prend seulement 5 valeurs différentes sur 2000 tests. Finalement, avec ces résultats nous sommes capables de dessiner une courbe (figure 7.4), dépendant de l'expansion s et de la taille de

n	256	512	1024	2048	4096
$\mathsf{s} = 1.45$	4	7	11	18	27
$\mathsf{s} = 1.4$	9	15	23	37	58
$\mathsf{s} = 1.3$	20	34	56	94	156

Table 7.2 – Valeur théorique de ℓ.

n	256	512	1024	2048	4096
$\mathsf{s} = 1.45$	4	6	9	14	21
$\mathsf{s} = 1.4$	6	11	17	27	44
$\mathsf{s} = 1.3$	13	23	39	65	110

Table 7.3 – Valeur expérimentale de ℓ.

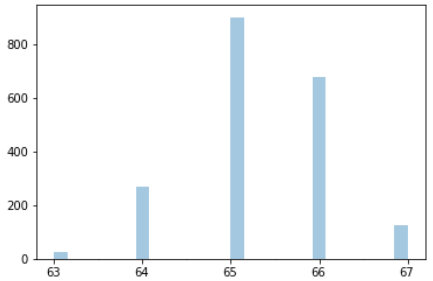

Figure 7.3 – Valeur de ℓ pour $n = 2048$ et $s = 1.3$ avec 2000 tests.

la graine, au-dessus de laquelle notre attaque coûte moins de 2^{80} opérations.

7.4 Une autre attaque algébrique

Dans cette section, nous étudions une autre méthode pour résoudre le système, basée sur des méthodes de base de Gröbner. La complexité des algorithmes existant [Buc76, Fau99, Fau02] est relativement difficile à décrire de manière générale. Cependant, dans notre cas, le système d'équations que l'on cherche à résoudre possède une forme très particulière : les équations sont quadratiques ; le nombre de monômes est toujours le même et les équations sont extrêmement creuses.

Le problème que nous rencontrons vient principalement du trop petit nombre d'équations que l'on a. Il convient donc d'essayer de trouver d'autres équations, linéairement indépendantes des premières, de manière à faire grossir la taille de notre système.

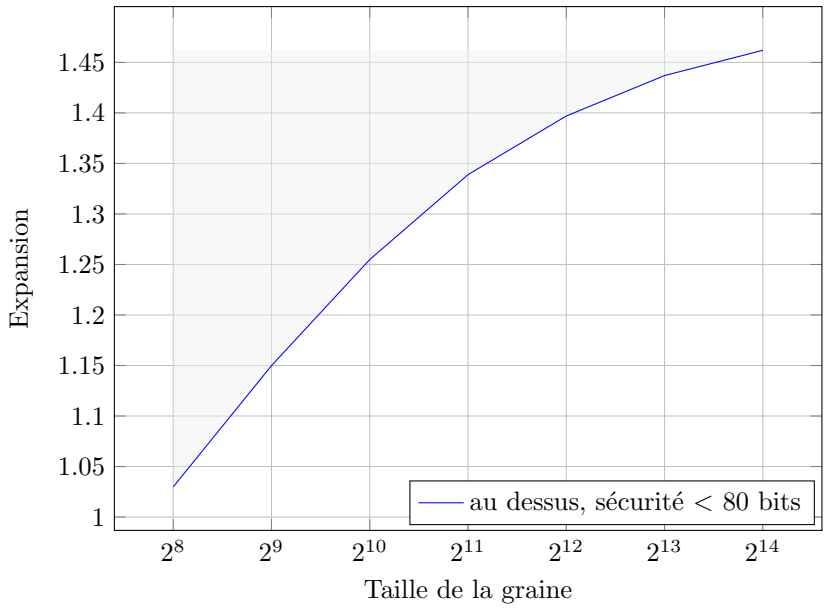

Figure 7.4 – Sécurité du PRG de Goldreich. La zone grise au-dessus de la courbe correspond à un choix de paramètres pour lesquels notre attaque coûte moins de 2^{80} opérations.

7.4.1 Trouver d'autres équations

L'idée générale consiste ici à dériver des équations de celles que l'on a initialement. En fait, une équation peut "en cacher une autre". L'idée peut aussi être reliée à la notion d'immunité algébrique, où l'on déduit aussi une nouvelle équation de l'équation initiale. Nous rappelons que la forme de nos équations de départ est la suivante.

$$\forall 1 \leq i \leq m,\ x_{\sigma_1^i} + x_{\sigma_2^i} + x_{\sigma_3^i} + x_{\sigma_4^i} x_{\sigma_5^i} = y_i\ . \qquad (E_i)$$

Au lieu de baisser le degré des équations en fixant la valeur de quelques bits en entrée, nous allons réaliser des opérations sur ces équations, afin d'obtenir plusieurs équations quadratiques linéairement indépendantes des précédentes, de manière à en obtenir au moins $\mathcal{N}_{var}$, où

$$\mathcal{N}_{var} = n + \binom{n}{2}\ .$$

Nous utilisons trois méthodes différentes pour générer nos équations.

Première méthode. Nous pouvons engendrer de nouvelles équations à partir de chaque équation, indépendamment des autres. En effet, si l'on considère

l'équation (E_i) :

$$x_{\sigma_1^i} + x_{\sigma_2^i} + x_{\sigma_3^i} + x_{\sigma_4^i} x_{\sigma_5^i} = y_i \ ,$$

alors comme nous sommes dans un corps de caractéristique 2, cela signifie que nos équations peuvent être quotientées par $x^2 + x = 0$. Donc, en multipliant l'équation (E_i) par $x_{\sigma_4^i}$ et $x_{\sigma_5^i}$, nous obtenons les deux équations quadratiques suivantes :

$$x_{\sigma_1^i} x_{\sigma_5^i} + x_{\sigma_2^i} x_{\sigma_5^i} + x_{\sigma_3^i} x_{\sigma_5^i} + x_{\sigma_4^i} x_{\sigma_5^i} = y_i x_{\sigma_5^i}$$

$$x_{\sigma_1^i} x_{\sigma_4^i} + x_{\sigma_2^i} x_{\sigma_4^i} + x_{\sigma_3^i} x_{\sigma_4^i} + x_{\sigma_4^i} x_{\sigma_5^i} = y_i x_{\sigma_4^i}$$

Ceci nous permet donc de considérer l'ensemble d'équations

$$\{zE_i | \forall z \in \{x_{\sigma_4^i}, x_{\sigma_5^i}\}\}$$

qui est de cardinalité 2, ce qui nous permet d'obtenir $2m = 2n^{\mathsf{s}}$ nouvelles équations.

Deuxième méthode. En utilisant les collisions sur les monômes de degré 2 utilisées dans l'étape 1 de notre attaque de type "supposer et déterminer", nous pouvons aussi engendrer pléthore de nouvelles équations quadratiques. Supposons que nous avons une collision sur le monôme de degré 2 entre (E_i) et (E_j). Alors, l'équation $(E_i + E_j)$ est de la forme

$$x_{\sigma_1^i} + x_{\sigma_2^i} + x_{\sigma_3^i} + x_{\sigma_1^j} + x_{\sigma_2^j} + x_{\sigma_3^j} = y_i + y_j \ .$$

Cette équation étant linéaire, nous pouvons la multiplier par n'importe quel monôme de degré 1, conduisant à l'ensemble d'équations

$$\{z(E_i + E_j) | \ \forall z \in \{1, x_1, x_2, \ldots, x_n\}\}$$

qui est de cardinalité $(n+1)$, ce qui nous permet d'assurer qu'environ

$$(n+1)\mathbb{E}[C] = (n+1)n^{\mathsf{s}} + (n+1)\binom{n}{2}\left(\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{n^{\mathsf{s}}} - 1\right)$$

nouvelles équations peuvent être créées, en utilisant la proposition 7.4.

Troisième méthode. Nous pouvons engendrer encore d'autres équations lorsque qu'une semi-collision apparaît, au sens de la définition suivante.

Définition 7.6 (Semi-collision)**.** *Une semi-collision est un couple* (i, j) *avec* $1 \leq i, j \leq m$ *tel que*

- $i \neq j$;
- (i, j) *n'est pas une collision ;*
- $\{\sigma_4^i, \sigma_5^i\} \cap \{\sigma_4^j, \sigma_5^j\} \neq \emptyset$.

Par exemple, les deux équations suivantes forment une semi-collision.

$$x_1 + x_2 + x_3 + x_7x_{10} = y_1 \tag{E_1}$$

$$x_4 + x_5 + x_6 + x_7x_8 = y_2 \tag{E_2}$$

Lorsque nous avons une semi-collision, il est possible d'engendrer 2 nouvelles équations. En effet, si l'on a une semi-collision (i, j), alors si l'on suppose sans perdre de généralité que $\sigma_4^i = \sigma_4^j$, nous pouvons engendrer l'équation $x_{\sigma_5^j} E_i + x_{\sigma_5^i} E_j$ qui est quadratique, puisque le monôme de degré 3 obtenu par $x_{\sigma_5^j} E_i$ est le même que celui obtenu par $x_{\sigma_5^i} E_j$.

En reprenant l'exemple ci-dessus, nous obtenons l'équation $x_8E_1 + x_{10}E_2$, *i.e.*

$$x_8x_1 + x_8x_2 + x_8x_3 + x_{10}x_4 + x_{10}x_5 + x_{10}x_6 = x_8y_1 + x_{10}y_2 \ .$$

7.4.2 Nombre réel d'équations

En utilisant ces trois méthodes, on se rend bien compte que le nombre d'équations de degré 2 que l'on obtient est bien plus grand que la taille de la sortie. Plus précisément, le nombre moyen d'équations que l'on a est donné par la proposition suivante.

Proposition 7.7 (Nombre d'équations de degré 2)**.** *Le nombre d'équations quadratiques $\mathcal{N}_{eq}(n, \mathsf{s})$ que l'on engendre avec les trois méthodes est en moyenne égal à*

$$\mathcal{N}_{eq} = n\binom{2n^{\mathsf{s}-1}}{2} + (n+2)n^{\mathsf{s}} + (n-1)\binom{n}{2}\left(\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{n^{\mathsf{s}}} - 1\right)$$

ce qui est de l'ordre de

$$O(n^{2\mathsf{s}-1}) \ .$$

Démonstration. Nous avons déjà déterminé le nombre d'équations dérivées en utilisant les deux premières méthodes : il vaut au total $2n^{\mathsf{s}} + (n+1)\mathbb{E}[C]$. Il nous reste donc à montrer que le nombre d'équations générées par la troisième méthode est en moyenne

$$n\binom{2n^{\mathsf{s}-1}}{2} - 2\mathbb{E}[C] \tag{7.16}$$

On suppose ici que les équations sont distinctes, ce qui n'est pas restreignant car si deux équations sont identiques, alors un distingueur évident sur la sortie existe avec une probabilité de succès de $\frac{1}{2}$.

Soit p la probabilité pour une variable d'apparaître dans un terme quadratique. Alors $\mathsf{p} = \frac{2}{n}$. Pour une variable x_i, il y a donc $m\mathsf{p} = 2n^{\mathsf{s}-1}$ équations en moyenne

qui font intervenir x_i dans l'unique terme quadratique. Parmi cet ensemble, le nombre de couples d'équations vaut donc

$$\binom{2n^{\mathsf{s}-1}}{2}.$$

En multipliant ceci par n, nous obtenons exactement le nombre de collisions et de semi-collisions. Pour obtenir le nombre de semi-collisions, il faut donc soustraire deux fois la valeur moyenne du nombre de collisions, donné par la proposition 7.4, puisqu'une collision est comptée deux fois dans notre comptage précédent (une fois pour x_i et une fois pour x_j).

Le nombre total d'équations est obtenu en additionnant les valeurs obtenues pour chacune des 3 méthodes de génération d'équations :

$$\begin{aligned}\mathcal{N}_{eq} &= 2n^{\mathsf{s}} + (n+1)\mathbb{E}[C] + n\binom{2n^{\mathsf{s}-1}}{2} - 2\mathbb{E}[C] \\ &= n\binom{2n^{\mathsf{s}-1}}{2} + (n+2)n^{\mathsf{s}} + (n-1)\binom{n}{2}\left(\left(\frac{\binom{n}{2}-1}{\binom{n}{2}}\right)^{n^{\mathsf{s}}} - 1\right)\end{aligned}$$

□

Comme $\mathsf{s} < 1.5$, cela implique que nous avons $\mathcal{N}_{eq}(n,\mathsf{s}) < \mathcal{N}_{var}(n)$ (en moyenne). En effet,

$$\begin{aligned}\mathcal{N}_{eq} &= 2n^{\mathsf{s}} + (n+1)\mathbb{E}[C] + n\binom{2n^{\mathsf{s}-1}}{2} - 2\mathbb{E}[C] \\ &\simeq 2n^{\mathsf{s}} + n \times n^{2(\mathsf{s}-1)} + n \times (2n^{\mathsf{s}-1})^2 - 2n^{2(\mathsf{s}-1)} \\ &\in \mathcal{O}(n^{2\mathsf{s}-1})\end{aligned}$$

d'après la proposition 7.4. Le fait que $\mathcal{N}_{eq}(n,\mathsf{s}) < \mathcal{N}_{var}(n)$ rend a priori impossible de linéariser le système et de retrouver la graine. Or, pour un nombre non-négligeable d'instances pratiques, il apparaît que $\mathcal{N}_{eq}(n,\mathsf{s}) \simeq \mathcal{N}_{var}(n)$, situation que nous analysons dans ce qui suit.

7.4.3 Description de l'attaque par linéarisation

7.4.3.1 Dépendances linéaires

Nous pouvons engendrer un grand nombre de nouvelles équations, cependant, il est possible qu'un nombre non-négligeable de ces équations soient linéairement dépendantes. De telles dépendances impliquent que le nombre d'équations est plus petit que celui attendu, mais ces équations peuvent parfois être utilisées par l'attaquant.e comme nous l'expliquons dans ce qui suit.

7.4.3.2 Description de l'attaque par linéarisation

Étape 1 À partir des équations $(E_1, \ldots, E_m)$, engendrer toutes les équations quadratiques dérivées correspondantes en suivant les trois méthodes décrites précédemment.

Étape 2 Réécrire le système correspondant en séparant les variables quadratiques des variables linéaires et constantes. Soit q_i la partie quadratique du système et b_i la partie affine, alors nous avons un système de la forme suivante.

$$\begin{aligned} q_1 &= b_1 \\ q_2 &= b_2 \\ &\vdots \\ q_{\mathcal{N}_{eq}(n,\mathsf{s})} &= b_{\mathcal{N}_{eq}(n,\mathsf{s})} \end{aligned}$$

Maintenant nous linéarisons le système en considérant chaque monôme de degré 2 comme une nouvelle variable. Soit Q la matrice binaire de taille $\mathcal{N}_{eq}(n,\mathsf{s}) \times \binom{n}{2}$ la matrice représentant les coefficients dans les formes quadratiques q_i et B la matrice de taille $\mathcal{N}_{eq}(n,\mathsf{s}) \times (n+1)$ la matrice qui représente les coefficients dans les formes affines b_i. Ces matrices sont représentées à la figure 7.5 où le vecteur gris foncé représente la liste des variables quadratiques et le vecteur gris clair représente les variables linéaires, en considérant la constante comme une variable, de manière à pouvoir écrire le terme de gauche égal à 0.

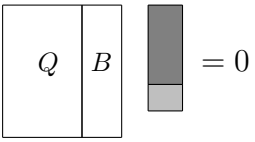

Figure 7.5 – Système linéarisé.

Étape 3 Calculer le rang des matrices Q et $(Q|B)$.

— Si $\mathsf{rang}(Q|B) = \mathcal{N}_{var}(n)$, alors inverser le système renvoie la valeur de la graine utilisée $(x_1, \ldots, x_n)$.

— Sinon $\mathsf{rang}(Q|B) < \mathcal{N}_{var}(n)$.

3.a Nous réécrivons le système comme décrit à la figure 7.6 qui suit.

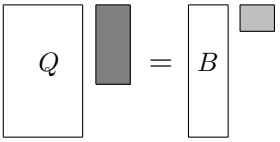

Figure 7.6 – Système linéarisé modifié.

3.b Trouver une base du noyau de Q. Alors, d'après le théorème du rang, il existe une matrice binaire Λ de taille $(\mathcal{N}_{eq}(n, \mathsf{s}) - \mathsf{rang}(Q)) \times \mathcal{N}_{eq}(n, \mathsf{s})$ de rang plein telle que $\Lambda Q = 0$.

3.c Multiplier le système par Λ, afin de trouver un système de la forme décrite à la figure 7.7.

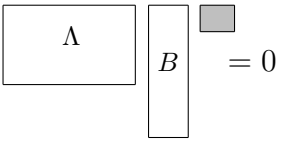

Figure 7.7 – Système linéaire faisant intervenir uniquement les variables linéaires.

3.d Si $\mathsf{rang}\Lambda B = n$, alors le système peut être inversé pour retrouver la graine.

L'amélioration précédente consiste à utiliser suffisamment d'équations quadratiques linéairement dépendantes (en les monômes de degré 2), pour retrouver des équations uniquement linéaires, ce qui nous permet d'améliorer sensiblement l'attaque par linéarisation ainsi que de résoudre quand même le système même si $\mathcal{N}_{var}(n) \geq \mathcal{N}_{eq}(n, \mathsf{s})$.

La complexité de cet algorithme est polynomiale, plus précisément elle est de l'ordre de $\mathcal{O}(n^{2\times\omega})$ où ω est l'exposant de la complexité de l'inversion matricielle. En effet, nous avons un système dont le nombre d'équations est de l'ordre de n^2, étant donné que $\mathsf{s} < 1.5$. Réécrire le système, calculer les rangs et dériver la base du noyau sont des opérations dont la complexité est bornée par la complexité de l'inversion d'un système, ce qui nous donne une complexité inférieure à $\mathcal{O}(n^{2\omega})$. Si $\mathsf{rang}(Q|B) = \mathcal{N}_{var}(n)$, alors le système est de taille n^2, ce qui nous donne la même complexité. Si on est dans le second cas, alors le système est de taille n, la complexité de l'inversion est alors de $\mathcal{O}(n^{\omega})$.

7.4.4 Vérification expérimentale

Cette attaque a été vérifiée expérimentalement avec Magma, où il est apparu que dès que $\mathcal{N}_{eq}(n, \mathsf{s}) - \mathcal{N}_{var}(n) \simeq n$, alors $\mathsf{rang}(\Lambda B) = n$. Nous avons pu alors extrapoler la faisabilité de l'attaque pour de plus grandes tailles de graines,

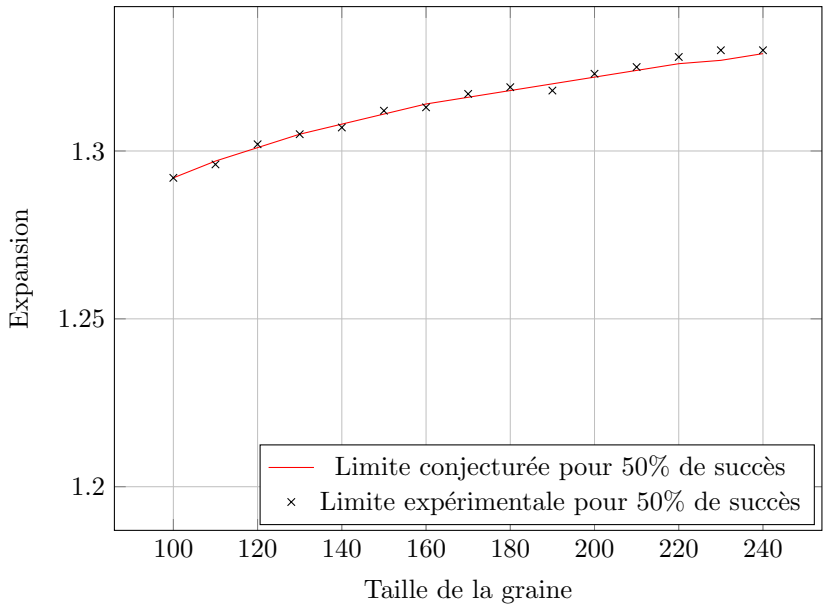

Figure 7.8 – Sécurité du PRG de Goldreich au regard de l'attaque par linéarisation. La zone au-dessus de la courbe correspond à un choix de paramètres pour lesquels notre attaque a une probabilité de succès plus grande que $\frac{1}{2}$.

jusqu'à $n = 2^{14}$, où nous vérifions la fréquence des instances qui vérifient $\mathcal{N}_{eq}(n, \mathsf{s}) - \mathcal{N}_{var}(n) \geq n$. Nous décrivons alors $\mathsf{s}_{50\%}$ comme l'expansion au-dessus de laquelle la probabilité que l'attaque fonctionne est plus grande que $\frac{1}{2}$.

Les résultats sont donnés à la figure 7.8 [6] où, au-dessus de la courbe, la probabilité de succès est supérieure à 50%. Au regard du faible nombre d'expérimentations qui ont été menées, il ne faut pas prendre cette courbe comme une vérité absolue, mais ces résultats mettent en évidence la fragilité de certains de certains choix des m d-uplets, pour lesquels il existe une attaque algébrique non-triviale en combinant les équations quadratiques.

7.5 Comparaison des deux attaques

Pour les tailles de graine pratiques que l'on considère, l'attaque sous-exponentielle est plus performante. Principalement, cette deuxième attaque a une probabilité d'échec qui peut devenir grande lorsque l'expansion est proche de 1, mais

6. Étant donné que l'attaque doit être menée à chaque fois pour déterminer sa faisabilité, nous n'avons pas pu réalisé autant de tests que pour l'attaque de type "supposer et déterminer", le nombre de tests est de l'ordre de 10 à chaque fois.

elle est moins coûteuse, alors que l'attaque de type "supposer et déterminer" fonctionne tout le temps, mais comme elle est sous-exponentielle, elle devient impraticable quand les tailles des graines deviennent trop grandes.

Afin de pouvoir comparer les deux approches, nous avons extrapolé la courbe décrite à la figure 7.8 sur de plus grandes tailles de graines. La comparaison des deux attaques est décrite à la figure 7.9.

Le PRG de Goldreich instancié avec le prédicat P_5 était considéré sûr, tant que l'expansion vaut $1.5 - \varepsilon$, pour une constante arbitraire ε non nulle. Nous réfutons donc ceci, puisque comme le montre notre analyse, il est nécessaire de prendre des graines de grande taille afin d'assurer une sécurité raisonnable. Typiquement, pour une expansion de 1.4, une graine de taille inférieure à 5120 bits ne peut pas assurer une sécurité de 80 bits. Même pour une expansion proche de 1, par exemple 1.1, la taille de la graine doit être plus grande que 512 bits, pour assurer une sécurité de 80 bits au regard de notre attaque.

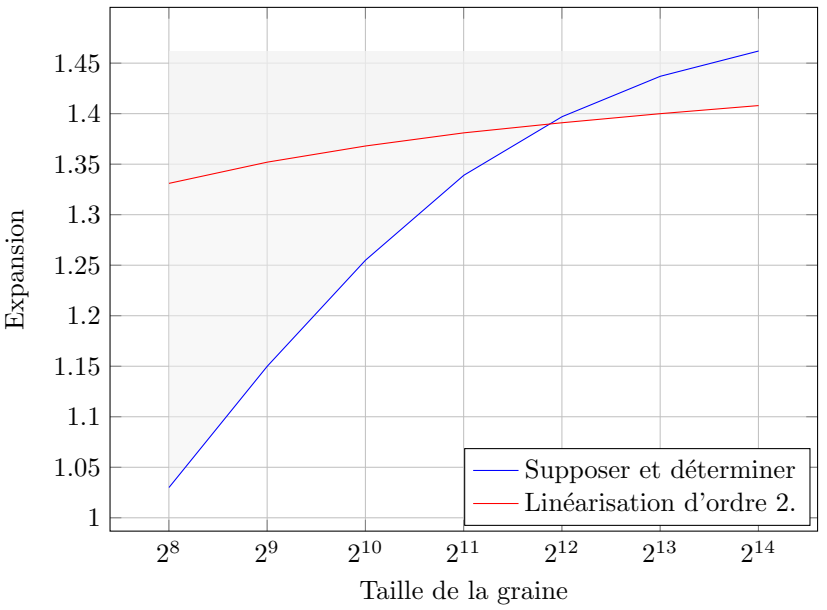

Figure 7.9 – Expansion maximale s pouvant être utilisée en fonction de la taille de la graine, assurant une sécurité de 80 bits, au regard des deux attaques décrites : l'attaque de type "supposer et déterminer" de la section 7.3 et l'attaque par linéarisation d'un système de degré 2 décrite à la section 7.4. La zone grise au-dessus des deux courbes correspond à un choix de paramètres ayant un niveau de sécurité inférieur à 80 bits.

7.6 Attaques sur la construction générique

Dans cette section, nous nous intéressons à généraliser les deux attaques précédentes. Alors qu'il est assez difficile de comprendre les dépendances entre les équations utilisées dans l'attaque par linéarisation, il est une notion qui n'a pas été utilisée précédemment et qui est cependant fondamentale pour évaluer la sécurité des constructions : la dimension de l'espace des annulateurs de prédicat. Nous verrons en quoi cette valeur permet d'affiner la borne donnée par Applebaum et Lovett dans [AL16] ($\mathsf{AI}(P) > \mathsf{s}$).

Pour l'attaque décrite à la section 7.3, la technique se généralise facilement à tout prédicat utilisé dans une construction *à la* Goldreich, ce qui nous permet de décrire un algorithme générique de type "supposer et déterminer" qui fonctionne sur toute instance du PRG de Goldreich et qui est meilleure que l'algorithme générique décrit en [BQ09] dont la complexité est de l'ordre de

$$2^{\mathcal{O}(n^{1-(\mathsf{s}-1)/2d})},$$

où d est la localité du prédicat. Nous appliquerons une variante de cette généralisation aux prédicats du type $\mathsf{XOR}_\ell\mathsf{MAJ}_k$, considérés comme une instance prometteuse du PRG de Goldreich. Cette technique peut aussi être vue comme une "attaque hybride" définie dans la thèse de Luk Bettale [Bet11].

7.6.1 Généralisation de la technique de type "supposer et déterminer"

Comme les prédicats doivent avoir une résilience d'au moins 3 et que la manière classique de construire des fonctions résilientes est d'ajouter des variables linéaires indépendantes des autres, il est tout naturel de considérer les prédicats de type $\mathsf{XOR}_\ell\mathsf{MAJ}_k$.

7.6.1.1 Le principe

Soit n la taille de la graine utilisée dans le PRG, d'expansion s et de localité d. Alors l'idée est de fixer la valeur de r bits, de telle sorte que suffisamment d'équations soient de petit degré, pour pouvoir résoudre le système correspondant. Nous faisons alors en sorte de fixer $k-1$ variables parmi les k variables pris dans la partie non-linéaire du prédicat de type $\mathsf{XOR}_\ell\mathsf{M}_k$, afin de trouver une équation linéaire. En effet, une fonction à une seule variable est toujours affine. Dans ces conditions, notre attaque fonctionne de la même manière que sur le prédicat P_5.

Étape 1 Fixer r variables de la graine, notées $x_{i_1}, \ldots, x_{i_r}$, telles qu'au moins n équations soient linéaires, *i.e.* telles que, pour chacune de ces n équations, $k-1$ variables soient fixées dans la partie non-linéaire du prédicat.

Étape 2 Pour les 2^r valeurs possibles de ces variables, déterminer l'ensemble des équations linéaires. Résoudre le système et ajouter la solution s'il n'y a pas de contradiction.

7.6.1.2 Complexité

Nous répétons la résolution d'un système linéaire exactement 2^r fois, ce qui nous donne une complexité de l'ordre de

$$\mathcal{O}\left(2^r \times n^\omega\right) ,$$

où ω est l'exposant défini par l'algorithme de résolution du système utilisé.

Nous devons donc évaluer la valeur du nombre r de bits sur lesquels nous devons faire une recherche exhaustive. Tout d'abord, nous approximons le nombre d'équations linéaires que l'on obtient. À r fixé, la probabilité qu'au moins $k-1$ variables figurent parmi les k bits impliqués dans M est donnée par

$$\frac{\binom{r}{k-1}\binom{n-r}{1}}{\binom{n}{k}} + \frac{\binom{r}{k}\binom{n-r}{0}}{\binom{n}{k}} .$$

En effet, il y a deux cas favorables : ou exactement 1 position parmi les k n'est pas dans $\{x_{i_1}, \ldots, x_{i_r}\}$, ou bien toutes les k positions intervenant dans M sont fixées, ce qui donne la valeur ci-dessus.

En multipliant cette probabilité par le nombre d'équations, c'est-à-dire n^{s}, on obtient le nombre moyen d'équations linéaires que l'on a en fixant r bits. En supposant que très peu d'équations sont linéairement dépendantes, dès que ce nombre est plus grand que n, on obtient un système d'équations linéaires avec suffisamment d'équations, que l'on peut résoudre ce qui permet de décider si l'hypothèse est bonne ou pas.

En prenant $r \simeq n^{\frac{k-\mathsf{s}}{k-1}}$, nous pouvons réaliser les approximations suivantes : $\binom{r}{k-1} \simeq r^{k-1}$, $\binom{r}{k} \simeq r^k$ et $\binom{n}{k} \simeq n^k$. Donc, le nombre d'équations que l'on obtient en prenant $r \simeq n^{\frac{k-\mathsf{s}}{k-1}}$ est approximativement

$$n^{\mathsf{s}} \cdot \left(\frac{r^{k-1}(n-r) + r^k}{n^k}\right) \simeq n^{\mathsf{s}} \cdot \left(\frac{r^{k-1}}{n^{k-1}}\right) \simeq n .$$

Donc, notre algorithme a une complexité sous-exponentielle en la taille de la graine, donnée par

$$\mathcal{O}\left(n^\omega \times 2^{n^{1-\frac{\mathsf{s}-1}{k-1}}}\right) .$$

Comme $k-1$ est toujours strictement plus petit que la localité d, par définition, il est important de remarquer que notre algorithme en temps sous-exponentiel ici est bien plus performant que celui décrit dans [BQ09] dont la complexité asymptotique est

$$2^{\mathcal{O}(n^{1-(\mathsf{s}-1)/2d})} .$$

L'intérêt de la construction réside dans la petite localité. Or, nous remarquons ici que la partie non-linéaire du prédicat joue un rôle, tout comme la résilience. Il convient donc de décrire des prédicats qui sont résilients, mais qui ne sont pas construits en rajoutant des variables linéaires indépendantes, puisque cela augmenterait la localité.

7.6.2 Encore un nouveau critère...

L'algorithme décrit précédemment s'attelle à déterminer des équations linéaires, or, en y regardant de plus près, on pourrait généraliser cette idée, en gardant des équations de degré plus grand. En effet, nous sommes largement contraints ici par le nombre r de bits fixés. Pour diminuer ce nombre, nous pouvons autoriser un nombre plus petit que $k-1$ de bits fixés dans la partie non-linéaire.

Alors qu'Applebaum et Lovett [AL16] n'ont considéré que l'immunité algébrique et la notion de "degré à r bits fixés", nous considérons un nouveau critère qui capte beaucoup mieux la sécurité du PRG au regard de notre attaque.

En fixant un certain nombre de bits dans le prédicat, nous nous intéressons au plus petit degré que nous pouvons obtenir de cette nouvelle équation. Comme nous avons déjà vu plusieurs fois qu'une équation peut en cacher une autre, il apparaît donc que le critère pertinent ici est la généralisation de l'immunité algébrique dans le sens suivant, qui généralise la notion de degré à r bits fixés.

Définition 7.8 (Immunité algébrique à r bits fixés)**.** *Soit f une fonction booléenne à n variables. Pour tout $0 \leq r \leq n$ et $b = (b_1, \ldots, b_r) \in \{0,1\}^r$, $I = (i_1, \ldots, i_r) \in \{1, \ldots, n\}^r$ tel que $i_1 < i_2 < \cdots < i_r$, on note $f_{b,I}$ la restriction de f où les r variables indexées par I sont fixées à la valeur b. Alors f est d'immunité algébrique à r bits fixés égale à a si*

$$\min(\mathsf{AI}(f_{b,I}) | I \in [d]^r, b \in \{0,1\}^r) = a \ .$$

Cette notion est aussi définie dans [MJSC16] comme l'immunité algébrique récurrente, afin de résister à l'attaque sur FLIP.

Afin de d'étudier cette généralisation, nous devons tout d'abord exhiber une borne sur ce critère pour les prédicats de type $\mathsf{XOR}_\ell\mathsf{M}_k$.

Proposition 7.9. *Soit P un prédicat de type $\mathsf{XOR}_\ell\mathsf{M}_k$, soit $j \leq k$, alors pour tout ensemble J de j variables pris dans la partie M, $z_{i_1}, \ldots, z_{i_j}$ et tout $b \in \{0,1\}^j$, la fonction booléenne f à $d-j$ variables définie par*

$$f = P_{b,J}$$

est d'immunité algébrique inférieure ou égale à $\left\lceil \frac{k-j}{2} \right\rceil + 1$.

Démonstration. f est de la forme $\mathsf{XOR}_\ell + g$ où $g \in \mathcal{B}_{k-j}$. L'immunité algébrique de g est inférieure ou égale à $\left\lceil \frac{k-j}{2} \right\rceil$. Supposons qu'elle est atteinte pour un annulateur non-nul h de g. La fonction booléenne h' à $\ell + k - j$ variables définie par

$$h' = (1 + \mathsf{XOR}_\ell)h$$

est donc un annulateur de f. En effet, si $\mathsf{XOR}_\ell(x) = 1$, alors $h' = 0$, sinon, alors $\mathsf{XOR}_\ell(x) = 0$, et $h' = h$, donc $f = g$ et par définition de h, on a $h'f = hg = 0$. De plus, h' est une fonction non-nulle car h' est construite en multipliant h par une fonction impliquant des variables indépendantes, et la fonction $(1 + \mathsf{XOR}_\ell)$ est

elle aussi non-nulle, donc h' est nécessairement non-nulle. Comme h est de degré au plus $\left\lceil \frac{k-j}{2} \right\rceil$ et que $(1 + \mathsf{XOR}_\ell)$ est de degré 1, h' est nécessairement de degré plus petit que $\left\lceil \frac{k-j}{2} \right\rceil + 1$. Si l'immunité algébrique de g est atteinte pour un annulateur de $(g+1)$, on obtient le résultat par un raisonnement similaire. □

En utilisant la proposition 7.9, nous sommes capables de décrire la généralisation de notre attaque de type "supposer et déterminer". Le principe est exactement le même : nous fixons r variables, et nous regardons le nombre d'équations de degré au plus $\left\lceil \frac{k-j}{2} \right\rceil + 1$ que nous pouvons obtenir, lorsque j variables sont fixées dans la partie non-linéaire du prédicat. Pour approximer le nombre d'équations d'un tel degré que nous pouvons récupérer, nous devons tout d'abord estimer la probabilité que j variables soient fixées dans la partie non-linéaire, sachant que r variables de la graine sont fixées.

La généralisation de notre attaque ne fonctionne que si l'expansion du PRG vérifie

$$\mathsf{s} > \left\lceil \frac{k-j}{2} \right\rceil + 1$$

La probabilité qu'au moins j variables soient fixées dans la partie non-linéaire (qui comporte k variables) est donnée par

$$\sum_{i=j}^{k} \frac{\binom{r}{i}\binom{n-r}{k-i}}{\binom{n}{k}} .$$

En multipliant cette probabilité par le nombre de bits de sortie du PRG, c'est-à-dire n^{s}, nous obtenons le nombre moyen d'équations de degré au plus $\left\lceil \frac{k-j}{2} \right\rceil + 1$. Toujours en supposant qu'une fraction négligeable d'équations est linéairement dépendante des autres, nous voulons que ce nombre moyen d'équations soit supérieur aux nombre de monômes de degré inférieur ou égal à $\left\lceil \frac{k-j}{2} \right\rceil + 1$, *i.e.* à

$$\sum_{i=1}^{\left\lceil \frac{k-j}{2} \right\rceil + 1} \binom{n}{i} .$$

Comme k est une constante et que nous voulons évaluer la complexité asymptotique de notre algorithme, nous pouvons réaliser l'approximation suivante :

$$n^{\mathsf{s}} \left(\sum_{i=j}^{k} \frac{\binom{r}{i}\binom{n-r}{k-i}}{\binom{n}{k}} \right) \simeq n^{\mathsf{s}} \sum_{i=j}^{k} \frac{r^i (n-r)^{k-i}}{n^k} .$$

En prenant

$$r \simeq n^{\frac{1+j-\mathsf{s}+\lceil (k-j)/2 \rceil}{j}} ,$$

on obtient, comme $\mathsf{s} > \left\lceil \frac{k-j}{2} \right\rceil + 1$,

$$n^{\mathsf{s}} \sum_{i=j}^{k} \frac{r^i n^{k-i}}{n^k} \simeq n^{\mathsf{s}} \frac{r^j}{n^j} \sum_{i=0}^{k-j} \frac{r^i}{n^i} \geq n^{\mathsf{s}} \frac{r^j}{n^j} \ .$$

En remplaçant r par sa valeur, on obtient

$$n^{\mathsf{s}} \frac{r^j}{n^j} \simeq n^{\lceil \frac{k-j}{2} \rceil + 1} \ ,$$

ce qui nous permet de résoudre le système correspondant en linéarisant les monômes. La complexité de cet algorithme est donc de l'ordre de

$$\mathcal{O}\left(2^r n^{\omega\left(\lceil \frac{k-j}{2} \rceil + 1\right)}\right) \ ,$$

où $r \simeq n^{\frac{1+j-\mathsf{s}+\lceil (k-j)/2 \rceil}{j}}$.

La description de cet algorithme est relativement compliquée, mais elle permet de capter quelque chose de plus que le précédent. Si le but est de construire un PRG dont la sortie est largement plus grande que la taille de l'entrée, alors il est possible que cette dernière attaque soit plus adaptée que la première, car elle permet de réduire le nombre de suppositions (r), en acceptant des équations de plus haut degré (et en augmentant la complexité de résolution de chaque système). Naturellement, pour le cas du prédicat P_5 avec $\mathsf{s} = 1.5$, il n'est d'aucune utilité d'utiliser cet algorithme, mais il doit être pris en compte dans l'utilisation d'autres instanciations dont l'objectif serait d'augmenter l'expansion.

7.6.3 Application au prédicat $\mathsf{XOR}_\ell\mathsf{MAJ}_k$

Dans l'algorithme précédent, nous considérons que la valeur des bits fixés n'a pas d'influence, ce qui ne correspond pas exactement à la notion d'immunité algébrique à r bits fixés. Afin de montrer en quoi ce critère est bien le critère pertinent, nous appliquons une variante sur les prédicats de type $\mathsf{XOR}_\ell\mathsf{MAJ}_k$, définis comme la somme directe d'une fonction linéaire à ℓ variables et de la fonction majorité sur k variables. La localité est ici égale à $\ell + k$.

L'exemple que nous allons étudier est assez révélateur de la pertinence du critère d'immunité algébrique à r-bits fixés : il met en évidence que toutes les valeurs prises des bits fixés ne sont pas équivalentes. Ainsi, au lieu de fixer la position de certains bits et de parcourir l'ensemble des valeurs possibles, nous appliquons exactement la même technique que dans l'attaque sur FLIP [DLR16] (chapitre 5) : nous fixons la valeur des bits, mais nous parcourons plusieurs positions, jusqu'à tomber sur une supposition correcte.

Nous considérons un prédicat de la forme $\mathsf{XOR}_\ell\mathsf{MAJ}_k$. Alors notre algorithme retrouve la graine utilisée dans un PRG instancié avec ce prédicat avec une complexité de l'ordre de

$$\mathcal{O}\left(n^\omega 2^{n^{(k-\mathsf{s})/(k-1)}}\right) \ .$$

7.6.3.1 Observations

La fonction majorité est connue pour atteindre la valeur maximale de l'immunité algébrique classique. Dans ces conditions, toutes les valeurs de bits fixés en entrées ne sont pas équivalentes. Les deux exemples flagrants sont les suivants :

— Si l'on fixe un quart des bits à 0 et un quart des bits en entrée à 1 de la fonction majorité, alors la fonction qui en découle, est la fonction majorité, qui prend un entrée la moitié des bits, et donc elle atteint la borne supérieure de l'immunité algébrique.

— Si en revanche on fixe la moitié des bits à 0 ou la moitié des bits à 1, alors par définition la fonction qui en découle est une fonction constante et le prédicat $\mathsf{XOR}_\ell\mathsf{MAJ}_k$ devient linéaire.

7.6.3.2 L'algorithme

Au regard de ces observations, il convient donc de supposer qu'un certain nombre de bits est à 1 (respectivement à 0), de manière à ce que suffisamment d'équations deviennent linéaires.

Étape 1 Choisir r variables.

Étape 2 Supposer que ces bits sont tous égaux à 0 ou tous égaux à 1.

Étape 3 Résoudre le système linéaire correspondant, s'il y a contradiction, retourner à l'étape 1, sinon renvoyer la solution.

7.6.3.3 Complexité

L'analyse de complexité est similaire à tout ce qui a été fait jusqu'à maintenant. Nous reprenons la généralisation de l'algorithme précédent, avec la valeur de $j = \left\lceil \frac{k}{2} \right\rceil + 1$, pour être sûr que dès que j variables sont fixées à 0 ou à 1, une équation linéaire en découle. Dans ces conditions, la probabilité de retrouver une équation linéaire est égale à

$$\sum_{i=j}^{k} \frac{\binom{r}{i}\binom{n-r}{k-i}}{\binom{n}{k}} .$$

En multipliant cette probabilité par le nombre de bits de sortie du PRG, c'est-à-dire n^{s}, nous obtenons le nombre moyen d'équations linéaires. Toujours en supposant qu'une fraction négligeable d'équations est linéairement dépendante des autres, nous voulons que ce nombre moyen d'équations soit supérieur à n, pour pouvoir inverser le système. Comme k est une constante et que nous voulons évaluer la complexité asymptotique de notre algorithme, nous pouvons réaliser l'approximation suivante :

$$n^{\mathsf{s}} \left(\sum_{i=j}^{k} \frac{\binom{r}{i}\binom{n-r}{k-i}}{\binom{n}{k}} \right) \simeq n^{\mathsf{s}} \sum_{i=j}^{k} \frac{r^i (n-r)^{k-i}}{n^k} .$$

En prenant

$$r \simeq n^{\frac{1+j-\mathsf{s}}{j}} ,$$

on obtient

$$n^{\mathsf{s}} \sum_{i=j}^{k} \frac{r^i n^{k-i}}{n^k} \simeq n^{\mathsf{s}} \frac{r^j}{n^j} \sum_{i=0}^{k-j} \frac{r^i}{n^i} \geq n^{\mathsf{s}} \frac{r^j}{n^j} .$$

En remplaçant r par sa valeur, on obtient

$$n^{\mathsf{s}} \frac{r^j}{n^j} \simeq n .$$

La complexité de notre algorithme est donc de l'ordre de

$$\mathcal{O}\left(n^{\omega} 2^{n^{1-\frac{\mathsf{s}-1}{\lceil k/2 \rceil + 1}}} \right) ,$$

puisque nous avons une probabilité de 2^{-r} d'avoir réalisé une bonne supposition, nous devons répéter la première étape en moyenne 2^r fois, où $r \simeq n^{1-\frac{\mathsf{s}-1}{\lceil k/2 \rceil + 1}}$.

La complexité de notre algorithme est aussi bonne pour une raison assez particulière : nous profitons ici de la symétrie des valeurs que l'on considère. Pour certains prédicats où la valeur des bits qui engendre des équations linéaires n'a pas autant de symétrie que "tout le monde à 0" ou "tout le monde à 1", la probabilité d'obtenir une équation linéaire pourrait drastiquement chuter.

7.6.4 Sur la dimension de l'espace des annulateurs

Dans [AL16], Applebaum et Lovett ont montré que s doit être strictement plus petit que l'immunité algébrique (ce qui est en fait relativement évident au vu de tout ce que l'on a fait ici). En cryptographie symétrique, l'immunité algébrique est le critère pertinent à prendre en compte, et la dimension de l'espace des annulateurs est utilisée uniquement pour diminuer la quantité de données nécessaire à l'attaque. Cependant, la complexité en données n'est souvent pas une limitation pour l'attaquant.e dans les attaques algébriques. A priori, la quantité de données nécessaire dans une attaque algébrique est de l'ordre de $\mathcal{O}(n^d)$ où d est le degré des équations que l'on considère, alors que le coût de la résolution du système est de l'ordre de $\mathcal{O}(n^{\omega d})$.

Or, comme nous l'avons déjà dit, le contexte du PRG de Goldreich est fondamentalement différent, puisque la difficulté d'inversion du système réside uniquement dans le trop petit nombre d'équations disponibles. Ainsi, la dimension de l'espace des annulateurs joue ici un rôle extrêmement important, puisqu'il définit exactement le nombre d'équations indépendantes d'un certain degré pouvant être obtenues à partir d'une seule équation. Certains résultats sur la dimension de cet espace peuvent être trouvés dans [Car06].

7.6.4.1 Observation

Prenons par exemple l'équation suivante :

$$x_1 + x_2x_3x_4x_5 = 0 \ . \tag{7.17}$$

Alors, nous pouvons dériver de l'équation (7.17) l'équation suivante :

$$x_1(1 + x_2) = 0 \ .$$

En effet, on a $1 + x_1 + x_2x_3x_4x_5 = 1$. Comme x_1 annule $(1 + x_1)$ et que $(1 + x_2)$ est un annulateur de $(x_2x_3x_4x_5)$, $(x_1)(1+x_2)(1+x_1+x_2x_3x_4x_5) = 0$. Comme on sait que le terme de droite est égal à 1, on peut dériver l'équation $x_1(1 + x_2) = 0$.

De plus, comme x_2, x_3, x_4 et x_5 jouent un rôle symétrique, cela signifie que l'équation (7.17) implique exactement 4 autres équations de degré 2, qui sont linéairement indépendantes.

Finalement, une équation peut en cacher d'autres ! Ce phénomène capte le fait que l'on peut avoir moins d'équations que de variables, mais pourtant plus d'information qu'on ne le croît. Ces équations engendrées ici correspondent aux équations engendrées avec la première méthode dans notre attaque par linéarisation d'ordre 2.

7.6.4.2 Amélioration du théorème d'Applebaum et Lovett

Nous reprenons un PRG qui suit la construction de Goldreich, de localité d, de prédicat P et d'expansion s. De plus, nous notons N_a^0 la dimension de l'espace des annulateurs du prédicat P de degré au plus a, et N_a^1 la dimension de l'espace des annulateurs de $P+1$ de degré au plus a. Par définition de l'immunité algébrique, on a $\forall a < e, N_a^1 = N_a^0 = 0$.

Ainsi, plutôt que de dire simplement que s doit être strictement plus petit que e, nous pouvons donner le théorème suivant.

Théorème 7.10. *Soit un PRG qui suit la construction de Goldreich, de prédicat P d'immunité algébrique e. Soit $N_a = \min(N_a^1, N_a^0)$, pour $a \geq e$. Alors si*

$$\mathsf{s} \geq a - \frac{\log(N_a)}{\log(n)} \ ,$$

alors, en considérant que les équations sont majoritairement indépendantes, il existe un algorithme en temps polynomial qui trouve une pré-image x pour un sortie $y \in \{0,1\}^{n^s}$ (ou qui certifie qu'il n'existe pas de pré-image).

Démonstration. Soit $a \geq e$. Pour chaque bit de sortie y_i, l'attaquant.e peut dériver N_a^1 ou N_a^0 équations linéairement indépendantes entre elles, de degré au plus a. On note c_0 le nombre de 0 et c_1 le nombre de 1 dans la sortie y. Alors l'attaquant.e a accès à exactement $c_0N_a^0 + c_1N_a^1$ équations. Comme les sous-ensembles utilisés dans la construction sont choisis aléatoirement de manière uniforme, nous pouvons faire l'hypothèse que le système correspondant

possède très peu d'équations linéairement dépendantes. Nous reviendrons sur cette hypothèse à la section qui va suivre. Quand ce nombre d'équations est plus grand que le nombre de monômes, *i.e.* que

$$\sum_{i=1}^{a} \binom{n}{i} \simeq n^a \ ,$$

alors nous obtenons un système d'équations que nous pouvons résoudre par linéarisation, pour une complexité de $\mathcal{O}(n^{a\omega})$.

Afin de relier ce nombre à s, comme $c_0 + c_1 = n^{\mathsf{s}}$, on a $c_0 N_a^0 + c_1 N_a^1 \geq n^{\mathsf{s}} N_a$. La condition est vérifiée quand

$$\mathsf{s}\log(n) + \log(N_a) \geq a\log(n) \ .$$

□

En considérant les prédicats du type $\mathsf{XOR}_\ell\mathsf{M}_k$, nous avons toujours égalité entre N_a^0 et N_a^1, dès que $\ell > 0$.

Proposition 7.11. *Soit P un prédicat de type $\mathsf{XOR}_\ell\mathsf{M}_k$ avec $\ell > 0$, alors*

$$\forall a, N_a^0 = N_a^1 = N_a \ .$$

Démonstration. Pour un prédicat de ce type, on a $P(x_1 + 1, x_2, \ldots, x_d) = 1 + P(x_1, x_2, \ldots, x_d)$. Soit f un annulateur non-nul de P, alors on a

$$f(x_1 + 1, x_2, \ldots, x_d)(P + 1) = f(x_1 + 1, x_2, \ldots, x_d)P(x_1 + 1, x_2, \ldots, x_d) = 0 \ .$$

En d'autres termes, si $f(x_1, x_2, \ldots, x_d)$ est un annulateur de P, alors $f' = f(x_1 + 1, x_2, \ldots, x_d)$ est un annulateur de $P + 1$.

De plus, si l'on note A_0 l'espace des annulateurs de P et A_1 l'espace des annulateurs de $P + 1$, alors nécessairement $A_0 \cap A_1 = \{0\}$: si f annule P et $P + 1$, alors f annule $P + P + 1 = 1$ et est donc nécessairement nulle.

Finalement, la transformation qui envoie x_1 sur $x_1 + 1$ et qui laisse inchangée les autres variables est linéaire et bijective, elle préserve donc le degré, et cela implique directement que $N_a^0 = N_a^1$ pour tout a.

□

En combinant la proposition 7.11 avec le théorème 7.10, et en prenant $a = e$, on obtient le corollaire suivant qui améliore le théorème d'Applebaum et Lovett pour les prédicats de type $\mathsf{XOR}_\ell\mathsf{M}_k$.

Corollaire 7.12. *Soit P un prédicat de type $\mathsf{XOR}_\ell\mathsf{M}_k$ avec $\ell > 0$ et soit e l'immunité algébrique de P et N_e la dimension de l'espace vectoriel des annulateurs de degré e, alors si*

$$\mathsf{s} \geq e - \frac{\log(N_e)}{\log(n)} \ ,$$

alors il existe un algorithme en temps polynomial qui trouve une pré-image pour une sortie donnée ou réfute celle-ci comme sortie valide du PRG.

7.6.5 Sur les équations linéairement indépendantes

Dans cette dernière partie, nous avons dû faire l'hypothèse que le système d'équations que l'on obtient par nos différentes techniques ("supposer et déterminer" ou grâce à l'immunité algébrique) ne possède que peu d'équations linéairement dépendantes. Le "peu" peut être simplement une proportion constante des équations : si la moitié des équations sont linéairement dépendantes des autres, alors nous devons diviser par deux le nombre d'équations, or comme on s'intéresse dans cette dernière partie à la complexité asymptotique de nos algorithmes, ce facteur 2 intervient seulement comme une constante et ne change pas l'ordre de grandeur.

Dans la pratique, comme nous l'avons vérifié expérimentalement pour le prédicat P_5, cette hypothèse est vérifiée. Elle est cependant à prendre avec des pincettes : étant donnée la très faible localité des prédicats utilisés dans les constructions, nous ne pouvons clairement pas considérer que ces équations se comportent de manière aléatoire.

D'un autre côté, comme nous l'avons vu dans l'attaque par linéarisation d'ordre 2, les équations linéairement indépendantes ne sont pas nécessairement à éliminer et peuvent servir. Intuitivement, si beaucoup d'équations sont linéairement dépendantes, on peut s'attendre à avoir un distingueur assez fort sur la sortie du PRG.

Dans le cas où l'on n'utilise pas d'annulateur mais où l'on utilise directement les équations en tant que telles, il est clair qu'à chaque équation dépendante des autres, on rajoute de la redondance. Plus précisément, à chaque équation linéairement dépendante, on a une probabilité de $\frac{1}{2}$ de réfuter l'hypothèse considérée. Ainsi, les dépendances entre équations ne remettent pas a priori en cause la faisabilité de notre attaque.

Dans le cas où l'on utilise des annulateurs, que ce soit avec des suppositions ou pas, le distingueur associé aux redondances des équations est potentiellement moins puissant. Dans un second travail, il serait intéressant de regarder ce que signifie "avoir des redondances sur des équations obtenues en utilisant les annulateurs".

7.7 Conclusion et perspectives

Finalement, nous avons appliqué des techniques provenant de la cryptographie symétrique sur le PRG de Goldreich, ce qui nous a permis d'améliorer considérablement la complexité des attaques connues. En regardant dans les détails, nous avons clairement montré que des algorithmes sous-exponentiels peuvent être bien meilleurs que des algorithmes polynomiaux même pour des tailles de graines qui peuvent être bien plus grandes que ce que l'on utilise en pratique.

Pour poursuivre ce travail, il faudrait tout d'abord précisément étudier l'implication des dépendances entre les équations que l'on obtient. Je suis persuadé que ces dépendances ne peuvent être qu'un bonus pour l'attaquant.e.

Pour généraliser l'attaque par linéarisation d'ordre 2 et être encore plus précis, il faudrait aussi prendre en compte les dépendances entre les différentes équations non-linéaires. Or, étudier ces dépendances ne peut se faire qu'une fois le prédicat choisi, puisque cela nécessite une étude approfondie de la structure du système considéré. Il convient donc, plutôt que d'assurer que le système est sûr tant que le prédicat choisi est "suffisamment bon", de fixer une famille de prédicats, afin de pouvoir étudier correctement leur sécurité.

Ensuite, la borne donnée sur l'immunité algébrique à r bits fixée est relativement triviale, et n'est probablement pas atteignable. De plus, comme nous l'avons dit à la fin de l'attaque sous-exponentielle sur le prédicat $\mathsf{XOR}_\ell\mathsf{MAJ}_k$, nous profitons d'une symétrie sur la valeur des bits que l'on fixe qui engendre une fonction linéaire. Le critère d'immunité algébrique à r bits fixés est donc à considérer avec prudence, car sans symétrie dans la valeur des bits considérés fixés, l'attaquant.e doit aussi exploiter une certaine régularité des m d-uplets choisis. L'étude de la sécurité concrète dans ce sens semble ardue et dépend encore une fois du prédicat choisi, qu'il convient d'étudier de manière approfondie.

Une autre question apparaît et semble relativement complexe : nous n'avons pour l'instant pas mélangé les deux techniques d'attaque. Ainsi comment se comporterait un algorithme hybride sur ce système, c'est-à-dire où l'on engendrerait plus d'équations, tout en faisant des hypothèses sur certaines variables ? En continuant sur cette idée, il n'est pas non plus très clair que les instances qui résistent à la première attaque sont aussi les instances qui résistent à la seconde. De même, est-ce que ces instances résistantes correspondent à des graphes suffisamment expansifs déduits des différentes variables choisies. En d'autre termes, il faudrait déterminer si les différentes attaques existantes exploitent les mêmes propriétés des m d-uplets choisis.

Pour conclure, la définition des "bons prédicats" pour la construction du PRG de Goldreich reste à trouver ; et la seule manière de savoir si des prédicats sont "bons" reste d'en proposer de concrets et de les analyser correctement.

Chapitre 8

Cryptanalyse de Ketje

Dans ce chapitre, nous analysons la sécurité de l'algorithme de chiffrement authentifié Ketje [BDP+16], soumis à la compétition internationale CAESAR [1] et conçu par G. Bertoni, J. Daemen, M. Peeters, G. Van Assche et R. Van Keer. Ce travail a été réalisé dans le cadre du projet collaboratif BRUTUS [2], avec María Naya Plasencia et Thomas Fuhr et a été publié dans le journal *IACR Transactions on Symmetric Cryptology* en 2018 [FNR18]. Notre cryptanalyse est une attaque de type *diviser pour mieux régner* qui permet de retrouver l'état interne plus rapidement que la recherche exhaustive sur la variante Ketje Jr, adaptée à la cryptographie légère, où le paramètre du *ratio* est augmenté. Nos résultats ne remettent pas en cause la sécurité du chiffrement Ketje, mais doivent être pris en considération pour de futures modifications potentielles visant à améliorer les performances de l'algorithme.

8.1 Les fonctions éponges et le chiffrement authentifié

8.1.1 Les fonctions éponges

La construction en éponge est une construction générique proposée par G. Bertoni, J. Daemen, M. Peeters et G. Van Assche [BDPA07] qui peut être utilisée dans diverses primitives : fonctions de hachage, chiffrements à flot... Cette construction prend en entrée une chaîne de bits de longueur arbitraire et retourne une chaîne de taille spécifiée par l'utilisateur. Dans le cas des fonctions de hachage, la taille de la sortie est fixée. La construction éponge opère sur un état interne mis à jour de manière itérative par une permutation et l'insertion de bits de l'entrée considérée.

En 2008, les concepteurs de la construction éponge ont prouvé la sécurité de

1. `https://competitions.cr.yp.to/caesar.html`
2. Agence Nationale de la Recherche, contrat ANR-14-CE28-0015

cette construction générique, *i.e.* en supposant que la permutation interne est une permutation aléatoire [BDPV08]. L'exemple le plus connu de l'utilisation de cette construction est la fonction de hachage KECCAK [BDPA13] conçue par les mêmes auteurs, vainqueure de la compétition du NIST et qui correspond désormais au standard SHA-3.

8.1.1.1 La construction éponge

Le but de cette construction est de définir une fonction F dont l'entrée est de taille arbitraire et dont la taille de la sortie peut aussi être arbitraire, à partir d'une permutation p qui opère elle sur une entrée de taille fixe, égale à b bits.

Dans cette construction, la taille de l'état interne sur lequel opère p, égale à b, est appelée la *largeur*. Elle est décomposée en deux quantités : $b = r + c$ où r est appelé le ratio et c est appelée la capacité.

Dans un premier temps, la chaîne de bits que l'on souhaite traiter est rembourrée selon une certaine règle réversible [3] afin d'atteindre une taille multiple de r bits. Alors les b bits de l'état sont initialisés à la valeur nulle et la construction consiste en deux étapes comme le décrit la figure 8.1 :

— **Phase d'absorption :** durant cette première étape, chaque bloc de données de taille r est ajouté par XOR aux r premiers bits de l'état interne, où ces insertions sont entrelacées avec l'application de la transformation bijective p. Cette phase se termine lorsque toute l'information en entrée a été traitée de cette façon.

— **Phase d'essorage :** durant cette deuxième étape, les r premiers bits de l'état sont renvoyés comme des blocs de sortie, et ces extractions sont entrelacées avec l'application de la transformation bijective p. Le nombre de blocs de sortie peut alors être choisi par l'utilisateur.

La capacité est le paramètre important car il définit le niveau de sécurité : il existe notamment des attaques génériques sur cette construction en $2^{c/2}$ où c désigne la capacité. Le choix du ratio, pour une capacité donnée, relève d'un compromis entre le coût d'implémentation de p et le nombre d'itérations.

8.1.1.2 Utilisation de la construction éponge

Comme l'information en entrée et en sortie de la construction en éponge peut être de longueur arbitraire, la construction éponge permet de construire plusieurs types de primitives cryptographiques telles que des fonctions de hachage, des chiffrements à flot, ou des codes d'authentification de message (MAC [4]). Selon l'utilisation, le type de données à traiter change, comme cela est décrit à la table 8.1 suivante.

Dans certains cas, la taille de l'entrée est petite (par exemple une clef secrète), alors que la taille de la sortie est grande (chiffrements à flot) ; dans d'autres cas, c'est l'inverse qui se produit (fonctions de hachage).

3. Padding
4. Message Authentication Code, permet d'assurer l'intégrité des messages

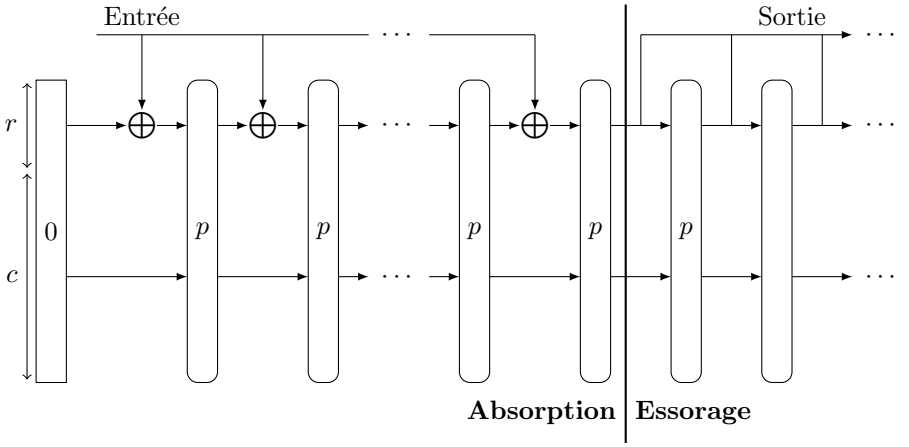

Figure 8.1 – La construction éponge.

Application	Valeur en entrée
Fonction de hachage	M
MAC	$K\|\|IV\|\|M$
Chiffrement à flot	$K\|\|IV$

Table 8.1 – Utilisations de la construction éponge.

8.1.1.3 La construction duplex

Dans ces conditions, il peut être utile d'utiliser la construction *duplex*, qui est une variante de la construction en éponge et dont la sécurité a été prouvée équivalente [BDPV12]. La construction duplex permet d'alterner des blocs en entrée et en sortie comme décrit à la figure 8.2, ce qui permet d'implémenter de manière efficace des modes de chiffrement authentifié en utilisant seulement un appel à p par bloc généré en sortie.

L'algorithme de chiffrement authentifié KETJE que nous détaillerons dans la section 8.2 suit de très près cette construction duplex.

8.1.2 Le chiffrement authentifié et la compétition CAESAR

Le chiffrement authentifié (AE [5]) et le chiffrement authentifié avec données associées (AEAD [6]) sont des chiffrements qui permettent d'assurer à la fois la *confidentialité* et l'*authenticité* des messages, en n'utilisant qu'une seule primitive

5. Authenticated Encryption
6. Authenticated encryption with Associated Data

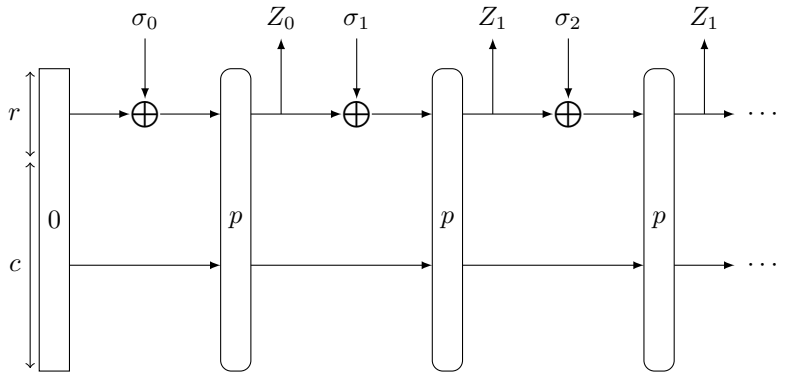

Figure 8.2 – La construction duplex : les σ_i sont les entrées après *padding* et la sortie est donnée par les Z_i.

cryptographique. Par exemple, `AES-GCM` est largement utilisé, mais sa sécurité est souvent discutée, notamment dans un scénario où le *nonce* est utilisé plusieurs fois.

C'est une des raisons pour lesquelles l'intérêt du chiffrement authentifié a grandi dans la communauté cryptographique ces dernières années, avec pour conséquence la tenue de la compétition CAESAR [7], annoncée en 2013. Cette année 2018 est la dernière année de cette compétition, car elle devrait être celle de l'annonce des vainqueurs, ce qui permettrait d'avoir un porte-feuille de bons algorithmes de chiffrement authentifié.

Dans le chiffrement authentifié, le *nonce* est une valeur publique, qui ne doit pas être utilisée plusieurs fois avec la même clef secrète. Cependant, pour se prémunir d'une mauvaise utilisation, il y a quand même un intérêt à concevoir des algorithmes dont la sécurité ne s'écroule pas complètement lorsque le *nonce* est répété. En plus du texte chiffré, un algorithme de chiffrement authentifié produit une marque (tag) qui est une suite de bits concaténée au texte chiffré qui assure que le message n'a pas été modifié.

Parfois, l'utilisateur peut souhaiter authentifier certaines données sans pour autant chiffrer celles-ci, ce qui justifie l'utilisation du chiffrement authentifié avec données associées.

Il existe plusieurs constructions de chiffrement authentifié sur lesquelles nous ne nous attarderons pas ici. Les candidats CAESAR retenus pour la phase finale de la compétition sont séparés en trois catégories :

1 Chiffrements à bas coût, pour des applications matérielles contraintes : ACORN [Wu17] et Ascon [DEMS17]

2 Chiffrements rapides sur CPU : OCB [KR17], MORUS [WH17] et AEGIS [WP17]

7. `http://competitions.cr.yp.to/caesar.html`

3 Chiffrements assurant une grande sécurité, même dans un scénario où le nonce est réutilisé : COLM [ABD+17], AES-COPA [ABD+16a], ELmD [ABD+16b] et Deoxys-II [JNPS17]

8.2 Description de Ketje Jr

Ketje [BDP+16] est une famille d'algorithmes de chiffrement authentifié qui a été soumise à la compétition CAESAR, et qui était encore en lice lors du troisième tour de la compétition mais n'a pas été retenue dans les finalistes. Ketje réutilise les composants de Keccak [BDPA13], la fonction de hachage gagnante de la compétition SHA-3. Ketje Jr et Ketje Sr sont deux algorithmes AEAD à bas coût dérivés de la famille Ketje. Ces algorithmes sont basés sur la construction MonkeyWrap [BDPA12], extrêmement proche de la construction duplex. Dans cette section, nous détaillons les caractéristiques de Ketje Jr, algorithme de chiffrement sur lequel nous avons proposé une cryptanalyse de type *diviser pour mieux régner*.

8.2.1 Le mode d'opération MonkeyWrap

Le mode d'opération MonkeyWrap [BDPA12] est un mode d'opération adapté au chiffrement authentifié et est similaire à la construction duplex décrite précédemment. Ce mode permet de traiter des suites de message clair, ainsi que les données associées. Comme ce n'est pas utile pour notre cryptanalyse, nous considérons le mode sans données associées. Le traitement du message clair M est le suivant.

Initialisation. Un état interne de taille b est noté A, et est initialisé avec la clef secrète K, de taille k bits[8], ainsi qu'un *nonce* N. L'état interne A est divisé en deux parties $A_r||A_c$ de tailles respectives r et c correspondant respectivement au *ratio* et à la *capacité* de la construction vue précédemment. Plus précisément, la valeur initiale de A est

$$enc(k/8)||K||pad_K||N||pad$$

où $enc(k/8)$ est un encodage de l'entier $k/8$, et où pad_K et pad sont des constantes de rembourrage fixées. Ensuite, la permutation Keccak-p que nous définirons plus tard est appliquée 12 fois, afin de "mélanger" la clef avec les valeurs publiques.

Les encodages utilisés ainsi que les manières de rembourrer l'état ne sont pas nécessaires à la compréhension ni à la faisabilité de l'attaque, car nous ne cryptanalysons pas la phase d'initialisation.

Traitement du texte clair. Le texte clair est rembourré et divisé en blocs de r bits que l'on note $M_0, \ldots, M_{\ell-1}$, qui sont traités de manière relativement simple.

8. Les tailles de clef recommandées par les auteurs se situent entre 96 bits et 182 bits.

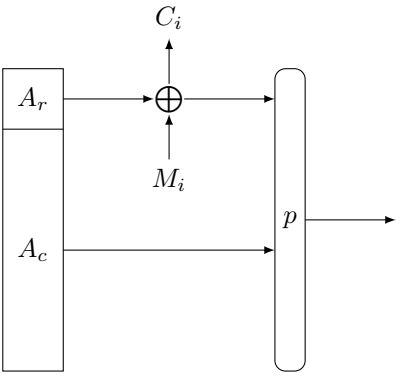

Figure 8.3 – Traitement du texte clair de KETJE.

Le i-ème bloc de chiffré est obtenu en suivant le principe des chiffrements à flot : $C_i = M_i + A_r$ et l'état interne A devient $C_i||A_c$. Finalement, la permutation KECCAK-p est appliquée, comme décrit à la figure 8.3.

Extraction de la marque. L'étape finale consiste à extraire la marque. Une fois que la totalité du texte clair est traitée, 6 tours de la permutation KECCAK-p sont appliqués à l'état interne, puis la marque est calculée comme une suite de blocs de r bits, qui sont définis par la partie A_r de l'état, lui-même mis à jour par l'application de KECCAK-p entre chaque extraction de r bits.

8.2.2 L'état interne de Ketje Jr

Comme nous montons une attaque sur KETJE JR, nous pouvons décrire celui-ci avec ses paramètres concrets. L'état interne de KETJE JR est de taille 200 bits, et est vu comme un tableau à trois dimensions d'éléments de $\mathbb{F}_2$, de taille $5 \times 5 \times 8$. Dans toute la suite, nous utiliserons le vocabulaire suivant :

- $A_{x,y,z}$ désigne le bit à la position (x, y, z) ;
- une *ligne* (*row*) est un ensemble de 5 bits où y et z sont constants $(A_{*,y,z})$;
- une *colonne* (*column*) est un ensemble de 5 bits où x et z sont constants $(A_{x,*,z})$;
- un *tube* (*lane*) est un ensemble de 8 bits où x et y sont constants $(A_{x,y,*})$;
- un *pan* (*sheet*) est un ensemble de 40 bits où x est constant $(A_{x,*,*})$;
- un *plan* (*plane*) est un ensemble de 40 bits où y est constant $(A_{*,y,*})$;
- une *tranche* (*slice*) est un ensemble de 25 bits où z est constant $(A_{*,*,z})$.

Les 200 bits de l'état interne sont numérotés de 0 à 199, en appliquant l'opération $(x, y, z) \mapsto 40x + 8y + z$. Dans cette partie de spécification de KETJE, nous reprenons les notations des auteurs. Chaque tranche est représentée par

une matrice de taille 5×5, où les indices sont notés à partir du centre où $(x, y) = (0, 0)$, *i.e.*,

$$T_z = A_{*,*,z} = \begin{bmatrix} (3,2) & (4,2) & (0,2) & (1,2) & (2,2) \\ (3,1) & (4,1) & (0,1) & (1,1) & (2,1) \\ (3,0) & (4,0) & (0,0) & (1,0) & (2,0) \\ (3,4) & (4,4) & (0,4) & (1,4) & (2,4) \\ (3,3) & (4,3) & (0,3) & (1,3) & (2,3) \end{bmatrix}$$

8.2.3 La permutation Keccak-p

La fonction de mise à jour de l'état interne notée KECCAK-p consiste en 5 étapes composées les unes aux autres :

$$\text{KECCAK-}p = \iota \circ \chi \circ \pi \circ \pi \circ \rho \circ \theta \ .$$

Nous décrivons chacune de ces opérations indépendamment

8.2.3.1 La fonction θ

La transformation θ est linéaire et décrite à la figure 8.4. Cette opération

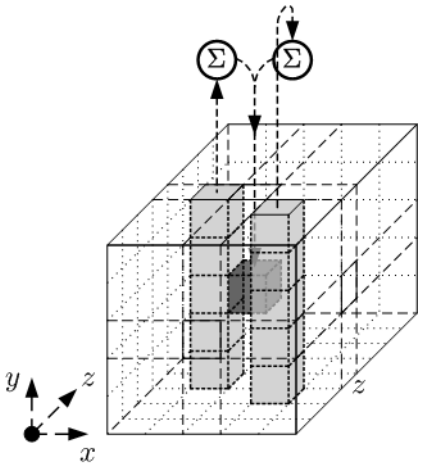

Figure 8.4 – Transformation d'un bit par θ.

apporte de la diffusion en calculant les bits de parité de deux colonnes (y constant) de l'état :

$$p(A)_{x,z} = \sum_{y=0}^{4} A_{x,y,z} \ .$$

Ces bits de parité des colonnes seront, pour alléger la lecture, souvent désignés par l'expression *bit de parité* car ce sont les seuls qui apparaissent dans ce travail.

Pour l'application θ, chaque bit $A_{x,y,z}$ de l'état est alors XORé avec les deux bits de parité $p(A)_{x-1,z}$ et $p(A)_{x+1,z-1}$, où les indices sont pris modulo 5 pour la coordonnée x et 8 pour z. L'application de θ est finalement donnée par la relation

$$\forall x, y, z,\ A_{x,y,z} \leftarrow A_{x,y,z} + p(A)_{x-1,z} + p(A)_{x+1,z-1}\ .$$

8.2.3.2 La fonction ρ

La fonction ρ est aussi une application linéaire, mais qui, elle, apporte de la diffusion entre les différentes tranches de l'état. L'application consiste en une rotation circulaire des bits de chaque tube indépendamment et est représentée à la figure 8.5. La sortie de ρ est donnée par la relation

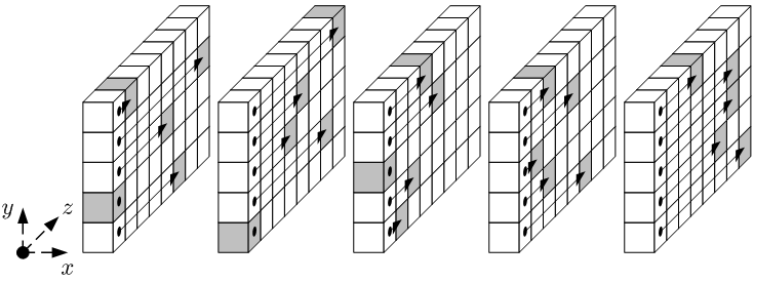

Figure 8.5 – L'application ρ.

$$\forall x, y, z,\ A_{x,y,z} \leftarrow A_{x,y,z-(t+1)(t+2)/2}$$

où t est compris entre 0 et 24 et satisfait la relation

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{F}_5^{2\times 2}$$

ou $t = -1$ si $x = y = 0$. Plus précisément, le nombre de positions de chaque tube est décalé selon les valeurs suivantes.

$$\begin{bmatrix} 1 & 7 & 3 & 2 & 3 \\ 7 & 4 & 4 & 4 & 6 \\ 4 & 3 & 0 & 1 & 6 \\ 0 & 6 & 2 & 2 & 5 \\ 5 & 0 & 1 & 5 & 7 \end{bmatrix}$$

8.2.3.3 La fonction π

L'application π est elle aussi linéaire et permet de "mélanger" les positions entre les tubes. Elle est appliquée sur chaque tranche indépendamment, de la manière décrite à la figure 8.6. Concrètement, l'application consiste à envoyer les

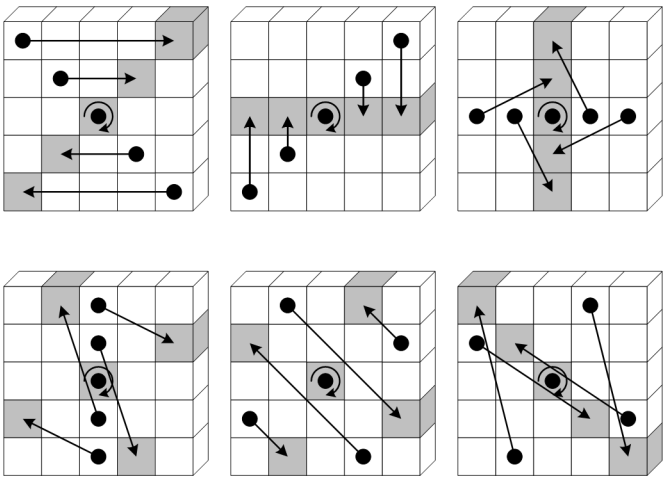

Figure 8.6 – L'application π.

bits à la position (x', y', z) à la position (x, y, z) où x, y, x', y' vérifient la relation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} .$$

8.2.3.4 La fonction χ

La fonction χ est la seule fonction non-linéaire dans la fonction de tour de Ketje Jr. Elle consiste en l'application en parallèle de 40 boîtes-S opérant sur chaque ligne (5 bits) indépendamment. χ est de degré algébrique 2, comme on peut le voir dans sa représentation à la figure 8.7. Principalement, chaque bit en sortie de l'application χ dépend de 3 bits en entrée appartenant à la même ligne, liés par la relation suivante.

$$\forall x, y, z,\ A_{x,y,z} \leftarrow A_{x,y,z} + (A_{x+1,y,z} + 1)A_{x+2,y,z} .$$

8.2.3.5 La fonction ι

ι est l'addition d'une constante de tour. Les constantes de tour sont définies de la manière suivante :

$$\mathsf{RC}_{i_r}[0, 0, 2^j - 1] = \mathrm{rc}[j + 7i_r] \text{ pour tout } 0 \leq j \leq \ell ,$$

où i_r désigne le numéro du tour et où $\ell = 3$ dans notre cas [9]. Les valeurs $\mathrm{rc}[t] \in \mathbb{F}_2$ sont définies par la sortie d'un registre à décalage à rétroaction linéaire (LFSR) de longueur 8 et de polynôme de rétroaction $x^8 + x^6 + x^5 + x^4 + 1$.

9. La valeur de ℓ dépend de la version de Ketje utilisée : la taille de l'état de Ketje vaut $b = 25 \times 2^\ell$, d'autres versions de Ketje étant censées assurer une meilleure sécurité, leur état peut être plus grand.

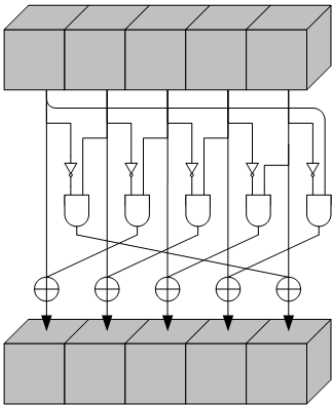

Figure 8.7 – La boîte-S de KECCAK : χ.

8.2.4 Génération de la suite chiffrante

La construction MonkeyWrap avec un *ratio* r consiste à extraire les r premiers bits de l'état à chaque itération. Comme les bits sont numérotés par leur coordonnée x, puis y, et enfin z, les bits sont extraits tube par tube. Pour KETJE v1 [BDP+14], tant que le ratio ne dépasse pas 40 bits, les bits extraits sont concentrés sur le premier plan ($y = 0$), où les tubes extraits correspondent à $x = 0, \ldots, \lceil r/8 \rceil$. Si le ratio est égal à 40 bits, alors tout le plan $y = 0$ est extrait, et nous montrerons en quoi ceci rend KETJE JR vulnérable. Pour KETJE JR, le ratio ne doit pas dépasser 16 bits.

Cependant, dans la deuxième version de KETJE [BDP+16], les auteurs ont modifié légèrement la permutation KECCAK-p, en prenant une variante de celle-ci appelée *permutation tordue* [10]. Cette permutation notée KECCAK-$p^\star$ est définie par la relation suivante.

$$\text{KECCAK-}p^\star[b, n_r] = \pi \circ \text{KECCAK-}p \circ \pi^{-1} \ .$$

En fait, l'application de cette nouvelle permutation est équivalente à appliquer la précédente (KECCAK-p). La seule différence est que le texte clair est ajouté et la suite chiffrante est produite dans et depuis des tubes différents. La suite chiffrante est obtenue en extrayant les bits de la *diagonale* satisfaisant l'équation $x = y$ pour $0 \leq x \leq \lceil r/8 \rceil$ (au lieu du plan $y = 0$).

La sécurité supposée par les concepteurs de KETJE JR est supposée être $\min(96, k)$ où k désigne la taille de la clef.

Depuis la compétition SHA-3, la permutation KECCAK décrite ci-dessous a suscité énormément d'intérêt et a été largement étudiée [BCD11, DGPW12, JN15]. Récemment, des versions réduites de KETJE ont été cryptanalysées avec des attaques par cube [11] [DLWQ17], ainsi que des techniques de linéarisation

10. Twisted permutation
11. cube attacks

[GLS16]. Cependant, les précédentes cryptanalyses de Ketje attaquent la phase d'initialisation, alors que nous travaillons sur la génération de la suite chiffrante.

8.3 Diviser pour mieux régner sur 3 blocs de sortie

L'idée principale de notre cryptanalyse de Ketje consiste à séparer l'état en deux parties, à construire deux listes couvrant toutes les valeurs possibles de chaque sous partie indépendamment, puis à fusionner ces deux listes en utilisant de l'information obtenue par les bits de suite chiffrante, *i.e.* une partie de l'état interne à des tours différents. Afin de comprendre plus précisément ce qui se passe, nous décrivons le principe général de notre attaque dans cette section. Ce principe ouvre une attaque sur les deux versions de Ketje qui utilise 3 blocs de sortie consécutifs. Les deux sections suivantes améliorent le principe général de l'attaque.

La stratégie appliquée est relativement générique, et nous montrons que la complexité de cet algorithme décroît lorsque le *ratio* augmente [12]. La complexité de cette attaque générique ne remet en aucun cas en cause la sécurité présumée par les auteurs de Ketje.

Nous utilisons 3 tours consécutifs. Donc, le nombre de bits d'information auxquels nous avons accès vaut $3 \times r$, ce qui implique immédiatement que la complexité d'une attaque par recouvrement de l'état ne peut être en-dessous de $2^{200-3\times r}$ opérations, qui est égal au nombre de solutions possibles étant donné le nombre de bits d'information auxquels nous avons accès. Ainsi, pour un *ratio* de 40 bits, il est impossible d'avoir une attaque dont le nombre d'opérations est plus petit que 2^{80}.

8.3.1 Attaque sur Ketje Jr v1 avec un ratio de 40 bits

Ici, on suppose que le *ratio* utilisé est égal à 40 bits, et nous travaillons sur Ketje Jr v1, c'est-à-dire qu'un plan entier ($y = 0$) est connu par l'attaquant.e à chaque tour.

8.3.1.1 L'attaque de type "diviser pour mieux régner"

Comme nous l'avons dit précédemment, notre attaque se fonde sur une technique de type *diviser pour mieux régner*. Le problème général que nous cherchons à résoudre peut être formulé de la manière suivante :

Soit deux ensembles $\mathcal{U}$ et $\mathcal{V}$, et deux fonctions $f : \mathcal{U} \to \mathbb{F}_2^c$, $g : \mathcal{V} \to \mathbb{F}_2^c$, ainsi qu'un élément $t \in \mathbb{F}_2^c$, trouver tous les éléments $(u, v) \in \mathcal{U} \times \mathcal{V}$ tels que $f(u) + g(v) = t$.

12. Ce qui est logique étant donné que le *ratio* correspond exactement à la quantité d'information à laquelle l'attaquant.e a accès.

Ce problème générique revient très souvent en cryptographie : par exemple dans le cadre des attaques de type *rencontre au milieu* [13] [DSP07, BR11, Sas11], ou encore des attaques par rebond [14] [LMR+09, KNRS11].

Algorithme générique. De manière générale, nous calculons toutes les valeurs $f(u)$ pour tout u dans $\mathcal{U}$ et enregistrons les valeurs $(u, f(u))$ dans une table, triée selon la valeur $f(u) + t$. Ensuite, nous calculons $g(v)$ pour tout $v \in \mathcal{V}$ afin de trouver une correspondance dans la table. Pour tout u qui satisfait $f(u)+t = g(v)$, (u, v) est une solution du problème.

Analyse de la complexité.

— La complexité en mémoire correspond ici au stockage des valeurs $(u, f(u))$ pour tout $u \in \mathcal{U}$. Il y a donc $|\mathcal{U}|$ éléments à stocker, chacun coûtant

$$\log_2(|\mathcal{U}|2^c) = \log_2(|\mathcal{U}|) + c$$

bits pour représenter ces éléments, ce qui donne une complexité en mémoire équivalente à $|\mathcal{U}|(\log_2(|\mathcal{U}|) + c)$ bits.

— La complexité en temps est, elle, dominée par le calcul de $|\mathcal{U}|$ fois la fonction f, ainsi que $|\mathcal{V}|$ fois la fonction g, ainsi que $|\mathcal{U}| \times |\mathcal{V}| \times 2^{-c}$ pour décrire l'ensemble des solutions si le système n'est pas dégénéré [15].

8.3.1.2 "Diviser pour mieux régner" sur Ketje Jr

Comme nous l'avons déjà dit, nous allons appliquer la technique précédente à trois blocs consécutifs de la suite chiffrante de KETJE JR, c'est-à-dire deux tours consécutifs de la fonction KECCAK-p.

Notations. On note A^0 l'état contenant le premier bloc de bits connus, B^0 la valeur de l'état après l'application de θ à l'état A^0, et C^0 la valeur de l'état après l'application de $\pi \circ \rho$ à l'état B^0. Ensuite, χ est appliquée et un nouveau bloc de 40 bits est extrait (le ratio est pris égal à $r = 40$). On note similairement au premier tour les états A^1, B^1 et C^1 les mêmes valeurs de l'état mais prises un tour plus tard, comme décrit à la figure 8.8. L'état à la fin du deuxième tour est noté de la même manière A^2.

Dans ces conditions, nous pouvons décrire notre attaque. L'idée consiste à retrouver la valeur de l'état A^1 en utilisant la stratégie de type "diviser pour mieux régner" décrite précédemment et en criblant les listes avec l'information contenue dans les états A^0 et A^2. Tout d'abord, nous remarquons qu'il est possible d'inverser immédiatement χ entre C^1 et A^2, puisque nous avons accès

13. meet-in-the middle attacks
14. Rebound attacks
15. A priori, c'est censé être le cas pour la majorité des systèmes cryptographiques : la valeur de t correspond à la valeur de c bits d'information, par lesquels on crible l'ensemble des possibilités.

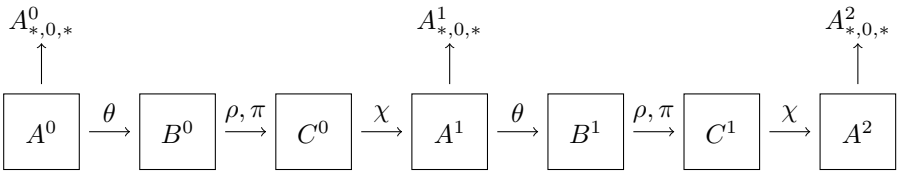

Figure 8.8 – Notations utilisées pour l'attaque sur 2 tours de Ketje Jr.

à un plan entier. Ensuite, nous exprimons $C^1_{*,0,*}$ comme une fonction de A^1, et $A^0_{*,0,*}$ comme une fonction de C^0. La clef de notre attaque est la suivante : comme θ, ρ, π ainsi que leurs inverses sont des permutations linéaires, nous pouvons diviser les états A^1 et C^0 en deux parties (A^u, A^v) (respectivement (C^u, C^v)).

Plus précisément, on définit A^u (respectivement C^u) comme les quatre premières tranches ($z = 0, 1, 2, 3$) de A^1 (respectivement C^1), ainsi que A^v (respectivement C^v) comme les quatre dernières tranches ($z = 4, 5, 6, 7$) de A^1 (respectivement C^1). On peut alors exprimer les bits du *plan* ($y = 0$) de C^1 (respectivement de A^0) en fonction de A^u et A^v (respectivement C^u et C^v).

$$\begin{array}{rcl} C^1_{*,0,*} & = & f_1(A^u) + g_1(A^v) \\ A^0_{*,0,*} & = & f_2(C^u) + g_2(C^v) \end{array}$$

où f_1, f_2, g_1 et g_2 sont des fonctions de $\mathbb{F}_2^{100}$ à valeurs dans $\mathbb{F}_2^{20}$.

Description de l'attaque. Maintenant nous pouvons décrire le cœur de notre attaque. On réalise une hypothèse sur la moitié de l'état A^u, de taille 100 bits. Cependant, pour chaque tranche, les 5 bits à la position $y = 0$ sont déjà connus, ce qui nous laisse 20 bits inconnus par tranche, impliquant un total de 80 bits sur lesquels nous faisons une hypothèse. Alors, nous pouvons calculer le résultat correspondant de C^u en appliquant χ^{-1} (voir figure 8.9). On définit ensuite f de la manière suivante :

$$f(A^u) = (f_1(A^u), f_2(C^u)) = (f_1(A^u), f_2'(A^u)) \ ,$$

où, comme 20 bits de A^u et 20 bits de C^u sont fixés, nous pouvons voir f comme une fonction de $\mathbb{F}_2^{80}$ à valeurs dans $\mathbb{F}_2^{40}$. En faisant exactement la même chose pour A^v et C^v, nous décrivons une fonction $g : \mathbb{F}_2^{80} \to \mathbb{F}_2^{40}$ avec la relation

$$g(A^u) = (g_1(A^v), g_2(C^v)) = (g_1(A^v), g_2'(A^v)) \ .$$

L'attaque consiste maintenant à appliquer la stratégie *diviser pour mieux régner* détaillée précédemment où $\mathcal{U}$ décrit les 2^{80} possibilités pour A^u, $\mathcal{V}$ décrit les 2^{80} possibilités pour A^v et $t = (C^1_{*,0,*}, A^0_{*,0,*})$. Dans ce cas, nous avons $|\mathcal{U}| = |\mathcal{V}| = 2^{80}$ et $c = 80$. Donc, la complexité de l'attaque est donnée par 2^{80}

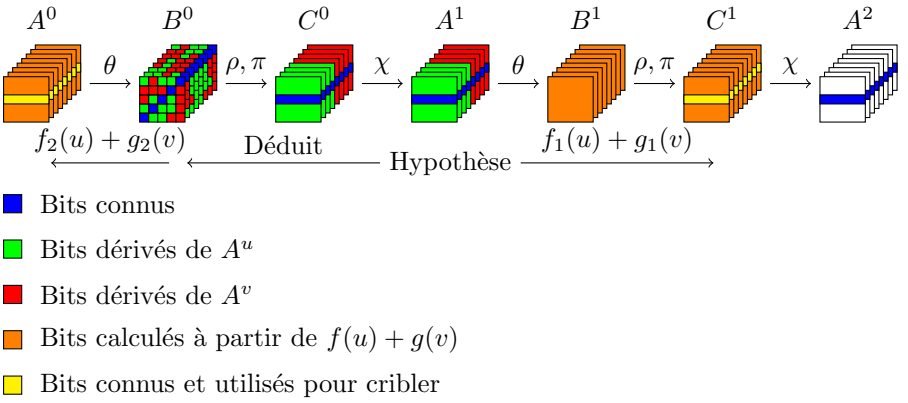

Figure 8.9 – Description de la stratégie de base sur KETJE JR.

évaluations de f et de g. Puis les 2^{80} candidats restant peuvent être parcourus par une recherche exhaustive qui a un coût de 2^{80}. L'idée générale de cette attaque est décrite à la figure 8.9

8.3.2 Attaque sur Ketje Jr v2 avec un ratio de 40 bits

L'attaque précédente était sur la première version de KETJE. Ici, nous montrons qu'il est possible de modifier légèrement ce qui précède afin de monter une attaque sur la deuxième version de KETJE, *i.e.* celle avec la permutation "tordue".

En regardant étape par étape l'attaque précédente, on remarque que le seul point qui change est l'inversion de l'étage non-linéaire entre C^1 et A^2. En effet, toutes les autres étapes fonctionnent pareillement, mais comme les bits connus en A^2 appartiennent à la première diagonale et non au même plan, ils n'appartiennent donc pas aux mêmes lignes, ce qui rend impossible l'inversion de χ. Cependant, nous allons montrer qu'il est possible de cribler directement avec l'information contenue en A^2 sans appliquer χ^{-1}.

Afin de parvenir à nos fins, nous devons modifier sensiblement notre algorithme de criblage. En effet, l'attaque sur la version initiale de KETJE JR consiste à exprimer les bits connus comme la somme de deux valeurs calculées indépendamment de $u \in \mathcal{U}$ et $v \in \mathcal{V}$. Or, maintenant, notre système fait apparaître une expression non-linéaire en u et v. Dans ces conditions, nous montrons comment adapter l'algorithme dans notre cas particulier, sans pour autant en augmenter significativement la complexité.

8.3.2.1 La stratégie de type "diviser pour mieux régner" avec des équations non-linéaires

Nous définissons le problème suivant en utilisant les notations précédentes comme une généralisation du problème initial. Soit α, β et γ trois entiers. Nous considérons l'ensemble de fonctions suivantes :

— 2α fonctions $f_i : \mathcal{U} \to \{0,1\}$ et $g_i : \mathcal{V} \to \{0,1\}$, pour $0 \leq i \leq \alpha - 1$;

— 3β fonctions $f'_i : \mathcal{U} \to \{0,1\}$ et $g_i, g'_i : \mathcal{V} \to \{0,1\}$, pour $\alpha \leq i \leq \alpha + \beta - 1$;

— 3γ fonctions $f_i, f'_i : \mathcal{U} \to \{0,1\}$ et $g'_i : \mathcal{V} \to \{0,1\}$, pour $\alpha + \beta \leq i \leq \alpha + \beta + \gamma - 1$.

Avec ces notations, le problème que l'on considère consiste à décrire l'ensemble des solutions $(u, v) \in \mathcal{U} \times \mathcal{V}$ qui vérifient l'ensemble des équations suivantes :

$$f_i(u) + g_i(v) = 0 \text{ for } 0 \leq i \leq \alpha - 1 \tag{8.1}$$

$$f'_i(u)g'_i(v) + g_i(v) = 0 \text{ for } \alpha \leq i \leq \alpha + \beta - 1 \tag{8.2}$$

$$f_i(u) + f'_i(u)g'_i(v) = 0 \text{ for } \alpha + \beta \leq i \leq \alpha + \beta + \gamma - 1 \tag{8.3}$$

Dit autrement, nous considérons un système de $c = \alpha + \beta + \gamma$ équations, où nous nous autorisons des équations ne faisant intervenir qu'un seul produit entre les éléments de u et de v. Il est à noter qu'un problème similaire a été considéré dans [LN15].

8.3.2.2 Lien avec Ketje

Pour comprendre pourquoi nous nous intéressons à ce problème spécifique, il est nécessaire de regarder en détail l'application de la couche non-linéaire de Ketje. En effet, la couche non-linéaire de Ketje effectue seulement un seul produit pour chaque bit en sortie : $(a, b, c) \to a + bc$. Ainsi, nous avons à faire à un système spécifique dans notre contexte pour lequel :

— chaque bit en entrée de χ peut être calculé en fonction de A^u et A^v de manière linéaire ;

— quelques bits en sortie sont connus par l'attaquant.e.

Comme chaque bit en entrée de χ (l'application entre C^1 et A^2) peut être calculé en fonction de A^u et A^v de manière linéaire, alors il convient que toutes les équations que l'on obtient sont toutes de la forme (8.1), (8.2) ou (8.3), et l'on se retrouve exactement dans le cadre du problème décrit ci-dessus.

8.3.2.3 La stratégie modifiée

Comme dans le cas du problème initial, notre solution consiste à construire deux listes triées telles que chaque solution à notre problème puisse être calculée en fusionnant ces deux listes d'une certaine manière. Nous voulons aussi que notre algorithme n'ait pas de faux positifs, mais surtout qu'il ne rate aucune solution. Dans ce qui suit nous expliquons comment construire la liste $L_{\mathcal{U}}$ correspondant à $\mathcal{U}$.

Nous construisons $L_{\mathcal{U}}$ de la manière suivante : chaque élément de cette liste est un couple (F, u) où F est une valeur encodée sur $c = \alpha + \beta + \gamma$ bits. Le bit à la position i de la valeur F sera alors comparé avec le même bit (i) des éléments $(G, v) \in L_{\mathcal{V}}$ où $v \in \mathcal{V}$ et G est aussi une valeur encodée sur c bits, afin de vérifier les équations (8.1), (8.2) et (8.3).

8.3.2.4 Prendre en compte les équations indépendamment

Afin de résoudre le problème efficacement, nous décrivons comment prendre en compte chaque équation. Pour chaque valeur u et v, ainsi que pour chaque équation (indexée par i allant de 0 à $c-1$), nous définissons les ensembles $R_i(u) \subseteq \mathbb{F}_2$ et $S_i(v) \subseteq \mathbb{F}_2$ de manière à ce que le couple (u, v) est solution de l'équation (i) si et seulement si $R_i(u) \cap S_i(v) \neq \emptyset$.

Cas (8.1) Ce cas correspond au problème initial de résolution d'équations linéaires où les contributions de u et de v sont indépendantes. L'équation est satisfaite si et seulement si $f_i(u) = g_i(v)$. Dans ce cas, $R_i(u) = \{f_i(u)\}$ et $S_i(v) = \{g_i(v)\}$.

Cas (8.2) Dans ce cas là, seulement $f'_i(u)$ est dépendante de u dans l'équation. Donc, nous fixons $R_i(u) = \{f'_i(u)\}$. Pour construire $S_i(v)$, nous prenons les valeurs de $g'_i(v)$ et de $g_i(v)$. Alors selon leur valeur il est nécessaire de construire différents ensembles $S_i(v)$. Typiquement, si $g'_i(v) = 1$, alors l'équation $f'_i(u)g'_i(v) + g_i(v) = 0$ devient $f'_i(u) = g_i(v)$, et donc nous devons fixer $S_i(v)$ à $\{g_i(v)\}$ comme pour le cas (8.1). Si en revanche $g'_i(v) = 0$, alors l'équation devient $g_i(v) = 0$. Dans ce cas, nous devons prendre en compte la valeur de $g_i(v)$: si $g_i(v) = 1$, alors l'équation (i) ne peut être satisfaite, et donc $S_i(v) = \emptyset$; si $g_i(v) = 0$, alors l'équation est toujours satisfaite, et donc nous fixons $S_i(v)$ à la valeur $\{0, 1\}$.

Cas (8.3) Ce dernier cas est symétrique par rapport au précédent en échangeant simplement le rôle de u et de v, et nous le traitons donc de manière équivalente.

Une telle construction des ensembles $R_i(u)$ et $S_i(v)$ pour tout $u \in \mathcal{U}$ et $v \in \mathcal{V}$ et $0 \leq i \leq c-1$ respecte bien la propriété suivante :

$$\forall i, u, v, R_i(u) \cap S_i(v) \neq \emptyset \Leftrightarrow (u, v) \text{ satisfait l'équation } (i) \ .$$

Afin de prendre en compte toutes les équations simultanément, nous considérons le produit cartésien des ensembles définis ci-dessus, *i.e.*

$$\begin{aligned} R(u) &= R_0(u) \times R_1(u) \times \cdots \times R_{c-1}(u) \\ S(v) &= S_0(v) \times S_1(v) \times \cdots \times S_{c-1}(v) \ . \end{aligned}$$

Ainsi, si (u, v) est une solution de toutes les équations, alors par construction il y aura une collision entre les ensembles $R(u)$ et $S(v)$ et réciproquement.

8.3.2.5 Résoudre le problème

Une fois cette propriété assurée, il nous est possible de décrire un algorithme générique qui résout le problème, ainsi que d'en évaluer la complexité.

Notre algorithme consiste à construire la liste $L_{\mathcal{U}}$ en énumérant tous les éléments (F, u), où F est calculée en fonction de $R(u)$, pour tout $u \in \mathcal{U}$. Les éléments sont ensuite triés en fonction de la valeur de F. La même chose est réalisée pour $L_{\mathcal{V}}$. Alors, l'ensemble des solutions (u, v) à notre problème peut être retrouvé en cherchant des collisions dans ces listes.

La valeur de F est calculée de la manière suivante : pour les $\alpha + \beta$ premières équations, on concatène seulement les valeurs de $f_i(u)$ pour i allant de 0 à $\alpha - 1$ (c'est le cas linéaire classique). Pour les équations i , i allant de α à $\beta - 1$, alors, comme u n'est impliqué qu'une seule fois dans l'équation, on concatène la valeur $f'_i(u)$ aux autres, comme cela est décrit dans l'algorithme 5. Ensuite l'algorithme 4 est appliqué, afin de construire la liste $L_{\mathcal{U}}$ complète en complétant par les γ équation restantes. Les mêmes algorithmes sont utilisés pour construire $L_{\mathcal{V}}$ par symétrie du problème.

Cependant, comme nous pouvons le voir à la description des algorithmes, la valeur concaténée aux valeurs de u dans la liste est bien de taille $c = \alpha + \beta + \gamma$. Or, d'un côté, certaines valeurs de u (et de v) ne seront pas ajoutés à la liste si une équation de type (8.2) ou (8.3) ne peut jamais être satisfaite, ce qui diminue potentiellement la taille des listes ; d'un autre côté, lorsque une équation de type (8.2) ou (8.3) est toujours satisfaite, alors l'élément u (ou v) correspondant est compté deux fois dans la liste, ce qui augmente la taille des listes construites. Il est donc nécessaire d'évaluer correctement la taille des listes $L_{\mathcal{U}}$ et $L_{\mathcal{V}}$.

8.3.2.6 Taille des listes

Nous devons évaluer la taille des listes car c'est un paramètre essentiel pour évaluer la complexité de la résolution de notre problème. Comme nous l'avons dit précédemment, la taille des listes est liée aux valeurs $(f_i(u), f'_i(u))$, respectivement $(g_i(u), g'_i(u))$.

Comme u et v jouent un rôle symétrique, nous faisons l'analyse sur u. Selon l'algorithme 4, une valeur $u \in \mathcal{U}$ est ajoutée à la liste si et seulement si aucune des valeurs de $(f_i(u), f'_i(u))$ pour i allant de $\alpha + \beta$ à $c - 1$ n'est égale à $(1, 0)$, sinon il existe une équation impossible à satisfaire, peu importe la valeur de $v \in \mathcal{V}$. En considérant que les fonctions que l'on manipule sont équilibrées (ce qui est le cas dans Ketje car elles sont affines), la probabilité qu'une valeur u apparaisse dans la liste $L_{\mathcal{U}}$ est donc égale à $(3/4)^{\gamma}$.

Nous notons alors ω cet événement, et nous notons $N(u)$ le nombre de valeurs ajoutées à la liste $L_{\mathcal{U}}$ qui correspondent à la valeur u.

Toujours selon la description de l'algorithme 4, le nombre de fois qu'une valeur fixée u est ajoutée à la liste est directement lié au nombre d'équations qui sont toujours satisfaites, peu importe la valeur de v, *i.e.* le nombre de fois où l'on

Algorithme 4 Construction de L_u, liste d'éléments Z tels que les équations $\alpha + \beta$ à $\alpha + \beta + \gamma - 1$ sont satisfaites pour tout (u, v) tel que $Z = (g'_{\alpha+\beta}(v), \ldots, g'_{\alpha+\beta+\gamma-1}(v))$.

Entrée: $u \in \mathcal{U}$
Sortie: une liste L_u contenant tous les éléments $R_{\alpha+\beta}(u) \times \ldots \times R_{\alpha+\beta+\gamma-1}(u)$

$L_u \leftarrow (\epsilon)$ // Initialiser L_u avec la chaîne vide
pour $i = \alpha + \beta$ **à** $\alpha + \beta + \gamma - 1$ **faire**
    // Vérifier si une équation n'est jamais satisfiable
    si $f_i(u) = 1$ et $f'_i(u) = 0$ **alors**
        Renvoyer ()
    fin si
fin pour
pour $i = \alpha + \beta$ **à** $\alpha + \beta + \gamma - 1$ **faire**
    si $f'_i(u) = 1$ **alors**
        pour $Z \in L_u$ **faire**
            Ajouter $f_i(u)$ à Z
        fin pour
    else
        //$f_i(u) = 0$ et $f'_i(u) = 0$, *i.e.* équation toujours satisfaite
        pour $Z \in L_u$ **faire**
            Remplacer Z avec deux éléments $Z||0$ et $Z||1$ dans L_u
        fin pour
    fin si
fin pour
    Renvoyer L_u

Algorithme 5 Énumération des valeurs de $L_{\mathcal{U}}$.

Entrée: $\mathcal{U}$, fonctions f_i, $0 \leq i < \alpha$ et $\alpha + \beta \leq i < \alpha + \beta + \gamma$, fonctions f'_i, $\alpha \leq i < \alpha + \beta + \gamma$
Sortie: La liste $L_{\mathcal{U}}$

$L_{\mathcal{U}} \leftarrow ()$
pour $u \in \mathcal{U}$ **faire**
    Calculer $X = f_0(u)|| \ldots ||f_{\alpha-1}(u)$
    Calculer $Y = f'_\alpha(u)|| \ldots ||f'_{\alpha+\beta-1}(u)$
    Calculer L_u en utilisant l'algorithme 4
    pour $Z \in L_u$ **faire**
        Ajouter $(X||Y||Z, u)$ à $L_{\mathcal{U}}$
    fin pour
fin pour
    Renvoyer $L_{\mathcal{U}}$

a $(f_i(u), f'_i(u)) = (0, 0)$ pour i allant de $\alpha + \beta$ à $c - 1$. Plus précisément, lorsque ce nombre est égal à j, u apparaît exactement 2^j fois, c'est-à-dire $N(u) = 2^j$.

Dans ces conditions, toujours en supposant que nos fonctions sont équilibrées, nous avons

$$\Pr[N(u) = 2^j|\omega] = \binom{\alpha}{j}\left(\frac{1}{3}\right)^j\left(\frac{2}{3}\right) .$$

Donc, l'espérance de $N(u)$ vaut

$$\begin{aligned}\mathbb{E}[N(u)] &= \sum_{j=0}^{\alpha}\left(\Pr\left[N(u) = 2^j|\omega\right]\Pr\left[\omega\right] \times 2^j\right)\\ &= (3/4)^\alpha \times \sum_{j=0}^{\alpha}\binom{\alpha}{j}(1/3)^j(2/3)^{\alpha-j}2^j\\ &= (3/4)^\alpha \times (2/3 + 2/3)^\alpha\\ &= 1.\end{aligned}$$

Ainsi, la taille de la liste $L_{\mathcal{U}}$ construite en utilisant l'algorithme 5 est donc $|\mathcal{U}|$, et le nombre de valeurs v testées est approximativement $|\mathcal{V}|$.

8.3.2.7 Complexité

Il semble possible d'optimiser les algorithmes décrits précédemment, cependant comme la complexité de cet algorithme ne sera pas un facteur limitant pour notre attaque sur Ketje Jr, nous utilisons l'algorithme initial. Pour chaque valeur de $u \in L_{\mathcal{U}}$, nous devons calculer exactement $\alpha + \beta + 2\gamma$ fonctions booléennes f_i et f'_i (construction de la liste $L_{\mathcal{U}}$). Ensuite, pour chaque équation parmi les γ dernières, nous devons réaliser 2 comparaisons afin de déterminer quelles valeurs doivent être ajoutées à la liste. Chaque valeur ajoutée dans cette partie requiert au plus γ modifications [16] dans la sous-liste correspondant à l'élément u considéré, ainsi qu'une insertion dans la liste $L_{\mathcal{U}}$. La complexité est donc bornée supérieurement par $2c(|\mathcal{U}| + |\mathcal{V}|)$ calculs de fonctions et comparaisons et $c(|\mathcal{U}| + |\mathcal{V}|)$ comparaisons, étant donné que la taille de la liste $L_{\mathcal{U}}$ est $|\mathcal{U}|$ et que la taille de $L_{\mathcal{V}}$ est $|\mathcal{V}|$.

Les listes étant construites, d'après les remarques précédentes, notre algorithme recherche toutes les valeurs (u, v) valides de la même manière que l'algorithme qui crible avec des relations linéaires. En effet, la manière dont nous avons construit nos nouvelles listes permet de considérer notre problème comme dans le cas linéaire, puisque nous avons, en quelque sorte "linéarisé" nos équations en séparant les différents cas selon les valeurs de $(f_i(u), f'_i(u))$, respectivement $(g_i(v), g'_i(v))$. Dans ces conditions, l'étape de criblage de notre algorithme correspond à la recherche des bonnes valeurs (u, v) dans des listes de tailles respectives $|\mathcal{U}|$ et $|\mathcal{V}|$, ce qui est équivalent au cas où le criblage est linéaire.

8.3.2.8 Application à Ketje Jr v2

En utilisant l'algorithme précédent, nous pouvons monter une attaque en utilisant le même principe, mais sur la deuxième version de Ketje, avec la

16. Le pire cas étant que toutes les γ équations soient satisfaites indépendamment la valeur de v.

permutation "tordue". Nous reprenons les notations de la section 8.3.1.

L'état A^1 est divisé en deux parties A^u et A^v, où nous augmentons la taille des deux listes qui correspondent à l'énumération de l'ensemble des deux moitiés d'état possibles. De plus, nous ajoutons 5 bits redondants correspondant à des bits de parité de colonnes. Plus précisément, A^u est défini comme les quatre premières tranches ($0 \leq z \leq 3$) ainsi que les 5 bits de parité des colonnes $A^1_{i,*,7}$, pour $i \in \{0,1,2,3,4\}$, et A^v est défini comme les quatre dernières tranches ($4 \leq i \leq 7$) ainsi que les 5 bits de parité des colonnes $A^1_{i,*,3}$, pour $i \in \{0,1,2,3,4\}$. En ajoutant ces bits de parité qui correspondent à des bits redondants de l'information contenue dans l'autre liste, l'application θ devient complètement transparente dans le sens où B^u et B^v (les 4 premières et 4 dernières tranches de B^1) peuvent être directement calculées à partir de A^u et A^v indépendamment.

Pour la moitié de l'état *augmenté* [17] A^u, nous connaissons la valeur contenue à la position $y = 0$ (par la tranche $A^1_{*,0,*}$). Donc, la taille de la liste couvrant l'ensemble des états possibles est de 2^{85}, car l'état est constitué de 100 bits, plus 5 bits de parité, moins 20 bits connus.

Comme nous avons rajouté 5 bits de parité des colonnes dans chaque liste, et que cette information est redondante, cela nous permet de rajouter 10 équations linéaires entre les deux moitiés de l'état, que nous rajoutons aux équations initiales. En d'autres termes, il existe f_1 et g_1 deux fonctions de $\{0,1\}^{85} \to \{0,1\}^{10}$ telles que

$$0 = f_1(A^u) + g_1(A^v) \ .$$

Pour les mêmes raisons que dans le cas classique où les relations sont linéaires, nous avons aussi deux fonctions f_2 et g_2 de $\{0,1\}^{85} \to \{0,1\}^{40}$ telles que

$$A^0_{*,0,*} = f_2(A^u) + g_2(A^v) \ .$$

Application à Ketje Jr v1. Si on reprend l'attaque décrite initialement sur KETJE JR v1, alors comme l'étage non-linéaire peut être inversé directement et que π et ρ ne sont que des applications qui échangent les positions des bits, cela signifie que dans ce cas, nous avons accès directement à 20 bits de plus connus dans B^u, et 20 bits connus dans B^v. Comme B^u et B^v sont calculés de manière transparente par A^u et A^v, cela signifie que la taille des listes représentant l'ensemble des valeurs possibles pour les deux sous-parties de l'état devient 2^{65}, en utilisant directement l'information de A^1 et de A^2. Dans ce cas, il existe f_1 et g_1 deux fonctions de $\{0,1\}^{65} \to \{0,1\}^{10}$ telles que

$$0 = f_1(A^u) + g_1(A^v) \ .$$

ainsi que deux fonctions f_2 et g_2 de $\{0,1\}^{65} \to \{0,1\}^{40}$ telles que

$$A^0_{*,0,*} = f_2(A^u) + g_2(A^v) \ .$$

Chaque liste étant de taille 2^{65}, la complexité totale de l'attaque est donc, dans ce cas, dominée par la recherche exhaustive après avoir criblé et fusionné

17. en rajoutant 5 bits de parité des colonnes

les deux listes, *i.e.* 2^{80}. L'attaque améliorée utilisant 3 blocs de sortie consécutifs sur Ketje Jr v1 avec un ratio de 40 bits est décrite à la figure 8.10

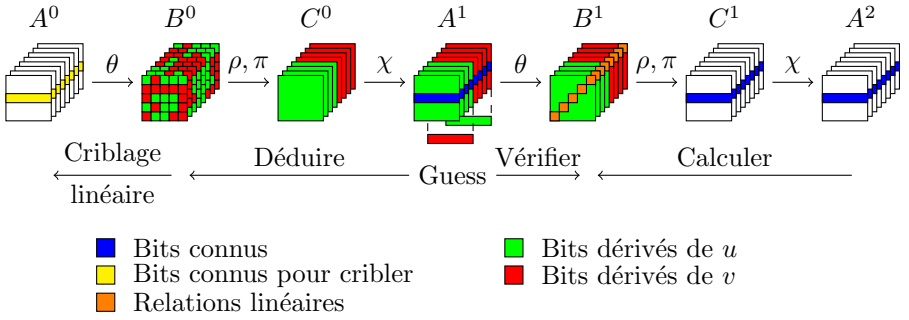

Figure 8.10 – Attaque de type diviser pour mieux régner sur Ketje Jr v1 avec un ratio de 40 bits et bits de parité.

Avec un ratio plus petit. Peu importe l'utilisation de la deuxième version ou de la première version de Ketje Jr, utiliser des sous-parties de l'état augmenté en rajoutant les bits de parité des colonnes nous permet de passer l'application θ sans encombre. De plus comme nous pouvons le voir à la figure 8.11, ρ et π ne consistent qu'en des échanges de bits : les bits de l'état C^1 sont directement dérivés d'un unique bit de B^u ou de B^v. De plus, les bits de sortie de l'application χ sont obtenus par la relation

$$\forall x, y, z,\ A_{x,y,z} \leftarrow A_{x,y,z} + (A_{x+1,y,z} + 1)A_{x+2,y,z}\ ,$$

ce qui nous permet d'assurer directement que les équations décrivant l'information contenue dans A^2 sont nécessairement de la forme (8.1), (8.2) ou (8.3), indépendamment de la position des bits produits en sortie dans l'état A^2.

Donc, l'attaque se généralise en utilisant l'algorithme 5 à Ketje v1 ou v2, quel que soit le *ratio* utilisé [18].

8.3.2.9 Analyse de la complexité

Soit r le ratio utilisé. Alors, chaque sous-partie d'état consiste en quatre tranches de 25 bits et 5 bits de parité de A^1, mais $r/2$ bits sont connus dans chacune de ces sous-parties. Dans ces conditions, le temps nécessaire pour construire une liste est de

$$T_{\text{liste}} = 2^{105-r/2}\ .$$

Ensuite, le nombre de solutions donné par notre technique de type "diviser pour mieux régner" adaptée à certaines équations non-linéaires dépend du nombre

18. La complexité de l'attaque reste naturellement dépendante du ratio.

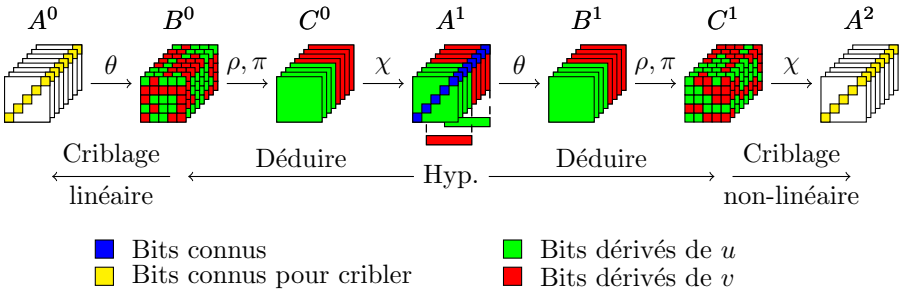

Figure 8.11 – Schéma de l'attaque par criblage non-linéaire.

d'équations disponibles. Nous avons par définition du ratio, exactement r relations de criblage venant de A^2, r venant de A^0, ainsi que 10 venant des bits de parité. Dans ces conditions, le nombre de valeurs restantes après criblage est

$$\begin{aligned} T_{\text{recherche}} &= 2^{105-r/2} \times 2^{105-r/2} \times 2^{-10-r-r} \\ &= 2^{200-3r} \ , \end{aligned}$$

ce qui n'est pas surprenant puisque $3r$ est exactement le nombre de bits d'information auxquels nous avons accès, parmi les 200 bits de l'état interne.

Finalement, quand le ratio est plus petit que 40, c'est le coût de la recherche exhaustive qui domine la complexité de l'attaque, alors que lorsque le ratio vaut 40 bits, c'est le coût de la description des listes $L_{\mathcal{U}}$ et $L_{\mathcal{V}}$ qui domine la complexité.

Amélioration de la complexité pour Ketje Jr v2, $r = 40$. En regardant les détails de l'application formée de la composition de ρ et de π, on constate que les bits $A^2_{1,1,1}$, $A^2_{1,1,2}$, $A^2_{2,2,6}$, $A^2_{2,2,7}$, sont calculables à partir de B^v uniquement (donc de A^v) et que les bits $A^2_{1,1,5}$, $A^2_{1,1,6}$, $A^2_{2,2,2}$ et $A^2_{2,231}$ sont calculables à partir de B^u uniquement (donc de A^u). Le.a lecteur.rice peut vérifier cela à la figure 8.15 page 225. Or, comme ces bits sont connus car on regarde la deuxième version de KETJE JR et que le ratio vaut 40, cela nous permet de cribler en amont les deux listes $L_{\mathcal{U}}$ et $L_{\mathcal{V}}$ indépendamment, réduisant la taille de ces listes de 2^{85} à 2^{81}, ce qui donne une complexité totale de l'ordre de $2 \times 2^{81} + 2^{80} \approx 2^{82}$.

8.3.2.10 Résultats de l'attaque avec 3 blocs consécutifs

Pour KETJE JR v2 avec $r = 40$, nous avons pu améliorer sensiblement la complexité de l'attaque en regardant précisément les détails des applications ρ et π. Pour KETJE JR v1 avec $r = 40$, nous avons aussi vu que la complexité est en réalité dominée par le coût de la recherche exhaustive. Pour toutes les autres versions, dès que le ratio est plus petit, c'est le coût de la recherche exhaustive

après criblage qui domine la complexité totale de l'attaque. Les résultats sont résumés à la table 8.2.

Version	Ratio	Complexité
Ketje Jr v1 et v2	16	2^{152}
Ketje Jr v1 et v2	24	2^{128}
Ketje Jr v1 et v2	32	2^{104}
Ketje Jr v2	40	2^{82}
Ketje Jr v1	40	2^{80}

Table 8.2 – Complexités des attaques sur 2 tours de Ketje Jr.

D'après ces résultats, comme c'est le coût de la recherche exhaustive qui domine la complexité de l'attaque, cela signifie que l'on peut difficilement faire mieux sans utiliser d'autres bits d'information. En effet, dans la description de l'attaque précédente, absolument toute l'information des 3 blocs de sortie est utilisée. Dans ces conditions, il convient de considérer non pas 3 blocs consécutifs, mais 4, afin d'espérer pouvoir améliorer notre attaque sur des ratios plus petits, et c'est ce que nous faisons dans la suite de ce chapitre.

8.4 Attaque sur 4 blocs consécutifs

Nous décrivons dans cette section une attaque sur Ketje Jr v1, *i.e.* sans la permutation "tordue". Cette attaque utilise de l'information contenue dans 4 sorties consécutives, avec un ratio égal à 40 bits, quitte à diminuer celui-ci par la suite en vue d'une amélioration.

Dans ce cas, le plus petit nombre de solutions ne peut être plus petit que $2^{200-4r} = 2^{40}$. Le principe de notre attaque est dans la même ligne que ce que nous avons décrit précédemment. Cependant, nous devons utiliser un autre criblage non-linéaire, en utilisant la suite chiffrante extraite de l'état A^3, où les notations sont les mêmes que précédemment. Cribler avec l'information contenue dans A^3 est la partie la plus compliquée de notre attaque, et nous emploierons deux méthodes différentes pour y parvenir. Comme nous avons exactement 40 bits de ratio, et que nous nous intéressons à la première version de Ketje Jr, cela signifie que nous pouvons inverser la couche non-linéaire (χ), et que cribler avec A^3 revient en réalité à cribler avec B^2. En effet, en inversant la couche linéaire et en appliquant l'application inverse de $\rho \circ \pi$, nous retrouvons la valeur de l'ensemble des tubes $B^2_{i,i,*}$.

Pour cribler nos deux listes, nous utiliserons deux méthodes : la première, décrite à la section 8.4.1 consiste à fixer certains bits de l'état, afin de rendre le passage des applications non-linéaires plus simple, et la deuxième méthode, décrite à la section 8.4.2 consiste à utiliser un algorithme de fusion de listes comme dans [Nay11], puis affiné dans [CNV13] avec des idées provenant de [DDKS12] :

l'algorithme de correspondance instantané et de correspondance parallèle [19] sans mémoire.

Nous considérons donc 4 blocs consécutifs de KETJE JR v1, ce qui couvre 3 tours consécutifs. On coupe donc l'état A^1 en deux parties comme précédemment, en rajoutant les bits de parité des colonnes, puis nous allons cribler et fusionner ces deux listes couvrant l'état augmenté en utilisant l'information contenue dans A^0, A^2 et A^3. Comme on travaille sur KETJE JR v1 avec un ratio de 40 bits, il est possible de calculer les plans $C^0_{*,0,*}$, $C^1_{*,0,*}$ et $C^2_{*,0,*}$ en appliquant χ^{-1}, puis, en appliquant $(\pi \circ \rho)^{-1}$ nous pouvons calculer l'ensemble des tubes $B^0_{i,i,*}$, $B^1_{i,i,*}$ et $B^2_{i,i,*}$ pour $i \in \{0,1,2,3,4\}$ comme décrit à la figure 8.12.

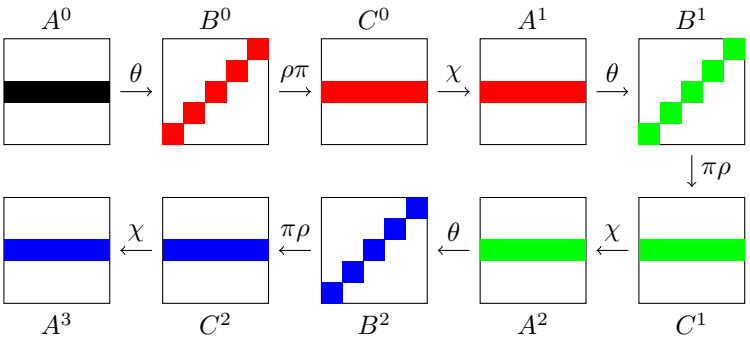

Figure 8.12 – Représentation de 3 tours de KETJE. Chaque partie colorée correspond à des tubes pouvant être directement calculés à partir des 4 blocs de suite chiffrante connus.

Nous découpons l'état A^1 en deux parties A^u et A^v comme nous le faisons depuis le début en y ajoutant les 5 bits de parité des colonnes. Ainsi, chaque liste décrivant l'ensemble des deux sous-parties de l'état est de taille 2^{65} : il y a 100 bits pour la moitié de l'état, plus 5 bits de parité des colonnes, moins 20 bits correspondant à l'information contenue dans A^1 ainsi que 20 bits contenus dans B^1, calculables de manière transparente grâce aux 5 bits de parité et provenant de l'information contenue dans A^2.

Nous avons toujours 10 équations linéaires qui lient les deux parties de l'état dues aux bits de parité des colonnes ainsi que 40 conditions provenant de l'information contenue dans A^1. Donc, il existe f_1 et g_1 deux fonctions de $\{0,1\}^{65}$ dans $\{0,1\}^{10}$ telles que

$$0 = f_1(A^u) + g_1(A^v) \ ,$$

ainsi que deux fonctions f_2 et g_2 de $\{0,1\}^{65}$ dans $\{0,1\}^{40}$ telles que

$$A^0_{*,0,*} = f_2(A^u) + g_2(A^v) \ .$$

19. instant matching et parallel matching

0	2	1	6	1
5	5	4	4	3
5	6	0	4	3
1	2	6	7	7
7	0	3	4	2

Figure 8.13 – Valeurs de décalage de l'application $\pi \circ \rho$: positions dans chaque tube des bits de la tranche $z = 0$ avant application de $\pi \circ \rho$.

8.4.1 Première méthode : fixer quelques bits

L'idée de cette technique consiste à fixer un certain nombre de bits n_b de manière à ce que l'application χ entre C^1 et A^2 devienne linéaire. Si n_b bits sont fixés, notre attaque doit être réalisée pour chaque valeur possible de ces n_b bits, multipliant la complexité finale par 2^{n_b}, mais faisant aussi décroître la taille de nos listes.

Pour décider des n_b bits que nous allons fixer, nous regardons en détail l'application de $\pi \circ \rho$ sur chaque tranche indépendamment, puisque nous fonctionnons tranche par tranche. Plus précisément, si B désigne l'état avant l'application de $\pi \circ \rho$, alors chaque bit dans le tube $B_{0,4,*}$ restera dans la même tranche, chaque bit de $B_{1,4,*}$ sera décalé de 2 tranches,... La valeur de ces décalages est résumée à la figure 8.13 et correspond à ce que les auteurs de KETJE ont appelé les compensations des décalages [20] de ρ. Ainsi, en regardant bit par bit, nous nous intéressons à la position exacte dans laquelle les bits de A^u et de A^v se retrouvent dans l'état C^1 après l'application de $\pi \circ \rho$. Or, nous devons encore traverser l'application χ, non-linéaire, ainsi que l'application θ, qui se traverse facilement en partie dès lors que deux tranches consécutives sont connues. Notre objectif est en effet de cribler avec les bits connus de B^2 (colorés en bleu dans la figure 8.12). Afin de traverser χ, nous fixons la valeur de certains bits de manière à ce que des lignes complètes ne dépendent que de A^u ou de A^v. Les choix "raisonnables" de bits à fixer sont décrits en couleur à la figure 8.14

Finalement, pour passer l'application θ, il est nécessaire d'avoir accès à au moins deux tranches consécutives, de manière à pouvoir cribler linéairement avec l'information contenue dans B^2 : on a un crible sur 5 bits si l'on a deux tranches consécutives, 10 si l'on a trois tranches consécutives, 10 si l'on a deux fois deux tranches consécutives,... du fait que l'application θ nécessite un bit de parité appartenant à la même tranche que le bit transformé, ainsi qu'un bit de parité appartenant à une tranche adjacente.

Comme nous l'avons déjà dit, fixer des bits nécessitera de réaliser notre

20. ρ-shift offsets

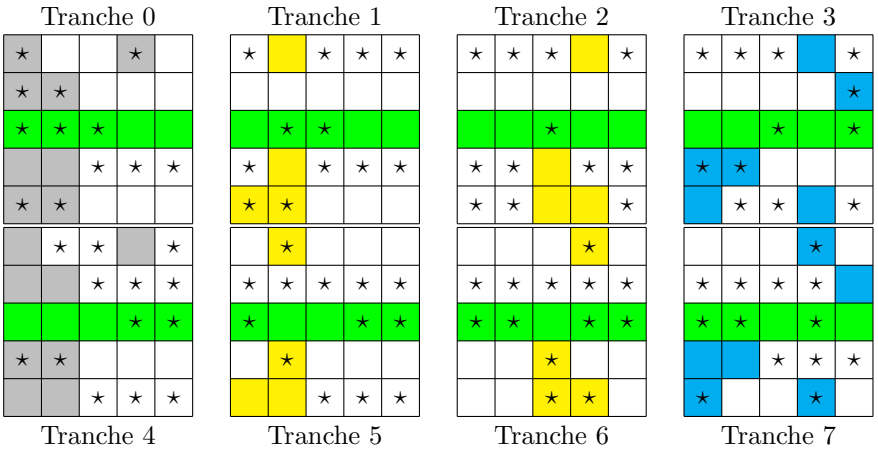

Figure 8.14 – C^1 : la partie verte est connue, les bits marqués d'une étoile ($\star$) correspondent à la moitié de l'état A^u, les autres correspondent à la moitié A^v. Les bits colorés en jaune, gris ou bleu correspondent aux meilleurs choix de bits pour fixer une tranche entière après application de χ.

attaque 2^{n_b} fois mais va faire décroître la taille des deux listes, et va nous permettre de cribler en utilisant plus d'information : une partie de l'information provenant de A^3.

Afin d'être plus compréhensible, nous expliquons le fonctionnement d'un choix particulier de bits : ceux en jaune à la figure 8.14, c'est-à-dire que l'on va rester sur les tranches 1, 2, 5 et 6. Il y a 16 bits représentés en jaune : 8 viennent de A^u et 8 de A^v. Ainsi, comme nous fixons ces bits, la taille de chaque liste devient $2^{65-8} = 2^{57}$. Ayant fixé ces bits, nous pouvons calculer les 4 tranches de coordonnées $z \in \{1, 2, 5, 6\}$ de A^2, indépendamment en fonction de A^u et de A^v. En effet, chaque ligne de ces tranches dans l'état C^1 ne dépend, après avoir fixé la valeur des bits jaunes, que de bits de A^u ou que de bits de A^v.

Ensuite, comme nous avons calculé 2 fois deux tranches consécutives de A^2, les caractéristiques de θ nous permettent de cribler avec 10 bits connus de B^2. Plus précisément, il existe deux fonctions f_3 et $g_3 : \{0,1\}^{57} \to \{0,1\}^{10}$ telles que

$$\left(B^2_{0,0,2}, \ldots, B^2_{4,4,2}, B^2_{0,0,6}, \ldots, B^2_{4,4,6}\right) = f_3(A^u) + g_3(A^v) \; .$$

En concaténant nos fonctions f_1, f_2 et f_3 ainsi que g_1, g_2 et g_3, nous pouvons donc définir deux fonctions f et $g : \{0,1\}^{57} \to \{0,1\}^{60}$:

$$f(A^u) = (f_1(A^u), f_2(A^u), f_3(A^u))$$
$$g(A^v) = (g_1(A^v), g_2(A^v), g_3(A^v)) \; .$$

Maintenant que tout ceci est posé, notre attaque consiste simplement à appliquer la stratégie du tout premier algorithme de ce chapitre qui fusionne

Choix	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{V}\rvert$	Rel.	n_b	T_r	$N_{eval}(f,g)$	Complexité
	2^{65}	2^{65}	0	0	2^{80}	2^{65}	2^{80}
$\{1,2\}$	2^{63}	2^{59}	5	8	2^{75}	2^{71}	2^{75}
$\{1,2,5,6\}$	2^{57}	2^{57}	10	16	2^{70}	2^{73}	2^{73}
$\{1,2,3,5,6\}$	2^{54}	2^{54}	15	22	2^{65}	2^{76}	2^{76}
$\{1,2,3,5,6,7\}$	2^{51}	2^{51}	20	28	2^{60}	2^{79}	2^{79}
$\{1,2,3,4,5,6\}$	2^{52}	2^{48}	25	30	2^{55}	2^{82}	2^{82}
$\{1,2,3,4,5,6,7\}$	2^{49}	2^{45}	30	36	2^{50}	2^{85}	2^{85}
$\{0,\ldots,7\}$	2^{43}	2^{43}	40	44	2^{40}	2^{87}	2^{87}

Table 8.3 – Complexité des attaques pour différents choix de bits fixés. n_b est le nombre de bits que l'attaquant.e fixe et Rel. est le nombre de nouvelles relations que l'on peut utiliser pour cribler. T_r est la complexité de la recherche exhaustive après criblage. Naturellement, la complexité de l'attaque est calculée comme le maximum entre le coût de la recherche exhaustive et le nombre d'évaluations de nos fonctions f et g.

deux listes en criblant uniquement avec des équations linéaires comme décrit en 8.3.1 avec les valeurs

$$(0,\ldots,0,A^0_{*,0,*},B^2_{0,0,2},\ldots,B^2_{4,4,2},B^2_{0,0,6},\ldots,B^2_{4,4,6}) ,$$

où les 0 viennent des équations dues aux bits de parité des colonnes $A^1_{i,*,3}$ et $A^1_{i,*,7}$ pour $i \in \{0,1,2,3,4\}$.

Dans le cas où nous avons fixé les bits colorés en jaune à la figure 8.14, nous avons, en reprenant les notations de la section précédente,

$$|\mathcal{U}| = |\mathcal{V}| = 2^{57}$$

et

$$c = 10 + 40 + 10 = 60 .$$

Cependant, notre algorithme doit être répété 2^{16} fois, pour chaque valeur possible des bits jaunes, ce qui donne une complexité finale de l'ordre de $2^{16} \times 2^{57} = 2^{73}$ évaluations de f et de g. Le coût de la recherche exhaustive du nombre de candidats restant après l'application de notre algorithme 2^{16} fois est lui égal à

$$2^{57} \times 2^{57} \times 2^{-10} \times 2^{-40} \times 2^{-10} \times 2^{16} = 2^{70} .$$

Les complexités correspondant à tous les choix "raisonnables" de bits à fixer sont décrites à la table 8.3 où il apparaît que le choix que nous avons détaillé est le plus performant...

8.4.2 Seconde méthode : fusion de listes

Dans cette section, nous décrivons une deuxième méthode pour cribler avec l'information contenue dans B^2 qui permet de réduire encore la complexité de notre attaque.

Nous notons comme précédemment la liste $L_{\mathcal{U}}$ (respectivement $L_{\mathcal{V}}$) qui correspond à A^u (respectivement A^v). Chacune de ces listes est de taille 2^{65} comme déjà expliqué. Nous avons toujours nos 50 relations linéaires, provenant de A^0 ainsi que des bits de parité. Ainsi, nous pouvons partitionner nos listes en 2^{50} sous-listes, de taille moyenne 2^{15}, selon la valeur de $(f_1(A^u), f_2(A^u))$ pour $L_{\mathcal{U}}$ et $(g_1(A^v), g_2(A^v))$ pour $L_{\mathcal{V}}$. Et, à chacune des 2^{50} sous-listes de $L_{\mathcal{U}}$, correspond une seule sous-liste de $\mathcal{V}$, déterminée par la valeur de l'information que nous avons sur A^0.

Nous proposons alors un algorithme qui, pour chacune des 2^{50} sous-listes associée à la valeur $(f_1(A^u), f_2(A^u))$, considère l'unique sous-liste de taille 2^{15} de $L_{\mathcal{V}}$ qui lui correspond, et fusionne ces deux sous-listes en utilisant l'information contenue dans B^2. Le coût total de l'algorithme sera donc 2^{50} multiplié par le coût de fusion des deux listes de taille 2^{15}. Ces sous-listes sont notées $L'_{\mathcal{U}}$ et $L'_{\mathcal{V}}$, où les éléments de ces listes sont les seules sous-parties de l'état conduisant à une valeur fixée de $(f_1(A^u), f_2(A^u))$, respectivement $(g_1(A^v), g_2(A^v))$.

Pour chacune de ces sous-listes de taille 2^{15}, étant données les propriétés de ρ et de π, ainsi que le fait que l'application χ ne dépend en entrée que de trois bits consécutifs appartenant à la même ligne dans l'état C^1, il nous est possible de calculer certains bits de l'état A^2 directement. Lorsque trois bits consécutifs sur la même ligne proviennent de la même moitié de l'état A^1, alors un bit de A^2 ne dépend que de cette moitié. Les détails des bits qui peuvent être calculés indépendamment sont représentés en jaune et en violet à la figure 8.15 qui décrit en détail les 3 tours de KETJE JR v1 en tenant compte de notre séparation de l'état en 2 parties. De plus, il est possible de calculer les valeurs provenant de A^u et de A^v qui, XORées l'une à l'autre déterminent la valeur des bits marqués d'un L à la figure 8.15.

Fusionner les deux listes $L'_{\mathcal{U}}$ et $L'_{\mathcal{V}}$ de manière naïve coûte 2^{30} opérations (parcourir les deux listes). Afin de réduire significativement ce coût, nous regardons en détail les relations entre C^1 et les bits connus de B^2.

Fusionner les listes $L'_{\mathcal{U}}$ et $L'_{\mathcal{V}}$ efficacement. Nous détaillons dans cette partie comment appliquer au contexte de KETJE l'idée décrite dans [CNV13].

Tout d'abord, nous devons identifier quelle partie de l'information nous allons utiliser. En effet, comme nos listes sont de taille 2^{15}, nous n'avons pas besoin d'exploiter tous les bits d'information, car au delà d'un certain nombre, parcourir les listes sera plus efficace que cribler. Les relations entre les bits connus de B^2 et les sous-parties de l'état A^u et A^v sont décrites à la figure 8.16.

Plus précisément, nous nous intéressons aux bits notés e_0, b_1, d_1, a_2, d_2, e_2, b_5, d_5, a_6, d_6, e_6 et c_7, représentés à la figure 8.16. Ces bits (ou des combinaisons de ces bits) ont la particularité de ne faire intervenir de manière non-linéaire qu'un nombre de variables restreint, comme nous allons le montrer avec les équations (8.16) à (8.25). C'est cette particularité que nous allons exploiter pour réduire le coût de l'algorithme de criblage.

Pour simplifier la lecture, nous contractons les notations : x_{ijk} est défini comme le bit $C^1_{i,j,k}$ si ce bit appartient à A^u, *i.e.* à la liste $L'_{\mathcal{U}}$ et y_{ijk} si ce bit

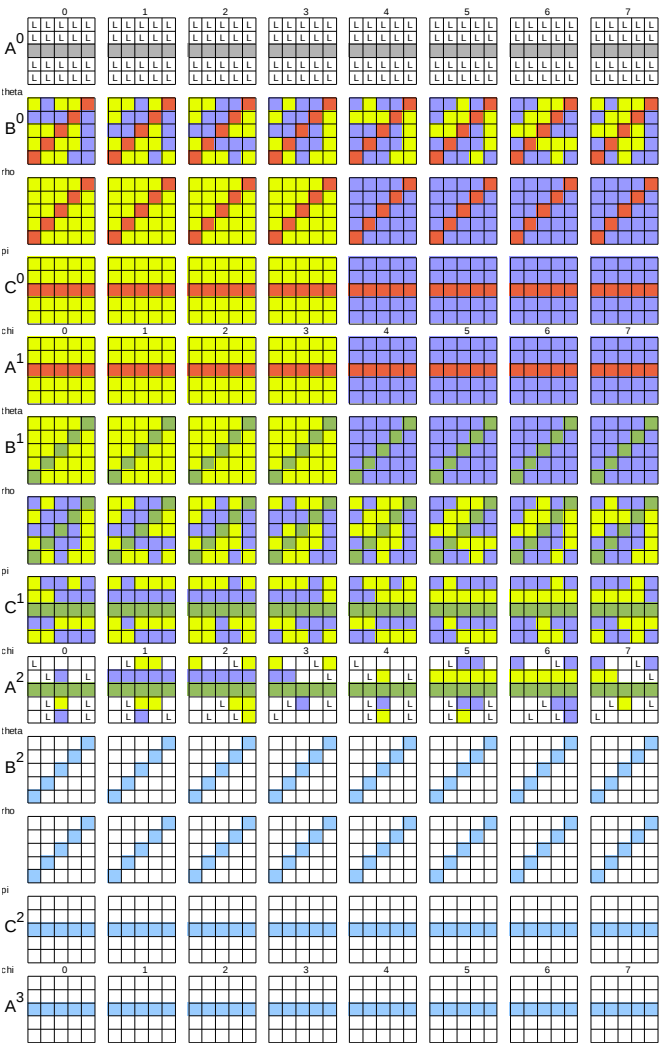

Figure 8.15 – Représentation de l'attaque sur 3 tours. Chaque carré de taille 5 représente une tranche, chaque ligne de 8 carrés représente l'état interne complet de Ketje Jr. Les bits colorés en jaune (respectivement en violet) sont calculables indépendamment à partir de $L_{\mathcal{U}}$ (respectivement $L_{\mathcal{V}}$). Les bits colorés en rouge (respectivement en vert) sont connus et proviennent de l'état A^1 (respectivement A^2) et ont déjà été pris en compte pour réduire la taille des listes de 2^{105} à 2^{65}. Les bits en noir et bleu sont connus et vont être utilisés pour cribler. Les bits marqués d'un L peuvent être calculés comme combinaison linéaire de A^u et de A^v.

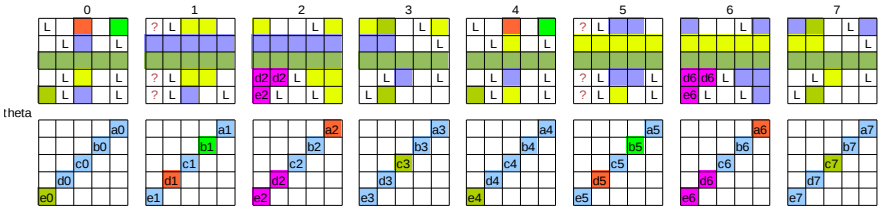

Figure 8.16 – Détail des relations quadratiques entre les bits connus de B^2 et de A^u et A^v.

appartient à A^v, *i.e.* à la liste $L'_{\mathcal{V}}$.

Afin de comprendre comment nous obtenons nos équations, nous expliquons par exemple en détail ce qui se produit pour le bit b_5.

Après l'application de θ, on obtient l'équation

$$b_5 = A^2_{1,1,5} + \sum_{i=0}^{4} (A^2_{0,i,5} + A^2_{2,i,4}) \; .$$

Cependant, dans cette équation, seulement $A^2_{2,2,4}$ amène une combinaison non-linéaire de variables provenant de chaque liste. En d'autres termes, il existe deux fonctions booléennes linéaires ℓ_1 et ℓ_2 telles que

$$\begin{aligned} b_5 =& \ell_1(A^u) + \ell_2(A^v) + A^2_{2,2,4} \\ =& \ell_1(A^u) + \ell_2(A^v) + C^1_{2,2,4} + (C^1_{3,2,4} + 1)C^1_{4,2,4} \; . \end{aligned}$$

Avec nos notations, nous avons donc $x_{224} = C^1_{2,2,4}$, $y_{324} = C^1_{3,2,4}$ ainsi que $x_{424} = C^1_{4,2,4}$. Nous définissons alors deux fonctions linéaires notées $\ell^x_{b_5}$ et $\ell^y_{b_5}$ telles que

$$b_5 = \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) + y_{324}x_{424} \; .$$

En faisant la même chose pour tous les autres bits que nous regardons en détail, nous obtenons des fonctions linéaires notées ℓ^x_b et ℓ^y_b pour $b \in \{e_0, b_1, d_1, a_2, d_2, e_2, b_5, d_5, a_6, d_6, e_6, c_7\}$ telles que les équations suivantes soient vérifiées.

$$b_5 = \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) + y_{324}x_{424} \tag{8.4}$$
$$b_1 = \ell^x_{b_1}(A^u) + \ell^y_{b_1}(A^v) + x_{320}y_{420} \tag{8.5}$$
$$c_7 = \ell^x_{c_7}(A^u) + \ell^y_{c_7}(A^v) + y_{027}x_{127} + y_{037}x_{137} \tag{8.6}$$
$$e_0 = \ell^x_{e_0}(A^u) + \ell^y_{e_0}(A^v) + x_{430}y_{030} + x_{320}y_{420} + y_{027}x_{127} + y_{037}x_{137} \tag{8.7}$$
$$a_2 = \ell^x_{a_2}(A^u) + \ell^y_{a_2}(A^v) + y_{421}x_{021} + y_{441}x_{041} + x_{431}y_{031} \tag{8.8}$$
$$d_1 = \ell^x_{d_1}(A^u) + \ell^y_{d_1}(A^v) + y_{421}x_{021} + y_{441}x_{041} + x_{431}y_{031} + x_{120}y_{220} \tag{8.9}$$
$$a_6 = \ell^x_{a_6}(A^u) + \ell^y_{a_6}(A^v) + x_{425}y_{025} + x_{435}y_{035} + y_{445}x_{045} \tag{8.10}$$
$$d_5 = \ell^x_{d_5}(A^u) + \ell^y_{d_5}(A^v) + x_{425}y_{025} + x_{435}y_{035} + y_{445}x_{045} + y_{124}x_{224} \tag{8.11}$$
$$e_2 = \ell^x_{e_2}(A^u) + \ell^y_{e_2}(A^v) + x_{432}y_{032} \tag{8.12}$$
$$d_2 = \ell^x_{d_2}(A^u) + \ell^y_{d_2}(A^v) + y_{042}x_{142} + x_{432}y_{032} + x_{442}y_{042} \tag{8.13}$$
$$e_6 = \ell^x_{e_6}(A^u) + \ell^y_{e_6}(A^v) + y_{436}x_{036} \tag{8.14}$$
$$d_6 = \ell^x_{d_6}(A^u) + \ell^y_{d_6}(A^v) + x_{046}y_{146} + y_{436}x_{036} + y_{446}x_{046} \tag{8.15}$$

En examinant ces équations, nous pouvons de plus constater que beaucoup de variables apparaissent plusieurs fois dans ce système. Par exemple, les équations (8.5), (8.6) et (8.7) ne font intervenir que 14 variables différentes, 7 calculables à partir de $L'_{\mathcal{U}}$ et 7 calculables à partir de $L'_{\mathcal{V}}$. De plus, il est possible de combiner linéairement certaines de ces équations : par exemple, on a

$$a_2 + d_1 = x_{120}y_{220} + \ell^x_{a_2}(A^u) + \ell^y_{a_2}(A^v) + \ell^x_{d_1}(A^u) + \ell^y_{d_1}(A^v)\ .$$

Finalement, nous pouvons aussi factoriser certaines variables dans ces équations, de manière à les simplifier : les termes y_{042} et x_{046} peuvent se factoriser de la manière suivante.

$$d_6 = \ell^x_{d_6}(A^u) + \ell^y_{d_6}(A^v) + x_{046}(y_{446} + y_{146}) + y_{436}x_{036}$$

$$d_2 = \ell^x_{d_2}(A^u) + \ell^y_{d_2}(A^v) + y_{042}(x_{442} + x_{142}) + x_{432}y_{032}$$

Avec ces transformations, nous obtenons le système d'équations suivant, qui totalise 10 équations quadratiques pour seulement 21 variables calculables à partir de $L'_{\mathcal{U}}$ et 21 variables calculables à partir de $L'_{\mathcal{V}}$. Si l'on considère d'autres bits, alors nous aurons au moins 2 termes quadratiques supplémentaires, augmentant la complexité totale du criblage : nous expliquerons plus tard pourquoi notre choix apporte la meilleure complexité.

Les équations que nous considérons sont donc les suivantes.

$$b_1 = x_{320}y_{420} + \ell^x_{b_1}(A^u) + \ell^y_{b_1}(A^v) \tag{8.16}$$
$$c_7 = y_{027}x_{127} + y_{037}x_{137} + \ell^x_{c_7}(A^u) + \ell^y_{c_7}(A^v) \tag{8.17}$$
$$e_0 = x_{430}y_{030} + x_{320}y_{420} + y_{027}x_{127} + y_{037}x_{137} + \ell^x_{e_0}(A^u) + \ell^y_{e_0}(A^v) \tag{8.18}$$
$$a_2 + d_1 = x_{120}y_{220} + \ell^x_{a_2+d_1}(A^u) + \ell^y_{a_2+d_1}(A^v) \tag{8.19}$$
$$a_6 + d_5 = y_{124}x_{224} + \ell^x_{a_6+d_5}(A^u) + \ell^y_{a_6+d_5}(A^v) \tag{8.20}$$
$$b_5 = y_{324}x_{424} + \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) \tag{8.21}$$
$$e_2 = x_{432}y_{032} + \ell^x_{e_2}(A^u) + \ell^y_{e_2}(A^v) \tag{8.22}$$
$$e_6 = y_{436}x_{036} + \ell^x_{e_6}(A^u) + \ell^y_{e_6}(A^v) \tag{8.23}$$
$$e_2 + d_2 = y_{042}(x_{142} + x_{442}) + \ell^x_{e_2+d_2}(A^u) + \ell^y_{e_2+d_2}(A^v) \tag{8.24}$$
$$e_6 + d_6 = x_{046}(y_{146} + y_{446}) + \ell^x_{e_6+d_6}(A^u) + \ell^y_{e_6+d_6}(A^v) \tag{8.25}$$

Idée générale de l'algorithme. Étant données deux listes $L'_{\mathcal{U}}$ et $L'_{\mathcal{V}}$, l'idée est de trouver toutes les paires d'éléments appartenant chacun à une des deux listes qui satisfont un certain nombre de relations ($n_{aux} + n_{res}$), en testant en parallèle n_{aux} relations, puis les n_{res} restantes. Dans ce qui suit ainsi qu'à la figure 8.17, nous décrivons un algorithme qui trouve ces paires plus rapidement que la recherche exhaustive.

Pour accomplir cette tâche, nous considérons n_{aux} relations, et nous classons la première liste $L'_{\mathcal{U}}$ selon les v^u_{aux} variables dérivées de A^u qui apparaissent dans ces équations. Pour chacune de ces valeurs, nous considérons toutes les correspondances possibles qui satisfont les n_{aux} relations, avec les v^v_{aux} variables qui apparaissent et qui appartiennent à A^v. Avec tous les éléments de la liste $L'_{\mathcal{V}}$ qui satisfont les relations, nous construisons une liste auxiliaire, de taille plus petite que la première, que nous ordonnons selon la valeur des v^v_{res} variables qui apparaissent dans les n_{res} équations restantes non prises en compte jusqu'à présent.

Ceci doit être réalisé pour toutes les valeurs possibles de v^u_{aux}. Une fois la liste auxiliaire dérivée de $L'_{\mathcal{V}}$ construite, nous revenons à la première liste $L'_{\mathcal{U}}$, où les éléments ont déjà été ordonnés selon la valeur des v^u_{aux} variables, et nous savons que les éléments de la liste auxiliaire sont tels que les n_{aux} premières équations sont satisfaites. Nous regardons ensuite les différentes valeurs des v^u_{res} variables de la première liste impliquées dans les n_{res} relations restantes, et nous vérifions dans la liste auxiliaire si les valeurs des v^v_{res} variables satisfont les n_{res} relations restantes. Le gain de l'algorithme vient du fait que nous ne regardons pas toutes les paires de groupe possibles, mais nous subdivisons les équations, de manière à ne regarder ces équations que partie par partie.

Nous détaillons la complexité cet algorithme, représenté à la figure 8.17, dans le paragraphe suivant. Naturellement, la complexité de la fusion des listes ne

pourra pas être meilleure que le nombre de solutions restant à la fin, *i.e.*

$$|L'_{\mathcal{U}}| \times |L'_{\mathcal{V}}| \times 2^{-n_{aux}-n_{res}} .$$

Dans notre cas, nous avons $v_{aux} = v^u_{aux} = v^v_{aux}$ ainsi que $v_{res} = v^u_{res} = v^v_{res}$. Afin d'appliquer l'algorithme, nous devons décider quelles sont les n_{aux} relations et quelles sont les n_{rem} relations. Nous évaluons donc la complexité de l'algorithme en fonction de n_{aux}, n_{res}, v_{aux} et v_{res}, pour pouvoir séparer en deux nos équations de manière optimale. L'algorithme de correspondance parallèle va, pour chacune des $2^{v_{aux}}$ valeurs possibles des v_{aux} variables associées aux n_{aux} premières relations dans $L'_{\mathcal{U}}$, réaliser la chose suivante :

1 Utiliser les n_{aux} relations pour construire la liste auxiliaire d'éléments de $L'_{\mathcal{V}}$ qui satisfont lesdites relations. Approximativement $2^{v_{aux}-n_{aux}}$ sous-listes de $L'_{\mathcal{V}}$ vont correspondre, chacune de ces sous-listes étant composée d'en moyenne $|L'_{\mathcal{V}}|/2^{v_{aux}}$ éléments, car nous avons séparé $L'_{\mathcal{V}}$ en fonction des valeurs possibles des v_{aux} variables.

2 Ensuite, cette liste auxiliaire est réordonnée selon la valeur des v_{res} variables impliquées dans les n_{res} équations.

3 Pour chacun de ces éléments dans la sous-liste de $L'_{\mathcal{U}}$, nous regardons toutes les correspondances possibles au regard des n_{res} équations, en examinant toutes les valeurs possibles des v_{res} variables.

La complexité de cet algorithme peut donc être calculée en fonction des paramètres n_{aux}, n_{res}, v_{aux} et v_{res}. Tout d'abord, nous devons parcourir l'ensemble des valeurs possibles des v_{aux} variables, ce qui coûte $2^{v_{aux}}$. Pour chacune des sous-listes de $L'_{\mathcal{U}}$, il y a $2^{v_{aux}-n_{aux}}$ valeurs qui correspondent dans $L'_{\mathcal{V}}$ et dont les éléments doivent être examinés, sachant que ces sous-listes sont de taille $|L'_{\mathcal{V}}|/2^{v_{aux}} = 2^{15-v_{aux}}$. Donc, chaque liste auxiliaire, que nous ordonnons en fonction des v_{res} variables, est en moyenne de taille $2^{15-n_{aux}}$, ce qui donne un coût de

$$2^{v_{aux}} \times \left[2^{v_{aux}-n_{aux}} \times 2^{15-v_{aux}}\right]$$

et qui correspond au premier terme de la formule (8.26).

Ensuite, nous reprenons la première sous-liste, de taille $2^{15-v_{aux}}$, que nous parcourons, et nous cherchons les éléments de la liste auxiliaire, construite à partir de $L'_{\mathcal{V}}$, tels que les v_{res} valeurs satisfont les n_{res} relations restantes. Il y a donc, pour chaque élément de la sous-liste de $L'_{\mathcal{U}}$, $2^{v_{res}-n_{res}}$ valeurs qui conviennent et qui satisfont les équations restantes, ce qui correspond au deuxième terme de la formule (8.26).

Comme la complexité ne peut être plus petite que le nombre final de solutions, cela signifie que dans cette fusion de listes, à v_{aux} variables fixées, la complexité ne peut pas être plus petite que $2^{30-v_{aux}-n_{aux}-n_{res}}$. En effet, on fusionne une liste de taille $2^{15-v_{aux}}$ (la sous-liste de $L'_{\mathcal{U}}$) avec une liste de taille $2^{15-n_{aux}}$ (la liste auxiliaire construite à partir de $L'_{\mathcal{V}}$), au regard de n_{res} relations.

En additionnant le coût de ces étapes, nous pouvons donc assurer que notre algorithme de fusion des listes de taille 2^{15} coûte

$$2^{15+v_{aux}-n_{aux}} + 2^{v_{aux}} \max(2^{15-v_{aux}} \times 2^{v_{res}-n_{res}}, 2^{30-v_{aux}-n_{aux}-n_{res}}) \quad (8.26)$$

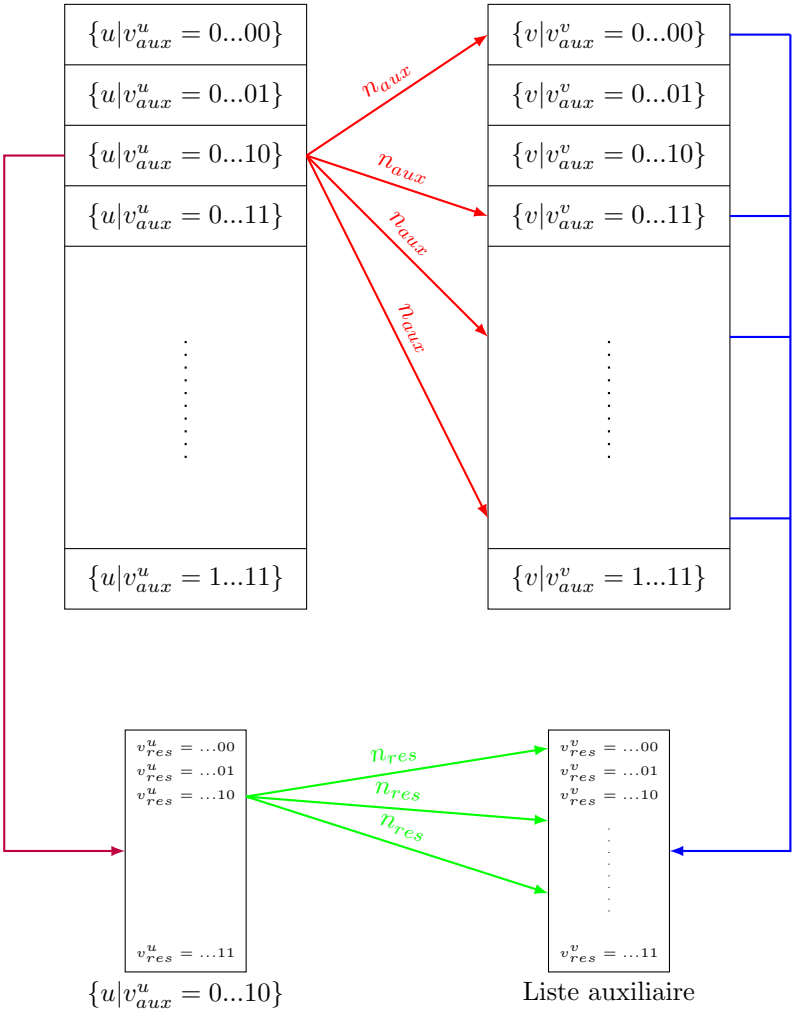

Figure 8.17 – Schéma de l'algorithme de correspondance parallèle.

opérations.

Dans ces conditions, nous pouvons choisir de manière appropriée v_{aux} et n_{aux} : nous prenons les équations associées à b_1, e_0, c_7 (7 variables provenant de chaque liste) ainsi que celles associées à a_2 et a_6. Alors, on a $n_{aux} = 5$, $v_{aux} = 11$, et les listes auxiliaires construites dans notre algorithme seront donc de taille $2^{11-5+4} = 2^{10}$. Les autres équations sont au nombre de $n_{res} = 5$, avec $v_{res} = 10$, ce qui nous donne une complexité de l'ordre de

$$2^{15+11-5} + \max(2^{15+10-5}, 2^{20}) = 2^{21} + 2^{20} = 2^{21.5}$$

opérations.

Complexité de l'attaque complète. Comme détaillé précédemment, la complexité en temps de l'attaque complète correspond à la répétition de l'algorithme décrit ci-dessus pour chacune des 2^{50} valeurs qui déterminent les relations linéaires. Nous obtenons donc une complexité de l'ordre de

$$2^{50} \times 2^{21.5} = 2^{71.5}$$

opérations.

Les opérations réalisées dans notre attaque coûtent moins cher que l'application d'un tour de Ketje, ce qui implique que notre attaque est plus rapide que la recherche exhaustive. Pour une comparaison plus fine, il faudrait investiguer le coût des fonctions f et g que nous appliquons aux sous-parties de l'état au coût de l'application d'un tour de Ketje Jr.

La complexité en mémoire de notre attaque est dominée par le stockage de nos listes, qui sont de taille 2^{65}, car la taille des listes auxiliaires construites est bien plus petite. De plus, comme nous trions nos listes selon des valeurs qui sont de taille fixe (sur 50 bits ou moins), nous n'avons pas à ajouter de facteur logarithmique dû au tri, car celui-ci peut se faire avec des tables de hachage en parcourant une seule fois chaque liste.

La complexité de notre attaque étant maintenant dominée par l'algorithme de fusion de listes et non par le coût de la recherche exhaustive, il semble ardu d'étendre l'attaque en utilisant plus de tours, alors qu'il était naturel de passer de 2 tours à 3 tours. Il semble donc que notre technique atteint ses limites ici. Cependant, le nombre d'opérations nécessaires pour mener à bien notre attaque est largement en-dessous du coût de la recherche exhaustive, *i.e.* 2^{96} pour une clef de taille minimale. Donc, il semble possible d'adapter notre attaque pour un ratio plus petit, *i.e.* 32 bits, d'autant plus que sur 4 tours, on dispose encore de suffisamment d'information : $2^{200-4\times32} = 2^{72} \ll 2^{96}$. De plus, prendre une clef plus grande ne modifie pas la complexité de notre attaque, celle-ci étant une attaque par recouvrement de l'état interne.

8.5 Amélioration pour un ratio de 32 bits

Dans cette section, nous décrivons une version améliorée de notre attaque sur 4 blocs consécutifs, toujours sur la première version de Ketje Jr, en considérant

un ratio plus petit de 32 bits. Notre attaque sur ces paramètres a une complexité de 2^{92} opérations.

Idées supplémentaires. Notre amélioration se base sur deux points :

— Comme nous connaissons 32 bits sur le même plan dans les états A^0, A^1, A^2 et A^3, nous avons 4 des 5 sorties des 8 boîtes-S χ. Nous pouvons alors inverser partiellement ces boîtes-S.

— Nous pouvons réduire le nombre d'interactions non-linéaires en fixant quelques bits de C^1 avant même d'appliquer notre technique de type "diviser pour mieux régner".

8.5.1 Exploiter l'information de A^0, A^1 et A^2

Pour les bits connus provenant de A^1, colorés en rouge à la figure 8.15, il n'y a rien à changer, mais comme nous avons 1 bit connu en moins par tranche, cela augmente la taille de nos listes d'un facteur 2^4 (4 bits dans chaque moitié d'état qui totalise 4 tranches).

Les 32 bits connus provenant de l'information sur l'état A^0 peuvent encore être calculés comme des relations linéaires des bits des 2 listes, mais comme nous n'avons que 32 bits et non plus 40, cela nous enlève exactement 8 relations linéaires de criblage, passant de 50 relations linéaires au total à 42 en comptant les bits de parité des colonnes.

Pour les bits provenant de A^2, nous décidons de fixer la valeur des 8 bits restant, et donc d'appliquer notre attaque 2^8 fois, afin de pouvoir inverser l'application non-linéaire χ et de cribler directement sur l'état B^1 indépendamment sur chaque liste.

Finalement, la taille de nos deux listes est donc égale à $2^{100+5-16-20} = 2^{69}$, mais nous aurons à répéter notre attaque 2^8 fois, pour chacune des valeurs supposées des bits dans A^2 aux positions $A^2_{4,0,i}$ pour $0 \leq i \leq 7$. Le nombre de relations linéaires est égal à $32 + 10 = 42$. Il ne nous reste plus qu'à utiliser l'information contenue dans le dernier bloc.

8.5.2 Exploiter l'information de A^3

Le problème auquel nous faisons face maintenant est que nous ne pouvons plus inverser l'application χ entre les états C^2 et A^3, et donc que nous n'avons plus accès à la valeur des bits dans l'état B^2. De plus, fixer 8 bits comme nous le faisons pour A^2 nous coûterait trop cher. En revanche, comme il ne manque qu'un seul bit par ligne de l'application χ, nous pouvons assurer qu'il y a exactement 32 équations linéaires indépendantes entre les bits aux tranches C^2 et les bits de A^3. En effet, comme un seul bit manque par application de χ, alors ce bit manquant peut être obtenu par combinaison linéaire des bits de C^2 et des bits connus de A^3, sachant que ces combinaisons linéaires dépendent de la valeur des 32 bits connus de A^3. Dit autrement, comme il y a un seul bit manquant pour chaque sortie de la permutation sur 5 bits χ, chaque bit en entrée dépend donc

linéairement de cette valeur, une fonction booléenne à 1 variable étant toujours affine. Dans notre cas, les bits connus sont

$$\left\{A^3_{0,0,*}, A^3_{1,0,*}, A^3_{2,0,*}, A^3_{3,0,*}\right\} ,$$

les bits $A^3_{4,0,*}$ étant inconnus. Pour simplifier la lecture, nous contractons les notations : $x_i = A^3_{4,0,i}$ pour tout $0 \leq i \leq 7$ et les bits connus de A^3 sont notés α_j pour j allant de 0 à 31. De plus, on définit le vecteur $(\alpha||X)$:

$$(\alpha||X) = (\alpha_0, \alpha_1, \ldots, \alpha_{31}, x_0, x_1, \ldots, x_7)^\top ,$$

ainsi que le vecteur β qui correspond aux bits colorés en bleu à la figure 8.15 dans l'état C^2, de taille 40 bits.

Alors, avec ces notations et les remarques précédentes, on sait qu'il existe une matrice M de taille 40×8 ainsi qu'un vecteur β_0 tel que

$$\beta = MX + \beta_0 .$$

Comme nous l'avons dit précédemment, les relations affines représentées par M entre les bits de la tranche $C^2_{*,0,*}$ et le vecteur X (correspondant aux bits inconnus de A^3) dépendent de la valeur $(\alpha_0, \alpha_1, \ldots, \alpha_{31})$. Donc M dépend des 32 bits connus par l'attaquant.e en A^3. De plus, les applications π et ρ sont uniquement des permutations des positions des bits, ce qui signifie qu'en réutilisant les mêmes notations que précédemment (figure 8.16), nous pouvons assurer qu'il existe une matrice M' de taille 40×8 ainsi qu'un vecteur β'_0 tels que

$$(a_0, \ldots, a_1, \ldots, e_6, a_7, \ldots, e_7)^\top = M'X + \beta'_0 .$$

En d'autres termes, les autres bits n'interviennent pas dans nos relations linéaires. Less équations linéaires dues à l'information contenue dans A^3 gardent la même "forme" et font toujours intervenir le même nombre de bits de l'état, assurant que l'on travaille dans un espace vectoriel de même dimension.

8.5.3 Récupérer l'information

Comme les bits $a_0, \ldots, a_1, \ldots, e_6, a_7, \ldots, e_7$ ne peuvent pas être exprimés comme une fonction linéaire des bits de A^u et de A^v, nous ne pouvons pas les exploiter en l'état pour cribler. Pour surmonter ce problème, nous adoptons la technique déjà utilisée dans la première méthode pour cribler à la section 8.4.1 : nous fixons la valeur de quelques bits, afin que l'expression des bits $a_0, \ldots, a_1, \ldots, e_6, a_7, \ldots, e_7$ devienne linéaire en les bits indéterminés.

Les bits que nous fixons sont au nombre de 20 et sont les suivants :

— $C^1_{3,2,0}$, $C^1_{1,2,0}$;
— $C^1_{4,2,1}$, $C^1_{4,4,1}$;
— $C^1_{1,2,2}$, $C^1_{0,4,2}$;
— $C^1_{3,3,3}$, $C^1_{1,3,3}$, $C^1_{1,2,3}$, $C^1_{2,1,3}$;

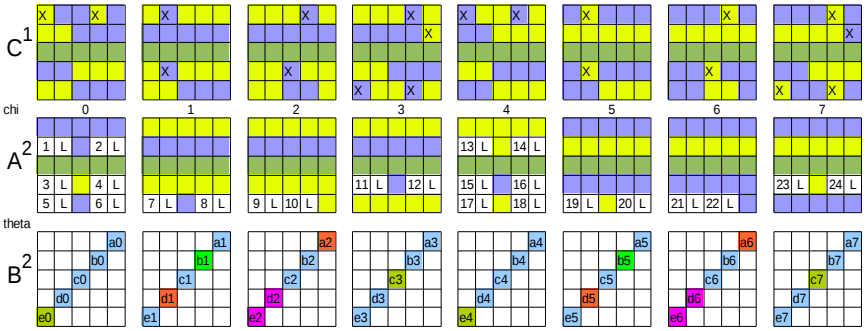

Figure 8.18 – Les bits marqués d'une croix dans C^1 sont les 20 bits dont nous fixons la valeur, ce qui implique que seules 24 variables (notées de 1 à 24) sont quadratiques dans A^2, les autres dépendent linéairement des bits des deux parties d'état.

— $C^1_{3,2,4}$, $C^1_{1,2,4}$;
— $C^1_{4,2,5}$, $C^1_{4,4,5}$;
— $C^1_{1,2,6}$, $C^1_{0,4,6}$;
— $C^1_{3,3,7}$, $C^1_{1,3,7}$, $C^1_{1,2,7}$, $C^1_{2,1,7}$.

Nous fixons ces bits là spécifiquement car, si l'on note (x, y, z) leur position, alors les deux bits à la position $(x-1, y, z)$ et $(x+1, y, z)$ appartiennent à la même moitié d'état, ce qui évite les dépendances entre les états A^u et A^v dans le calcul de l'application χ entre C^1 et A^2. Cette partie est détaillée à la figure 8.18. Finalement, nous obtenons 40 équations pour les bits $a_0, \ldots, e_0, a_1, \ldots, e_6, a_7, \ldots, e_7$, où 24 variables quadratiques (monômes de degré 2 impliquant les deux parties de l'état) apparaissent. Une variable quadratique apparaît à l'application de la fonction non-linéaire χ seulement quand deux bits adjacents dans l'état C^1 appartiennent à deux parties d'état différentes. Les 24 monômes de degré 2 obtenus de cette manière à l'état A^2 sont ensuite considérés comme des nouvelles variables auxquelles nous faisons référence par l'expression "variables quadratiques".

Comme nous avons seulement 24 variables quadratiques qui interviennent dans les 40 bits $a_0, \ldots, e_0, a_1, \ldots, e_6, a_7, \ldots, e_7$, nous pouvons assurer que, par un simple pivot de Gauss, nous pouvons récupérer au moins $(40-24)$ équations linéaires entre les deux listes $L_\mathcal{U}$ et $L_\mathcal{V}$. En fait, en regardant dans les détails, nous nous apercevons que nous obtenons non pas 16, mais 20 équations linéaires indépendantes qui ne font pas intervenir les variables quadratiques.

Plus précisément, si l'on note q_i pour $1 \leq i \leq 24$ ces variables quadratiques, décrites à la figure 8.18, alors les équations suivantes sont satisfaites.

— $a_0 = q_2 + q_4 + q_6 + q_{23} + l_{a_0}(L_\mathcal{U}, L_\mathcal{V})$
— $b_0 = q_2 + l_{b_0}(L_\mathcal{U}, L_\mathcal{V})$
— $c_0 = q_{24} + l_{c_0}(L_\mathcal{U}, L_\mathcal{V})$

— $d_0 = q_1 + q_3 + q_5 + l_{d_0}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_0 = q_5 + l_{e_0}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_1 = q_1 + q_3 + q_5 + q_8 + l_{a_1}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_1 = l_{b_1}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_1 = q_2 + q_4 + q_6 + l_{c_1}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_1 = q_7 + l_{d_1}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_1 = q_7 + l_{e_1}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_2 = q_7 + l_{a_2}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_2 = q_{10} + l_{b_2}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_2 = q_8 + l_{c_2}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_2 = q_9 + l_{d_2}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_2 = q_9 + l_{e_2}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_3 = q_9 + q_{12} + l_{a_3}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_3 = l_{b_3}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_3 = l_{c_3}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_3 = q_{10} + q_{11} + l_{d_3}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_3 = l_{e_3}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_4 = q_{11} + q_{14} + q_{16} + q_{18} + l_{a_4}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_4 = q_{14} + l_{b_4}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_4 = q_{12} + l_{c_4}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_4 = q_{13} + q_{15} + q_{17} + l_{d_4}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_4 = q_{17} + l_{e_4}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_5 = q_{13} + q_{15} + q_{17} + q_{20} + l_{a_5}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_5 = l_{b_5}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_5 = q_{14} + q_{16} + q_{18} + l_{c_5}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_5 = q_{19} + l_{d_5}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_5 = q_{19} + l_{e_5}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_6 = q_{19} + l_{a_6}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_6 = q_{22} + l_{b_6}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_6 = q_{20} + l_{c_6}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_6 = q_{21} + l_{d_6}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $e_6 = q_{21} + l_{e_6}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $a_7 = q_{21} + q_{24} + l_{a_7}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $b_7 = l_{b_7}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $c_7 = l_{c_7}(L_{\mathcal{U}}, L_{\mathcal{V}})$
— $d_7 = q_{22} + q_{23} + l_{d_7}(L_{\mathcal{U}}, L_{\mathcal{V}})$

— $e_7 = l_{e_7}(L_{\mathcal{U}}, L_{\mathcal{V}})$

Comme nous pouvons le voir, les 20 équations linéaires indépendantes que nous obtenons sont données par :

— $a_0 + c_1 + b_6 + d_7$;
— $c_0 + a_7 + e_6$;
— $d_0 + a_1 + c_2\ b_1$;
— $d_1 + e_1$;
— $d_1 + a_2$;
— $d_2 + e_2$;
— $b_2 + d_3 + a_4 + c_5$;
— $d_2 + a_3 + c_4$;
— b_3 ;
— c_3 ;
— e_3 ;
— $d_4 + a_5 + c_6$;
— b_5 ;
— $d_5 + e_5$;
— $d_5 + a_6$;
— $d_6 + e_6$;
— b_7 ;
— c_7 ;
— e_7.

Cribler avec 12 équations linéaires. Nous avons donc exactement 20 équations linéairement indépendantes entre les deux listes qui permettent de cribler les valeurs de l'espace vectoriel de dimension 40 qui correspond aux bits $a_0, b_0, \ldots, e_7$. On note

$$\beta' = (a_0, \ldots, e_0, a_1, \ldots, e_6, a_7, \ldots, e_7)^\top .$$

De plus, comme il existe M' de taille 40×8 et β'_0 tels que $\beta' = M'X + \beta'_0$, cela signifie qu'il existe 40 fonctions linéaires $(\ell_i)_{1 \leq i \leq 20}$ et $(\ell'_i)_{1, \leq i \leq 20}$, telles que, pour tout $1 \leq i \leq 20$,

$$\ell_i(L_{\mathcal{U}} + L_{\mathcal{V}}) = \ell'_i(\beta') = \ell'_i \circ M'X + \ell'_i(\beta_0) .$$

Finalement, en appliquant un pivot de Gauss sur les 8 premières équations qui nous permet d'éliminer les valeurs inconnues $x_0, \ldots, x_7$, nous obtenons au moins 12 équations linéaires qui lient nos deux listes.

8.5.4 Complexité de l'attaque

Rappelons que nous avons fixé 8 bits pour pouvoir directement cribler chacune des deux listes avec l'information contenue dans A^2 (bits en vert à la figure 8.18), ainsi que 20 bits pour obtenir des équations linéaires. La taille de nos deux listes peut donc être calculée. Nous avons 100 bits d'état et 5 bits de parité des colonnes, et on connaît 16 bits d'information sur A^1, 20 bits sur A^2 car on a fixé 8 bits, ainsi que 10 bits fixés dans chaque liste pour obtenir des équations linéaires, ce qui donne

$$|L_{\mathcal{U}}| = |L_{\mathcal{V}}| = 2^{100} \times 2^{-16} \times 2^5 \times 2^{-20} \times 2^{-10} = 2^{59} \ .$$

Le coût de notre attaque sur Ketje Jr v1 avec un ratio de 32 bits vaut donc, en répétant 2^{28} fois notre technique de type "diviser pour mieux régner"

$$2^{28} \times 2^{59}$$

pour construire les listes à chaque fois, plus

$$2^{28} \times 2^{2\times 59} \times 2^{-32} \times 2^{-10} \times 2^{12}$$

pour décrire l'ensemble des solutions, ce qui donne

$$2^{87} + 2^{92} \ .$$

Cette complexité peut être un petit peu améliorée, en fixant 4 bits de plus dans la dernière partie, afin de pouvoir utiliser plus d'équations, ce qui permettrait d'équilibrer les deux termes intervenant dans la conplexité, et d'obtenir une complexité proche de 2^{90} opérations.

8.6 Conclusion

Nous avons appliqué une technique de type "diviser pour mieux régner" sur l'algorithme de chiffrement authentifié Ketje Jr. Il est important de noter que notre attaque la plus performante, sur la première version de Ketje Jr, ne s'adapte pas en l'état à la deuxième version, *i.e.* quand la permutation "tordue" est utilisée.

Notre attaque ne remet pas en cause la sécurité assurée par les auteurs de Ketje Jr, puisque le ratio utilisé ici est bien plus grand que celui préconisé par les auteurs, qui est maximum 16 bits. Cependant, notre attaque apporte une nouvelle borne non-triviale sur la sécurité de Ketje et met en garde l'utilisateur contre la tentation d'augmenter le ratio dans le but d'avoir un meilleur débit. De plus, l'utilisation de la permutation "tordue" dans la deuxième version de Ketje empêche de mener à bien notre attaque sur 4 blocs consécutifs de sortie. Son ajout dans la deuxième version de Ketje semble donc très pertinente, puisqu'il empêche l'inversion de l'application χ.

Finalement, dans notre travail, nous nous rendons aussi compte que le nombre de variables intervenant dans les équations non-linéaires ne doit pas être trop petit, sinon cela fournit un crible plus efficace que la recherche exhaustive. Ainsi, nous remarquons une fois de plus que les équations non-linéaires qui régissent les systèmes algébriques que l'on considère ne doivent pas être creuses, *i.e.* qu'il faut qu'un nombre suffisant de variables indépendantes apparaissent.

Ce travail a été récompensé par les auteurs de KETJE, puisque nous avons gagné le prix de cryptanalyse du concours organisé par les concepteurs décerné lors de la conférence *FSE 2018*.

Par ailleurs, nos résultats sont limités par le fait que la taille de nos deux listes est encore très grande. Pour espérer améliorer notre technique, il faudrait essayer de fusionner plus de deux listes, en utilisant certaines équations. Ce nouveau problème générique semble difficile, mais il serait intéressant de l'étudier, car cela pourrait généraliser ces attaques de type "diviser pour mieux régner", en divisant l'état en plus que deux parties.

Conclusion et perspectives

Conclusion générale

L'ensemble de mes travaux montre qu'une vision structurelle des objets mathématiques mis en jeu dans les systèmes cryptographiques peut avoir un apport important en cryptanalyse. Cette vision permet de découvrir de nouvelles attaques et de mettre en évidence de nouveaux critères de conception. Il est donc nécessaire de faire des allers-retours entre la conception de chiffrements, la cryptanalyse et les notions mathématiques fondamentales exploitées dans les attaques afin de déterminer précisément le niveau de sécurité pratique des algorithmes de chiffrement.

En particulier, les différentes études menées pendant ma thèse mettent bien en évidence la fragilité des systèmes qui utilisent des objets mathématiques possédant une représentation très structurée. Par exemple, nous avons attaqué des générateurs pseudo-aléatoires comme FLIP ou le PRG de Goldreich en tirant partie d'équations multivariées creuses, alors que nos attaques sur les LFSR filtrés exploitent le caractère creux de la représentation univariée du polynôme employé. La représentation multivariée identifie l'espace d'entrée comme un espace vectoriel, alors que la représentation univariée l'identifie à un corps fini de caractéristique 2. Ces représentations sont naturellement liées, mais les équations peuvent être creuses dans un cas, et denses dans l'autre, ce qui montre que les deux approches ne captent pas le même phénomène. Cette multiplicité des représentations possibles ouvre un vaste potentiel de recherche. On peut par exemple s'interroger sur la pertinence d'éventuelles représentations intermédiaires fondées sur des sous-corps de $\mathbb{F}_2^n$, par exemple l'utilisation de polynômes bivariés à coefficients dans $\mathbb{F}_{2^{n/2}}$ quand n est pair, plutôt que de polynômes univariés à coefficients dans $\mathbb{F}_{2^n}$.

Par ailleurs, le lien entre les différentes représentations des fonctions booléennes n'est pas bien compris et est un sujet difficile d'autant plus que la cardinalité des ensembles (le nombre de fonctions booléennes à n variables est 2^{2^n}) interdit toute recherche exhaustive au-delà de 6 variables. L'ensemble de ces représentations amène à de nouveaux critères, qu'il faut prendre en compte, tout comme leur généralisation qui consisterait à restreindre l'espace en entrée des fonctions. Cette restriction peut être liée à l'emploi soit à l'appartenance à l'ensemble des antécédents d'un élément connu par une fonction augmentée, ou éventuellement

à d'autres partitions, qui reflèteraient l'ensemble des possibilités de l'attaquant.e.

Nous mentionnons maintenant quelques conclusions et pistes de recherche plus précises sur les différents sujets abordés dans cette thèse.

Changement de base sur les SPN

Nous avons montré que, pour les chiffrements de type SPN à bas coût qui utilisent un cadencement de clef très simple, la faiblesse au regard des attaques par invariant est extrêmement liée à la forme canonique rationnelle de la matrice de diffusion (chapitre 3). Ce qui est étonnant, c'est que ce critère n'est pas lié au caractère MDS des couches de diffusion utilisées. Ainsi, nous amenons un tout nouveau critère sur la couche de diffusion des chiffrements SPN ainsi que sur le choix des constantes de tour.

Résilience par sous-groupe

Dans le chapitre 4, nous avons étudié la construction générique des registres filtrés. Il est apparu que la bonne représentation des fonctions booléennes est, dans ce cas, la représentation univariée, que ce soit pour capter la résistance aux attaques algébriques ou la résistance aux attaques par corrélation. Nous avons montré que le critère de non-linéarité généralisée est le critère pertinent à prendre en compte dans ce cadre, à la place de la non-linéarité classique. Plus étonnant encore, nous avons réussi à "découper" l'état interne en fonction des sous-groupes multiplicatifs du corps fini $\mathbb{F}_{2^n}$. Cette nouvelle attaque de type "diviser pour mieux régner" est régie par la distance de la fonction de filtrage aux fonctions de la forme $H(X^k)$ où $\mathsf{pgcd}(k, 2^n - 1) > 1$, ce qui nous amène à définir un nouveau critère pour les fonctions booléennes que nous appelons *résilience par sous-groupe*.

Définition 8.1 (Résilience par sous-groupe)**.** *Soit F une fonction booléenne à n variables et soit τ un diviseur de $2^n - 1$. Alors F est dite* résiliente par sous-groupe d'ordre τ *si pour toute fonction booléenne G de la forme $H(x^k)$, avec k multiple de $\frac{2^n-1}{\tau}$,*

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{F(x)+G(x)} = \left(1 + \frac{\tau}{d}\right)$$

où $d = \mathsf{pgcd}(k, \tau)$.

Ce critère n'est a priori pas lié avec les autres critères. Nous savons que, pour chaque τ qui divise $2^n - 1$, il existe des fonctions résilientes par sous-groupe d'ordre τ. Suite à des expérimentations personnelles sur un petit nombre de variables, et par intuition, nous conjecturons la propriété suivante.

Conjecture 8.2. *Pour tout $n \in \mathbb{N}^*$, il existe une fonction booléenne à n variables, résiliente par sous-groupe d'ordre τ pour tout τ qui divise $2^n - 1$.*

Lien avec les fonctions à entrées restreintes

Finalement, on peut aussi s'apercevoir que, lorsque $\mathsf{pgcd}(k, \tau) = 1$, alors la résilience par sous-groupe est directement liée à la notion de fonctions presque parfaitement équilibrées (chapitre 6). Au lieu d'imposer que la fonction soit presque équilibrée sur chacun des ensembles formé par les mots de poids constants, on demande ici qu'elle soit presque équilibrée sur chacun des translatés du sous-groupe multiplicatif d'ordre τ.

Attaques algébriques

Nous avons vu au chapitre 4 que ni le degré algébrique ni l'immunité algébrique ne sont les critères pertinents pour évaluer la sécurité des LFSR filtrés face aux attaques algébriques. C'est en réalité l'immunité spectrale des suites binaires [GRHH11, Définition 1], *i.e.* le caractère creux de la représentation univariée de nos équations qui est le critère pertinent dans ce contexte.

Ceci est directement à mettre en parallèle avec la cryptanalyse de FLIP(voir chapitre 5), du PRG de Goldreich (chapitre 7) et de celle du chiffrement authentifié Ketje (chapitre 8), où nous exploitons aussi un caractère creux des équations, mais cette fois dans leur représentation multivariée. De plus, la manière d'exploiter ce caractère creux est différente avec la technique de type "supposer et déterminer" et dans Ketje, où nous réalisons la fusion de deux listes.

Ainsi, nous obtenons deux manières de représenter nos équations (univariée ou multivariée), dans lesquelles le caractère "creux" est plus pertinent que simplement le degré.

"Supposer et déterminer"

Quantifier la résistance d'une primitive cryptogrqphique aux attaques de type "supposer et déterminer" est chose ardue. Tout d'abord, il faut comprendre quelles sont les hypothèses appropriées que peut faire l'attaquant.e. En effet, pour l'instant, nous avons réalisé des hypothèses sur la valeur de certains bits, ce qui a amené les auteurs de FLIP à considérer la notion d'*immunité algébrique récurrente* [MJSC16] (définition 7.8 du chapitre 7). Or, je pense que ceci ne capte pas suffisamment l'attaque dans sa vision globale, et il faut plutôt considérer la possibilité pour l'attaquant.e de faire des hypothèses sur des combinaisons linéaires de bits, ce qui est bien plus réaliste. Faire des hypothèses sur des combinaisons linéaires de bits revient alors à considérer les fonctions booléennes restreintes à des sous-ensembles qui sont des sous-espace affines de $\mathbb{F}_2^n$, ce qui nous ramène alors au chapitre 6, mais dans un contexte encore différent.

Dans ces conditions, il convient d'analyser les propriétés cryptographiques des fonctions booléennes lorsque l'on restreint l'entrée à des sous-espaces affines de grande taille, permettant ainsi d'identifier précisément quels sont les systèmes algébriques induits qui sont faciles et difficiles à résoudre.

Pour résumer, nous avons donc deux visions : univariée et multivariée, pour lesquelles les critères pertinents changent, sachant que ces critères doivent aussi

être généralisés en considérant la restriction à des sous-espaces affines, afin de se prémunir de l'attaque de type "supposer et déterminer". Ceci est résumé à la table 8.4 ci-dessous.

Multivarié	Univarié
Immunité algébrique	Immunité spectrale
Résilience	Résilience par sous-groupe
Non-linéarité	Non-linéarité généralisée

Table 8.4 – Critères cryptographiques liés aux deux représentations polynomiales des fonctions booléennes utilisées.

Utiliser les fonctions augmentées

Bien entendu, le but du ou de la cryptographe est aussi de concevoir des algorithmes suffisamment simples à décrire. En effet, définir un système avec une fonction booléenne qui n'est creuse ni en représentation univariée ni en représentation multivariée n'a que peu d'intérêt en cryptographie : si le coût de stockage de la fonction est de l'ordre de 2^n, alors il est aussi coûteux de chiffrer que de cryptanalyser le système. Il est donc nécessaire de garder des définitions simples pour les fonctions booléennes et de les combiner avec d'autres composantes, notamment une fonction de mise à jour de l'état interne qui interagit bien avec les propriétés de la fonction de filtrage.

Ainsi, dans le contexte des générateurs pseudo-aléatoires, l'attaque de type "supposer et déterminer" n'exploite pas la fonction de filtrage seule, elle prend aussi en compte la fonction de mise à jour de l'état interne. La fonction de mise à jour de FLIP comme du PRG de Goldreich consiste en une permutation des bits, il est donc tout à fait logique de fixer la valeur des bits et non une combinaison linéaire de ceux-ci. C'est en ce sens que la fonction de mise à jour et la fonction booléenne interagissent mal.

Dans un contexte plus général, cette fonction de mise à jour peut être plus complexe. De plus, pour les critères liés à la représentation multivariée, se restreindre par exemple à un hyperplan revient à diminuer la dimension de l'espace de 1, et par conséquent modifie significativement la structure de l'ensemble sur lequel on travaille. On change ainsi la parité du nombre de variables, ce qui a un impact important sur les propriétés : si n est pair, alors il y a un sous-corps de taille $\frac{n}{2}$, ce qui, comme nous l'avons vu au chapitre 4 affecte significativement les propriétés des objets que l'on considère.

Une étude bien plus complète consiste à considérer les fonctions augmentées. Soit f une fonction de filtrage booléenne à n variables et Φ une fonction de mise à jour de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ utilisée dans un générateur pseudo-aléatoire, la fonction augmentée [And95] de taille m notée F_m associée est définie par

$$\begin{array}{rccc} F_m & : & \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^m \\ & & x & \longmapsto & (f(x), f\circ\Phi(x), f\circ\Phi^2(x), \ldots, f\circ\Phi^{m-1}(x)) \ . \end{array}$$

En utilisant cette fonction augmentée, nous captons correctement les “bonnes” ou les “mauvaises” interactions entre la fonction de mise à jour et la fonction de filtrage. Toujours dans ce contexte, notre travail sur les fonctions à entrées restreintes ressort alors directement ici en considérant les sous-ensembles pré-images, qui forment une partition de $\mathbb{F}_2^n$. Cependant, calculer cette partition, même pour un m petit semble hors de portée au regard des tailles de n considérées, mais cette partition est évidemment directement exploitable en fonction de la suite chiffrante observée et son utilisation pratique dans une attaque est un champ de recherche vaste, qui reste à explorer.

Bibliographie

[ABD+16a] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser et Kan Yasuda : AES-COPA. *Soumission à la compétition CAESAR*, 2016.

[ABD+16b] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser et Kan Yasuda : ELmD. *Soumission à la compétition CAESAR*, 2016.

[ABD+17] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser et Kan Yasuda : COLM. *Soumission à la compétition CAESAR*, 2017.

[AEL+18] Tomer Ashur, Maria Eichlseder, Martin M. Lauridsen, Gaëtan Leurent, Brice Minaud, Yann Rotella, Yu Sasaki et Benoit Viguier : Cryptanalysis of full morus. *ASIACRYPT 2018*, pages 1–30, 2018.

[AFMV07] Jean-Philippe Aumasson, Matthieu Finiasz, Willi Meier et Serge Vaudenay : TCHo : A hardware-oriented trapdoor cipher. *In* Josef Pieprzyk, Hossein Ghodosi et Ed Dawson, éditeurs : *ACISP 07*, volume 4586 de *LNCS*, pages 184–199. Springer, Heidelberg, juillet 2007.

[AHI04] Michael Alekhnovich, Edward A. Hirsch et Dmitry Itsykson : Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *In* Josep Díaz, Juhani Karhumäki, Arto Lepistö et Donald Sannella, éditeurs : *ICALP 2004*, volume 3142 de *LNCS*, pages 84–96. Springer, Heidelberg, juillet 2004.

[AIK04] Benny Applebaum, Yuval Ishai et Eyal Kushilevitz : Cryptography in NC^0. *In 45th FOCS*, pages 166–175. IEEE Computer Society Press, octobre 2004.

[AIK08] Benny Applebaum, Yuval Ishai et Eyal Kushilevitz : On pseudorandom generators with linear stretch in NC^0. *Computational Complexity*, 17(1):38–69, 2008.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic et Samuel Neves : NORX : Parallel and scalable AEAD. *In* Miroslaw Kutylowski et Jaideep Vaidya, éditeurs : *ESORICS 2014, Part II*, volume 8713 de *LNCS*, pages 19–36. Springer, Heidelberg, septembre 2014.

[AL16] Benny Applebaum et Shachar Lovett : Algebraic attacks against random local functions and their countermeasures. *In* Daniel Wichs et Yishay Mansour, éditeurs : *48th ACM STOC*, pages 1087–1100. ACM Press, juin 2016.

[And95] Ross J. Anderson : Searching for the optimum correlation attack. *In* Bart Preneel, éditeur : *FSE'94*, volume 1008 de *LNCS*, pages 137–143. Springer, Heidelberg, décembre 1995.

[App15] Benny Applebaum : The cryptographic hardness of random local functions – survey. Cryptology ePrint Archive, Report 2015/165, 2015. `http://eprint.iacr.org/2015/165`.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen et Michael Zohner : Ciphers for MPC and FHE. *In* Elisabeth Oswald et Marc Fischlin, éditeurs : *EUROCRYPT 2015, Part I*, volume 9056 de *LNCS*, pages 430–454. Springer, Heidelberg, avril 2015.

[Ava17] Roberto Avanzi : The QARMA block cipher family. *IACR Trans. Symm. Cryptol.*, 2017(1):4–44, 2017.

[AZ04] Martin Aigner et Günter M. Ziegler : *Proofs from THE BOOK (3. ed.)*. Springer, 2004.

[Bab95] Steve Babbage : A space/time trade-off in exhaustive search attacks on stream ciphers. *In European Convention on Security and Detection*, numéro 408. IEEE Conference Publication, 1995.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita et Francesco Regazzoni : Midori : A block cipher for low energy. *In* Tetsu Iwata et Jung Hee Cheon, éditeurs : *ASIACRYPT 2015, Part II*, volume 9453 de *LNCS*, pages 411–436. Springer, Heidelberg, novembre / décembre 2015.

[BCD11] Christina Boura, Anne Canteaut et Christophe De Cannière : Higher-order differential properties of Keccak and Luffa. *In* Antoine Joux, éditeur : *FSE 2011*, volume 6733 de *LNCS*, pages 252–269. Springer, Heidelberg, février 2011.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen et Tolga Yalçin : PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. *In* Xiaoyun Wang et Kazue Sako, éditeurs :

ASIACRYPT 2012, volume 7658 de *LNCS*, pages 208–225. Springer, Heidelberg, décembre 2012.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander et Yann Rotella : Proving resistance against invariant attacks : How to choose the round constants. *In* Jonathan Katz et Hovav Shacham, éditeurs : *CRYPTO 2017, Part II*, volume 10402 de *LNCS*, pages 647–678. Springer, Heidelberg, août 2017.

[BDP+14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche et Ronny Van Keer : Ketje v1. *Soumission à la compétition CAESAR*, 2014.

[BDP+16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche et Ronny Van Keer : Ketje v2. *Soumission à la compétition CAESAR*, 2016.

[BDPA07] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Sponge functions. *Ecrypt Hash Workshop 2007*, 2007.

[BDPA12] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Permutation-based encryption, authentication and authenticated encryption. *In DIAC*, 2012.

[BDPA13] Guido Bertoni, Joan Daemen, Michael Peeters et Gilles Van Assche : Keccak. *In* Thomas Johansson et Phong Q. Nguyen, éditeurs : *EUROCRYPT 2013*, volume 7881 de *LNCS*, pages 313–314. Springer, Heidelberg, mai 2013.

[BDPV08] Guido Bertoni, Joan Daemen, Michael Peeters et Gilles Van Assche : On the indifferentiability of the sponge construction. *In* Nigel P. Smart, éditeur : *EUROCRYPT 2008*, volume 4965 de *LNCS*, pages 181–197. Springer, Heidelberg, avril 2008.

[BDPV12] Guido Bertoni, Joan Daemen, Michael Peeters et Gilles Van Assche : Duplexing the sponge : Single-pass authenticated encryption and other applications. *In* Ali Miri et Serge Vaudenay, éditeurs : *SAC 2011*, volume 7118 de *LNCS*, pages 320–337. Springer, Heidelberg, août 2012.

[Bet11] Luk Bettale : *Cryptanalyse algébrique : outils et applications*. Thèse de doctorat, Université Pierre et Marie Curie, 2011.

[BIM18] Subhadeep Banik, Takanori Isobe et Masakatu Morii : On design of robust lightweight stream cipher with short internal state. *IEICE Transactions*, 101-A(1):99–109, 2018.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich et Siang Meng Sim : The SKINNY family of block ciphers and its low-latency variant MANTIS. *In* Matthew Robshaw et Jonathan Katz, éditeurs : *CRYPTO 2016, Part II*, volume 9815 de *LNCS*, pages 123–153. Springer, Heidelberg, août 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin et C. Vikkelsoe : PRESENT : An ultra-lightweight block cipher. *In* Pascal Paillier et Ingrid Verbauwhede, éditeurs : *CHES 2007*, volume 4727 de *LNCS*, pages 450–466. Springer, Heidelberg, septembre 2007.

[BL16] Karthikeyan Bhargavan et Gaëtan Leurent : On the practical (in-)security of 64-bit block ciphers : Collision attacks on HTTP over TLS and openvpn. *In* Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers et Shai Halevi, éditeurs : *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 456–467. ACM, 2016.

[Bla83] Richard E Blahut : *Theory and practice of error control codes*. Addison-Wesley, 1983.

[Bla85] Richard E. Blahut : *Fast algorithms for digital signal processing*. Addison-Wesley, 1985.

[BLM+05] A. Braeken, J. Lano, N. Mentens, B. Preneel et I. Verbauwhede : SFINKS : a synchronous stream cipher for restricted hardware environments. Soumission au projet eSTREAM [ECR05], 2005. `http://www.ecrypt.eu.org/stream/`.

[BQ09] Andrej Bogdanov et Youming Qiao : On the security of Goldreich's one-way function. *In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 392–405. Springer, 2009.

[BR11] Andrey Bogdanov et Christian Rechberger : A 3-subset meet-in-the-middle attack : Cryptanalysis of the lightweight block cipher KTANTAN. *In* Alex Biryukov, Guang Gong et Douglas R. Stinson, éditeurs : *SAC 2010*, volume 6544 de *LNCS*, pages 229–240. Springer, Heidelberg, août 2011.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks et Louis Wingers : The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `http://eprint.iacr.org/2013/404`.

[Buc76] B. Buchberger : A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, août 1976.

[BW99] Alex Biryukov et David Wagner : Slide attacks. *In* Lars R. Knudsen, éditeur : *FSE'99*, volume 1636 de *LNCS*, pages 245–259. Springer, Heidelberg, mars 1999.

[BW00] Alex Biryukov et David Wagner : Advanced slide attacks. *In* Bart Preneel, éditeur : *EUROCRYPT 2000*, volume 1807 de *LNCS*, pages 589–606. Springer, Heidelberg, mai 2000.

[Can06] Christophe De Cannière : Trivium : A stream cipher construction inspired by block cipher design principles. *In* Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis et Bart Preneel, éditeurs : *ISC 2006*, volume 4176 de *LNCS*, pages 171–186. Springer, Heidelberg, août / septembre 2006.

[Car06] Claude Carlet : On the higher order nonlinearities of algebraic immune functions. *In* Cynthia Dwork, éditeur : *CRYPTO 2006*, volume 4117 de *LNCS*, pages 584–601. Springer, Heidelberg, août 2006.

[Car07] Claude Carlet : Boolean Functions for Cryptography and Error Correcting Codes. *In* Yves Crama et Peter Hammer, éditeurs : *Boolean Methods and Models*. Cambridge University Press, 2007.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier et Renaud Sirdey : Stream ciphers : A practical solution for efficient homomorphic-ciphertext compression. *In* Thomas Peyrin, éditeur : *FSE 2016*, volume 9783 de *LNCS*, pages 313–333. Springer, Heidelberg, mars 2016.

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi et Aaram Yun : Batch fully homomorphic encryption over the integers. *In* Thomas Johansson et Phong Q. Nguyen, éditeurs : *EUROCRYPT 2013*, volume 7881 de *LNCS*, pages 315–335. Springer, Heidelberg, mai 2013.

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi et Yann Rotella : On the concrete security of Goldreich's pseudorandom generator. *ASIACRYPT 2018*, pages 1–55, 2018.

[CEMT14] James Cook, Omid Etesami, Rachel Miller et Luca Trevisan : On the one-way function candidate proposed by Goldreich. *TOCT*, 6(3):14 :1–14 :35, 2014.

[CFG+17] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean et Jean-René Reinhard : Cryptanalysis of NORX v2.0. *IACR Trans. Symm. Cryptol.*, 2017(1):156–174, 2017.

[CJM02] Philippe Chose, Antoine Joux et Michel Mitton : Fast correlation attacks : An algorithmic point of view. *In* Lars R. Knudsen, éditeur : *EUROCRYPT 2002*, volume 2332 de *LNCS*, pages 209–221. Springer, Heidelberg, avril / mai 2002.

[CJS01] Vladimor V. Chepyzhov, Thomas Johansson et Ben J. M. Smeets : A simple algorithm for fast correlation attacks on stream ciphers. *In* Bruce Schneier, éditeur : *FSE 2000*, volume 1978 de *LNCS*, pages 181–195. Springer, Heidelberg, avril 2001.

[CM01] Mary Cryan et Peter Bro Miltersen : On pseudorandom generators in NC. *In* Jirí Sgall, Ales Pultr et Petr Kolman, éditeurs :

Mathematical Foundations of Computer Science 2001, MFCS 2001, volume 2136 de *LNCS*, pages 272–284. Springer, 2001.

[CM03] Nicolas Courtois et Willi Meier : Algebraic attacks on stream ciphers with linear feedback. *In* Eli Biham, éditeur : *EUROCRYPT 2003*, volume 2656 de *LNCS*, pages 345–359. Springer, Heidelberg, mai 2003.

[CMR17a] Claude Carlet, Pierrick Méaux et Yann Rotella : Boolean functions with restricted input and their robustness ; application to the FLIP cipher. *IACR Trans. Symm. Cryptol.*, 2017(3):192–227, 2017.

[CMR17b] Claude Carlet, Pierrick Méaux et Yann Rotella : Boolean functions with restricted input and their robustness ; application to the FLIP cipher. Cryptology ePrint Archive, Report 2017/097, 2017. `http://eprint.iacr.org/2017/097`.

[CN12] Anne Canteaut et María Naya-Plasencia : Correlation attacks on combination generators. *Cryptography and Communications*, 4(3-4):147–171, 2012.

[CNV13] Anne Canteaut, María Naya-Plasencia et Bastien Vayssière : Sieve-in-the-middle : Improved MITM attacks. *In* Ran Canetti et Juan A. Garay, éditeurs : *CRYPTO 2013, Part I*, volume 8042 de *LNCS*, pages 222–240. Springer, Heidelberg, août 2013.

[Cou03] Nicolas Courtois : Fast algebraic attacks on stream ciphers with linear feedback. *In* Dan Boneh, éditeur : *CRYPTO 2003*, volume 2729 de *LNCS*, pages 176–194. Springer, Heidelberg, août 2003.

[CR16] Anne Canteaut et Yann Rotella : Attacks against filter generators exploiting monomial mappings. *In* Thomas Peyrin, éditeur : *FSE 2016*, volume 9783 de *LNCS*, pages 78–98. Springer, Heidelberg, mars 2016.

[CT00] Anne Canteaut et Michaël Trabbia : Improved fast correlation attacks using parity-check equations of weight 4 and 5. *In* Bart Preneel, éditeur : *EUROCRYPT 2000*, volume 1807 de *LNCS*, pages 573–588. Springer, Heidelberg, mai 2000.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller et Adi Shamir : Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. *In* Reihaneh Safavi-Naini et Ran Canetti, éditeurs : *CRYPTO 2012*, volume 7417 de *LNCS*, pages 719–740. Springer, Heidelberg, août 2012.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gergor Leander, Eik List, Florian Mendel et Christian Rechberger : Rasta : A cipher with low ANDdepth and few ANDs per bit. *CRYPTO 2018*, 2018.

[DEMS17] Christoph Dobraunig, Maria Eichlseder, Florian Mendel et Martin Schläffer : Ascon. *Soumission à la compétition CAESAR*, 2017.

[DF04] David Steven Dummit et Richard M Foote : *Abstract algebra*. John Wiley and Sons, Inc., 2004.

[DGPW12] Alexandre Duc, Jian Guo, Thomas Peyrin et Lei Wei : Unaligned rebound attack : Application to Keccak. *In* Anne Canteaut, éditeur : *FSE 2012*, volume 7549 de *LNCS*, pages 402–421. Springer, Heidelberg, mars 2012.

[DH76] Whitfield Diffie et Martin E. Hellman : New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[DHS16] Yarkin Doröz, Yin Hu et Berk Sunar : Homomorphic AES evaluation using the modified LTV scheme. *Des. Codes Cryptography*, 80(2):333–358, 2016.

[Dil74] J.F. Dillon : *Elementary Hadamard difference sets*. Thèse de doctorat, Univ Maryland, 1974.

[DK13] Ilya Dumer et Olga Kapralova : Spherically punctured biorthogonal codes. *IEEE Trans. Information Theory*, 59(9):6010–6017, 2013.

[DK17] Ilya Dumer et Olga Kapralova : Spherically punctured reed-muller codes. *IEEE Trans. Information Theory*, 63(5):2773–2780, 2017.

[DL18] Sébastien Duval et Gaëtan Leurent : MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2018(2):48–78, 2018.

[DLR16] Sébastien Duval, Virginie Lallemand et Yann Rotella : Cryptanalysis of the FLIP family of stream ciphers. *In* Matthew Robshaw et Jonathan Katz, éditeurs : *CRYPTO 2016, Part I*, volume 9814 de *LNCS*, pages 457–475. Springer, Heidelberg, août 2016.

[DLWQ17] Xiaoyang Dong, Zheng Li, Xiaoyun Wang et Ling Qin : Cube-like attack on round-reduced initialization of Ketje Sr. *IACR Trans. Symm. Cryptol.*, 2017(1):259–280, 2017.

[Dob95] Hans Dobbertin : Construction of Bent functions and balanced Boolean functions with high nonlinearity. *In* Bart Preneel, éditeur : *FSE'94*, volume 1008 de *LNCS*, pages 61–74. Springer, Heidelberg, décembre 1995.

[DPAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche et Vincent Rijmen : Noekeon. *In Proceedings of the first NESSIE Workshop*, 2000.

[DR02] Joan Daemen et Vincent Rijmen : *The Design of Rijndael : AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DSES14] Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth et Berk Sunar : Toward practical homomorphic evaluation of block ciphers using Prince. *In* Rainer Böhme, Michael Brenner, Tyler Moore et Matthew Smith, éditeurs : *FC 2014 Workshops*, volume 8438 de *LNCS*, pages 208–220. Springer, Heidelberg, mars 2014.

[DSP07] Orr Dunkelman, Gautham Sekar et Bart Preneel : Improved meet-in-the-middle attacks on reduced-round DES. *In* K. Srinathan, C. Pandu Rangan et Moti Yung, éditeurs : *INDOCRYPT 2007*, volume 4859 de *LNCS*, pages 86–100. Springer, Heidelberg, décembre 2007.

[DW97] Ed Dawson et Chuan-Kun Wu : On the linear structure of symmetric Boolean functions. *Australasian Journal of Combinatorics*, 16:239–243, 1997.

[ECR05] ECRYPT - European Network of Excellence in Cryptology : The eSTREAM Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`, 2005.

[EJ00] P. Ekdahl et T. Johansson : SNOW - a new stream cipher. *In Proceedings of First NESSIE Workshop*, Heverlee, Belgique, 2000.

[Fau99] Jean-Charles Faugere : A new efficient algorithm for computing Grobner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61 – 88, 1999.

[Fau02] Jean Charles Faugere : A new efficient algorithm for computing Grobner bases without reduction to zero (F5). *In Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM.

[FNR18] Thomas Fuhr, María Naya-Plasencia et Yann Rotella : State-recovery attacks on modified Ketje Jr. *IACR Trans. Symmetric Cryptol.*, 2018(1):29–56, 2018.

[FS09] Philippe Flajolet et Robert Sedgewick : *Analytic Combinatorics*. Cambridge University Press, 2009.

[Gen09] Craig Gentry : Fully homomorphic encryption using ideal lattices. *In* Michael Mitzenmacher, éditeur : *41st ACM STOC*, pages 169–178. ACM Press, mai / juin 2009.

[GG04] Solomon W. Golomb et Guang Gong : *Signal Design for Good Correlation : For Wireless Communication, Cryptography, and Radar*. Cambridge University Press, 2004.

[GGM86] Oded Goldreich, Shafi Goldwasser et Silvio Micali : How to construct random functions. *Journal of the ACM*, 33(4):792–807, octobre 1986.

[GHS12] Craig Gentry, Shai Halevi et Nigel P. Smart : Homomorphic evaluation of the AES circuit. *In* Reihaneh Safavi-Naini et Ran

Canetti, éditeurs : *CRYPTO 2012*, volume 7417 de *LNCS*, pages 850–867. Springer, Heidelberg, août 2012.

[Gie95] Mark Giesbrecht : Nearly optimal algorithms for canonical matrix forms. *SIAM J. Comput.*, 24(5):948–969, 1995.

[GJN+16] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki et Siang Meng Sim : Invariant subspace attack against Midori64 and the resistance criteria for S-box designs. *IACR Trans. Symm. Cryptol.*, 2016(1):33–56, 2016. `http://tosc.iacr.org/index.php/ToSC/article/view/534`.

[GLS16] Jian Guo, Meicheng Liu et Ling Song : Linear structures : Applications to cryptanalysis of round-reduced Keccak. *In* Jung Hee Cheon et Tsuyoshi Takagi, éditeurs : *ASIACRYPT 2016, Part I*, volume 10031 de *LNCS*, pages 249–274. Springer, Heidelberg, décembre 2016.

[GM17] Shay Gueron et Nicky Mouha : SPHINCS-simpira : Fast stateless hash-based signatures with post-quantum security. Cryptology ePrint Archive, Report 2017/645, 2017. `http://eprint.iacr.org/2017/645`.

[Gol97] Jovan Dj. Golic : Cryptanalysis of alleged A5 stream cipher. *In* Walter Fumy, éditeur : *EUROCRYPT'97*, volume 1233 de *LNCS*, pages 239–255. Springer, Heidelberg, mai 1997.

[Gol00] Oded Goldreich : Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. `http://eprint.iacr.org/2000/063`.

[Gon11] Guang Gong : A closer look at selective DFT attacks. CACR report 2011-35, University of Waterloo, 2011.

[Got66] D. H. Gottlieb : A certain class of incidence matrices. *Proceedings of the American Mathematical Society*, 17(6):1233–1237, 1966.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann et Matthew J. B. Robshaw : The LED block cipher. *In* Bart Preneel et Tsuyoshi Takagi, éditeurs : *CHES 2011*, volume 6917 de *LNCS*, pages 326–341. Springer, Heidelberg, septembre / octobre 2011.

[GRHH11] Guang Gong, Sondre Rønjom, Tor Helleseth et Honggang Hu : Fast discrete Fourier spectra attacks on stream ciphers. *IEEE Trans. Information Theory*, 57(8):5555–5565, 2011.

[HCJ02] Shai Halevi, Don Coppersmith et Charanjit S. Jutla : Scream : A software-efficient stream cipher. *In* Joan Daemen et Vincent Rijmen, éditeurs : *FSE 2002*, volume 2365 de *LNCS*, pages 195–209. Springer, Heidelberg, février 2002.

[Hel80] Martin E. Hellman : A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.

[Hel11] Tor Helleseth : Maximal-length sequences. *In Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 763–766. Springer, 2011.

[Her75] Israel Nathan Herstein : *Topics in Algebra.* John Wiley & Sons, Lexington, USA, 1975.

[Her86] Tore Herlestam : On linear shift registers with permuted feedback. *In* Ingemar Ingemarsson, éditeur : *EUROCRYPT'86*, LNCS, pages 38–39. Springer, Heidelberg, mai 1986.

[HJB09] M. Hell, T. Johansson et L. Brynielsson : An overview of distinguishing attacks on stream ciphers. *Cryptography and Communications*, 1(1):71–94, 2009.

[HJM05] M. Hell, T. Johansson et W. Meier : Grain : A stream cipher for constrained environments. Soumission au projet eSTREAM [ECR05], 2005. `http://www.ecrypt.eu.org/stream/`.

[HR00] Philip Hawkes et Gregory G. Rose : Exploiting multiples of the connection polynomial in word-oriented stream ciphers. *In* Tatsuaki Okamoto, éditeur : *ASIACRYPT 2000*, volume 1976 de *LNCS*, pages 303–316. Springer, Heidelberg, décembre 2000.

[HR11] Tor Helleseth et Sondre Rønjom : Simplifying algebraic attacks with univariate analysis. *In Information Theory and Applications - ITA 2011*, pages 153–159. IEEE, 2011.

[Jea16] Jérémy Jean : Cryptanalysis of Haraka. *IACR Trans. Symm. Cryptol.*, 2016(1):1–12, 2016. `http://tosc.iacr.org/index.php/ToSC/article/view/531`.

[JJ99] Thomas Johansson et Fredrik Jönsson : Improved fast correlation attacks on stream ciphers via convolutional codes. *In* Jacques Stern, éditeur : *EUROCRYPT'99*, volume 1592 de *LNCS*, pages 347–362. Springer, Heidelberg, mai 1999.

[JJ00] Thomas Johansson et Fredrik Jönsson : Fast correlation attacks through reconstruction of linear polynomials. *In* Mihir Bellare, éditeur : *CRYPTO 2000*, volume 1880 de *LNCS*, pages 300–315. Springer, Heidelberg, août 2000.

[JK77] N.L. Johnson et S. Kotz : *Urn models and their application : an approach to modern discrete probability theory.* Wiley Series in Probability and Statistics : Applied Probability and Statist ICS Sesction Series. Wiley, 1977.

[JN15] Jérémy Jean et Ivica Nikolic : Internal differential boomerangs : Practical analysis of the round-reduced Keccak-f permutation. *In* Gregor Leander, éditeur : *FSE 2015*, volume 9054 de *LNCS*, pages 537–556. Springer, Heidelberg, mars 2015.

[JNPS17] Jérémy Jean, Ivica Nikolić, Thomas Peyrin et Yannick Seurin : Deoxys-II. *Soumission à la compétition CAESAR*, 2017.

[Jou09] Antoine Joux : *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 2009.

[Ker83] Auguste Kerckhoffs : La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, janvier 1883.

[Key76] Edwin L. Key : An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Trans. Information Theory*, 22:732–736, 1976.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel et Christian Rechberger : Haraka - efficient short-input hashing for post-quantum applications. Cryptology ePrint Archive, Report 2016/098, 2016. `http://eprint.iacr.org/2016/098`.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann et Matthew J. B. Robshaw : PRINTcipher : A block cipher for IC-printing. *In* Stefan Mangard et François-Xavier Standaert, éditeurs : *CHES 2010*, volume 6225 de *LNCS*, pages 16–32. Springer, Heidelberg, août 2010.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen et Friedrich Wiemer : Shorter linear straight-line programs for MDS matrices. *IACR Trans. Symm. Cryptol.*, 2017(4):188–211, 2017.

[KNRS11] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck et Martin Schläffer : Cryptanalysis of Luffa v2 components. *In* Alex Biryukov, Guang Gong et Douglas R. Stinson, éditeurs : *SAC 2010*, volume 6544 de *LNCS*, pages 388–409. Springer, Heidelberg, août 2011.

[KR07] Lars R. Knudsen et Vincent Rijmen : Known-key distinguishers for some block ciphers. *In* Kaoru Kurosawa, éditeur : *ASIACRYPT 2007*, volume 4833 de *LNCS*, pages 315–324. Springer, Heidelberg, décembre 2007.

[KR17] Ted Krovetz et Philip Rogaway : OCB. *Soumission à la compétition CAESAR*, 2017.

[KSC78] V.F. Kolchin, B.A. Sevastianov et V.P. Chistiakov : *Random allocations*. Scripta series in mathematics. V. H. Winston, 1978.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi et Erik Zenner : A cryptanalysis of PRINTcipher : The invariant subspace attack. *In* Phillip Rogaway, éditeur : *CRYPTO 2011*, volume 6841 de *LNCS*, pages 206–221. Springer, Heidelberg, août 2011.

[Lan93] G. Landsberg : Ueber eine Anzahlbestimmung und eine damit zusammenhängende Reihe. *J. reine angew. Math. 111*, pages 87–88, 1893.

[LMR+09] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen et Martin Schläffer : Rebound distinguishers :

Results on the full Whirlpool compression function. *In* Mitsuru Matsui, éditeur : *ASIACRYPT 2009*, volume 5912 de *LNCS*, pages 126–143. Springer, Heidelberg, décembre 2009.

[LMR15] Gregor Leander, Brice Minaud et Sondre Rønjom : A generic approach to invariant subspace attacks : Cryptanalysis of robin, iSCREAM and Zorro. *In* Elisabeth Oswald et Marc Fischlin, éditeurs : *EUROCRYPT 2015, Part I*, volume 9056 de *LNCS*, pages 254–283. Springer, Heidelberg, avril 2015.

[LN83] Rudolf Lidl et Harald Niederreiter : *Finite Fields*. Cambridge University Press, 1983.

[LN14] Tancrède Lepoint et Michael Naehrig : A comparison of the homomorphic encryption schemes FV and YASHE. *In* David Pointcheval et Damien Vergnaud, éditeurs : *AFRICACRYPT 14*, volume 8469 de *LNCS*, pages 318–335. Springer, Heidelberg, mai 2014.

[LN15] Virginie Lallemand et María Naya-Plasencia : Cryptanalysis of full Sprout. *In* Rosario Gennaro et Matthew J. B. Robshaw, éditeurs : *CRYPTO 2015, Part I*, volume 9215 de *LNCS*, pages 663–682. Springer, Heidelberg, août 2015.

[Luc78] Edouard Lucas : Théorie des fonctions numériques simplement périodiques. *American Journal of Mathematics*, 1(2):184–196, 1878.

[LV04] Yi Lu et Serge Vaudenay : Faster correlation attack on Bluetooth keystream generator E0. *In* Matthew Franklin, éditeur : *CRYPTO 2004*, volume 3152 de *LNCS*, pages 407–425. Springer, Heidelberg, août 2004.

[Mac16] Hugo Daniel Macedo : Gaussian elimination is not optimal, revisited. *J. Log. Algebr. Meth. Program.*, 85(5):999–1010, 2016.

[Mas69] James L. Massey : Shift-register synthesis and BCH decoding. *IEEE Trans. Information Theory*, 15(1):122–127, 1969.

[McE87] Robert J. McEliece : *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.

[MFI01] Miodrag J. Mihaljevic, Marc P. C. Fossorier et Hideki Imai : A low-complexity and high-performance algorithm for the fast correlation attack. *In* Bruce Schneier, éditeur : *FSE 2000*, volume 1978 de *LNCS*, pages 196–212. Springer, Heidelberg, avril 2001.

[MJB04] Alexander Maximov, Thomas Johansson et Steve Babbage : An improved correlation attack on A5/1. *In* Helena Handschuh et Anwar Hasan, éditeurs : *SAC 2004*, volume 3357 de *LNCS*, pages 1–18. Springer, Heidelberg, août 2004.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert et Claude Carlet : Towards stream ciphers for efficient FHE with low-noise ciphertexts. *In* Marc Fischlin et Jean-Sébastien Coron, éditeurs : *EUROCRYPT 2016, Part I*, volume 9665 de *LNCS*, pages 311–343. Springer, Heidelberg, mai 2016.

[MPC04] Willi Meier, Enes Pasalic et Claude Carlet : Algebraic attacks and decomposition of boolean functions. *In* Christian Cachin et Jan Camenisch, éditeurs : *EUROCRYPT 2004*, volume 3027 de *LNCS*, pages 474–491. Springer, 2004.

[MS77] F. Jessie MacWilliams et Neil J.A. Sloane : *The theory of error-correcting codes*. North-Holland, 1977.

[MS89] Willi Meier et Othmar Staffelbach : Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[MS94] James L. Massey et Shirlei Serconek : A Fourier transform approach to the linear complexity of nonlinearly filtered sequences. *In* Yvo Desmedt, éditeur : *CRYPTO'94*, volume 839 de *LNCS*, pages 332–340. Springer, Heidelberg, août 1994.

[MST03] Elchanan Mossel, Amir Shpilka et Luca Trevisan : On e-biased generators in NC0. *In FOCS 2003*, pages 136–145. IEEE Computer Society, 2003.

[MZD18] Sihem Mesnager, Zhengchun Zhou et Cunsheng Ding : On the nonlinearity of Boolean functions with restricted input. *Cryptography and Communications*, mars 2018.

[Nay07] María Naya-Plasencia : Cryptanalysis of Achterbahn-128/80. *In* Alex Biryukov, éditeur : *FSE 2007*, volume 4593 de *LNCS*, pages 73–86. Springer, Heidelberg, mars 2007.

[Nay11] María Naya-Plasencia : How to improve rebound attacks. *In* Phillip Rogaway, éditeur : *CRYPTO 2011*, volume 6841 de *LNCS*, pages 188–205. Springer, Heidelberg, août 2011.

[NG05] Yassir Nawaz et Guang Gong : The WG stream cipher. Soumission au projet eSTREAM [ECR05], 2005. `http://www.ecrypt.eu.org/stream/`.

[NLV11] Michael Naehrig, Kristin E. Lauter et Vinod Vaikuntanathan : Can homomorphic encryption be practical ? *In* Christian Cachin et Thomas Ristenpart, éditeurs : *CCSW 2011*, pages 113–124. ACM, 2011.

[OW14] Ryan O'Donnell et David Witmer : Goldreich's PRG : evidence for near-optimal polynomial stretch. *In IEEE 29th Conference on Computational Complexity, CCC 2014*, pages 1–12. IEEE Computer Society, 2014.

[RC10] Sondre Rønjom et Carlos Cid : Nonlinear equivalence of stream ciphers. *In* Seokhie Hong et Tetsu Iwata, éditeurs : *FSE 2010*, volume 6147 de *LNCS*, pages 40–54. Springer, Heidelberg, février 2010.

[RGH07] Sondre Rønjom, Guang Gong et Tor Helleseth : On attacks on filtering generators using linear subspace structures. *In* Solomon W. Golomb, Guang Gong, Tor Helleseth et Hong-Yeop Song,

éditeurs : *Sequences, Subsequences, and Consequences, SSC 2007*, volume 4893 de *LNCS*, pages 204–217. Springer, 2007.

[RH07] Sondre Rønjom et Tor Helleseth : A new attack on the filter generator. *IEEE Information Theory*, 53(5):1752–1758, 2007.

[Røn15] Sondre Rønjom : Powers of subfield polynomials and algebraic attacks on word-based stream ciphers. Cryptology ePrint Archive, Report 2015/495, 2015. `http://eprint.iacr.org/2015/495`.

[Røn16] Sondre Rønjom : Invariant subspaces in Simpira. Cryptology ePrint Archive, Report 2016/248, 2016. `http://eprint.iacr.org/2016/248`.

[RS87] Rainer A. Rueppel et Othmar Staffelbach : Products of linear recurring sequences with maximum complexity. *IEEE Trans. Information Theory*, 33(1):124–131, 1987.

[RSA78] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman : A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Rue86] Rainer A. Rueppel : *Analysis and Design of stream ciphers*. Springer-Verlag, 1986.

[Sas11] Yu Sasaki : Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. *In* Antoine Joux, éditeur : *FSE 2011*, volume 6733 de *LNCS*, pages 378–396. Springer, Heidelberg, février 2011.

[Sav94] Petr Savický : On the bent boolean functions that are symmetric. *Eur. J. Comb.*, 15(4):407–410, 1994.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini et Yarik Markov : The first collision for full SHA-1. *In* Jonathan Katz et Hovav Shacham, éditeurs : *CRYPTO 2017, Part I*, volume 10401 de *LNCS*, pages 570–596. Springer, Heidelberg, août 2017.

[Sha49] C. Shannon : Communication theory of secrecy systems. *Bell System Technical Journal, Vol 28, pp. 656–715*, octobre 1949.

[Sie84] Thomas Siegenthaler : Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Information Theory*, IT-30(5):776–780, 1984.

[Sie85] Thomas Siegenthaler : Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Computers*, 34(1):81–85, 1985.

[TIM+18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki et Bin Zhang : Fast correlation attack revisited, cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. *CRYPTO 2018*, 2018.

[TLS16] Yosuke Todo, Gregor Leander et Yu Sasaki : Nonlinear invariant attack - practical attack on full SCREAM, iSCREAM, and Midori64. *In* Jung Hee Cheon et Tsuyoshi Takagi, éditeurs : *ASIACRYPT 2016, Part II*, volume 10032 de *LNCS*, pages 3–33. Springer, Heidelberg, décembre 2016.

[WH17] Hongjun Wu et Tao Huang : MORUS. *Soumission à la compétition CAESAR*, 2017.

[Wil90] Richard M. Wilson : A diagonal form for the incidence matrices of t-subsets vs. k-subsets. *Eur. J. Comb.*, 11(6):609–615, 1990.

[WP17] Hongjun Wu et Bart Preneel : AEGIS. *Soumission à la compétition CAESAR*, 2017.

[Wu17] Hongjun Wu : ACORN. *Soumission à la compétition CAESAR*, 2017.

[YG01] Amr M. Youssef et Guang Gong : Hyper-bent functions. *In* Birgit Pfitzmann, éditeur : *EUROCRYPT 2001*, volume 2045 de *LNCS*, pages 406–419. Springer, Heidelberg, mai 2001.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang et Ingrid Verbauwhede : RECTANGLE : A bit-slice lightweight block cipher suitable for multiple platforms. Cryptology ePrint Archive, Report 2014/084, 2014. `http://eprint.iacr.org/2014/084`.

[ZGM17] Bin Zhang, Xinxin Gong et Willi Meier : Fast correlation attacks on Grain-like small state stream ciphers. *IACR Trans. Symm. Cryptol.*, 2017(4):58–81, 2017.

[Zie59] Neal Zierler : Linear recurring sequences. *J. Soc. Indus. Appl. Math.*, 7:31–48, 1959.