Codes de Reed-Muller et cryptanalyse du registre filtré.

Frédéric Didier

Thèse

présentée à
l´École Polytechnique

pour obtenir le titre de
Docteur de l´École Polytechnique
en Informatique

par
Frédéric Didier

Codes de Reed-Muller et cryptanalyse du registre filtré

Soutenue le 18/12/2007

Rapporteurs :	Thomas Johansson
	Gilles Zémor
Directeur :	Jean-Pierre Tillich
Jury :	Anne Canteaut
	Jean-Charles Faugère
	Antoine Joux
	François Morain
	Amin Shokrollahi

Table des matières

Remerciements

Je tiens à remercier toutes les personnes qui ont rendu ces trois années de thèse particulièrement enrichissantes. D'une part, ceux avec qui j'ai travaillé et qui ont directement contribué à mes travaux scientifiques et à l'élaboration de ce document, mais aussi tous les autres avec qui j'ai passé de bons moments et qui m'ont ainsi permis de réfléchir dans les meilleures conditions.

Je remercie donc tout d'abord mon directeur Jean-Pierre Tillich, non seulement parce qu'il m'a donné la chance de travailler sur un sujet que j'ai beaucoup aimé mais aussi pour son encadrement exemplaire. Sans toutes les heures qu'il a passé à écouter mes idées, me conseiller, me relire et m'aider dans la rédaction des preuves et des divers documents que j'ai été amené à produire, mon travail ne serait certainement pas ce qu'il est.

Je remercie les autres membres permanents du projet pour leurs disponibilités, leurs conseils avisés et toutes les discussions que nous avons pu avoir. Dans le désordre, Anne Canteaut, Daniel Augot, Pascale Charpin et Nicolas Sendrier. Merci aussi à Christelle pour tout son travail administratif et sa compagnie de tous les jours au projet.

Je remercie Mathieu, Scarabé, Yann, Andrea, Thomas et Maria qui, plus que des collègues de travail, sont devenus de véritables amis. Un grand merci à Matthieu Finiasz pour m'avoir, entre autre, fait découvrir le projet. Merci aussi à tous les autres CODEURS : chercheurs, post-doctorants, thésards et stagiaires qui ont contribué à rendre la vie au projet si animée.

Je tiens à remercier Thomas Johansson et Gilles Zémor pour avoir accepté d'être rapporteurs de ma thèse ainsi que François Morain, Antoine Joux, Jean-Charles Faugère, Anne Canteaut et Amin Shokrollahi de faire partie de mon jury.

Je remercie également tous mes amis de leur présence hors du cadre du travail. Merci donc à Raph, Fab, Vaness, Jip, Elisa, Anne-Ruxandra, Charles, Jules, Jen, Hélène, Audrey, Tristan, Matt, Olive, Rémi, Pascal, Titi, Adrian, Virginie & Uliana. Je remercie également tous les autres, notamment les mites, les chimistes, les grimpeurs, les joueurs de Volley, mes compagnons de navette, les Coulange, les habitants de RedCastle et ceux du boulevard Lefebvre et assimilés.

Les dernières lignes de ces remerciements vont à mes parents, ma sœur Anne-Laure et toute ma famille qui m'a toujours soutenu durant cette thèse.

Aperçu de la thèse

Les résultats présentés dans ce document sont le fruit de mes trois années de thèse, de septembre 2004 à 2007, au sein du projet CODES à L'INRIA Rocquencourt. Ces travaux, appartenant au domaine de la cryptographie mais aussi des codes correcteurs, ont fait l'objet de plusieurs publications :

[Did06a] Frédéric Didier. A new bound on the block error probability after decoding over the erasure channel. *IEEE Transactions on Information Theory*, 52(10), 4496–4503, Octobre 2006.

[DT06] Frédéric Didier et Jean-Pierre Tillich. Computing the algebraic immunity efficiently. *Fast Software Encryption, FSE*, *LNCS* 4047, 359–374, Mars 2006.

[Did06b] Frédéric Didier. Using Wiedemann's algorithm to compute the immunity against algebraic and fast algebraic attacks. *Indocrypt 2006*, *LNCS* 4329, 236–250, Décembre 2006.

[DLC07] Frédéric Didier et Yann Laigle-Chapuy. Finding low-weight polynomial multiples using discrete logarithm. *IEEE International Symposium on Information Theory - ISIT 2007*, Juin 2007.

[Did07] Frédéric Didier. Attacking the filter generator by finding zero inputs of the filtering function. *Indocrypt 2007*, Décembre 2007.

L'ensemble de ces résultats a été obtenu en s'intéressant de plus ou moins loin aux différentes attaques sur le registre filtré qui est une des constructions les plus simples de chiffrement à flot. C'est à travers cette application que nous avons choisi de les présenter. Le document est organisé de la manière suivante :

Nous commençons dans le chapitre 1 par donner une rapide introduction aux codes correcteurs, à la cryptographie et surtout aux chiffrements à flot. Les chapitres 2 et 3 posent ensuite les bases mathématiques nécessaires à la compréhension des deux éléments-clefs du registre filtré que sont les LFSR et les fonctions booléennes. Le registre filtré proprement dit est présenté au chapitre 4 avec un rapide état de l'art des différentes attaques que l'on peut mener contre une telle construction.

Le chapitre 5 s'intéresse aux calculs des relations de parité de poids

faible vérifiées par la séquence produite par un LFSR et qui servent dans de nombreuses attaques de nature statistique sur le registre filtré. Une attaque qui les utilise est d'ailleurs présentée juste après, au chapitre 6. Ces deux chapitres décrivent en fait le travail effectué durant la dernière année de ma thèse et correspondent respectivement aux articles [DLC07] et [Did07].

Le reste de la thèse abandonne le domaine des attaques de nature statistique pour s'intéresser aux attaques de nature algébrique sur le registre filtré. On introduit ainsi la notion de complexité linéaire et une attaque qui l'utilise au chapitre 7, puis on décrit les attaques algébriques au chapitre 8. Nos travaux portent en fait sur l'immunité algébrique d'une fonction booléenne qui est un entier qui permet de mesurer le degré de résistance du registre filtré face aux attaques du chapitre 8.

L'immunité algébrique est une notion cryptographique qui est intimement liée au problème du décodage des codes de Reed-Muller sur le canal à effacements, problème appartenant au domaine des codes correcteurs. Nous présentons ces codes au chapitre 9 et leur relation avec l'immunité algébrique au chapitre 10. Cette interaction entre codes correcteurs et cryptographie qui offre une vision différente du même problème est très intéressante et nous a permis d'obtenir des résultats nouveaux dans les deux disciplines.

Du côté des codes correcteurs, nous avons ainsi obtenu des résultats sur le comportement des codes linéaires sur le canal à effacements publiés dans [Did06a] et présentés au chapitre 11. Ces résultats nous donnent également une borne sur l'immunité algébrique d'une fonction booléenne aléatoire. Nous avons également obtenu un algorithme de décodage sur le canal à effacements qui est efficace pour les codes qui s'encodent efficacement ([Did06b], chapitre 14).

Ce problème de décodage, identique à celui du calcul de l'immunité algébrique, occupe la dernière partie de ce document. Nous donnons ainsi un état de l'art des algorithmes pour calculer l'immunité algébrique d'une fonction booléenne au chapitre 12. Puis, dans les deux derniers chapitres (13 et 14), nous détaillons deux d'entre eux qui correspondent respectivement aux articles [DT06] et [Did06b].

Chapitre 1

Introduction

Aujourd'hui l'information transite presque entièrement sous une forme numérisée sur une très grande variété de support et cela avec des quantités, des débits et des services impressionnants. Tout cela met en jeu énormément de disciplines pour lesquelles la recherche est encore très active comme la théorie des codes correcteurs ou la cryptographie.

Nous décrivons ici très rapidement et succinctement ce que sont ces deux disciplines dans lesquelles s'inscrit cette thèse. Dans les deux dernières sections de ce chapitre introductif, nous détaillerons un peu plus les chiffrements à flot et leur construction qui est basée sur des générateurs pseudo-aléatoires. Pour une introduction un peu plus détaillée de la cryptographie, nous invitons le lecteur à consulter les articles [CL01] et [Can02]. Un bon cours introductif peut également être trouvé dans [Zém00].

1.1 Une introduction aux codes correcteurs

Les codes correcteurs d'erreurs sont des outils qui permettent d'améliorer la fiabilité d'un échange d'information sur un canal bruité. Ils sont utilisés dans la plupart des technologies de communications modernes comme dans le WIFI, l'ADSL, les téléphones portables, les communications satellites, etc. Ils servent aussi lors du stockage de l'information dans les disques durs, sur les CD ou les DVD.

Le principe des codes correcteurs est d'introduire une redondance dans l'information que l'on veut transmettre ou stocker. Si le nombre d'erreurs n'est pas trop élevé et que la redondance est bien structurée, il sera alors possible de reconstruire l'information initiale sans aucune erreur. Il existe plusieurs techniques pour réaliser cela, mais nous nous concentrerons dans cette introduction sur les codes en bloc linéaires.

Dans de tels codes, l'information est codée comme une suite de lettres sur un corps fini $\mathbf{F}$. Pour simplifier, nous considérerons uniquement des messages binaires où chaque lettre vaut soit 0 soit 1, on se place donc sur le corps fini

à deux éléments $\mathbf{F}_2$. On découpe alors un message à transmettre en blocs de k bits qui seront encodés indépendamment les uns des autres. Chaque mot de k bits est en fait transformé par une application linéaire en un mot plus long de n bits avant d'être transmis. La *longueur* n du code est donc plus grande que la *dimension* k du code et c'est ainsi que la redondance est ajoutée. La quantité k/n qui mesure cet ajout est appelée le *rendement* du code.

Un code linéaire désigné par $\mathcal{C}$ est donc entièrement déterminé par la matrice de l'application linéaire, notée M et appelée *matrice génératrice* du code, qui transforme un mot de k bits en un mot de code de n bits. Les k bits initiaux à transmettre sont usuellement appelés *bits d'information*. Si l'on ne s'intéresse pas à la manière d'encoder l'information, un code $\mathcal{C}$ est alors simplement un sous-espace vectoriel de dimension k (l'image de M) de l'espace vectoriel des mots de n bits.

Avant de voir comment utiliser cette redondance, il convient dans un premier temps de définir la nature des erreurs qui dépendent en fait du *canal de transmission* que l'on va utiliser pour envoyer nos messages. Une des modélisations d'un canal la plus simple, et aussi l'une des plus utilisées, est ce que l'on appelle le *canal binaire symétrique* ou BSC de l'anglais Binary Symmetric Channel. Ce canal est paramétré par une *probabilité d'erreur* p qui est la probabilité pour tout bit transitant par le canal de changer de valeur (un 0 devient 1 et vice versa). Dans la suite, nous nous placerons sur ce canal, mais il en existe de nombreux autres comme le canal à bruit blanc gaussien ou le canal à effacements que nous décrirons en détail au chapitre 10.

Maintenant, lorsque l'on transmet un mot de code c sur le canal BSC, on reçoit un mot de n bits y qui n'est pas forcément dans le code $\mathcal{C}$. Effectuer un décodage optimal revient alors à retrouver le mot de code c qui maximise la probabilité d'observer y sachant que l'on a émis c, on parle de décodage ML au *maximum de vraisemblance*.

Sur le canal BSC, décoder au maximum de vraisemblance est équivalent au problème de trouver le mot de code le plus proche de y au sens de la distance de Hamming qui correspond au nombre de positions sur lesquelles les bits des deux mots diffèrent. Cependant, effectuer un *décodage complet*, c'est-à-dire pour n'importe quel mot y, est très difficile. On se restreint en général à des algorithmes qui fonctionnent lorsqu'il n'y a pas trop d'erreurs.

On introduit ainsi la *distance minimale* d_1 d'un code qui est la plus petite distance entre deux de ses mots. Pour un code linéaire, cela correspond également au plus petit poids de Hamming d'un mot du code non nul. Si le nombre d'erreurs sur le canal BSC est strictement inférieur à $\lfloor \frac{d_1-1}{2} \rfloor$, un décodage au maximum de vraisemblance renverra toujours la bonne solution. La plupart du temps, les algorithmes de décodage ne fonctionnent pas si le nombre d'erreurs est plus grand que la moitié de la distance minimale. Mais ce n'est pas vrai pour tous les codes et dans cette thèse, on verra

plusieurs algorithmes qui permettent un décodage complet au maximum de vraisemblance.

Mentionnons enfin qu'en théorie de l'information, il est possible de définir la *capacité* du canal qui correspond au rendement maximal d'un code au delà duquel un décodage au maximum de vraisemblance a toutes les chances de se tromper. On considère pour cela des codes dont la longueur n devient infinie et en pratique il est difficile de construire des codes qui atteignent la capacité d'un canal. On peut ainsi montrer que la capacité du canal BSC de probabilité d'erreur p est de $1-h(p)$ où h est la fonction d'entropie binaire : $h(p) : p \mapsto -p\log_2(p) - (1-p)\log_2(1-p)$.

1.2 Une introduction à la cryptographie

La cryptographie, de la même manière que les codes correcteurs d'erreurs, s'intéresse au traitement de l'information. Un des ses buts premiers est de garantir la confidentialité des données que l'on veut transmettre ou stocker en *chiffrant* l'information. On veut ainsi empêcher des entités non habilitées d'apprendre quoi que ce soit sur les données chiffrées. Nous nous contenterons de cette application ici, mais la cryptographie en a beaucoup d'autres, comme par exemple garantir l'intégrité des données via des signatures numériques ou encore montrer que l'on connaît de l'information sans la divulguer (ce qui sert par exemple pour ouvrir votre porte d'immeuble avec un RFID).

On sait, depuis le célèbre article de Kerckhoffs *La cryptographie militaire* de 1883 que, pour offrir une sécurité raisonnable, un système de chiffrement doit reposer sur le secret d'une quantité courte, la clef, et non sur le secret du procédé employé. Ce principe fondamental a été énoncé par Kerckhoffs : "Il faut que [le système] n'exige pas le secret [...] Et ici j'entends par secret, non la clef proprement dite, mais ce qui constitue la partie matérielle du système : tableaux, dictionnaires ou appareils mécaniques quelconques qui doivent en permettre l'application. "

Aujourd'hui, on peut distinguer deux grandes catégories d'algorithmes de chiffrement, ceux dit à *clef publique* (ou *asymétrique*) et ceux dit à *clef secrète* (ou *symétrique*). Les premiers, dont l'exemple le plus célèbre est certainement RSA (du nom de ses 3 concepteurs Rivest, Shamir et Adelman [RSA77]) permettent d'échanger de l'information chiffrée sans s'être mis d'accord au préalable sur un secret. Ces algorithmes sont asymétriques car ils utilisent en fait deux clefs, l'une publique et l'autre secrète. Tout le monde peut ainsi chiffrer un message avec la clef publique du destinataire, mais seul ce dernier pourra le déchiffrer avec sa clef secrète qu'il est le seul à connaître. C'est grâce à ces techniques à clef publique que la plupart des applications de la cryptographie autres que le simple chiffrement sont devenues possibles.

Mais, si ces algorithmes à clef publique offrent de nombreux avantages,

ils sont en général beaucoup plus lents que les algorithmes basés sur l'utilisation d'une clef secrète qui a été échangée au préalable entre les participants. Ces contraintes de performance imposent l'usage d'algorithmes de chiffrement à clef secrète, puisque les algorithmes à clef publique connus actuellement n'offrent pas un débit suffisant pour permettre le chiffrement ou le déchiffrement en ligne. En pratique dans les applications modernes, on utilise des algorithmes à clef publique pour se mettre d'accord sur une clef secrète qui sera ensuite utilisée dans un algorithme symétrique pour l'échange proprement dit d'information.

Dans cette thèse, nous nous intéresserons uniquement aux chiffrements symétriques et plus précisément aux *chiffrements à flot* dont nous verrons le principe dans la prochaine section. Mais avant cela, donnons quand même un aperçu des *chiffrements par blocs* qui constituent la plus grande famille de chiffrement symétrique.

Comme pour les codes en bloc, un chiffrement par blocs découpe l'information en blocs de n bits et traite chaque bloc indépendamment (ou presque). Contrairement aux codes correcteurs, il n'y a pas de redondance et utiliser un chiffrement par blocs revient à effectuer une permutation sur l'ensemble des mots de n bits qui va transformer un message de n bits en un mot chiffré de n bits. Cette permutation est paramétrée par une clef secrète, et un chiffrement par blocs idéal associe à chaque clef une permutation aléatoire.

Nous n'expliquerons pas comment un tel système peut être construit, mais il existe deux exemples très connus qui sont le DES et l'AES. Le DES (Data Encryption Standard) était l'ancien standard de chiffrement par blocs, il opère sur des blocs de 64 bits et utilise une clef secrète de 56 bits. La taille de la clef est maintenant trop petite et un nouveau standard a été adopté, il s'agit justement de l'AES (Advanced Encryption Standard). Ce dernier a été choisi en octobre 2000 parmi les 15 systèmes proposés en réponse à l'appel d'offre lancé par le NIST (National Institute of Standards and Technology). Cet algorithme, initialement appelé RIJNDAEL, a été conçu par deux cryptographes belges, V. Rijmen et J. Daemen. Il opère sur des blocs de 128 bits et est disponible pour trois tailles de clefs secrètes différentes : 128, 192 et 256 bits. Les spécifications de l'AES ainsi que diverses implémentations sont disponibles sur la page Web du NIST `http://csrc.nist.gov/encryption/aes/rijndael`.

Mentionnons quand même qu'une fois un chiffrement par blocs construit, il ne suffit pas de l'appliquer bêtement à chaque bloc pour obtenir un chiffrement sûr. Il y a en effet un grand nombre d'applications où l'on peut être amené à chiffrer des blocs identiques, et l'on ne veut certainement pas que leurs chiffrés le soient. Un exemple est le chiffrement d'une image avec des zones complètement uniformes qu'un tel mode de chiffrement ne cacherait pas. Il existe donc des *modes opératoires* qui étant donné une fonction de chiffrement par blocs, expliquent comment l'appliquer sur un message réel.

Un exemple qui nous sera utile, car il va en fait transformer un chiffrement par blocs en chiffrement à flot, est le mode compteur décrit sur la figure 1.1.

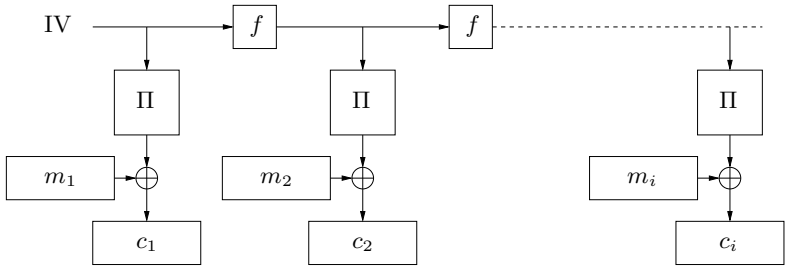

FIG. 1.1 – Mode compteur pour un chiffrement par blocs qui correspond ici à la permutation Π qui dépend de la clef. Le message à émettre est découpé en blocs de n bits, les m_i, et chaque bloc est chiffré en c_i à l'aide d'un XOR. L'entrée de la fonction de chiffrement est initialisée avec une valeur publique, l'IV (pour Initialisation Value) qui est mise à jour d'un bloc sur l'autre par une fonction f qui est souvent un simple compteur qui incrémente la valeur de 1.

1.3 Les chiffrements à flot

Les procédés de chiffrement à flot sont des techniques qui permettent d'assurer la confidentialité d'une communication dans des contextes où il est primordial de pouvoir chiffrer et déchiffrer très rapidement au moyen de ressources, notamment d'une capacité de stockage, très limitées. La plupart des systèmes embarqués entrent dans ce cadre, des communications téléphoniques aux communications satellites. Dans ce contexte, les algorithmes à clef publique connus sont inutilisables et même les algorithmes à clef secrète par blocs peuvent présenter des inconvénients.

En effet, la structure par blocs présente un inconvénient majeur pour certains systèmes embarqués. Nous rappelons que ces techniques consistent à découper le message à transmettre en blocs de taille fixe (généralement en blocs de 128 bits), puis à chiffrer chacun de ces blocs. On ne peut donc commencer à chiffrer un message que si l'on connaît la totalité d'un bloc. Il en est bien sûr de même pour le déchiffrement. Ceci occasionne naturellement un délai dans la transmission et nécessite également le stockage successif des blocs dans une mémoire tampon.

Au contraire, dans les procédés de chiffrement par flot l'unité de base du chiffrement et du déchiffrement n'est plus le bloc mais le bit. Chaque nouveau bit transmis peut être chiffré ou déchiffré indépendamment des autres, en particulier sans qu'il soit nécessaire d'attendre les bits suivants. Un autre avantage de ces techniques est que, contrairement aux algorithmes par blocs,

le processus de déchiffrement ne propage pas les erreurs de transmission. Supposons qu'une erreur survenue au cours de la communication ait affecté un bit du message chiffré. Dans le cas d'un chiffrement à la volée, cette erreur affecte uniquement le bit correspondant du texte clair, et ne le rend donc généralement pas complètement incompréhensible. Par contre, dans le cas d'un chiffrement par blocs, c'est tout le bloc contenant la position erronée qui devient incorrect après déchiffrement. Ainsi, une erreur sur un seul bit lors de la transmission affecte en réalité 128 bits du message clair. C'est pour cette raison que les chiffrements à flot sont également utilisés pour protéger la confidentialité dans les transmissions bruitées.

Les systèmes de chiffrement à flot reposent sur le célèbre chiffrement à usage unique, appelé aussi technique du masque jetable, et également connu sous son appellation anglo-saxonne "one-time-pad". Ce procédé, inventé par Vernam pour protéger les communications télégraphiques pendant la première guerre mondiale, consiste simplement à effectuer un XOR (ou exclusif) bit à bit entre le message clair et une suite de bits aléatoire de même longueur qui constitue la clef secrète du système. Rappelons que l'opération XOR entre deux bits, représentée par le symbole $\oplus$, est définie par :

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1 \quad \text{et} \quad 1 \oplus 1 = 0 \, .$$

Le chiffrement du message binaire 10100110 (représentation binaire de la lettre e) avec la clef secrète 01001011 se fait donc de la manière suivante :

	message clair	1	0	1	0	0	1	1	0
$\oplus$	clef secrète	0	1	0	0	1	0	1	1
=	message chiffré	1	1	1	0	1	1	0	1

Le déchiffrement est similaire, on retrouve le message d'origine à partir du message chiffré et de la clef par :

	message chiffré	1	1	1	0	1	1	0	1
$\oplus$	clef secrète	0	1	0	0	1	0	1	1
=	message clair	1	0	1	0	0	1	1	0

On peut démonter que le chiffrement à usage unique est incassable dans la mesure où la connaissance du message chiffré n'apporte aucune information sur le message clair. Mais cette propriété n'est garantie que si la clef secrète est bien une suite totalement aléatoire aussi longue que le message clair, et qu'elle n'est utilisée que pour transmettre un seul message. L'emploi de la même clef pour chiffrer plusieurs messages peut notamment avoir des conséquences désastreuses. C'est ce qui est arrivé durant la seconde guerre mondiale où des opérateurs du KGB ont eu l'imprudence d'utiliser de nombreuses fois les mêmes clefs à usage unique. Cette observation a permis à la National Security Agency (NSA) de déchiffrer des télégrammes soviétiques de première importance. L'opération secrète était connue sous le nom de

projet VENONA et son existence n'a été révélée par la NSA qu'à la fin des années 90.

L'utilisation d'une clef secrète aléatoire à usage unique et de même longueur que le message à transmettre est malheureusement nécessaire pour obtenir un chiffrement inconditionnellement sûr, c'est-à-dire pour lequel on peut prouver qu'il est impossible de retrouver le message clair à partir du chiffré sans connaître la clef secrète. Cette condition rend généralement tous les chiffrements parfaits, comme le chiffrement à usage unique, inutilisables puisqu'il est rarement envisageable de s'échanger préalablement un secret aussi long que la totalité du message à transmettre. L'usage de ces systèmes est donc réservé aux communications exigeant un niveau de sécurité extrêmement élevé comme les communications diplomatiques (le canal sécurisé utilisé pour transmettre les suites aléatoires secrètes n'est autre que la valise diplomatique). C'était par exemple le cas du célèbre "téléphone rouge" entre Washington et Moscou pendant la guerre froide.

1.4 Générateur pseudo-aléatoire

Dans les applications usuelles, le chiffrement à flot utilisé est une version affaiblie du chiffrement à usage unique. La suite que l'on additionne par XOR au message clair n'est plus une suite engendrée de manière totalement aléatoire, mais une suite *pseudo-aléatoire*. Cette suite est engendrée de manière déterministe par un dispositif appelé générateur pseudo-aléatoire qui est souvent de la forme donnée sur la figure 1.2. On a vu qu'il est possible de transformer un chiffrement par blocs en chiffrement à flot en utilisant comme mode opératoire le mode compteur. Mais des systèmes dédiés sont plus rapides et demandent moins de ressources physiques.

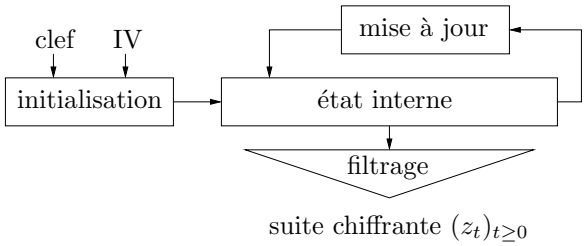

FIG. 1.2 – Forme classique d'un générateur pseudo-aléatoire avec une fonction de mise à jour de l'état interne et une fonction de filtrage pour produire la suite chiffrante $(z_t)_{t\geq 0}$.

Un générateur pseudo-aléatoire possède ainsi une mémoire interne qui contient l'état dans lequel il se trouve. À chaque top d'horloge, cet état est filtré par une fonction dite de *filtrage* pour produire un ou plusieurs bits de

suite chiffrante et il est mis à jour par une autre fonction. Dans cette thèse, nous nous intéresserons particulièrement au cas où la fonction de mise à jour est linéaire ce qui impose que celle de filtrage soit non linéaire si l'on veut que le système offre un minimum de sécurité.

En général, l'état interne qui est de taille relativement modeste (de l'ordre de 128 ou 256 bits) est initialisé par une valeur secrète (la clef du système) et une valeur publique (l'IV ou Initialisation Value) qui ont souvent comme taille la moitié de celle de l'état interne. L'IV est en général une valeur tirée au hasard au début d'un échange d'information et permet d'utiliser la même clef pour un grand nombre de communications. Il serait en effet très mauvais d'encoder deux messages avec la même suite chiffrante comme on l'a déjà mentionné dans la section précédente. Un simple XOR des deux messages nous donnerait alors le XOR des deux messages clairs et si les messages sont en français par exemple, le biais statistique est largement suffisant pour permettre de reconstituer les deux. On peut également noter que, l'IV étant publique, la procédure d'initialisation de l'état interne est souvent longue de manière à ce qu'un attaquant ait peu d'information sur la valeur initiale de l'état interne.

Ce type de procédé ne garantit plus une sécurité théorique : la connaissance du chiffré apportera toujours théoriquement une information sur le message clair. Par contre, cette information peut être très difficile à obtenir (déchiffrer le message peut nécessiter par exemple plusieurs milliers d'années sur un grand nombre d'ordinateurs en parallèle). On se place donc ici sur le plan de la sécurité calculatoire puisque l'on évalue la sécurité du système par le temps de calcul et la quantité de ressources nécessaires pour le cryptanalyser.

La sécurité d'un algorithme de chiffrement par flot repose entièrement sur les qualités cryptographiques du générateur pseudo-aléatoire employé. Celui-ci doit notamment résister à une attaque à clair connu, dans laquelle un attaquant dispose de divers messages chiffrés ainsi que des textes clairs correspondants — qu'il a deviné par exemple en utilisant le fait que les messages envoyés suivent un format particulier. Un simple XOR entre ces couples clairs-chiffrés fournit alors les valeurs de certains bits produits par le générateur. Dans ce contexte, il doit donc être impossible en pratique quand on connaît certains bits produits par le générateur de prédire la valeur des bits suivants.

Contrairement aux chiffrements par blocs et à l'AES, il n'existe pas de standard universellement reconnu pour les chiffrements à flot. Chaque application utilise ainsi son propre chiffrement comme l'algorithme A5/1 pour les communications GSM Européenne ou encore l'algorithme E0 utilisé dans le Bluetooth.

Récemment, en novembre 2004, le projet eSTREAM[1] a été lancé par

[1] `http://www.ecrypt.eu.org/stream/`

le réseau d'excellence Européen ECRYPT pour essayer de construire un chiffrement à flot qui pourrait être adopté comme un futur standard. Le projet arrivera à son terme en mai 2008. Il fait suite à NESSIE, un autre projet au terme duquel aucune proposition de chiffrement à flot n'a été jugée satisfaisante.

Ces projets stimulent énormément la recherche dans ce domaine et l'enchaînement de construction d'algorithmes et de leurs attaques fait de la cryptographie une science passionnante.

Chapitre 2

Récurrence linéaire sur les corps finis

Nous commençons dans ce chapitre par rappeler les principales propriétés des corps finis qui sont des objets mathématiques essentiels aussi bien pour la théorie des codes correcteurs que pour la cryptographie. Le corps que nous utiliserons le plus est sans conteste le corps fini à 2 éléments que nous noterons $\mathbf{F}_2$, mais il arrivera que l'on ait besoin de travailler sur des corps plus gros et tous les résultats de ce chapitre s'appliquent à n'importe quel corps fini.

Nous nous intéresserons ensuite aux LFSR qui sont des moyens de calculer des récurrences linéaires sur de tels corps, récurrences qui interviendront à de nombreuses reprises dans cette thèse. La plupart des démonstrations sont omises pour faciliter la lecture. On pourra les trouver dans le livre de McEliece [McE87] dont nous nous sommes largement inspiré pour écrire cette section. Pour en savoir plus sur les nombreuses propriétés des corps finis, une bonne référence est le livre de Lidl et Niederreiter [LN83].

2.1 Les corps finis

Un corps fini est un ensemble fini d'éléments que l'on peut additionner, soustraire, multiplier et diviser. Nous souhaitons bien sûr que ces opérations se comportent de manière "usuelle" ce qui va imposer une structure très forte à ces objets. Plus formellement :

Définition 2.1 (Corps fini)**.** Un corps fini est un ensemble $\mathbf{F}$ fini muni de deux opérations binaires "+" et "."(ou rien) qui vérifient :

- $\mathbf{F}$ est un groupe commutatif avec la loi "+" d'élément neutre 0.
- Les éléments non nuls de $\mathbf{F}$ forment un groupe avec la loi "$\cdot$".
- On a la propriété de distributivité $a(b+c) = ab + ac$.

On parle de corps commutatif lorsque le groupe multiplicatif du corps l'est, c'est en fait toujours le cas pour un corps fini d'après le théorème de Wedderburn. Le corps fini le plus petit est le corps fini à deux éléments $\{0, 1\}$ que l'on notera $\mathbf{F}_2$. L'addition correspond au "ou exclusif" (XOR) et la multiplication au "et" (AND). Ce corps fait partie des corps finis les plus simples qui sont définis pour p premier par

$$\mathbf{F}_p = \{0, 1, \ldots, p-1\}, \quad \text{avec une arithmétique modulo } p\ . \tag{2.1}$$

Il n'est pas immédiat que $\mathbf{F}_p$ soit un corps, mais on peut le montrer. Tous les autres corps finis vont en fait être des extensions algébriques de ces corps dits *premiers*. Ils peuvent être définis à l'aide de la notion importante de polynôme *irréductible* :

Définition 2.2 (Polynôme irréductible)**.** Soit $p(X)$ un polynôme de degré n à coefficients dans un corps $\mathbf{F}$. $p(X)$ est dit irréductible sur $\mathbf{F}$ s'il ne peut pas être écrit sous la forme d'un produit de deux polynômes dans $\mathbf{F}$ de degré strictement positif.

Théorème 2.3 (Construction de $\mathbf{F}_q$)**.** Pour tout p premier et n entier, il existe un unique corps fini à $q = p^n$ éléments. On peut le définir à un isomorphisme près par

$$\mathbf{F}_q \stackrel{\text{def}}{=} \mathbf{F}_p[X]/p(X)$$

pour $p(X)$ polynôme irréductible de degré n à coefficients dans $\mathbf{F}_p$.

On peut montrer que ce sont les seuls corps finis qui existent. Ainsi, le cardinal d'un corps fini est nécessairement de la forme p^n et pour un cardinal donné il y a unicité du corps à un isomorphisme près. Le nombre premier p qui divise le cardinal d'un corps fini est appelé la *caractéristique* du corps.

Il est important de noter que le théorème 2.3 est un théorème constructif. Ainsi $\mathbf{F}_q$ peut être vu comme un espace vectoriel sur $\mathbf{F}_p$ et tous ses éléments peuvent s'écrire dans la base dite canonique

$$(1, X, X^2, \ldots, X^{n-1}) \tag{2.2}$$

Calculer dans le corps se fait alors de la même manière que dans l'anneau des polynômes sur $\mathbf{F}_p$ avec en plus une réduction modulo $p(X)$. La structure des corps finis donne des règles de calcul assez particulières. Rappelons sans démonstration quelques propriétés intéressantes qui nous seront utiles par la suite :

Proposition 2.4 (Calcul sur $\mathbf{F}_{p^n}$)**.** Soit α et β deux éléments de $\mathbf{F}_{p^n}$, on a toujours

- $p\ \alpha = 0$
- $(\alpha + \beta)^p = \alpha^p + \beta^p$
- $\alpha^{p^n} = \alpha$

Nous aurons également besoin de travailler dans les sous-corps d'un corps fini. Le résultat essentiel est que $\mathbf{F}_{p^n}$ est un sous-corps de $\mathbf{F}_{p^m}$ si et seulement si n divise m. Ce dernier corps peut être construit par extension de $\mathbf{F}_{p^n}$ par un polynôme irréductible de degré m/n sur $\mathbf{F}_{p^n}$. De plus les éléments de ce sous-corps sont caractérisés par la troisième propriété de la proposition 2.4, ce sont les seuls à vérifier $\alpha^{p^n} = \alpha$.

Un polynôme irréductible $p(X)$ de degré n et à coefficients dans $\mathbf{F}_q$ possède toujours n racines distinctes dans $\mathbf{F}_{q^n}$. Ces racines jouent en fait toutes le même rôle et sont dites *conjuguées*. En effet les fonctions

$$\phi_i : x \mapsto x^{q^i}$$

définies sur $\mathbf{F}_{q^n}$ sont des automorphismes de corps qui laissent $\mathbf{F}_q$ invariant. Pour une racine α de $p(X)$, il est alors facile de vérifier que

$$p(\phi_i(\alpha)) = \phi_i(p(\alpha)) = 0 \ .$$

Dans le cas où $p(X)$ est irréductible, toutes ses racines se dérivent de α par application de ces automorphismes et sont $\alpha, \alpha^q, \ldots, \alpha^{q^n}$. En fait tous les polynômes irréductibles sur $\mathbf{F}_q$ sont à un facteur près de la forme suivante :

Proposition 2.5 (Polynôme minimal)**.** Soit α un élément d'une extension finie de $\mathbf{F}_q$, Soit n le plus petit entier tel que $\alpha^{q^n} = \alpha$ alors α est dans $\mathbf{F}_{q^n}$ et

$$\prod_{i=1}^{n} \left(X - \alpha^{q^i} \right)$$

est un polynôme irréductible de degré n à coefficients dans $\mathbf{F}_q$. On parle de polynôme minimal de α sur $\mathbf{F}_q$.

Enfin, nous aurons besoin de la notion de fonction trace.

Définition 2.6 (Fonction trace)**.** On appelle fonction trace de $\mathbf{F}_{q^n}$ dans $\mathbf{F}_q$ la fonction qui à α associe la somme de ses éléments conjugués. De manière formelle

$$\mathrm{Tr}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{n-1}} \ . \tag{2.3}$$

Il s'agit bien d'une fonction à valeurs dans $\mathbf{F}_q$ car toutes les images satisfont l'équation de corps $X^q = X$. De plus, on peut montrer que cette fonction est $\mathbf{F}_q$-linéaire. En fait toutes les fonctions $\mathbf{F}_q$-linéaires de $\mathbf{F}_{q^n}$ dans $\mathbf{F}_q$ sont de la forme $\alpha \mapsto \mathrm{Tr}(\theta\alpha)$ pour un θ dans $\mathbf{F}_{q^n}$. De plus, pour deux θ différents, ces fonctions sont différentes.

2.2 Les LFSR

L'un des éléments les plus courants dans un chiffrement à flot est sans conteste le *registre à décalage avec rétroaction linéaire*. Nous utiliserons dans

toute cette thèse l'abréviation LFSR de l'anglais Linear Feedback Shift Register. Le comportement d'un tel objet est intimement lié à la structure multiplicative des corps finis. Les résultats que nous allons voir sont en fait les premiers pas vers des circuits efficaces de multiplication dans les corps finis dérivés des travaux de Elwyn Berlekamp [Ber82]. Pour plus de détails, on peut aussi consulter la thèse de Cédric Lauradoux [Lau07].

Un LFSR produit une séquence $(s_t)_{t\geq 0}$ d'éléments d'un corps fini $\mathbf{F}_q$ qui vérifie une récurrence linéaire

$$s_{t+l} = \sum_{i=0}^{l-1} c_i s_{t+i} \quad \forall t \geq l \tag{2.4}$$

où l est la longueur du LFSR et les c_i sont des coefficients dans $\mathbf{F}_q$. La forme dite de Fibonacci d'un tel générateur est représentée sur la figure 2.1. Elle est constituée de l registres qui contiennent chacun un élément de $\mathbf{F}_q$. L'ensemble des valeurs de ces registres constitue l'*état* du LFSR. On peut noter que, l'ensemble des états possibles étant fini, la suite produite est périodique.

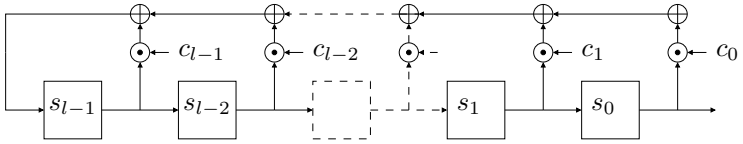

FIG. 2.1 – LFSR de longueur l en configuration de Fibonacci.

Initialement, le LFSR est rempli avec les l premiers éléments de la suite $(s_0, \ldots, s_{l-1})$ comme indiqué sur la figure, c'est ce que l'on appelle l'*état initial*. À chaque pas (ou top d'horloge), le contenu de chaque registre est décalé vers la droite dans le registre suivant et le registre tout à gauche est rempli avec un nouvel élément de la suite $(s_t)_{t\geq 0}$ grâce à la formule de récurrence (2.4). Par usage répété de cette formule de récurrence, il est possible d'exprimer s_t comme fonction linéaire des bits de l'état initial, nous noterons $s_t = S_t(s_0, \ldots, s_{l-1})$. En fait, S_t sera vu comme un polynôme de degré 1 en l variables, X_1 jusqu'à X_l.

Il est commode de représenter les coefficients de la récurrence sous la forme d'un polynôme $g(X)$ à coefficients dans $\mathbf{F}_q$ connu sous le nom de *polynôme générateur* du LFSR :

$$g(X) = X^l - \sum_{i=0}^{l-1} c_i X^i \ . \tag{2.5}$$

Nous noterons toujours $g(X)$ le polynôme générateur d'un LFSR de longueur l et désignerons par α une racine de $g(X)$ dans $\mathbf{F}_{q^l}$. Attention, dans la littérature un LFSR sous la forme de Fibonacci est souvent décrit par son

polynôme de rétroaction qui est le *polynôme réciproque* de $g(X)$ (c'est-à-dire $1 - \sum_i c_{l-i} X^i$). Une propriété particulièrement importante pour les polynômes qui définissent une séquence linéaire est la *primitivité* :

Définition 2.7 (Polynôme primitif)**.** Un polynôme irréductible $g(X)$ de degré n et à coefficients dans $\mathbf{F}_q$ est dit primitif si une racine α de $g(X)$ engendre le groupe multiplicatif $\mathbf{F}_{q^n}^*$ (l'étoile signifie $\mathbf{F}_{q^n}$ privé de 0).

On peut montrer que le groupe multiplicatif d'un corps fini est cyclique, il existe donc toujours des polynômes primitifs. Quand le polynôme générateur d'un LFSR est primitif, la suite engendrée possède de nombreuses bonnes propriétés statistiques intéressantes pour un chiffrement à flot. Le LFSR vérifie en particulier le théorème 2.8, et l'on parle alors de LFSR de période maximale.

Théorème 2.8 (m-séquences)**.** Soit un LFSR sur $\mathbf{F}_q$ de longueur l et de polynôme générateur $g(X)$ primitif. Alors, si l'état initial est différent de 0, le LFSR décrit tous les états non nuls possibles. La séquence produite est ainsi de période maximale, c'est-à-dire $q^l - 1$. Un telle séquence est connue sous le nom de m-séquence (abbréviation de l'anglais maximum length sequence).

Ce résultat découle du fait que l'on peut associer à l'état interne d'un LFSR un élément de $\mathbf{F}_{q^l}$ de telle manière qu'avancer le LFSR correspond à une simple multiplication par α (une racine de $g(X)$) dans le corps fini. Du moment que l'état initial n'est pas nul, l'état interne parcourt alors tout le groupe $\mathbf{F}_{q^l}^*$ puisque α est un générateur par primitivité de $g(X)$.

Avant de voir une preuve, intéressons-nous à ce que l'on appelle la forme de Galois d'un LFSR pour laquelle l'association est plus simple.

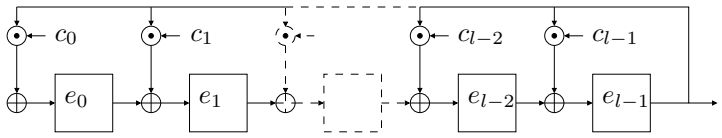

FIG. 2.2 – LFSR de longueur l en configuration de Galois.

Proposition 2.9 (LFSR de Galois)**.** Pour un LFSR sous la forme de Galois décrit dans la figure 2.2, avancer le LFSR correspond à multiplier par α l'état interne vu comme élément de F_{q^l} dans la base

$$(1, \alpha, \alpha^2, \ldots, \alpha^{l-1}) \ .$$

Ainsi, à un état interne $(e_0, \ldots, e_{l-1})$ on associe l'élément $e = \sum_i e_i \alpha^i$. Multiplier par α revient bien à décaler tous les coefficients de un vers la

droite sauf le terme en α^l, qui lui est réduit grâce au polynôme générateur $g(X)$. En effet, α étant une racine de $g(X)$ on a

$$\alpha^l = \sum_{i=0}^{l-1} c_i \alpha^i \ . \tag{2.6}$$

Remarquons que la suite produite par un LFSR de Galois vérifie parfaitement la formule de récurrence (2.4). En effet, l'état interne la vérifie car au temps t il est égal à $e\alpha^t$ et α est une racine de $g(X)$. Chaque coefficient de cet état la vérifie donc aussi et a fortiori c'est également le cas de la suite produite par le LFSR.

Les deux constructions sont en fait équivalentes et produisent les mêmes suites du moment que l'élément initial est bien choisi. Pour s'en convaincre, il faut voir l'état interne d'un LFSR de Fibonacci comme des coefficients d'un élément de $\mathbf{F}_{q^l}$ dans ce que l'on appelle la base duale de la base canonique $(1, \alpha, \ldots, \alpha^{l-1})$. La base duale est définie comme l'unique l-uplet $(\beta_0, \ldots, \beta_{l-1})$ de $\mathbf{F}_{q^l}$ qui vérifie

$$\mathrm{Tr}(\beta_i \alpha^j) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases} \ . \tag{2.7}$$

Nous ne démontrerons ni l'existence, ni l'unicité de cette base, mais le lecteur peut consulter [LN83]. On a maintenant la proposition suivante :

Proposition 2.10 (LFSR de Fibonacci)**.** Pour un LFSR sous la forme de Fibonacci décrit dans la figure 2.1, avancer le LFSR correspond à multiplier par α l'état interne vu comme élément de F_{q^l} dans la base

$$(\beta_0, \beta_1, \ldots, \beta_{l-1}) \ .$$

Démonstration. D'après la propriété (2.7) de la base duale, on peut voir que la coordonnée s_i d'un élément $s = \sum_i s_i \beta_i$ du corps dans cette base est donnée par $\mathrm{Tr}(s\alpha^i)$. Calculons maintenant les coordonnées s'_i de $s\alpha$ dans la base duale. Pour i variant de 0 à $l-2$, c'est facile :

$$s'_i = \mathrm{Tr}(s\alpha\alpha^i) = s_{i+1} \ .$$

Pour $i = l-1$, on a $s'_{l-1} = \mathrm{Tr}(s\alpha^l)$ et en utilisant (2.6), on obtient

$$s'_{l-1} = \sum_{i=0}^{l-1} c_i s_i \ .$$

Et on retrouve donc exactement la récurrence du LFSR de Fibonacci. Attention, nous avons inversé le sens de lecture de l'état interne entre les configurations de Galois et de Fibonacci. □

2.3 Quelques remarques

Les LFSR ou de manière équivalente les récurrences linéaires sur les corps finis vérifient plusieurs propriétés qui expliquent leur ubiquité dans de nombreux domaines de l'informatique. Nous mettons l'accent ici sur quelques unes d'entre elles qui nous seront utiles par la suite.

Forme générale d'une récurrence linéaire sur un corps fini

Intéressons-nous aux séquences qui vérifient une récurrence linéaire de polynôme générateur $g(X) = X^l - \sum_{i=0}^{l-1} c_i X^i$ de degré l et à coefficients dans $\mathbf{F}_q$, c'est-à-dire les séquences $(s_t)_{t\geq 0}$ qui vérifient

$$s_{t+l} = \sum_{i=0}^{l-1} c_i s_{t+i} \qquad \forall t \geq 0 \ . \tag{2.8}$$

De manière générale, on a le théorème suivant :

Théorème 2.11 (Récurrence linéaire sur $\mathbf{F}_{q^n}$)**.** Si $\alpha_1, \alpha_2, \ldots, \alpha_r$ sont les r racines distinctes de $g(X)$ dans $\mathbf{F}_{q^n}$ alors pour tout choix de constantes $\lambda_1, \ldots, \lambda_r$ de $\mathbf{F}_{q^n}$ la séquence d'éléments de $\mathbf{F}_{q^n}$

$$s_t \stackrel{\text{def}}{=} \lambda_1 \alpha_1^t + \lambda_2 \alpha_2^t + \cdots + \lambda_r \alpha_r^t \qquad t \geq 0$$

vérifie la récurrence linéaire (2.8).

Démonstration. Chaque α_i^t satisfait la relation de récurrence (2.8) car α_i est une racine de $g(X)$. La somme des $\lambda_i \alpha_i^t$ également car l'ensemble des séquences satisfaisant une récurrence linéaire forme un espace vectoriel. □

Dans le cas où $g(X)$ n'a pas de racine multiple, par un argument dimensionnel on peut montrer que toutes les séquences qui vérifient (2.8) sont de la forme donnée dans le théorème 2.11.

Ce théorème nous sera utile au chapitre 7 pour calculer la complexité linéaire d'une suite chiffrante produite par un LFSR filtré. Lorsque $g(X)$ est irréductible la situation est plus simple car ses racines sont toutes distinctes et sont conjuguées. On obtient alors le résultat suivant pour les séquences à valeur dans $\mathbf{F}_q$:

Théorème 2.12. (Séquences à valeur dans $\mathbf{F}_q$) Si $g(X)$ est irréductible, alors il existe un unique θ dans $\mathbf{F}_{q^n}$ tel que toutes les séquences à valeurs dans $\mathbf{F}_q$ qui vérifient (2.8) soient de la forme :

$$s_t = \mathrm{Tr}(\theta \alpha^t) \ .$$

Démonstration. En utilisant le LFSR associé de la section précédente on a vu qu'une telle séquence de $\mathbf{F}_q$ s'obtient comme une fonction linéaire de α^t. Le théorème découle alors directement des propriétés de la fonction trace vues dans la section précédente. □

Une séquence sur $\mathbf{F}_q$ qui vérifie une récurrence linéaire de polynôme générateur $g(X)$ non irréductible peut se mettre sous la forme d'une somme de séquences qui vérifient des récurrences linéaires de polynômes générateurs donnés par les facteurs irréductibles de $g(X)$. La situation est un peu plus compliquée si $g(X)$ admet des racines multiples, voir [McE87].

Equation de parité et polynôme

Il arrivera que l'on utilise le terme d'*équation de parité* pour désigner une relation de récurrence vérifiée par une séquence sur un corps fini. Pour un certains nombre de raisons, il est commode d'associer à une telle équation de parité un polynôme :

Définition 2.13 (Équations de parité associées à un polynôme)**.** À un polynôme $p(X) = \sum_{i=1}^{n} p_i X^i$ de degré n à coefficients dans $\mathbf{F}_2$ on associe une équation de parité sur une séquence $(s_t)_{t\geq 0}$ qui peut s'appliquer pour n'importe quel décalage dans le temps par :

$$\sum_i p_i s_{t+i} = 0 \quad \text{pour un décalage } t \geq 0 \ .$$

C'est exactement ce qui se passe pour le polynôme générateur $g(X)$ d'une récurrence linéaire par exemple. Avec cette association, on a aussi le résultat essentiel suivant sur lequel on reviendra au chapitre 5 :

Proposition 2.14 (Équations de parité pour un LFSR)**.** Soit une séquence $(s_t)_{t\geq 0}$ non nulle produite par un LFSR de polynôme générateur $g(X)$ primitif, alors l'équation de parité associée à un polynôme $p(X)$ à coefficients dans $\mathbf{F}_2$ est vérifiée pour tout décalage du temps si et seulement si $p(X)$ est égal à 0 modulo $g(X)$. C'est-à-dire si et seulement si $p(X)$ est un multiple de $g(X)$.

Démonstration. Si $p(X)$ est un multiple de $g(X)$, alors $p(X)$ peut s'écrire comme somme de $X^i g(X)$, il est alors facile de vérifier que l'équation de parité est vraie car l'équation associée à $g(X)$ l'est pour tout les décalages dans le temps.

Pour la réciproque, raisonnons par l'absurde. Si $p(X)$ n'est pas un multiple, alors le reste de la division par $g(X)$ est un polynôme $q(X)$ non nul et de degré au plus $l-1$. Par hypothèse, l'équation de parité associée à ce polynôme est vraie pour tous les décalages dans le temps. Ce n'est pas possible car sa valeur ne dépend que de l'état interne d'un LFSR qui engendre

la suite (vu que le degré est d'au plus l) et tous les états internes non nuls sont atteignables par primitivité de $g(X)$. □

On peut remarquer que le polynôme multivarié $S_t(X_1, \ldots, X_l)$ correspond à $X^t \mod g(X)$ si le coefficient devant X_i correspond à celui devant X^{i-1}.

Faire avancer un LFSR

Faire marcher un LFSR à l'envers, c'est-à-dire diviser par α se fait aussi de manière très efficace. En fait, même si cela est rarement mentionné, il est facile d'avancer un LFSR d'un grand nombre de pas d'un coup. Avancer de i pas correspond en effet à une multiplication par α^i dans $\mathbf{F}_{q^l}$. Le calcul de α^i peut se faire efficacement grâce à un algorithme d'exponentiation rapide et la multiplication dans le corps se fait en temps constant.

Énumération des éléments de $\mathbf{F}_2^l$

Il est intéressant de noter que les LFSR sont utilisés dans de nombreux autres domaines de l'informatique que la cryptographie. Par exemple, ils sont très utiles lorsque l'on veut décrire l'ensemble des états que peuvent prendre l bits. On aurait pu faire cela avec un simple compteur qui commence à 0 et qui ajoute 1 à chaque fois, mais en hardware l'addition est beaucoup plus coûteuse que faire avancer un LFSR d'un pas. Mentionnons également les codes de Gray, qui permettent de compter en passant d'une valeur à la suivante en ne changeant qu'un seul bit.

Propriétés statistiques des m-séquences

La suite produite par un LFSR primitif possède également de très bonnes propriétés statistiques ce qui fait de ce dernier un bon générateur pseudo-aléatoire. Non seulement tous les sous-mots de l bits non nuls apparaissent une et une seule fois, mais la séquence vérifie de nombreuses autres propriétés. Par exemple, les sous-séquences d'exactement n zéros consécutifs pour $n < l - 1$ apparaissent 2^{l-n-2} fois et la sous-séquence de $l - 1$ zéros apparaît une fois. L'ordre d'apparition suit quant à lui une loi assez compliquée. Il est facile d'avancer ou de reculer le LFSR d'un nombre donné de pas, mais très difficile de savoir de combien il faut l'avancer pour obtenir un état donné. C'est le problème du logarithme discret que l'on étudiera plus en détail au chapitre 5.

Utilisation d'un LFSR en cryptographie

Malgré ces bonnes propriétés, un LFSR ne peut pas être utilisé tel quel dans un chiffrement à flot car il est facile de retrouver l'état initial du mo-

ment que l'on connaît l bits de la séquence produite. Chaque bit connu nous donne en effet une équation linéaire sur l'état initial. Du moment que l'on en a un peu plus de l, on obtient un système linéaire inversible avec grande probabilité et dont la solution nous donne l'état initial. Nous verrons même au chapitre 7 que si l'on dispose de $2l$ bits consécutifs, on peut retrouver le polynôme générateur grâce à l'algorithme de Berlekamp-Massey. Une des façons de résoudre ce problème est justement de filtrer la sortie par une fonction booléenne comme nous le verrons au chapitre 4.

Chapitre 3

Les fonctions booléennes

On s'intéresse ici à ce que l'on appelle les fonctions booléennes qui sont avec les LFSR un autre élément-clef des chiffrements à flot. Elles sont aussi utilisées sous forme vectorielle dans les chiffrements par blocs et se retrouvent en fait au cœur de pratiquement tous les systèmes de chiffrement symétriques. Nous verrons ultérieurement au chapitre 9 que de nombreux liens existent avec les codes de Reed-Muller et la théorie des codes correcteurs.

3.1 Un mot sur les éléments de $\mathbf{F}_2^m$

Avant de rentrer dans le vif du sujet, nous avons besoin de définir plusieurs notations autour des vecteurs de m bits que l'on va constamment manipuler tout au long de cette thèse. Un tel vecteur sera toujours noté en gras comme $\mathbf{x}$, $\mathbf{u}$ ou encore $\mathbf{y}$. On aura de temps en temps besoin de manipuler les bits d'un tel vecteur, ils seront toujours notés par la même lettre que le vecteur (mais pas en gras) et indexés de 1 à m, par exemple $\mathbf{x} = (x_1, \ldots, x_m)$.

Nous utiliserons deux notions importantes sur les vecteurs de bits qui sont le *support* et le *poids de Hamming*. Ils sont définis pour des vecteurs de bits de longueur quelconque par :

Définition 3.1 (Support)**.** Soit un vecteur $\mathbf{b} = (b_1, \ldots, b_n)$ de $\mathbf{F}_2^n$. Le support de $\mathbf{b}$ que nous noterons $\sup(\mathbf{b})$ est l'ensemble des positions des bits non nuls de $\mathbf{b}$. De manière plus formelle on a

$$\sup(\mathbf{b}) = \{i, b_i = 1\} \ .$$

Définition 3.2 (Poids de Hamming)**.** Soit un vecteur $\mathbf{b} = (b_1, \ldots, b_n)$ de $\mathbf{F}_2^n$. Le poids de Hamming de $\mathbf{b}$ est égal à son nombre de composantes non nulles, on le notera $|\mathbf{b}|$. C'est-à-dire

$$|\mathbf{b}| \stackrel{\text{def}}{=} |\sup(\mathbf{b})| \ .$$

Nous utilisons aussi la notation | | pour le cardinal d'un ensemble.

Nous aurons également besoin d'ordonner les éléments de $\mathbf{F}_2^m$. L'ordre que nous utiliserons est l'ordre lexicographique inverse et nous le désignerons dans toute cette thèse par *ordre usuel*. En particulier les notations $<, \leq, >, \geq$ entre éléments de $\mathbf{F}_2^m$ se réfèreront toujours à cet ordre. Il est défini par :

Définition 3.3 (Ordre lexicographique inverse)**.** L'ordre lexicographique inverse sur les éléments de $\mathbf{F}_2^m$ est tel que $\mathbf{x} < \mathbf{y}$ si et seulement s'il existe $i_0 \in [1, m]$ tel que

$$\forall i \in]i_0, m] \quad x_i = y_i \quad \text{et} \quad x_{i_0} < y_{i_0} \ .$$

Cet ordre présente de nombreuses propriétés qui vont le rendre très utile. En particulier, si l'on voit un m-uplet de $\mathbf{F}_2$ comme la représentation binaire d'un entier, l'ordre lexicographique inverse est le même que l'ordre usuel sur les entiers. Il faut pour cela choisir la convention suivante :

Définition 3.4 (Entier associé)**.** On associe à un élément $\mathbf{x}$ de $\mathbf{F}_2^m$ un entier de 0 à $2^m - 1$ par

$$\sum_{i=1}^{m} x_i 2^{i-1} \ .$$

Les bits de poids faible sont donc ceux d'indice le plus petit. Quand on veut désigner un élément de $\mathbf{F}_2^m$ par un nombre, on notera ce nombre en gras comme $\mathbf{0}$ ou $\mathbf{2^m - 1}$.

Exemple Pour m égal à 5, le monôme $X_1X_2X_4$ est par exemple désigné par le 5-uplet $(1, 1, 0, 1, 0)$ et correspond au nombre 11.

Remarque L'ordre que nous venons de définir n'est pas le seul intéressant. On verra par exemple dans la proposition 9.11 que l'ordre induit par un LFSR peut aussi être très utile.

3.2 Définitions et représentations

Nous appellerons fonction booléenne à m variables toute fonction de $\mathbf{F}_2^m$ dans $\mathbf{F}_2$. Il y en a exactement 2^{2^m} et elles forment un espace vectoriel de dimension 2^m. La manière la plus simple d'en représenter une est simplement de lister l'ensemble de ses valeurs sur tous les éléments de $\mathbf{F}_2^m$:

Définition 3.5 (Vecteur de valeurs)**.** Soit f une fonction booléenne à m variables. On appelle table de vérité ou vecteur des valeurs de f le vecteur de 2^m bits suivant

$$f(\mathbf{0}), f(\mathbf{1}), \ldots, f(\mathbf{2^m - 1}) \ .$$

Notez que l'on a choisi l'ordre usuel défini dans la section précédente pour lister les éléments de $\mathbf{F}_2^m$.

Une des premières propriétés importantes d'une fonction booléenne est son poids de Hamming. Nous étendons la définition 3.2 à une fonction f en considérant son vecteur de valeurs. On notera cette quantité $|f|$. En cryptographie, on utilise presque tout le temps des fonctions dites *équilibrées* pour s'assurer qu'il n'y ait pas de biais statistique dans le chiffrement.

Définition 3.6 (Fonction équilibrée)**.** Une fonction booléenne à m variables est dite équilibrée si elle prend autant de fois la valeur 0 que la valeur 1 sur l'ensemble de ses entrées. De manière équivalente, son poids de Hamming est alors 2^{m-1}.

Il existe une deuxième représentation plus algébrique d'une fonction booléenne sous la forme d'un polynôme en m variables que nous noterons $F(X_1, \ldots, X_m)$ ou simplement F. Il s'agit de l'unique polynôme à m variables avec un degré en chaque variable d'au plus 1 qui s'évalue en tous les points $\mathbf{x} = (x_1, \ldots, x_m)$ de $\mathbf{F}_2^m$ comme f, voir la définition 3.7. L'existence et l'unicité de cette représentation polynomiale est en fait quelque chose de fondamental qui existe aussi pour des fonctions qui prennent leurs arguments dans un corps quelconque. Nous ne détaillerons pas cela ici, par contre dans le cas binaire nous verrons plus loin que ce résultat découle directement du théorème 3.11.

Définition 3.7 (Forme algébrique normale)**.** La forme algébrique normale ou ANF (pour Algebraic Normal Form) d'une fonction booléenne f à m variables est donnée par l'unique vecteur de 2^m bits $(f_{\mathbf{u}}, \mathbf{u} \in \mathbf{F}_2^m)$ tel que

$$f(\mathbf{x}) = \sum_{\mathbf{u} \in \mathbf{F}_2^m} f_{\mathbf{u}} \mathbf{x}^{\mathbf{u}} \quad \text{avec} \quad \mathbf{x}^{\mathbf{u}} \stackrel{\text{def}}{=} x_1^{u_1} \ldots x_m^{u_m} \ .$$

On choisit encore une fois l'ordre usuel pour les 2^m coefficients $f_{\mathbf{u}}$ de ce polynôme. Par abus de notation, on parlera de fonction monôme $\mathbf{u}$ pour la fonction $\mathbf{x}^{\mathbf{u}}$. L'ensemble de ces fonctions forme une base de l'espace vectoriel des fonctions booléennes. Le degré d'un monôme $\mathbf{u}$ est égal à son poids de Hamming $|\mathbf{u}|$.

Définition 3.8 (Degré d'une fonction booléenne)**.** Le degré d'une fonction booléenne f est défini comme le degré le plus élevé des monômes de coefficient non nul dans son ANF, c'est exactement le degré total du polynôme F.

Une classe de fonctions particulièrement importante est donnée par les *fonctions affines* qui sont les fonctions de degré au plus 1. Lorsqu'en plus le coefficient $f_{\mathbf{0}}$ est nul, on parle de fonctions linéaires qui sont caractérisées par la proposition 3.9 qui découle directement de l'ANF.

Proposition 3.9 (Fonctions linéaires)**.** Pour toute fonction linéaire f de $\mathbf{F}_2^m$ dans $\mathbf{F}_2$, il existe un unique élément $\mathbf{u}$ de $\mathbf{F}_2^m$ tel que

$$f(\mathbf{x}) = \mathbf{u} \cdot \mathbf{x} \quad \forall \mathbf{x} \in \mathbf{F}_2^m$$

ou le "$\cdot$" est ici le produit scalaire ($\sum_i u_i x_i$). Nous noterons cette fonction $\lambda_{\mathbf{u}}$.

Regardons maintenant d'un peu plus près le comportement d'une fonction monôme. En utilisant l'arithmétique sur $\mathbf{F}_2$ et la définition de $\mathbf{x}^{\mathbf{u}}$, il est facile d'obtenir le résultat suivant :

Proposition 3.10 (Fonction monôme)**.** Une fonction monôme $\mathbf{u}$ ne vaut 1 qu'aux points $\mathbf{x}$ qui vérifient $\mathbf{u} \subseteq \mathbf{x}$ où l'inclusion est définie par

$$\text{pour} \quad \mathbf{x}, \mathbf{u} \in \mathbf{F}_2^m \quad \mathbf{u} \subseteq \mathbf{x} \quad \text{ssi} \quad \{i \mid u_i = 1\} \subseteq \{i \mid x_i = 1\} \ .$$

En particulier, elle est nulle sur tous les points $\mathbf{x}$ strictement plus petits que $\mathbf{u}$ pour l'ordre usuel et vaut 1 en $\mathbf{u}$.

Il existe une transformation involutive (i.e qui est sa propre inverse) qui relie les deux principales représentations d'une fonction booléenne connue sous le nom de transformation de Möbius. Elle est aussi connue sous le nom de transformation de Reed-Muller ce qui est assez naturel car elle est intimement liée à ces codes que nous présenterons au chapitre 9. Nous verrons également que l'on peut la calculer très efficacement en $O(m2^m)$ dans l'annexe A.

Théorème 3.11 (Transformée de Möbius)**.** Soit f une fonction booléenne à m variables. Son vecteur de valeurs $(f(\mathbf{x})_{\mathbf{x} \in \mathbf{F}_2^m})$ et les coefficients de son ANF $(f_{\mathbf{u}})_{\mathbf{u} \in \mathbf{F}_2^m}$ sont reliés par

$$f(\mathbf{x}) = \sum_{\mathbf{u} \subseteq \mathbf{x}} f_{\mathbf{u}} \quad \text{et} \quad f_{\mathbf{u}} = \sum_{\mathbf{x} \subseteq \mathbf{u}} f(\mathbf{x}) \ .$$

Démonstration. La première égalité (ANF vers vecteur des valeurs) découle directement de la proposition 3.10 :

$$f(\mathbf{x}) = \sum_{\mathbf{u}} f_{\mathbf{u}} \mathbf{x}^{\mathbf{u}} = \sum_{\mathbf{u} \subseteq \mathbf{x}} f_{\mathbf{u}} \ . \tag{3.1}$$

Pour la seconde égalité, il suffit de démontrer l'involutivité de la transformation de Möbius. Soit f' la fonction booléenne définie par $f'_{\mathbf{u}} = \sum_{\mathbf{x} \subseteq \mathbf{u}} f(\mathbf{x})$. En utilisant la première égalité on a alors

$$f'(\mathbf{y}) = \sum_{\mathbf{u} \subseteq \mathbf{y}} f'_{\mathbf{u}} = \sum_{\mathbf{u} \subseteq \mathbf{y}} \sum_{\mathbf{x} \subseteq \mathbf{u}} f(\mathbf{x}) \ .$$

On peut inverser les deux sommes pour obtenir

$$f'(\mathbf{y}) = \sum_{\mathbf{x} \subseteq \mathbf{y}} \sum_{\mathbf{u} | \mathbf{x} \subseteq \mathbf{u} \subseteq \mathbf{y}} f(\mathbf{x}) = \sum_{\mathbf{x} \subseteq \mathbf{y}} 2^{|\mathbf{y}|-|\mathbf{x}|} f(\mathbf{x})$$

et comme on est en caractéristique 2, on retrouve bien $f'(\mathbf{y}) = f(\mathbf{y})$. Remarquons au passage qu'un simple argument de cardinalité nous montre l'unicité de l'ANF. □

3.3 La transformée de Walsh

La transformée de Walsh (ou de Walsh-Hadamard) est un outil assez puissant pour analyser les propriétés cryptographiques d'une fonction booléenne. C'est en fait une transformée de Fourier discrète qui s'applique aux fonctions de $\mathbf{F}_2^m$ à valeurs dans un corps. Ici nous nous servirons uniquement de fonctions à valeurs dans $\mathbf{R}$.

Définition 3.12 (Transformée de Walsh)**.** La transformée de Walsh d'une fonction W de $\mathbf{F}_2^m$ à valeurs dans $\mathbf{R}$, noté $\widehat{W}$, est définie par

$$\widehat{W}(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbf{F}_2^m} W(\mathbf{x})(-1)^{\mathbf{u} \cdot \mathbf{x}} \ .$$

De plus il existe une transformée inverse

$$W(\mathbf{x}) = \frac{1}{2^m} \sum_{\mathbf{u} \in \mathbf{F}_2^m} \widehat{W}(\mathbf{x})(-1)^{\mathbf{u} \cdot \mathbf{x}} \ .$$

La transformée de Walsh est calculable très efficacement en $O(m2^m)$ par un algorithme similaire à celui utilisé pour la transformée de Möbius que l'on a détaillé en annexe A. Cette transformée possède également plusieurs propriétés intéressantes qui vont nous être utiles par la suite comme son comportement vis à vis de la convolution de deux fonctions :

Proposition 3.13 (Produit de convolution de fonctions sur $\mathbf{F}_2^m$)**.** Le produit de convolution (noté $*$) entre deux fonctions W_1 et W_2 de $\mathbf{F}_2^m$ à valeurs dans $\mathbf{R}$ est définie par :

$$(W_1 * W_2)(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbf{F}_2^m} W_1(\mathbf{y} + \mathbf{x}) W_2(\mathbf{y}) \ .$$

Il se calcule très efficacement par la transformée de Walsh car l'on a

$$\widehat{W_1 * W_2}(\mathbf{u}) = \widehat{W_1}(\mathbf{u}) \widehat{W_2}(\mathbf{u}) \ .$$

Démonstration.

$$\begin{aligned}\widehat{W_1}(\mathbf{u})\widehat{W_2}(\mathbf{u}) &= \left(\sum_{\mathbf{x}} W_1(\mathbf{x})(-1)^{\mathbf{u}\cdot\mathbf{x}}\right)\left(\sum_{\mathbf{x}} W_2(\mathbf{x})(-1)^{\mathbf{u}\cdot\mathbf{x}}\right) \\ &= \sum_{\mathbf{x}}\sum_{\mathbf{y}} W_1(\mathbf{x})W_2(\mathbf{y})(-1)^{\mathbf{u}\cdot(\mathbf{x}+\mathbf{y})}.\end{aligned}$$

En inversant les sommes et en remplaçant la somme sur $\mathbf{x}$ par une somme sur $\mathbf{x}+\mathbf{y}$, on obtient

$$\widehat{W_1}(\mathbf{u})\widehat{W_2}(\mathbf{u}) = \sum_{\mathbf{x}}\left[\sum_{\mathbf{y}} W_1(\mathbf{x}+\mathbf{y})W_2(\mathbf{y})\right](-1)^{\mathbf{u}\cdot\mathbf{x}} = \widehat{W_1 * W_2}(\mathbf{u})$$

d'où le résultat. □

Pour appliquer la transformée de Walsh à une fonction booléenne f on va en fait l'appliquer à la fonction signe de f :

Définition 3.14 (Spectre de Walsh)**.** On notera S_f la fonction signe de f qui à $\mathbf{x}$ associe $(-1)^{f(\mathbf{x})}$. L'ensemble des valeurs de la transformée $\widehat{S_f}$ est connu sous le nom de *spectre de Walsh* de f.

Le spectre de Walsh de f nous donne une troisième représentation d'une fonction booléenne par inversibilité de la transformée. On a de plus la proposition suivante :

Proposition 3.15 (Égalité de Parseval)**.**

$$\sum_{\mathbf{u}} \widehat{S_f}(\mathbf{u})^2 = 2^{2m} \ .$$

Démonstration. Il suffit d'écrire ce que nous donne la somme des carrés :

$$\sum_{\mathbf{u}} \widehat{S_f}(\mathbf{u})^2 = \sum_{\mathbf{u}}\left(\sum_{\mathbf{x}}(-1)^{\mathbf{x}.\mathbf{u}+f(\mathbf{x})}\right)^2 = \sum_{\mathbf{u}}\sum_{\mathbf{x}}\sum_{\mathbf{y}}(-1)^{\mathbf{x}.\mathbf{u}+f(\mathbf{x})+\mathbf{y}.\mathbf{u}+f(\mathbf{y})} \ .$$

Et en inversant la somme sur $\mathbf{u}$, on voit que $\sum_{\mathbf{u}}(-1)^{(\mathbf{x}+\mathbf{y}).\mathbf{u}}$ fait 0 sauf si $\mathbf{x}$ est égal à $\mathbf{y}$. On obtient alors

$$\sum_{\mathbf{u}} \widehat{S_f}(\mathbf{u})^2 = \sum_{\mathbf{x}}\sum_{\mathbf{u}}(-1)^{2f(\mathbf{x})} = 2^{2m} \ .$$

□

La représentation en fréquence (définition 3.14) d'une fonction booléenne est très commode pour analyser et calculer certaines propriétés cryptographiques. Un exemple important de telle propriété est la *non-linéarité* qui va correspondre à la distance minimale d'une fonction aux fonctions affines :

Définition 3.16 (Non-linéarité)**.** La non-linéarité d'une fonction booléenne f à m variables est définie comme

$$\min_{\mathbf{u}\in\mathbf{F}_2^m,\ c\in\mathbf{F}_2} |f + \lambda_{\mathbf{u}} + c| \ .$$

Nous verrons au chapitre 9 qu'il s'agit aussi de la distance de f au code de Reed-Muller d'ordre 1. Il est facile de voir que la valeur au point $\mathbf{u}$ de la transformée de la fonction signe de f est directement reliée à la distance de f à $\lambda_{\mathbf{u}}$ par

$$|f - \lambda_{\mathbf{u}}| = 2^{m-1} - \frac{\widehat{S_f}(\mathbf{u})}{2}$$

car

$$\widehat{S_f}(\mathbf{u}) = \sum_{\mathbf{x}} S_f(\mathbf{x})(-1)^{\mathbf{u}\cdot\mathbf{x}} = \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})+\lambda_{\mathbf{u}}(\mathbf{x})} \ .$$

La non-linéarité de f est donc directement donnée par le maximum de la valeur absolue (pour tenir compte des fonctions affines) de la transformée de Walsh de sa fonction signe.

Une autre propriété intimement liée à la transformée de Walsh est ce que l'on appelle l'*autocorrélation* d'une fonction booléenne que l'on utilisera au chapitre 6 :

Définition 3.17 (Autocorrélation)**.** On appelle valeurs d'autocorrélation d'une fonction booléenne f l'ensemble des distances de cette fonction à ses translatées :

$$\{|\mathbf{x} \ \mapsto \ f(\mathbf{x}) + f(\mathbf{x}+\mathbf{y})|\,, \quad \mathbf{y} \in \mathbf{F}_2^m\} \ .$$

On voit que ces valeurs se calculent très bien avec la transformée de Walsh car elles s'obtiennent directement à partir des coefficients du produit de convolution de la fonction signe de f avec elle-même :

$$S_f * S_f(\mathbf{y}) = \sum_{\mathbf{x}} S_f(\mathbf{y}+\mathbf{x}) S_f(\mathbf{x}) = \sum_{\mathbf{x}} (-1)^{f(\mathbf{y}+\mathbf{x})+f(\mathbf{x})}$$

et l'on retombe sur 2^m moins la valeur du coefficient d'autocorrélation en $\mathbf{y}$.

3.4 Les fonctions booléennes cryptographiques

Le choix d'une bonne fonction booléenne est un problème difficile et en pratique le concepteur d'un système de chiffrement a un choix relativement restreint. Il convient en effet d'utiliser des fonctions avec de bonnes propriétés pour résister aux nombreuses attaques découvertes au cours de l'évolution de la cryptographie.

Une fonction cryptographique doit par exemple être équilibrée, de degré relativement élevé et posséder une non-linéarité élevée. Selon les cas, il existe en fait bien d'autres propriétés à considérer comme l'autocorrélation, la

résilience, le critère de propagation ou les structures linéaires de la fonction. Pour résister aux attaques algébriques, il y a aussi l'immunité algébrique définie au chapitre 8 et dont on discutera amplement dans cette thèse. Un bon document de référence sur les fonctions booléennes et leur propriétés cryptographiques est le rapport d'HDR d'Anne Canteaut [Can06].

Mais les critères cryptographiques ne font pas tout, il est aussi impératif que l'on puisse évaluer rapidement une fonction booléenne pour obtenir des systèmes efficaces. Cela est particulièrement vrai pour les chiffrements à flot où l'on recherche une efficacité maximale. Malheureusement, il existe une borne sur la complexité de calcul d'une fonction booléenne en terme de circuit électronique [Sha49b, Lup58] qui montre que la plupart des fonctions sont difficiles à calculer. L'on doit donc se restreindre à certaines classes de fonctions :

Les fonctions à petit nombre de variables

Lorsque le nombre de variables est faible (de l'ordre de 8) il est tout à fait envisageable et performant de stocker le vecteur des valeurs de la fonction. On peut alors choisir n'importe quelle fonction booléenne. En revanche, dès que le nombre de variables devient plus important on est obligé de considérer des fonctions avec une forte structure de manière à pouvoir les calculer efficacement.

Les fonctions avec une ANF simple

Une première solution consiste à considérer des fonctions qui ont peu de termes dans leur ANF, ou bien qui admettent une ANF qui se factorise, ce qui permet un calcul rapide par le biais de la forme polynomiale. Le problème est que de telle fonctions sont rarement optimales au niveau de la sécurité.

Les fonctions symétriques

Ce sont des fonctions qui ne dépendent que du poids de Hamming de l'entrée et qui se calculent donc très efficacement.

Définition 3.18 (fonctions symétriques)**.** Une fonction booléenne à m variables est dite symétrique s'il existe un vecteur de $m+1$ bits $v = (v_0, \ldots, v_m)$ (dit vecteur de valeurs simplifié) tel que

$$\forall \mathbf{x} \quad f(\mathbf{x}) = v_{|\mathbf{x}|} \ .$$

L'implémentation vraiment efficace de ces fonctions a conduit à plusieurs études sur leurs propriétés [Sav94, MS02, Gou04]. Plus récemment, Anne Canteaut et Marion Videau ont étudié leur propriétés cryptographiques de manière très poussée. Pour connaître les résultats de cette étude, le lecteur

est invité à lire les articles [Vid04, CV05, Vid05b] et la thèse de Marion Videau [Vid05a]. On verra également au chapitre 10 des fonctions symétriques d'immunité algébrique maximale.

Les fonctions puissances

Cette classe est dérivée de celle des fonctions puissances sur les corps finis. Ces dernières sont en fait des fonctions vectorielles que l'on peut rendre scalaires en considérant une fonction linéaire de la sortie. Elles demandent en général plus de travail que les fonctions symétriques pour être calculées mais cela reste tout à fait raisonnable, notamment quand l'exposant est bien choisi [ABMV93]. C'est d'ailleurs ce type de fonction qui a été retenu dans les boites S de l'AES même si dans ce cas précis le nombre de variables est suffisamment faible pour pouvoir tout tabuler.

Définition 3.19 (fonctions puissance)**.** Une fonction de $\mathbf{F}_{2^m}$ dans $\mathbf{F}_{2^m}$ est dite fonction puissance si elle est de la forme

$$\mathbf{x} \mapsto \mathbf{x}^a \ .$$

L'entier a est appelé l'exposant de la fonction.

Le choix de l'exposant est bien sûr crucial et de nombreuses familles d'exposants ont été étudiées du point de vue cryptographique, notamment les familles d'exposants qui donnent pour m impair des fonctions avec une très bonne non-linéarité, ce sont des fonctions dites AB pour Almost Bent. Citons également la fonction inverse d'exposant -1 qui est utilisée dans l'AES.

Chapitre 4

Cryptanalyses du registre filtré

Nous allons nous intéresser maintenant à l'une des constructions les plus simples de chiffrements à flot. Il s'agit d'un simple registre à rétroaction linéaire filtré par une fonction booléenne. Une grande partie de cette thèse porte sur certains types d'attaques que l'on peut mener sur ce chiffrement. Nous considérerons le cas d'un registre sur $\mathbf{F}_2$ pour simplifier la présentation, la plupart des résultats restent néanmoins valides si l'on se place sur un corps plus gros.

Après une présentation du registre filtré dans la première section, nous énumérons les attaques connues sur le registre filtré dans les sections qui suivent. Nous ne ferons qu'évoquer ici les attaques fondées sur la complexité linéaire et les attaques algébriques que nous verrons bien plus en détail dans les chapitres 7 et 8. Mentionnons également l'attaque du chapitre 6 qui se rapproche, par les méthodes utilisées, des attaques par corrélation. Toutes les analyses présentées ici sur le registre filtré considèrent la suite chiffrante partiellement connue, il s'agit donc d'attaques à clair connu.

4.1 Le registre filtré

Le registre filtré est constitué d'un registre à rétroaction linéaire (voir chapitre 2) filtré par une fonction booléenne (voir chapitre 3) comme décrit sur la figure 4.1. À chaque pas d'horloge, le LFSR est avancé d'un pas et certains bits de l'état interne du registre sont utilisés comme entrées de la fonction de filtrage qui produit ainsi un bit de la suite chiffrante.

Nous utiliserons les notations de cette figure dans toute la suite de cette thèse. Ainsi, le LFSR utilisé est toujours de longueur l et de polynôme générateur $g(X)$ primitif lui-même de degré l. La séquence binaire engendrée par ce LFSR est notée $(s_t)_{t\geq 0}$. L'état initial, qui est aussi la clef secrète du système, est donc $(s_0, \ldots, s_{l-1})$. La fonction de filtrage est f, elle est

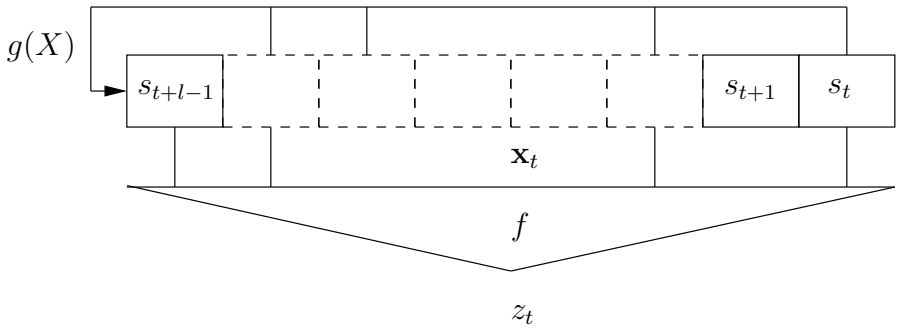

FIG. 4.1 – Le registre filtré. Seule une partie des bits de l'état courant est utilisée comme entrée $\mathbf{x}_t$ de la fonction de filtrage f.

toujours choisie équilibrée à m variables et de degré d. La position des m variables parmi les l n'a pas d'importance dans cette thèse bien qu'elle puisse en avoir pour certaines attaques. Nous noterons $t_1, \ldots, t_m$ ces positions et $\mathbf{x}_t = (s_{t+t_1}, \ldots, s_{t+t_m})$ le vecteur de m bits des entrées de f au temps t. Enfin, la suite chiffrante produite par ce registre filtré est notée $(z_t)_{t\geq 0}$. À chaque indice de temps t positif ou nul, l'équation suivante est vérifiée :

$$f(\mathbf{x}_t) = f(s_{t+t_1}, \ldots, s_{t+t_m}) = z_t \ .$$

Cette construction n'est pas utilisée telle quelle en pratique, les concepteurs de chiffrements à flot préférant souvent des constructions plus compliquées qui conduisent à de meilleures performances pour un même niveau de sécurité. Néanmoins, du moment que le système est correctement dimensionné, les attaques actuellement connues ne mettent pas vraiment un tel système en danger.

Le registre filtré présente en outre l'avantage d'utiliser les principaux blocs de constructions que l'on retrouve dans la majorité des chiffrements à flot. Sa simplicité permet ainsi une meilleure compréhension des différentes attaques qui s'appliquent bien souvent à des systèmes plus compliqués.

Par exemple, la plupart des attaques qui marchent sur le registre filtré s'appliquent presque directement à tous les chiffrements à flot dont la fonction de mise à jour de l'état interne est linéaire. On peut montrer qu'un tel système peut presque toujours se mettre sous la forme décrite sur la figure 4.2 où l'état interne est en fait constitué de plusieurs LFSR de polynôme générateur irréductible. Le cas du registre filtré est donc un cas particulier de cette famille générale de chiffrement à flot.

En pratique, pour une taille de l'état interne fixée, c'est celui qui offre la meilleure sécurité. Il semble par exemple difficile d'attaquer une sous-partie du système comme cela peut arriver lorsqu'il y a plusieurs registres (voir par exemple les attaques par corrélation décrites plus loin dans ce chapitre). En revanche, avoir plusieurs registres peut conduire à une implémentation plus efficace, notamment dans les systèmes embarqués les plus légers.

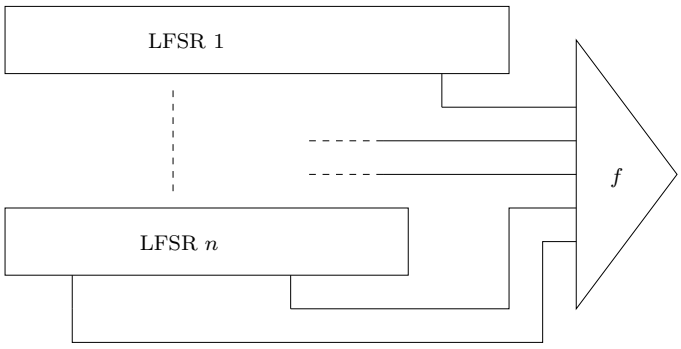

FIG. 4.2 – Forme générale d'un chiffrement à flot avec mise à jour de l'état interne de manière linéaire.

L'idée d'une preuve que toute fonction de mise à jour linéaire peut être représentée par des LFSR est la suivante. Si l'on appelle $\mathcal{L}$ la fonction de mise à jour linéaire de l'état interne qui va de $\mathbf{F}_2^l$ dans $\mathbf{F}_2^l$, on peut la représenter par une matrice. Le polynôme caractéristique de cette matrice nous donne alors une formule de récurrence linéaire vérifiée par l'état interne et donc par chacun de ses bits. On peut alors montrer (voir la discussion au début de la dernière section du chapitre 2) que la décomposition de ce polynôme en facteurs irréductibles nous donne les polynômes générateurs des LFSR de la figure 4.2 dont la somme de la longueur des états vaut l. On peut remarquer que lorsque le polynôme caractéristique de la matrice est irréductible, on retombe sur le cas d'un seul registre filtré.

4.2 Recherche exhaustive et compromis temps-mémoire

La manière la plus brutale mais souvent aussi la plus efficace pour retrouver la clef secrète d'un système de chiffrement est ce que l'on appelle la recherche exhaustive. Elle est applicable sur tous les chiffrements et consiste simplement à essayer toutes les clefs secrètes possibles jusqu'à ce que l'on retrouve la suite chiffrante observée. Pour une clef secrète de k bits, sa complexité est donc en $O(2^k)$ et correspond à la limite que l'on va essayer de battre en considérant des attaques plus évoluées. La complexité est en fait légèrement supérieure car, pour générer les premiers bits de suite chiffrante à partir de la clef, on doit souvent appliquer un processus d'initialisation assez long.

Dans les chiffrements à flot la longueur de clef est presque tout le temps plus petite que la taille de l'état interne, donc une recherche exhaustive sur ce dernier en $O(2^l)$ est moins efficace. En revanche, elle présente certains

avantages car elle va nous permettre de développer des algorithmes du type compromis "temps-mémoire-longueur de suite chiffrante disponible". Cette approche n'est pas possible en raisonnant directement sur la clef car l'IV est inconnue pendant la phase de précalcul. Remarquons qu'il est nénmoins possible d'utiliser un compromis temps-mémoire sur le couple (IV, clef) comme indiqué dans l'article [HS05]. Dans la suite nous noterons T pour la complexité de l'attaque, M pour la mémoire qu'elle utilise et B pour le nombre de bits de suite chiffrante requis.

Premièrement, comme l'ont fait remarquer indépendamment Babbage [Bab95] et Golic [Gol97], on peut améliorer la complexité de la recherche exhaustive sur l'état interne si l'on dispose de beaucoup de bits de suite chiffrante (ici B). En arrangeant les $B - l$ vecteurs de l bits consécutifs pouvant être construits à partir des B bits de suite chiffrante dans une table de hachage, il est possible de tester si un état initial est un antécédent d'un de ces vecteurs en $O(1)$. La probabilité pour un état pris au hasard d'être un tel antécédent est de $O(B/2^l)$. Il faut donc en tester de l'ordre de $T = O(2^l/B)$ en moyenne avant d'en trouver un et donc de retrouver la clef du système.

Comme ici $M = B$, on obtient ainsi un compromis qui suit la courbe $TM = 2^l$ et pas de précalcul. Un point intéressant sur cette courbe est obtenu pour un B de l'ordre de $2^{l/2}$ ce qui nous donne une attaque de complexité $2^{l/2}$ aussi bien en temps qu'en mémoire. C'est d'ailleurs pourquoi en pratique l est choisi deux fois plus grand que la longueur de la clef, même si obtenir autant de bits de suite chiffrante n'est pas vraiment envisageable car il y a souvent des limites sur le nombre de bits qu'un système est censé produire.

Hellman a montré qu'il est possible d'améliorer la complexité d'une recherche exhaustive sans avoir beaucoup de suite chiffrante. Dans [Hel80] Il obtient un compromis temps-mémoire pour seulement l bits de suite chiffrante qui suit la courbe $TM^2 = 2^{2l}$. En revanche, son attaque nécessite un précalcul de l'ordre de 2^l.

L'idée est de voir le système comme une fonction aléatoire $\mathcal{F}$ de $\mathbf{F}_2^l$ dans $\mathbf{F}_2^l$ qu'il est facile de calculer, mais bien sûr difficile d'inverser. Pour cela on associe à un état interne les l premiers bits de suite chiffrante produits à partir de cet état. L'algorithme va générer dans une phase de précalcul de nombreuses chaînes d'itérés de longueur t de cette fonction et stocker les paires (point de départ, point d'arrivée). Pour cela, on part d'un point aléatoire de $\mathbf{F}_2^l$, on itère t fois la fonction $\mathcal{F}$ et l'on arrive à un point final que l'on stocke. Si maintenant on cherche à inverser la fonction sur une valeur couverte par l'une de ces chaînes, en itérant successivement la fonction sur cette valeur, on va tomber sur l'un des points d'arrivée stockés. Il sera alors possible de repartir du point de départ associé pour inverser la fonction.

Pour que cela marche, il convient de prendre un nombre de chaînes suffisamment grand pour couvrir tout l'espace $\mathbf{F}_2^l$. Malheureusement, en pratique

la situation est plus compliquée que cela car pour une fonction aléatoire, une fraction constante de points n'ont pas d'antécédent et il sera impossible d'inverser leur image sauf si ces points ont été choisis comme points de départ. En plus, si l'on prend de plus en plus de chaînes, la probabilité qu'elles couvrent les mêmes points de l'espace devient importante. La solution retenue est d'utiliser des versions légèrement modifiées de la fonction $\mathcal{F}$, les $\mathcal{F}_i$ de la forme $\mathcal{F} \circ \mathcal{L}_i$ où les $\mathcal{L}_i$ sont des fonctions linéaires. Comme une fonction linéaire est facile à inverser, on peut résoudre les difficultés mentionnées plus haut en construisant un certain nombre de chaînes par fonction $\mathcal{F}_i$. On peut ainsi obtenir le compromis de [Hel80] qui, si l'on prend $T = M$, nous donne une attaque en temps et en mémoire de l'ordre de $O(2^{2l/3})$ pour un B très faible. Il existe plusieurs variantes de cet algorithme qui conduisent à des implémentations efficace, citons notamment l'utilisation de points distingués due a Rivest [Den82] et l'utilisation des "rainbow tables" [Oec03].

Sur le registre filtré, Biryukov et Shamir ont amélioré cette attaque dans [BS00] en montrant qu'il est possible d'exploiter un grand nombre de bits de suite chiffrante. Ils obtiennent un compromis qui suit la courbe $TM^2B^2 = 2^{2l}$. Pour le même temps de calcul que l'attaque de Hellman, on arrive alors à s'en sortir avec $M = B = O(2^{l/3})$. Le précalcul de leur attaque est quant à lui en $O(2^l/B)$, ce qui nous donne $O(2^{2l/3})$ avec nos choix de T, M et B. Cette attaque semble plus ou moins réalisable jusqu'à des l de l'ordre de 100, pour plus de détails voir [BS00].

4.3 Attaques par corrélation

Les attaques par corrélation forment une famille très importante d'attaque sur les chiffrements à flot. Elles ont été découvertes par Siegenthaler [Sie85] et s'appliquent dans leur forme initiale uniquement dans le cas de plusieurs registres filtrés par une fonction booléenne.

Si la fonction de filtrage est mal choisie, il peut exister une corrélation entre la sortie d'un des registres internes et la suite chiffrante produite. On peut alors faire une recherche exhaustive sur ce registre uniquement et retrouver l'état qui conduit à une corrélation maximale. La propriété de la fonction booléenne qui intervient ici est la résilience qui mesure l'équilibre de la fonction lorsque certaines de ses variables d'entrées sont fixées. Nous n'en dirons pas plus ici, mais le lecteur est invité à lire [Jön02] pour plus d'information.

Une vision intéressante des choses pour réaliser cette attaque est de voir la suite chiffrante comme une version bruitée du registre que l'on veut attaquer. On se ramène alors à un problème de décodage. Dans le cas où le registre interne que l'on attaque est un LFSR de longueur l, on verra au chapitre 9 qu'il est possible de retrouver l'état interne qui conduit à une corrélation maximale en $O(l2^l)$. Il faut bien sûr disposer d'assez de bits de

suite chiffrante pour que le vrai état initial soit distinguable des autres.

Avec la complexité ci-dessus, cette forme de l'attaque ne marche pas dans le cas qui nous intéresse où un seul registre filtré est utilisé. Mais il existe des attaques par corrélation de meilleure complexité que $O(l2^l)$, on parle alors d'*attaques par corrélation rapides*. Ces attaques ont été introduites par Meier et Staffelbach dans [MS88] et exploitent vraiment le fait qu'il s'agit d'un problème de décodage. Depuis, de nombreuses variantes ont été proposées [CJS00, JJ00, MFI01, JJ02, CF02].

L'idée est d'utiliser des algorithmes de décodage de meilleure complexité que le décodage au maximum de vraisemblance de complexité $O(l2^l)$. Bien sûr, la longueur de la suite chiffrante interceptée doit être plus grande pour que l'attaque fonctionne.

On se ramène donc à l'étude des algorithmes de décodage d'un code linéaire de dimension l et longueur 2^l qui est formé par toutes les séquences produites par le LFSR que l'on cherche à attaquer. Il existe de nombreuses approches et l'on peut en trouver une bonne synthèse dans la thèse de Jönsson [Jön02]. La plupart des attaques utilisent des équations de parité de poids faible vérifiées par la suite produite par le LFSR attaqué comme dans [CT00]. Nous verrons au chapitre 5 comment trouver de telles équations et présenterons une attaque qui les utilise au chapitre 6.

Pour concevoir et analyser les performances des algorithmes de décodage, la modélisation des erreurs (c'est-à-dire du canal sur lequel on se trouve) est cruciale. Ici, seule une partie des 2^l bits d'un mot de code bruité va être connue et la plupart des attaques supposent que l'on est en présence d'un canal binaire symétrique. Le taux d'erreur est calculé en fonction des propriétés de la fonction booléenne de filtrage.

Bien que cette approche présente l'avantage d'être simple, selon le type d'attaque elle peut être discutable. Parmi les simulations effectuées, il apparaît en effet que le comportement de certaines attaques est moins bon que celui prédit par la théorie [Lev04b]. De plus, nous verrons que l'analyse présentée au chapitre 6 qui nous semble basée sur une hypothèse plus raisonnable ne donne pas forcément les mêmes résultats.

4.4 Autres attaques

Il existe bien sûr de nombreuses autres approches pour attaquer un registre filtré. Les principales autres familles sont listées ci-dessous avec une rapide description et les principales références. Nous verrons également dans cette thèse plusieurs attaques en détail aux chapitres 6, 7 er 8.

Attaques Algébriques Ce sont les attaques qui vont nous intéresser le plus dans cette thèse. Elles sont essentiellement de nature algébrique contrairement aux attaques par corrélation par exemple, où les outils utilisés sont

plus de nature statistique. Par attaques algébriques, on désigne la plupart du temps les attaques décrites au chapitre 8, mais l'attaque basée sur la complexité linéaire du chapitre 7 rentre aussi dans la famille des attaques algébriques.

Attaques par inversion Ces attaques ont été initialement introduites par Golic [Gol96] et généralisées dans [GCD00], elles exploitent la position des entrées de la fonction de filtrage dans le registre. Rappelons que ces positions sont données par les variables t_1 jusqu'à t_m. En substance, la complexité de l'attaque va alors se ramener à une recherche exhaustive sur ce que l'on appelle la mémoire effective du registre qui vaut

$$\frac{t_m - t_1}{\mathrm{PGCD}_i(t_i - t_{i+1})} .$$

Si le PGCD entre l'écart des entrées vaut 1 et que $t_m - t_1$ est de l'ordre de l, ces attaques peuvent être facilement évitées. Il existe également une autre technique de complexité similaire fondée sur une représentation en treillis et sur l'algorithme de Viterbi, voir [LBGZ01].

Distingueur Mentionnons enfin les attaques qui ne visent pas directement à retrouver l'état initial du LFSR mais juste à distinguer la suite chiffrante d'une suite aléatoire. Ces attaques peuvent permettre de vérifier que l'on a correctement retrouvé la suite chiffrante ou nous aider à retrouver la structure du chiffrement dans le cas où elle est inconnue. Notez également que cette approche peut aussi conduire à des attaques de "type diviser pour régner" s'il est possible de faire une recherche exhaustive sur une sous partie de la clef comme dans [CF01].

L'idée est ici de trouver des équations qui présentent un biais sur la suite chiffrante. La plupart des techniques pour faire cela reposent, de la même manière que les attaques par corrélation rapides, sur les équations linéaires induites par les multiples de poids faible des polynômes générateurs des LFSR qui interviennent dans le système. On en verra un exemple au chapitre 6 qui reprend l'attaque présentée dans [CF01]. Pour des chiffrements à flot plus évolués, des techniques efficaces sont données dans [Gol94] et [CHJ02].

Chapitre 5

Calcul des multiples de poids faible

Beaucoup d'attaques par corrélation et l'attaque dont nous parlerons dans le prochain chapitre utilisent des relations de parité de poids faible vérifiées par les bits de sortie d'une séquence produite par un LFSR. Au chapitre 2 on a vu que ces relations sont en fait données par les multiples de poids faible du polynôme générateur du LFSR. Nous nous intéressons ici à plusieurs algorithmes pour calculer ces relations.

Nous commencerons par les algorithmes classiques qui reposent tous sur des compromis temps-mémoire et dont le plus efficace est certainement celui de [CJM02] que nous présenterons dans la deuxième section. Nous présenterons ensuite une autre approche basée sur le calcul de logarithmes discrets qui a donné lieu, avec la collaboration de Yann Laigle-Chapuy, à la publication [DLC07]. Cette approche s'avère utile pour calculer les multiples de poids 4 qui jouent un rôle essentiel dans certaines attaques ([CT00], chapitre 6).

5.1 Présentation du problème

Notre but est de rechercher des multiples de poids w fixé et de degré plus petit qu'une borne D d'un polynôme $g(X)$ à coefficients dans $\mathbf{F}_2$. Pour des applications pratiques, seuls les w très petits nous intéressent, typiquement 3, 4, 5, 6 ou 7. Avant de détailler les algorithmes, on rappelle deux résultats essentiels pour cette section que l'on a vus vers la fin du chapitre 2 et qui font le lien entre équations de parité pour le LFSR et multiples de $g(X)$.

Définition 5.1 (Équations de parité associées à un polynôme)**.** À un polynôme $p(X) = \sum_{i=1}^{n} p_i X^i$ de degré n à coefficients dans $\mathbf{F}_2$ on associe une équation de parité sur une séquence $(s_t)_{t\geq 0}$ qui peut s'appliquer pour

n'importe quel décalage dans le temps par :

$$\sum_i p_i s_{t+i} = 0 \quad \text{pour un décalage } t \geq 0 \ .$$

Proposition 5.2 (Équations de parité pour un LFSR)**.** Soit une séquence $(s_t)_{t\geq 0}$ non nulle produite par un LFSR de polynôme générateur $g(X)$ primitif, alors l'équation de parité associée à un polynôme $p(X)$ à coefficients dans $\mathbf{F}_2$ est vérifiée pour tout décalage du temps si et seulement si $p(X)$ est égal à 0 modulo $g(X)$. C'est-à-dire si et seulement si $p(X)$ est un multiple de $g(X)$.

Comme en général les LFSR utilisés ne sont pas dégénérés, $g(X)$ a un terme constant qui vaut 1 et nous nous intéresserons uniquement aux multiples $p(X)$ de terme constant 1. Les autres, qui sont divisibles par une puissance de X, ne représentent en effet que des équations de parité qui sont décalées dans le temps.

En supposant que les X^i modulo $g(X)$ sont aléatoires, on a le résultat suivant qui nous donne une idée du nombre de multiples attendus de poids w et de degré au plus D. Cette "heuristique" nous permettra de choisir D en fonction de nos besoins en nombre de multiples. En pratique, l'ordre de grandeur est correct pour la plupart des cas intéressants.

Heuristique 5.3 (Nombre de multiples)**.** Une approximation du nombre de multiples de poids w et de degré au plus D d'un polynôme $g(X)$ de degré l est donnée par :

$$\frac{\binom{D}{w-1}}{2^l} \sim \frac{D^{w-1}}{(w-1)!2^l} \quad \text{pour } w = o(D) \ .$$

Justification. On fait ici l'hypothèse qu'un polynôme de poids w pris au hasard parmi les D^{w-1} possibles (car on suppose qu'il a toujours un terme constant égal à 1) a une chance sur 2^l d'être égal à 0 modulo $g(X)$. Le comportement asymptotique s'en déduit directement du moment que w est un $o(D)$. □

L'algorithme standard pour trouver tous les multiples de poids w et de degré au plus D est d'utiliser un compromis temps-mémoire :

Algorithme 5.4 (Compromis temps-mémoire)**.** L'algorithme prend en entrée un polynôme $g(X)$, un degré maximal D et un poids $w = 1 + q_1 + q_2$ avec $q_1 \leq q_2$. Il retourne tous les multiples de $g(X)$ de poids w et de degré au plus D.

1. [Précalcul] Calculer tout les X^i modulo $g(X)$ pour $i = 1..D$ et les stocker dans un tableau `mod`$[i]$.

2. [Mise en table] Pour tous les q_1-uplets $\Gamma = (\gamma_1, \ldots, \gamma_{q_1})$ tel que $0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$, calculer et stocker les paires $(1 + \sum_i \texttt{mod}[\gamma_i], \Gamma)$ dans une table de hachage indexée par le premier élément.
3. [Recherche] Pour tous les q_2-uplets $\Delta = (\delta_1, \ldots, \delta_{q_2})$ tel que $0 < \delta_1 < \cdots < \delta_{q_2} \leq D$, calculer $\sum_i \texttt{mod}[\delta_i]$ et regarder dans la table si un élément est le même. Si c'est le cas, on trouve un multiple

$$1 + \sum_i X^{\gamma_i} + \sum_i X^{\delta_i} \ .$$

La complexité en mémoire de cet algorithme est en $O(D^{q_1})$ et celle en temps est en $O(D^{q_2})$. Remarquons que dans le cas impair, la phase de recherche est en fait inutile car tout le travail a déjà été fait en calculant la table, il suffit juste de détecter les collisions lors de sa création. Le compromis usuel est de choisir $q_1 = \lfloor \frac{w-1}{2} \rfloor$ et $q_2 = \lceil \frac{w-1}{2} \rceil$.

5.2 Algorithme de Chose Joux Mitton

Pour ce problème de calcul de multiples de poids faible, il est en fait possible d'utiliser uniquement la racine carrée de la mémoire nécessaire à l'algorithme précédent comme l'ont montré les auteurs de [CJM02]. L'idée est maintenant de découper w en 4 et l'amélioration en mémoire n'arrive qu'à partir du poids $w = 5$, dans les cas plus petits on n'a de toute manière pas besoin de plus de $O(D)$ mémoire.

On va présenter l'algorithme pour le poids $w = 5$ pour des raisons de simplicité, mais il est tout à fait généralisable et le lecteur peut lire [CJM02] pour avoir la version générale.

Algorithme 5.5 (Algorithme CJM)**.** L'algorithme prend en entrée un polynôme $g(X)$ de degré l, un degré maximal D et calcule tous les multiples de $g(X)$ de poids 5. Il utilise un paramètre a souvent choisi de l'ordre de $\log_2(D)$.

1. [Précalcul] Pour $i = 1..D$, stocker i dans une table de hachage H_1 où l'indice est donné par la valeur des a bits de poids fort de $X^i \mod g(X)$.
2. [Boucle 1] Effectuer les étapes suivantes pour toutes les valeurs v de a bits.
3. [Boucle 2] Pour tous les éléments de H_1 associés à un X^i chercher s'il n'existe pas une autre entrée j dans H_1 telle que la somme $X^i + X^j$ modulo $g(X)$ soit égale à v sur les bits de poids fort. Cette étape prend $O(1)$ par élément de H_1 en moyenne si a est de l'ordre de $\log_2(D)$. Si oui exécuter l'étape 4 avec le couple (i, j).
4. [Collision ?] Chercher à mettre le triplet $(X^i + X^j \mod g(X), i, j)$ dans une table H_2 qui ne tient pas compte du bit de poids faible. S'il y a collision

avec $(X^{i'} + X^{j'} \mod g(X), i', j')$, vérifier si la somme des $X^i + X^j + X^{i'} + X^{j'} \mod g(X)$ vaut 1, si oui on obtient un multiple :

$$1 + X^i + X^j + X^{i'} + X^{j'} \ .$$

L'algorithme obtient bien toutes les positions, car si l'on regarde un multiple de poids 5, les a bits de poids fort de la somme de deux puissances de X sont forcément égaux aux a bits de poids fort de la somme des deux autres. En plus la mémoire est réduite en moyenne d'un facteur D ce qui permet de trouver les multiples de poids 5 avec une mémoire linéaire en D. Dans le cas général, la mémoire est d'en fait $O(D^{\lceil \frac{w-1}{4} \rceil})$ pour une complexité en temps qui reste de $O(D^{\lceil \frac{w-1}{2} \rceil})$.

Si l'on ne fait pas attention, il est par contre possible d'obtenir plusieurs fois le même multiple. En pratique, pour le poids 5, on peut imposer des contraintes à chaque étape pour que ça n'arrive pas. L'idée est de commencer par prendre uniquement des couples tels que $i < j$ à l'étape 2. Ensuite, si la somme des 4 puissances de X doit valoir 1, il est toujours possible de s'arranger pour que le bit de poids fort de v soit nul. Cela enlève déjà pas mal de calculs inutiles. Il est possible de les enlever tous en regardant le deuxième bit de point fort de v et ainsi de suite mais nous ne détaillerons pas cela ici.

5.3 Utilisation des logarithmes discrets

Dans les algorithmes que nous avons détaillés dans la section précédente, la structure pourtant assez forte du corps fini $\mathbf{F}_2[X]/g(X)$ n'a absolument pas été exploitée. En fait, on a donné des algorithmes génériques pour trouver des combinaisons linéaires de très petit poids parmi un grand nombre de vecteurs plus ou moins aléatoires. Pour ce dernier problème, qui est NP-complet, il semble difficile d'obtenir des algorithmes plus efficaces que ceux que nous avons vus.

En revanche, si l'on arrive à exploiter d'une manière ou d'une autre la structure du corps fini, on peut espérer faire mieux. C'est ce que nous avons tenté de faire en utilisant les logarithmes discrets dans le groupe des éléments non nuls de $\mathbf{F}_2[X]/g(X)$. Nous noterons ce logarithme discret Log dans toute cette section.

Définition 5.6 (Logarithme discret)**.** Soit un groupe G engendré par un élément g. Le problème du logarithme discret est, étant donné un élément x de G, de retrouver l'entier i tel que $g^i = x$.

Selon le groupe, il existe plusieurs algorithmes assez efficaces pour calculer ces derniers que nous verrons plus loin. Ici, nous allons voir comment on peut les exploiter pour obtenir des multiples de petit degré. L'idée n'est

en fait pas nouvelle et date de l'article peu connu [PK95]. Nous généralisons leur approche et en donnons une analyse de complexité plus réaliste.

L'idée est cette fois de décomposer w en $2 + q_1 + q_2$ avec $q_1 \leq q_2$. Considérons maintenant le q_1-uplet

$$\Gamma = (\gamma_1, \ldots, \gamma_{q_1}) \quad \text{avec } 0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$$

et le q_2-uplet

$$\Delta = (\delta_1, \ldots, \delta_{q_2}) \quad \text{avec } 0 < \delta_1 < \cdots < \delta_{q_2} \leq D \ .$$

On peut alors considérer les logarithmes discrets suivants

$$L_\Gamma = \mathrm{Log}\left(1 + \sum_i X^{\gamma_i}\right) \quad \text{et} \quad L_\Delta = \mathrm{Log}\left(1 + \sum_i X^{\delta_i}\right) \ .$$

Par définition des logarithmes discrets, on obtient alors en notant

$$e(\Gamma, \Delta) \stackrel{\text{def}}{=} L_\Gamma - L_\Delta \mod 2^l$$

deux multiples de poids w (ou moins) de $g(X)$:

$$\left(1 + \sum_i X^{\gamma_i}\right) + X^{e(\Gamma,\Delta)}\left(1 + \sum_i X^{\delta_i}\right) \ ,$$
$$X^{e(\Delta,\Gamma)}\left(1 + \sum_i X^{\gamma_i}\right) + \left(1 + \sum_i X^{\delta_i}\right) \ .$$

De plus, le terme constant vaut 1, sauf si $e(\Gamma, \Delta)$ vaut zéro ce qui nous donne un multiple de poids plus faible que w qu'il faut diviser par une puissance de X pour le ramener à une forme standard. On obtient donc un multiple de poids w (au plus) et de degré au plus D si l'une des deux conditions suivantes est vérifiée :

$$e(\Gamma, \Delta) + \delta_{q_2} \leq D \ , \tag{5.1}$$

$$e(\Delta, \Gamma) + \gamma_{q_1} \leq D \ . \tag{5.2}$$

On pourra noter que ces conditions s'excluent l'une de l'autre dès que D est plus petit que 2^{l-1}. Un algorithme pour trouver tous les multiples de poids w et de degré au plus D est alors le suivant :

Algorithme 5.7 (Utilisation des logarithmes discrets [DLC07])**.** L'algorithme prend en entrée un polynôme $g(X)$, un degré maximal D et un poids w qui est décomposé cette fois en $2 + q_1 + q_2$ avec $q_1 \leq q_2$. On note Log le logarithme discret dans $(\mathbf{F}_2[X]/g(X))^*$ et l'algorithme calcule tous les multiples de poids w et de degré au plus D de $g(X)$.

1. [Mise en table] Pour tous les q_1-uplets $\Gamma = (\gamma_1, \ldots, \gamma_{q_1})$ tel que $0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$ calculer et stocker la paire (L_Γ, Γ) dans une table de hachage indexée par le Log.
2. [Multiples ?] Pour tous les q_2-uplets $\Delta = (\delta_1, \ldots, \delta_{q_2})$ tel que $0 < \delta < \cdots < \delta_{q_2} \leq D$ calculer le logarithme L_Δ et chercher dans la table de hachage s'il n'y a pas des éléments qui satisfont les conditions (5.1) ou (5.2). S'il y en a, sortir les multiples correspondants.

Si l'on met pour l'instant de côté la complexité du calcul des logarithmes discrets, cet algorithme a une complexité en temps de $O(D^{q_2})$ et en mémoire de $O(D^{q_1})$. En effet, si l'on met les logarithmes dans une table de hachage indexée par leurs bits de poids fort, il est possible de tester les conditions (5.1) et (5.2) en $O(1)$ pour des valeurs de D raisonnables.

On peut encore une fois choisir les paramètres de manière à équilibrer la complexité des deux phases, c'est-à-dire prendre $q_1 = \lfloor \frac{w-2}{2} \rfloor$ et $q_2 = \lceil \frac{w-2}{2} \rceil$. La complexité finale des divers algorithmes est indiquée sur la table suivante en fonction de la parité de w :

	$w = 2p$		$w = 2p+1$	
Algorithme	**temps**	**mémoire**	**temps**	**mémoire**
TMTO	D^p	D^{p-1}	D^p	D^p
CJM	D^p	$D^{\lceil p/2 \rceil}$	D^p	$D^{\lceil p/2 \rceil}$
LogTMTO	D^{p-1}	D^{p-1}	D^p	D^{p-1}

On voit donc que dans le cas ou w est impair, l'algorithme de Chose Joux et Mitton est toujours meilleur. Par contre, pour les petits poids pairs, 4 et éventuellement 6 on gagne quand même un facteur D en temps pour un algorithme qui reste avec une mémoire convenable. L'utilisation des logarithmes discrets peut donc permettre de calculer les multiples de poids 4 efficacement, modulo la complexité de calcul du logarithme.

Cet algorithme est donc très intéressant, car en pratique les logarithmes sont souvent calculables facilement et les équations de poids 4 sont très utiles pour certains types d'attaques. On en reparlera au chapitre suivant.

L'algorithme peut également s'optimiser un peu, car chaque multiple de degré au plus D et de poids w admet plusieurs décompositions comme somme d'un polynôme de poids $q_1 + 1$ et d'un de poids $q_2 + 1$. On a ainsi le résultat suivant qui se démontre facilement :

Lemme 5.8. Soit $p(X)$ un polynôme de poids $w = 2 + q_1 + q_2$ et de degré au plus D. Alors il existe un entier e et deux polynômes $a(X)$ et $b(X)$ de poids respectif q_1 et q_2 et de degré respectif au plus D et $Dq_2/(w-1)$ tel que $p(X) = 1 + a(X) + X^e(1 + b(X))$ ou tel que $p(X) = X^e(1 + a(X)) + 1 + b(X)$.

5.4 Calcul des logarithmes discrets

Intéressons-nous maintenant au calcul des logarithmes discrets dans le groupe multiplicatif du corps $\mathbf{F}_{2^l}$. C'est un problème qui a de nombreuses applications, notamment en cryptographie à clef publique et pour lequel il existe des algorithmes performants. L'algorithme de base qui marche sur n'importe quel groupe est l'algorithme "pas de bébé pas de géant" (Baby step giant step en anglais) découvert par Shanks [Sha71]. Lorsque le cardinal du groupe n'est pas premier, le calcul du logarithme discret peut se ramener à plusieurs calculs de logarithmes dans des sous-groupes plus petits, c'est l'algorithme de Pohlig-Hellman [PH78]. Dans le cas du groupe multiplicatif d'un corps fini, il existe des algorithmes plus évolués comme l'algorithme de Coppersmith [Cop84a, Cop84b].

Algorithme 5.9 (Baby step giant step)**.** On se place dans un groupe G engendré par g. L'algorithme prend en entrée x et calcule son Log, c'est-à-dire le i tel que $x = g^i$. Il utilise un paramètre T pour la taille des pas de géant qui est usuellement la racine carrée du cardinal du groupe.

1. [Baby step] Précalculer et stocker dans une table de hachage les g^i pour i allant de 0 jusqu'a $T-1$. On en profite pour calculer également la valeur de g^T.
2. [Giant step] Partir de x puis calculer les xg^{jT} jusqu'à ce que l'on tombe sur une valeur $xg^{jT} = g^i$ qui soit dans la table de l'étape 1.
3. [Log] Le logarithme de x est alors $i - jT$.

Il est important de noter que dans l'utilisation que nous en avons, nous allons calculer beaucoup de logarithmes. Heureusement, tous les algorithmes mentionnés plus haut peuvent bénéficier d'une étape de précalcul plus importante de manière à calculer ensuite des logarithmes plus rapidement.

Regardons par exemple l'algorithme de Pohlig-Hellman qui se ramène en fait à utiliser l'algorithme "pas de bébé pas de géant" sur chacun des sous-groupes de $\mathbf{F}_{2^l}^*$. Il est ainsi possible d'augmenter les pas de géant (en augmentant la mémoire requise) pour qu'un calcul de logarithme soit plus rapide.

En particulier, si l'entier $2^l - 1$ se factorise en petits nombres, il est possible de tabuler tous les logarithmes dans tous les sous-groupes de $\mathbf{F}_{2^l}^*$ de manière à rendre l'algorithme de Pohlig-Hellman très efficace. Cela correspond en fait à un pas de géant maximal et permet de calculer un logarithme en $O(1)$ une fois que les tables sont construites.

Cette approche est efficace pour de nombreuses longueurs l de LFSR. Ainsi elle peut être utilisée sans problème jusqu'à un l de 78 sauf pour $\{37, 41, 49, 59, 61, 62, 65, 67, 69, 71, 74, 77\}$. Elle marche également pour certaines longueurs plus importantes comme le montre la table 5.1.

l	53	96	110	156	210
Mémoire	439MB	510MB	1.7GB	940MB	210MB

TAB. 5.1 – Utilisation mémoire de l'algorithme de Pohlig-Hellman entièrement tabulé pour quelques valeurs de l telle que $2^l - 1$ soit friable.

Finalement, sauf pour certaines longueurs de LFSR, les logarithmes discrets se calculent plutôt efficacement. En tout cas, un logarithme se calcule en général beaucoup plus vite que $O(D)$ ce qui fait de l'algorithme qui les utilise le plus efficace pour la recherche de multiples de poids 4. De plus, l'implémentation est très aisée et peut donner lieu à quelques optimisations.

Pour les cas les plus intéressants ($w \in \{3, 4, 5\}$), on doit calculer des logarithmes de la forme $\mathrm{Log}(1+X^i)$. Ce logarithme est en fait le logarithme de Zech de i, et l'on peut exploiter certaines propriétés de ces derniers [Hub90] pour accélérer les calculs. En fait, pour chaque logarithme calculé, on en obtient $6l$ gratuitement. Bien sûr, ils ne nous sont pas tous utiles, mais il est néanmoins facile de diviser le temps de calcul par au moins 2.

Finalement, un autre cas qui apparaît vraiment en pratique et pour lequel le calcul de logarithme est facile est celui où l'on recherche des multiples de plusieurs LFSR à la fois. On se place alors dans le groupe produit des groupes multiplicatifs des corps finis associés à chaque LFSR. Calculer un logarithme se ramène alors via Pohlig-Hellman à un calcul de logarithme pour chaque sous-groupe, ce qui est usuellement rapide.

5.5 Résumé

Nous avons vu dans ce chapitre les diverses techniques pour calculer les multiples de poids faible d'un polynôme sur un corps fini. C'est un problème difficile, et la plupart des algorithmes n'exploitent pas vraiment la structure sous-jacente du corps pour arriver à leur fin.

Nous avons essayé de l'exploiter en utilisant les logarithmes discrets sur les corps finis, ce qui a conduit à quelques améliorations notamment pour le calcul des multiples de poids 4. Pour ces derniers, les autres algorithmes retournent tous les multiples jusqu'à un degré D fixé en $O(D^2)$ alors qu'avec les logarithmes discrets on s'en sort avec $O(D)$ calculs de logarithmes. Calculer des logarithmes peut sembler difficile, mais dans de nombreux cas pratiques ils se calculent très rapidement. Le gain en temps est alors conséquent, on a par exemple réussi pour $l = 53$ à calculer tous les multiples de poids 4 jusqu'à un degré de 2^{30} en quelques heures sur un ordinateur standard. Remarquons que l'on obtient des performances semblables pour toute valeur de l qui conduit à un calcul de logarithme en $O(1)$.

Chapitre 6

Une nouvelle attaque sur le registre filtré

Ce chapitre est consacré à une attaque de nature probabiliste sur le registre filtré et correspond à l'article [Did07]. On va utiliser les multiples de poids faible du polynôme générateur du LFSR qui se calculent comme on l'a expliqué dans la section précédente. L'attaque utilise essentiellement les multiples de poids 5, mais on verra aussi que les multiples de poids 4 sont très utiles pour distinguer la suite chiffrante d'une suite aléatoire. En particulier, ils peuvent servir à monter une attaque dans le cas où le système est constitué de plusieurs LFSR.

Le déroulement de l'attaque présentée ici n'est pas nouveau, mais l'analyse du biais statistique impliqué et son lien avec les propriétés de la fonction booléenne de filtrage mettent en lumière des éléments qui n'avaient jamais été publiés. Au final, on obtient une attaque des plus générique qui est certainement l'une des plus efficaces pour attaquer le registre filtré et ses variantes.

Le principe de cette attaque se rapproche des attaques par corrélation "vectorielles" [MH04, LZGB03, Lev04a, EJ04, GH05]. Le cœur de l'analyse est basé non pas sur un modèle de canal BSC comme pour beaucoup d'attaques par corrélation mais reprend les travaux de Sabine Leveiller [Lev04b] où le lien entre les relations de parité de poids faible et la fonction booléenne est complètement exploité. On obtient d'ailleurs des résultats qui nous semblent plus cohérents avec les expériences effectuées.

6.1 Présentation de l'attaque

Nous présentons ici le déroulement de l'attaque sur le registre filtré, nous montrerons en fin de section comment on peut l'adapter au cas de plusieurs LFSR filtrés.

Notre attaque utilise comme la plupart des attaques par corrélation les

multiples de poids faible du polynôme générateur du LFSR $g(X)$. Comme on l'a vu dans le chapitre précédent, chacun de ces multiples induit une équation de parité vérifiée par les bits de la suite produite par le LFSR et donc par les entrées de la fonction de filtrage f. Ainsi pour un multiple $p(X) = 1 + \sum_{i=1}^{w-1} X^{\delta_i}$ de poids w on a :

$$\mathbf{x}_t + \mathbf{x}_{t+\delta_1} + \cdots + \mathbf{x}_{t+\delta_{w-1}} = \mathbf{0} \quad \forall t \geq 0 \ . \tag{6.1}$$

Dans tout ce chapitre, nous allons supposer que pour un multiple donné et un point $\mathbf{x}_t$, les autres points $\mathbf{x}_{t+\delta_1}$ jusqu'à $\mathbf{x}_{t+\delta_{w-1}}$ prennent avec la même probabilité toutes les valeurs qui satisfont l'équation (6.1). C'est l'unique hypothèse sur laquelle est basée toute notre analyse. Elle nous semble justifiée par les bonnes propriétés statistiques d'un LFSR et surtout, comme on le verra plus loin, par le fait que les résultats expérimentaux sont très proches de ce que la théorie prévoit. Avec cette hypothèse, on définit :

$$P_{\mathbf{x}} \stackrel{\text{def}}{=} \Pr\left(f(\mathbf{x}_1) + \cdots + f(\mathbf{x}_{w-1}) = 0 \quad | \quad \sum_{i=1}^{w-1} \mathbf{x}_i = \mathbf{x} \right) \ .$$

C'est la probabilité pour un multiple donné de poids w que $z_{t+\delta_1} + \cdots + z_{t+\delta_{w-1}}$ soit égal à 0 sachant que $\mathbf{x}_t = \mathbf{x}$. Pour simplifier les notations, on a enlevé les δ_i qui de toute manière n'ont pas d'influence dans notre analyse. Le cœur de notre attaque est basé sur ces probabilités. Elles peuvent s'exprimer simplement comme on le verra dans la section suivante, et pour un w impair elles vont satisfaire deux propriétés intéressantes :

- $P_{\mathbf{0}}$ est la plus grande probabilité parmi les $P_{\mathbf{x}}$, elle est toujours plus grande que $1/2$.
- Si la fonction f a de bonnes propriétés d'autocorrélation alors il y a toujours un écart entre $P_{\mathbf{0}}$ et les autres $P_{\mathbf{x}}$.

Étant données ces propriétés, il devient facile de déduire ce que l'on va faire. En utilisant de nombreux multiples de $g(X)$, on sera capable d'avoir une bonne approximation des probabilités $P_{\mathbf{x}_t}$ associées à une position t. Si l'écart entre $P_{\mathbf{0}}$ et les $P_{\mathbf{x}}$ est assez grand (ce qui va dépendre du nombre de multiples à notre disposition) on sera alors capable de distinguer quelles positions t sont associées avec des $\mathbf{x}_t$ qui valent $\mathbf{0}$.

Chaque $\mathbf{x}_t$ égal à $\mathbf{0}$ nous indique que les m bits de la séquence $(s_t)_{t\geq 0}$ qui composent $\mathbf{x}_t$ sont égaux à 0. On obtient alors m équations linéaires sur l'état initial du LFSR qui sont données par les expressions linéaires des bits de $\mathbf{x}_t$ en fonction de $(s_0, \ldots, s_{l-1})$. En regroupant les équations de tous les $\mathbf{x}_t$ nuls ainsi détectés on obtient un système linéaire de rang au plus $l-1$ car à la fois l'état nul et le vrai état initial sont solutions. On espère ainsi, avec $\lceil l/m \rceil$ tels états $\mathbf{x}_t$ à $\mathbf{0}$, obtenir un système de rang $l-1$ dont l'unique solution non triviale est l'état initial du LFSR.

Le déroulement de l'attaque est décrit dans l'algorithme suivant qui utilise deux paramètres D et N dont on verra comment choisir la valeur plus loin :

Algorithme 6.1 (Détection des entrées de f à $\mathbf{0}$ [Did07])**.** Étant donnés deux paramètres D et N ainsi que $D+N$ bits de suite chiffrante produits par un LFSR filtré de polynôme générateur $g(X)$, cet algorithme essaie de retrouver l'état initial.

1. [Précalcul] Chercher tous les multiples de poids $2p+1$ de $g(X)$ et de degré au plus D.
2. [Approximation] Calculer une estimation de $P_{\mathbf{x}_t}$ pour les N premiers bits de la suite chiffrante. Pour une position donnée, il suffit de compter combien d'équations linaires associées aux multiples de $g(X)$ sont satisfaites par les bits de la suite chiffrante. Notons que parmi ces N bits, seuls ceux pour lesquels $z_t = f(\mathbf{0})$ sont à considérer.
3. [Supposition] On va supposer que les $\lceil l/m \rceil$ bits avec l'approximation de $P_{\mathbf{x}_t}$ la plus haute correspondent à des $\mathbf{x}_t$ tout à zéro.
4. [Fin] Résoudre le système linéaire induit par la connaissance de ces $\mathbf{x}_t$ en ces positions et retrouver l'état initial.

Une complexité détaillée sera donnée plus loin, mais en voici quand même une première idée. La meilleure complexité pour la première étape, qui est du précalcul, est donnée par l'algorithme de [CJM02] (voir le chapitre précédent). Elle est de $O(D^p)$ en temps et de $O(D^{p/2})$ en mémoire. La complexité de l'étape 2 est en N fois le nombre de multiples et requiert comme indiqué dans le préambule de l'algorithme $N+D$ bits de suite chiffrante. Les deux dernières étapes sont quant à elles négligeables dans la complexité finale. Il est possible dans l'étape 3 d'élargir le nombre de candidats et de résoudre plusieurs systèmes linéaires différents à l'étape 4 jusqu'à ce que l'on retrouve l'état initial. Il est en effet très facile de tester si l'on obtient le bon état initial ou pas.

Cas de plusieurs LFSR filtrés par une fonction booléenne Il existe dans ce cas une variante de cette attaque très efficace et dont la complexité ne dépend que du nombre de variables de la fonction de filtrage utilisée ! L'idée est de faire une recherche exhaustive sur un registre et d'utiliser des équations de parité de poids 4 satisfaites par tous les autres pour distinguer le bon état des mauvais.

Pour distinguer la suite chiffrante d'une suite aléatoire, on verra dans la section suivante que les équations de poids 4 sont un bon choix. En effet, sous notre hypothèse, elles sont toujours vérifiées par la suite chiffrante avec un biais positif plus grand que $1/2^{m+1}$ pour une fonction de m variables.

Sans rentrer dans les détails, il est alors possible de ne considérer que les équations qui, étant donnée la valeur du registre testé, sont telles que les 4 vecteurs d'entrée de la fonction de filtrage sont de somme nulle. Si l'état testé n'est pas le bon, il n'y a pas de raison particulière d'avoir un biais alors que dans le cas contraire, on est exactement dans le cas d'une équation linéaire de poids 4 avec des entrées aléatoires de somme nulle.

6.2 Calcul du biais

L'efficacité de notre attaque va essentiellement reposer sur l'écart entre $P_{\mathbf{0}}$ et les autres $P_{\mathbf{x}}$. Cet écart va être plutôt faible et représente le biais entre deux distributions binomiales que nous aurions à distinguer pour retrouver les états tout à 0. Pour cela, nous allons nous intéresser un peu plus en détail à la probabilité

$$P_{\mathbf{x}} \stackrel{\text{def}}{=} \Pr\left(f(\mathbf{x}_1) + \cdots + f(\mathbf{x}_{w-1}) = 0 \quad | \quad \sum_{i=1}^{w-1} \mathbf{x}_i = \mathbf{x} \right) \tag{6.2}$$

qui correspond à une équation de poids w. Nous l'utiliserons ensuite pour exprimer l'écart entre $P_{\mathbf{0}}$ et les autres $P_{\mathbf{x}}$. Commençons par introduire

$$\sigma_i(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{\mathbf{x}_1,\ldots,\mathbf{x}_i \in \mathbf{F}_2^m} (-1)^{f(\mathbf{x}_1)+\cdots+f(\mathbf{x}_i)+f(\mathbf{x}+\mathbf{x}_1+\cdots+\mathbf{x}_i)} .$$

Par définition, $\sigma_0(\mathbf{x})$ est simplement la fonction signe de f

$$\sigma_0(\mathbf{x}) = (-1)^{f(\mathbf{x})}$$

et la probabilité $P_{\mathbf{x}}$ pour une équation de poids w est directement reliée à $\sigma_{w-2}(\mathbf{x})$ par

$$P_{\mathbf{x}} = \frac{1}{2}\left(1 + \frac{1}{2^{(w-1)m}}\sigma_{w-2}(\mathbf{x})\right) . \tag{6.3}$$

En effet, la somme $\mathbf{x} + \mathbf{x}_1 + \cdots + \mathbf{x}_{w-2}$ correspond à $\mathbf{x}_{w-1}$ dans l'équation (6.2) car la somme de $\mathbf{x}_1$ à $\mathbf{x}_{w-1}$ doit être égale à $\mathbf{x}$. De plus, il est facile de démontrer la relation de récurrence suivante

$$\sigma_i(\mathbf{x}) = \sum_{\mathbf{y}\in\mathbf{F}_2^m} (-1)^{f(\mathbf{x}+\mathbf{y})}\sigma_{w-1}(\mathbf{y}) = \sigma_0 * \sigma_{i-1}$$

où $*$ est le produit de convolution défini dans la proposition 3.13. En utilisant les propriétés de la transformée de Walsh, on a alors $\widehat{\sigma_i}(\mathbf{u}) = \widehat{\sigma_0}(\mathbf{u})^{i+1}$ avec

$$\widehat{\sigma_0}(\mathbf{u}) = \sum_{\mathbf{x}\in\mathbf{F}_2^m} \sigma_0(\mathbf{x})(-1)^{\mathbf{u}\cdot\mathbf{x}}.$$

En utilisant maintenant la transformée de Walsh inverse on obtient

$$\sigma_i(\mathbf{x}) = \frac{1}{2^m}\sum_{\mathbf{u}\in\mathbf{F}_2^m} (-1)^{\mathbf{u}\cdot\mathbf{x}}\widehat{\sigma_1}(\mathbf{u})^{i+1} .$$

Ce qui donne

$$P_{\mathbf{x}} = \frac{1}{2}\left[1 + \sum_{\mathbf{u}\in\mathbf{F}_2^m} (-1)^{\mathbf{u}\cdot\mathbf{x}} \left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m}\right)^{w-1}\right] . \tag{6.4}$$

Ce résultat a déjà été observé par Sabine Leveiller dans sa thèse [Lev04b], nous en avons juste donné une preuve différente ici. Le lecteur peut également consulter l'article [LZGB03] où la même approche est utilisée.

Nous allons maintenant montrer que la probabilité $P_{\mathbf{0}}$ peut se distinguer assez bien des autres quand w est impair. Remarquons d'abord que pour un tel w, on a toujours une borne inférieure sur $P_{\mathbf{0}}$. Pour cela, notre modèle[1] est assez différent du canal BSC utilisé par la plupart des attaques par corrélation rapide. En effet, la probabilité qu'une équation soit vraie ne dépend pas de la non-linéarité de la fonction, même s'il y a des liens.

Lemme 6.2 (Valeur de ${P_{\mathbf{0}}}^2$.)**.** Avec les notations de cette section, on a toujours l'inégalité suivante pour un w de la forme $2p+1$:

$$P_{\mathbf{0}} \geq \frac{1}{2}\left(1 + \frac{1}{2^{m(p-1)}}\right) \ .$$

Il y a égalité dans le cas où f est une fonction courbe. $P_{\mathbf{0}}$ est aussi la probabilité qu'une équation de parité de poids $2p$ soit vraie sur les bits de la suite chiffrante, il suffit pour s'en convaincre de regarder (6.2).

Démonstration. Avec le w du lemme, l'équation (6.4) devient pour $P_{\mathbf{0}}$:

$$P_{\mathbf{0}} = \frac{1}{2}\left[1 + \sum_{\mathbf{u}\in\mathbf{F}_2^m}\left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m}\right)^{2p}\right] \ .$$

Pour le cas $p = 1$, on a directement grâce à l'égalité de Parseval (proposition 3.15) que $P_{\mathbf{0}} = 1$. Ce résultat indique que les équations de parité de poids 2 sont toujours vérifiées par la suite chiffrante. C'est trivial, mais cela confirme le bien fondé de notre modélisation. Pour les autres valeurs de p, le reste de la preuve découle directement du lemme 6.3 qui suit avec $n = 2^m$, $s = 1$ et les a_i égaux aux $\left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m}\right)^2$ pour tous les $\mathbf{u}$ de $\mathbf{F}_2^m$. □

Lemme 6.3 (Somme de puissance)**.** Étant donné n nombres réels positifs $(a_i)_{i=1\ldots n}$ de somme $s \stackrel{\text{def}}{=} \sum_{i=1}^n a_i$ et un entier $p > 0$ on a l'inégalité

$$\sum_{i=1}^{n} {a_i}^p \geq n\left(\frac{s}{n}\right)^p \ .$$

Il y a égalité quand tous les a_i sont égaux à s/n.

[1] L'idée d'utiliser un canal à "mémoire" comme dans la thèse de Sabine Leveiller est assez ancienne et se retrouve par exemple dans l'article [And94] sur la "fonction augmentée".

[2] Ce résultat n'est pas mentionné dans [Did07] car il n'est pas essentiel à l'attaque sur le registre filtré. Par contre, c'est lui qui nous indique que les équations de poids 4 font de très bon distingueurs. On le retrouve dans le rapport d'HDR de Anne Canteaut [Can06] mais il est déjà présent dans l'article [ZZ99].

Démonstration. Ce résultat est une conséquence presque directe de l'inégalité de Hölder :

$$\sum_{i=1}^{n} |a_i b_i| \leq \left(\sum_{i=1}^{n} |a_i|^p \right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} |b_i|^q \right)^{\frac{1}{q}}$$

où $(a_i)_{i=1\ldots n}, (b_i)_{i=1\ldots n}, p$ et q sont des réels avec en plus la contrainte $\frac{1}{p}+\frac{1}{q} = 1$. En l'appliquant pour les a_i positifs du lemme, les b_i tout à 1, le p du lemme et le q correspondant on obtient

$$\sum_{i=1}^{n} a_i \leq \left(\sum_{i=1}^{n} {a_i}^p \right)^{\frac{1}{p}} n^{\frac{1}{q}} \ .$$

Ce qui donne

$$\left(\sum_{i=1}^{n} {a_i}^p \right) \geq (s n^q)^p \ .$$

Comme $\frac{1}{q} = \frac{1-p}{p}$ on obtient bien

$$\left(\sum_{i=1}^{n} {a_i}^p \right) \geq s^p n^{pq} \geq n \left(\frac{s}{n} \right)^p \ .$$

□

Pour distinguer $P_{\mathbf{0}}$ des autres $P_{\mathbf{x}}$, on va s'intéresser à la distance minimale entre $P_{\mathbf{0}}$ et $P_{\mathbf{x}}$, c'est-à-dire $\min_{\mathbf{x}\neq\mathbf{0}} (P_{\mathbf{0}} - P_{\mathbf{x}})$. Commençons par définir Δ comme le minimum de $P_{\mathbf{0}} - P_{\mathbf{x}}$ quand $w = 3$. En utilisant (6.3) on a

$$\Delta \stackrel{\text{def}}{=} \frac{1}{2} \left[\frac{\sigma_1(\mathbf{0})}{2^m} - \max_{\mathbf{x}\neq\mathbf{0}} \left(\frac{\sigma_1(\mathbf{x})}{2^m} \right) \right] = \frac{1}{2} \left[1 - \max_{\mathbf{x}\neq\mathbf{0}} \left(\frac{\sigma_1(\mathbf{x})}{2^m} \right) \right] \ .$$

On a la deuxième égalité car $\sigma_1(\mathbf{0})$ est égal à 2^m, c'est encore l'égalité de Parseval que l'on a déjà utilisée dans le lemme sur $P_{\mathbf{0}}$. Si f a une bonne autocorrélation, alors ce Δ est très proche de $1/2$. En effet d'après (6.2) on obtient

$$\sigma_1(\mathbf{x}) = \sum_{\mathbf{u}} (-1)^{f(\mathbf{u})+f(\mathbf{u}+\mathbf{x})}$$

qui n'est rien d'autre qu'un coefficient d'autocorrélation et qui est censé être proche de 0. Notez que si ce n'est pas le cas, il y a toutes les raisons de penser que l'on pourra alors distinguer d'autres valeurs de $P_{\mathbf{x}}$ que $P_{\mathbf{0}}$, de plus il y a certainement d'autres attaques possibles.

Dans le cas plus général $w = 2p + 1$, en utilisant (6.4) on peut écrire la différence entre $P_{\mathbf{0}}$ et les $P_{\mathbf{x}}$ comme

$$\min_{\mathbf{x}\neq\mathbf{0}} (P_{\mathbf{0}} - P_{\mathbf{x}}) = \frac{1}{2} \min_{\mathbf{x}\neq\mathbf{0}} \left[\sum_{\mathbf{u}} \left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m} \right)^{2p} - \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{x}} \left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m} \right)^{2p} \right]$$

ce qui se simplifie en

$$\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}}) = \min_{\mathbf{x}\neq\mathbf{0}} \sum_{\mathbf{u}|\mathbf{u}\cdot\mathbf{x}=1} \left(\frac{\widehat{\sigma_0}(\mathbf{u})}{2^m}\right)^{2p} .$$

Remarquez que cette différence est toujours positive ce qui implique déjà que $P_{\mathbf{0}}$ est bien la plus grande de tous les $P_{\mathbf{x}}$. Dans le cas $p = 1$ ce minimum est par définition Δ et pour les autres p le lemme 6.3 nous montre que le pire cas est quand les $\widehat{\sigma_0}(\mathbf{u})$ sont tous égaux ce qui nous donne le lemme suivant.

Lemme 6.4 (Borne inférieure sur le biais [Did07])**.** Avec les notations de cette section, on obtient toujours l'inégalité suivante pour les équations de parité de poids $2p + 1$:

$$\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}}) \geq 2^{m-1}\left(\frac{\Delta}{2^{m-1}}\right)^p .$$

Démonstration. La preuve est une application directe du lemme 6.3 avec $n = 2^{m-1}$, $s = \Delta$ et les a_i égaux aux $\left(\frac{\widehat{\sigma_1}(\mathbf{u})}{2^m}\right)^2$ pour les $\mathbf{u}$ tel que $\mathbf{u}\cdot\mathbf{x} = 1$. □

Pour finir cette section, nous avons calculé le vrai biais pour 3 fonctions booléennes f_1, f_2 et f_3 de respectivement 8, 8 et 9 variables. Ces fonctions ont été choisies avec des propriétés cryptographiques correctes, leur coefficient de Fourier maximum est respectivement de 32, 24 et 48.

Fonctions	m	**Biais (pds 5)**	**Borne**	**Biais (pds 7)**	**Borne**
f_1	8	0.0039	0.002	0.000061	0.000008
f_2	8	0.0027	0.002	0.000021	0.000008
f_3	9	0.0014	0.001	0.000006	0.000002

TAB. 6.1 – Comparaisons de la borne théorique pour $\Delta = \frac{1}{2}$ avec le biais exact de certaines fonctions cryptographiques.

6.3 Complexité de l'attaque

Nous détaillons ici la complexité de l'attaque sur le registre filtré lorsque l'on utilise des multiples de poids $2p+1$. On va supposer que l'autocorrélation de la fonction de filtrage est bonne ce qui signifie pour nous un Δ proche de $1/2$ et un biais à détecter qui va être de l'ordre de

$$\text{biais} \simeq 2^{-(p-1)m-1} .$$

Pour le détecter nous aurons approximativement besoin d'autant d'équations que le carré de son inverse comme nous l'indique la borne de Chernoff (voir l'annexe B). Comme on regarde les multiples de poids $2p+1$ de $p(X)$ jusqu'au degré D, en utilisant l'heuristique 5.3 du chapitre précédent, on aura environ

$$\text{Nombre de multiples de degré au plus D} \simeq \binom{D}{2p}\frac{1}{2^l} \simeq \frac{D^{2p}}{(2p)!2^l} \ .$$

En regroupant les deux dernières équations, on devra donc choisir le degré D de telle manière que

$$\frac{D^{2p}}{(2p)!2^l} = 2^{2m(p-1)+2} \ .$$

Ce qui veut dire que nous devrons calculer les multiples de poids $2p+1$ jusqu'à un degré D tel que

$$\log_2 D = \frac{l}{2p} + m\left(1 - \frac{1}{p}\right) + \frac{1}{p} \ .$$

On a négligé le terme en $(2p)!$ ici, car p reste en pratique très faible. En plus, dans les algorithmes de calcul de multiples, si l'on s'arrange pour qu'un multiple ne puisse être obtenu que d'une seule manière, on va pouvoir gagner un facteur du même ordre sur les complexités données ci-dessous. En utilisant l'algorithme de [CJM02], la complexité pour trouver tous ces multiples est de D^p en temps et $D^{p/2}$ en mémoire. Remarquez que l'algorithme est complètement parallélisable sur un grand nombre de machines. On obtient donc finalement pour la complexité de la phase de précalcul

$$\log_2(\text{temps de précalcul}) = \frac{l}{2} + (p-1)m + 1,$$

$$\log_2(\text{mémoire pour le précalcul}) = \frac{l}{4} + (p-1)m/2 + 1/2 \ .$$

Pour la partie en ligne de l'attaque, on a besoin d'identifier de l'ordre de $\lceil \frac{l}{m} \rceil$ positions qui correspondent à un $\mathbf{x}$ nul. On aura donc besoin d'approximer $P_{\mathbf{x}}$ pour environ N bits de suite chiffrante avec

$$N = \left\lceil \frac{l}{m} \right\rceil 2^m \ .$$

En effet un $\mathbf{x}$ nul apparaît en moyenne tous les 2^m bits de suite chiffrante. On peut en fait gagner un facteur 2 car il n'est pas nécessaire de calculer l'approximation pour les positions t telles que $f(\mathbf{x}_t) \neq f(\mathbf{0})$. Pour chacun de ces N bits, on va calculer la valeur d'autant d'équations de parité que de multiples. En négligeant leur poids qui est très faible, on obtient une complexité pour la phase en ligne de

$$\log_2(\text{temps}/N) = 2 + 2m(p-1) \ .$$

C'est une complexité plutôt faible ! Quant à la mémoire on a juste besoin d'avoir accès à $N + D$ bits de suite chiffrante, c'est-à-dire en substance

$$\log_2(\text{longueur de la suite chiffrante}) = \frac{l}{2p} + m\left(1 - \frac{1}{p}\right) + \frac{1}{p} .$$

Pour avoir une idée des performances de cette attaque, comparons la au meilleur compromis "temps-mémoire-longueur de suite chiffrante" que l'on a donné au chapitre 4. Nous rappelons qu'une telle attaque suit la courbe $TM^2B^2 = 2^{2l}$ où T est le temps que prend l'attaque (sans le précalcul), M la mémoire utilisée et B la longueur de suite chiffrante dont on dispose. On avait vu qu'un bon compromis est obtenu pour $M = B = 2^{l/3}$, ce qui nous fait un temps de précalcul et un temps actuel en $O(2^{2l/3})$.

Par comparaison, avec des équations de poids 5 et un nombre de variables m de l'ordre de $l/8$ (ce qui semble normal en pratique) on a la complexité suivante : une complexité en ligne de $O(2^{l/4})$ et une mémoire de $O(2^{5l/16})$ pour une longueur de suite chiffrante nécessaire de $O(2^{5l/16})$ bits. C'est mieux que le compromis "temps-mémoire-longueur de suite chiffrante" précédent, d'autant que le temps pour calculer les multiples est de l'ordre de $O(2^{5l/8})$ ce qui est un peu mieux que $2^{2l/3}$. On remarquera également qu'en pratique, l est souvent plus grand que $8m$.

6.4 Résumé et performances

Nous venons de décrire une attaque sur le registre filtré qui utilise les multiples de poids impair (5 ou 7) pour détecter les entrées toutes à zéro de la fonction de filtrage. Il est ensuite facile d'en déduire la clef du système. Bien que cette attaque utilise des équations de poids impair, utiliser des équations de poids 4 permet de construire un très bon distingueur et peut être utile pour attaquer des systèmes à plusieurs LFSR.

Outre la longueur du registre, la complexité de l'attaque dépend essentiellement des deux paramètres que sont le nombre de variables de la fonction de filtrage et le maximum de son spectre d'autocorrélation. Pour les deux raisons qui suivent, l'attaque nous semble difficile à éviter et donc assez générique :

- Utiliser une fonction avec une mauvaise autocorrélation ouvrirait certainement la porte à d'autres attaques et permettrait peut-être une adaptation de notre approche en détectant d'autres antécédents que le vecteur nul. La fonction utilisée n'a donc a priori que peu d'influence si ce n'est son nombre de variables m.
- Il n'est pas possible d'avoir une fonction de filtrage avec un trop grand nombre de variables par rapport à la longueur l du registre, cela pour des raisons de performance mais également pour que les positions des entrées de f ne fragilisent pas le système.

La complexité de l'attaque a été analysée précisément en utilisant un modèle probabiliste dû à Sabine Leveiller qui nous semble pertinent. En effet, les performances réelles d'une implémentation de l'attaque qui sont données dans la table 6.2 sont très proches de ce que prévoit la théorie. Pour détecter le biais, le nombre théorique de multiples était ainsi de respectivement 65746, 137200 et 510000 pour chaque générateur. Cela nous donne en utilisant l'heuristique 5.3 sur le nombre de multiples de degré au plus D, un $\log_2 D$ théorique de 18.4, 20.16 et 21.14.

l	m	poids	$\log_2 D$	Nb de multiples(temps)	N	temps
53	8	5	18.6	100000(20min)	3200	10 sec
59	8	5	20.47	330000(1jour)	3200	30 sec
61	9	5	21	349034(2jours)	4000	1 min

TAB. 6.2 – Paramètres, temps de précalcul et temps de l'attaque en ligne sur plusieurs registres filtrés, les fonctions de filtrage utilisées sont respectivement f_1, f_2 et f_3. Les calculs ont été effectués sur un Pentium 4 cadencé à 3.6Ghz avec 2Mb de cache et 2Gb de mémoire.

Les deux premières attaques suivent exactement la méthode décrite dans ce chapitre. Nous avons utilisé quelques astuces supplémentaires pour le troisième registre pour ne pas passer trop de temps dans le précalcul des multiples. En doublant la taille de la suite chiffrante disponible, il est possible d'utiliser chaque multiple w fois (en décalant un multiple de l'une de ces 5 positions non nulles). Nous avons de plus passé plus de temps dans la dernière phase de l'attaque. Il y avait ainsi trois antécédents tout à zéro faux parmi les 10 positions qui maximisaient $P_{\mathbf{x}}$. Ces erreurs ont été facilement corrigées en testant tous les sous-ensembles de 7 positions parmi ces 10. Ces 7 positions suffisent car chacune d'elles induit 9 équations sur l'état initial et 7×9 est plus grand que la longueur du registre, 61.

Chapitre 7

Complexité linéaire

La complexité linéaire d'une suite binaire correspond à la longueur du plus petit LFSR qui peut l'engendrer. C'est une mesure intéressante de la complexité d'une suite donnée, une suite aléatoire ayant par exemple une complexité linéaire proche de la moitié de sa longueur. Nous nous intéressons dans ce chapitre à la complexité linéaire de la suite chiffrante produite par un registre filtré.

Nous commencerons par présenter l'algorithme de Berlekamp-Massey qui étant donnée une séquence permet de trouver sa complexité linéaire ainsi que le LFSR qui l'engendre. Nous étudierons ensuite une borne supérieure sur la complexité du LFSR filtré par une fonction de degré donné. Enfin, nous nous intéresserons à une attaque très récente et des plus efficaces.

7.1 L'algorithme de Berlekamp-Massey

L'algorithme de Berlekamp-Massey est un algorithme très important en informatique et qui a de nombreuses applications. Nous nous en servirons en particulier au chapitre 14 puisque c'est un élément clef de l'algorithme de Wiedemann. Il est introduit en 1968 par E. Berlekamp dans son livre [Ber68] comme algorithme de décodage pour des codes BCH. Mais c'est J.M. Massey qui un an plus tard a montré dans [Mas69] que l'on pouvait s'en servir pour calculer la complexité linéaire d'une suite et lui a donné sa forme actuelle. En fait, on peut montrer que cet algorithme est équivalent à l'algorithme d'Euclide, voir pour cela l'article de J.L. Dornstetter de 1987 [Dor87].

Étant donné une suite binaire $(s_t)_{t\geq 0}$ de longueur n, cet algorithme va nous permettre de retrouver le plus petit LFSR qui l'engendre et a fortiori la complexité linéaire de cette suite. Si $(s_t)_{t\geq 0}$ est de complexité linéaire l, l'algorithme a besoin de $2l$ bits pour être sûr de retrouver le polynôme minimal qui génère cette séquence. Cet algorithme fonctionne sur n'importe quel corps, mais nous ne le présenterons que sur $\mathbf{F}_2$ par souci de clarté.

L'algorithme est quadratique avec une complexité en $O(n^2)$ et utilise une

mémoire linéaire, en $O(n)$. Il existe en fait des versions sous-quadratiques directement déduites des versions sous-quadratiques de l'algorithme d'Euclide fondés sur la multiplication rapide de polynômes par le biais d'une transformée de Fourier discrète. La complexité de ces algorithmes est asymptotiquement en $O(n \log^2 n)$ mais ils ne deviennent intéressants que pour des très grandes longueurs. Dans cette thèse nous nous contenterons de la version quadratique de cet algorithme, le lecteur est invité à lire [GY79] pour plus de détails. Une bonne référence est également la thèse d'Emmanuel Thomé [Tho03].

Nous attirons l'attention sur le fait que l'algorithme de Berlekamp-Massey est usuellement présenté avec le calcul du polynôme de connexion qui est le polynôme réciproque du polynôme générateur. C'est d'ailleurs le cas dans l'article de Massey [Mas69]. Néanmoins, lorsque l'on travaille sur $\mathbf{F}_2$, utiliser le polynôme générateur facilite l'implémentation. Ce dernier est stocké sous la forme d'une séquence de bits et le passage au polynôme réciproque revient juste à changer le sens de lecture. Avec un polynôme générateur, savoir si l'équation de parité associée est vérifiée correspond à faire le XOR des deux séquences dans le même sens, ce qui peut se faire très efficacement avec les opérations vectorielles (sur 64 bits ou plus) des processeurs modernes. Nous utiliserons donc cette représentation ici.

L'algorithme va parcourir linéairement la suite $(s_t)_{t\geq 0}$ et construire le LFSR minimal qui engendre la séquence jusqu'au temps t. L'état initial de ce LFSR est donné par les premiers bits de la séquence, donc seul le polynôme générateur a besoin d'être calculé. Nous le noterons $g(X)$ et ses coefficients seront les c_i pour i variant de 0 à l, on aura toujours $c_l = 1$.

Pour passer à la position $t + 1$, si le LFSR génère encore la suite on ne fait rien. Dans le cas contraire, on parle "d'échec" et l'algorithme va corriger $g(X)$ en additionnant avec un décalage indiqué par la variable δ un polynôme générateur calculé précédemment. Ce dernier, noté $b(X)$, est en fait égal à la valeur de $g(X)$ avant correction lors du dernier échec. De manière plus formelle l'algorithme est le suivant :

Algorithme 7.1 (Berlekamp-Massey)**.** Étant donné une suite $(s_t)_{t\geq 0}$ de $\mathbf{F}_2$ et un entier n, l'algorithme calcule le polynôme générateur de degré minimal de la suite jusqu'à la position n.

1. [Initialisation] On commence avec $t \leftarrow 0$. $g(X) \leftarrow 1$ et son degré $l \leftarrow 0$. $b(X) \leftarrow 1$. On initialise également $\delta \leftarrow -1$, on aura toujours $\delta + t + 1 = 2l$.
2. [Fin ?] Si $t = n$ alors on a fini. Le polynôme minimal qui génère la suite est $g(X)$ et la complexité linéaire de la suite est l.
3. [Erreur ?] Si
$$\sum_{i=0}^{l} c_i s_{t+i} = 0$$
alors le polynôme $g(X)$ génère toujours la suite, aller en 5.

4. [Correction] Il y a deux cas à considérer :
   – Si $\delta > 0$ alors l ne change pas et $g(X) = g(X) + X^{\delta} b(X)$.
   – Sinon faire $\delta \leftarrow -\delta$. $g(X) \leftarrow b(X) + X^{\delta} g(X)$ et $b(X)$ prend l'ancienne valeur de $g(X)$. $l \leftarrow l + \delta$.
5. [Boucle] Faire $t \leftarrow t + 1$, $\delta \leftarrow \delta - 1$ et retourner en 2.

Il est assez facile de comprendre pourquoi à chaque fin de boucle $g(X)$ est bien un polynôme générateur de la suite. Regardons pour cela la figure 7.1.

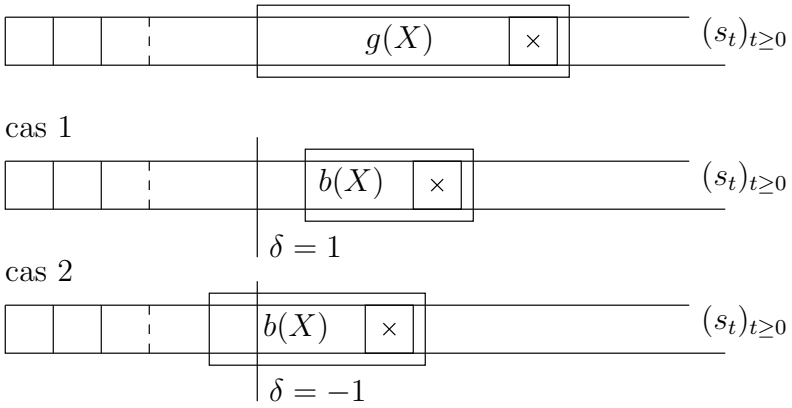

FIG. 7.1 – Illustration de l'algorithme de Berlekamp-Massey. Les fenêtres glissantes correspondent à l'application de l'équation de parité déduite du polynôme indiqué. Une croix est placée à la première position où l'équation n'est pas vérifiée.

L'idée quand on rencontre une erreur est de former le polynôme tel que l'équation de parité associée à la position de l'erreur corresponde à la somme de la fenêtre courante et de celles de la dernière erreur. En effet, pour ce nouveau polynôme l'équation de parité sera vérifiée au moins un bit de plus. Elle est vraie sur le dernier bit, car somme de deux équations fausses sur ce bit. Et elle l'est sur tous les bits d'avant, car c'était le cas pour les deux polynômes que l'on vient de sommer. Il y a deux cas à distinguer selon le signe de δ. Si δ est strictement positif, le nouveau polynôme est bien $g(X)+X^{\delta} b(X)$ et la complexité linéaire ne change pas. On ne change pas non plus $b(X)$, car le conserver tel quel aura toutes les chances de nous donner des polynômes de plus petit degré plus tard. Le deuxième cas intervient quand δ est négatif, le nouveau polynôme est alors donné par $X^{-\delta} g(X) + b(X)$, la complexité linéaire augmente et l'on change le polynôme $b(X)$ pour la même raison que l'on ne l'a pas changé précédemment. Dans le cas limite où δ est égal à 0, on peut changer le polynôme ou garder le même, cela ne changera rien au résultat.

La minimalité découle du lemme suivant qui correspond au théorème 1 de l'article de Massey [Mas69]. Nous ne le démontrerons pas ici.

Lemme 7.2. Soit l_{t-1} la longueur minimale d'un LFSR qui génère la suite $s_0, \ldots, s_{t-1}$ mais qui ne génère plus $s_0, \ldots, s_t$. Alors, la longueur minimale l_t du LFSR qui génère la suite $s_0, \ldots, s_t$ vérifie :

$$l_t \geq \max\left(t + 1 - l_{t-1}, l_{t-1}\right) \ .$$

On peut alors montrer que dans l'étape 4 de correction cette borne minimale est toujours atteinte. Pour cela il suffit de vérifier que l'on a toujours $\delta + t + 1 = 2l_{t-1}$ à l'entrée de la boucle. On finit par deux petites remarques.

Remarque 1 Le polynôme minimal d'une suite n'est pas toujours unique. En fait si $l < n/2$ on peut montrer qu'il y a unicité mais ce n'est en général pas vrai dans le cas contraire.

Remarque 2 Il arrive que le polynôme minimal soit "dégénéré", c'est-à-dire qu'il soit divisible par X. Cette notion sera assez importante dans notre étude de l'algorithme de Wiedemann que nous verrons au chapitre 14. Dans ce cas, si i est le plus grand entier tel que X^i divise $g(X)$, les i premiers bits du LFSR n'interviennent plus du tout dans la suite. En fait, après i pas, tout se passe comme si la suite était produite par un LFSR de longueur $l - i$ et de polynôme minimal $g(X)/X^i$. On peut aussi montrer que $g(X)$ et $b(X)$ ne sont jamais dégénérés en même temps ce qui est rassurant, car on peut toujours voir le début de la suite comme l'état initial d'un LFSR non dégénéré.

7.2 Complexité linéaire du registre filtré

Le registre filtré ayant une forte structure, il est possible d'obtenir une borne supérieure sur la complexité linéaire de la suite chiffrante produite. On va ainsi démontrer dans cette section le théorème suivant dû à E. Key dans son article de 1976 [Key76].

Théorème 7.3 (Complexité linéaire du registre filtré)**.** La complexité linéaire d'un registre de longueur l filtré par une fonction booléenne f de degré d est bornée par

$$L_d \stackrel{\text{def}}{=} 1 + \binom{l}{1} + \cdots + \binom{l}{d} \ .$$

De plus, la suite chiffrante est toujours engendrée linéairement par un polynôme de degré L_d qui ne dépend pas de f. Son expression est

$$g_d(X) \stackrel{\text{def}}{=} \prod_{J, |J| \leq d} \left(X - \alpha^J\right)$$

où α est une racine dans $\mathbf{F}_2^l$ du polynôme générateur $g(X)$ du LFSR et où le produit est pris sur les entiers dont le poids de Hamming de l'écriture en base 2 est d'au plus d.

Démonstration. Nous allons utiliser les résultats sur les récurrences linéaires présentés au début de la dernière section du chapitre 2.

Nous rappelons ainsi le théorème 2.12 qui montre que la suite $(s_t)_{t\geq 0}$ produite par un LFSR binaire de longueur l et de polynôme générateur irréductible $g(X)$ est de la forme $s_t = \mathrm{Tr}(\theta\alpha^t)$ pour un θ de $\mathbf{F}_2^l$. Regardons maintenant la séquence $(z_t)_{t\geq 0}$ produite si l'on filtre le tout par un monôme de degré d. On a alors un produit de la forme

$$z_t = s_{t+t_1} \dots s_{t+t_d} = \mathrm{Tr}(\theta\alpha^{t+t_1}) \dots \mathrm{Tr}(\theta\alpha^{t+t_d}) \ .$$

La valeur des positions t_i n'est pas importante ici, car du moment qu'elles sont différentes, z_t peut se mettre sous la forme

$$z_t = \sum_{J,|J|\leq d} \lambda_j \alpha^{tJ} \tag{7.1}$$

où les t_i sont "cachés" dans les λ. J est ici un entier et $|J|$ désigne le poids de Hamming de sa représentation en base 2. D'après le théorème 2.11, la séquence $(z_t)_{t\geq 0}$ vérifie donc la récurrence linéaire donnée par le polynôme

$$g_d(X) \stackrel{\text{def}}{=} \prod_{J,|J|\leq d} \left(X - \alpha^J\right) \ .$$

Ce polynôme est à coefficients dans $\mathbf{F}_2$ car pour une racine α^J donnée, tous les éléments conjugués de α^J sont aussi racines de $g_d(X)$. En effet, J, $2J$ et plus généralement $2^i J$ ont le même poids de Hamming modulo 2^l.

En comptant les J de poids de Hamming fixé on peut calculer le degré de $g_d(X)$ que nous noterons L_d :

$$L_d \stackrel{\text{def}}{=} \sum_{i=0}^{d} \binom{l}{i} \ .$$

La complexité linéaire de la séquence produite par un LFSR filtré par une fonction de degré au plus d est donc bornée supérieurement par L_d. Ceci est seulement une borne car $g_d(X)$ n'est pas forcément le polynôme minimal de la séquence. □

Il est en fait possible, dans l'expression de $g_d(X)$ de ne pas mettre le facteur $(X-1)$, ce terme n'est utile que si la fonction de filtrage a un coefficient $f_{\mathbf{0}}$ non nul. Or, comme f est connue, on peut prendre dans ce cas la négation bit à bit de toute la suite chiffrante et ne pas utiliser ce facteur. La complexité linéaire d'une suite filtrée peut donc être bornée par $L_d - 1$.

Ce polynôme va avoir une importance capitale dans la suite et il est intéressant de pouvoir le calculer explicitement. Une approche directe par multiplications successives nous donne une complexité de calcul en $O(L_d^2)$. Il

est possible de le calculer en $O(L_d \log^2 L_d)$ avec des transformées de Fourier discrètes comme explicité dans [HR04]. L'idée est simplement de regrouper les facteurs de $g_d(X)$ deux par deux, d'effectuer les $L_d/2$ multiplications de ces couples pour obtenir $L_d/2$ polynômes de degré 2, puis de regrouper ces derniers deux par deux et de continuer ainsi. Il existe des algorithmes rapides basés sur une transformée de Fourier discrète qui calculent la multiplication de deux polynômes de degré d en $O(d \log d)$. On peut alors borner la complexité du calcul par

$$L_d \left[\frac{1}{2} \log L_d + \frac{1}{4} 2 \log L_d + \cdots + \frac{1}{2^{\log L_d}} 2^{\log L_d - 1} \log L_d \right] \leq L_d \log^2 L_d \ .$$

7.3 L'attaque de Rønjom et Helleseth

On va voir dans cette section que si l'on dispose de L_d bits de suite chiffrante d'un LFSR filtré par une fonction de degré d, il est possible de retrouver l'état initial en $O(L_d)$ opérations. Ce résultat très joli a été montré récemment par Rønjom et Helleseth dans leur article [RH07b]. L'approche fonctionne sur n'importe quel corps (voir [RH07a]) mais nous resterons sur $\mathbf{F}_2$ pour plus de simplicité.

Tout d'abord, vu que la complexité linéaire de la suite chiffrante produite par un LFSR filtré par une fonction de degré d est d'au plus L_d, si l'on connaît L_d bits consécutifs de $(z_t)_{t \geq 0}$ alors on peut générer les autres. On parle souvent d'attaque de Berlekamp-Massey car on peut utiliser ce dernier algorithme pour calculer le polynôme générateur d'une telle séquence et s'en servir pour générer la suite.

Ici, un polynôme générateur étant connu, il est inutile d'appliquer cet algorithme. En plus nous rappelons que $g_d(X)$ est complètement indépendant de la fonction de filtrage. Il est donc très important que le degré d de la fonction de filtrage soit assez élevé pour qu'il soit inenvisageable qu'un attaquant ait accès à plus de L_d bits de suite chiffrante. Cela est d'autant plus vrai depuis [RH07b] car il est en fait possible avec L_d bits de suite chiffrante de retrouver l'état initial du LFSR sans beaucoup plus de calcul.

Pour réussir cela, l'idée de base est assez simple et tout repose sur le polynôme :

$$p(X) \stackrel{\text{def}}{=} g_d(X)/g(X) = \prod_{J, 1 < |J| \leq d} \left(X - \alpha^J\right) = \sum_{i=0}^{L_d - l - 1} p_i X^i \ .$$

On n'a pas tenu compte ici du terme $|J| = 0$, ce qui ne pose pas de problème comme on l'a remarqué dans la section précédente. On a également vu que l'on peut calculer les coefficients p_i de ce polynôme efficacement en

$O(L_d \log^2 L_d)$. On s'intéresse alors à la séquence

$$z_t^* \stackrel{\text{def}}{=} \sum_{i=0}^{L_d} p_i z_{i+t}$$

pour laquelle tous les termes en α^J pour $|J|$ strictement plus grands que 1 disparaissent dans la formule (7.1) car ils sont racines de $p(X)$. La suite $(z_t^*)_{t\geq 0}$ est donc engendrée par $g(X)$ et il existe un unique θ^* dans $\mathbf{F}_{2^l}$ tel que

$$z_t^* = \mathrm{Tr}(\theta^* \alpha^t) \ .$$

Nous supposerons pour l'instant que ce θ^* est non nul. La suite $(z_t^*)_{t\geq 0}$ est donc juste une suite décalée de la suite produite par le LFSR (en supposant $g(X)$ primitif). Si l'on arrive à calculer la valeur de ce décalage qui ne dépend que du système, alors en calculant l éléments consécutifs de $(z_t^*)_{t\geq 0}$ il sera facile de retrouver l'état initial.

L'approche retenue ici va revenir au même, on va chercher l'expression linéaire qui relie z_0^* à l'état initial $s_0, \ldots, s_{l-1}$. On pourra alors exprimer les l premiers z_t^* en fonction de l'état initial et une simple inversion du système linéaire nous donnera la clef. Notons que par unicité du θ^* (impliqué par la primitivité du LFSR), le système sera forcément de rang plein. Le but est donc de rechercher un polynôme linéaire Z^* en l variables $X_1, \ldots, X_l$ tel que

$$z_t^* = Z^*(s_t, \ldots, s_{t+l-1}) \ .$$

Pour déterminer cette fonction linéaire, Rønjom et Helleseth proposent la méthode suivante. Rappelons que S_t est le polynôme linéaire en $X_1, \ldots, X_l$ tel que $s_t = S_t(s_0, \ldots, s_{l-1})$. En utilisant maintenant la forme polynomiale F de l'ANF de f, on obtient alors par construction :

$$Z^*(X_1, \ldots, X_l) = \sum_{i=0}^{L_d - l - 1} p_i F(S_{i+t_1}, \ldots, S_{i+t_m}) \tag{7.2}$$

le calcul s'effectuant dans l'anneau des polynômes à l variables sur $\mathbf{F}_2$ où l'on réduit le degré en chaque variable à 1, c'est-à-dire dans l'anneau quotienté par $X_1^2 - X_1$, ..., $X_l^2 - X_l$. Comme on va évaluer ces polynômes sur $\mathbf{F}_2$, il est en effet inutile de garder des puissances d'une variable plus élevées que 1. Bien que de prime abord la partie droite de l'équation (7.2) soit de degré d, on vient de voir qu'elle est en fait linéaire ! L'idée est donc pour chaque i tel que p_i vaut 1, de calculer uniquement la partie linéaire de $f(\mathbf{x}_i)$ en fonction des bits de l'état initial, c'est-à-dire la partie linéaire de $F(S_{i+t_1}, \ldots, S_{i+t_m})$. Cela est assez facile car si l'on note $S_t = \sum_{i=j}^{l} a_j(t) X_j$ alors X_j est dans la partie linéaire de $F(S_{i+t_1}, \ldots, S_{i+t_m})$ si et seulement si $f(a_j(i+t_1), \ldots, a_j(i+t_m)) = 1$. Le calcul de Z^* se fait donc en $O(lL_d)$ évaluations de f.

Une autre méthode peut être plus simple à expliquer est de partir d'un état initial connu, de calculer $(z_t)_{t\geq 0}$ jusqu'au temps L_d, puis de calculer les l premiers termes de $(z_t^*)_{t\geq 0}$. On peut alors trouver Z^* en résolvant le système linéaire d'inconnues les coefficients de Z^* donnés par les l premières équations :

$$Z^*(S_t(s_0,\ldots,s_{l-1}),\ldots,S_{t+l-1}(s_0,\ldots,s_{l-1})) = z_t^* \quad \text{pour} \quad t \in [0, l-1] \ .$$

La complexité est asymptotiquement la même.

Une fois la fonction linéaire Z^* connue, il suffit comme expliqué plus haut de résoudre le système linéaire d'inconnues $X_1,\ldots,X_l$ donné par les équations

$$Z^*(S_t,\ldots,S_{t+l-1}) = z_t^* \quad \text{pour} \quad t \in [0, l-1] \ .$$

Le bit s_i de l'état initial sera alors donné par la valeur de l'inconnue X_{i+1}. La complexité finale de cette attaque est dominée par le calcul des l premiers termes de $(z_t^*)_{t\geq 0}$ qui se fait en $O(lL_d)$. En pratique, grâce aux instructions vectorielles des processeurs modernes, on peut compter avec un coût unitaire des opérations de XOR manipulant $O(1)$ mots de taille $O(l)$. Cela nous donne une complexité linéaire de $O(L_d)$ pour mener l'attaque, plus la phase de précalcul du polynôme $p(X)$ qui se fait en $O(L_d \log^2 L_d)$.

Le seul problème de cette attaque est le cas où Z^* est le polynôme nul, c'est-à-dire quand θ^* est nul. Heureusement, cela a peu de chance d'arriver et indiquerait en particulier une complexité linéaire d'au plus $L_d - l$. On se reportera à l'article [RH07b] pour plus de détails et un début de solution qui repose sur l'utilisation du polynôme $g_d(X)/g_2(X)$. On obtient ainsi une séquence z_t^* qui dépend de manière quadratique de l'état initial et on peut mener une attaque similaire dont la résolution du système final, quadratique, demande plus d'effort pour être résolu. Récemment, deux articles ont apporté une solution plus efficace à ce problème, [RGH07, Riz07].

7.4 Résumé

La complexité linéaire est une mesure importante de la qualité cryptographique d'une suite chiffrante. Il est possible de la calculer à l'aide de l'algorithme de Berlekamp-Massey. Pour un LFSR de longueur l filtré par une fonction de degré d, la complexité linéaire de la suite chiffrante $(z_t)_{t\geq 0}$ est bornée par L_d avec :

$$L_d \stackrel{\text{def}}{=} \sum_{i=0}^{d} \binom{l}{i} \ .$$

De plus il existe toujours un polynôme générateur de degré L_d indépendant de la fonction de filtrage, son expression est donnée par

$$g_d(X) = \prod_{J,|J|\leq d} \left(X - \alpha^J\right)$$

ou α est une racine dans $\mathbf{F}_{2^{L}_{d}}$ du polynôme générateur du LFSR $g(X)$.

Il est donc possible si l'on dispose de L_d bits consécutifs de $(z_t)_{t\geq 0}$ de générer la suite de la séquence avec une complexité de $O(L_d)$ par bit. En fait, Rønjom et Helleseth ont montré qu'il est possible de retrouver l'état initial du LFSR avec la même complexité linéaire en L_d.

En conclusion, il est nécessaire que le degré d de la fonction de filtrage soit assez important pour qu'un attaquant ne puisse avoir accès à L_d bits consécutifs de suite chiffrante. Souvent, les concepteurs de chiffrement à flot imposent une limite sur le nombre de bits de la suite chiffrante utilisables avant réinitialisation de l'IV. Une valeur cohérente pourrait être autour de $L_d/2$ pour éviter les attaques décrites ici qui sont très efficaces.

Chapitre 8

Les attaques algébriques sur le registre filtré

Nous décrivons ici ce que l'on appelle les attaques algébriques sur le registre filtré. L'attaque présentée dans le chapitre précédent est également de nature algébrique, mais dans la littérature, ce sont plutôt les attaques que nous allons voir maintenant qui sont désignées par ce terme.

Nous commençons par exposer le principe de ces attaques qui remonte aux travaux de Claude Shannon et son article de 1949 [Sha49a]. Un telle attaque se ramène en fait à la résolution d'un système algébrique sur un corps fini et l'on en profitera pour faire un rapide tour d'horizon des différentes méthodes pour obtenir des solutions.

Malgré l'ancienneté de cette approche, ce n'est que récemment que ces attaques ont connu un regain d'intérêt avec le développement de nouvelles techniques qui peuvent en faire des attaques très efficaces. Nous décrirons ainsi l'attaque algébrique de Courtois et Meier [CM03], puis une version dite rapide due à Courtois [Cou03] qui repose sur des algorithmes un peu plus évolués.

8.1 Une idée qui date de Shannon

L'idée derrière les attaques algébriques remonte à Shannon et à son fameux article de 1949 [Sha49a] où il explique que casser un bon chiffrement doit nécessiter "as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type". C'est en effet le cas dans la plupart des cryptosystèmes, où retrouver la clef peut se ramener à résoudre un gros système d'équations algébriques. En général, le problème décisionnel sous-jacent est un problème NP-complet et en pratique, les paramètres utilisés en cryptographie conduisent à des systèmes algèbrique complètement hors de portée des machines et des algorithmes actuels.

Pour le registre filtré par exemple, en notant $X_1,\ldots,X_l$ les bits inconnus

de l'état initial, il est facile d'obtenir un tel système algébrique. Rappelons que l'on a introduit dans la section 2.2 le polynôme linéaire $S_t(X_1, \ldots, X_l)$ en ces variables tel que $s_t = S_t(s_0, \ldots, s_{l-1})$ la suite $(s_t)_{t\geq 0}$ étant la suite produite par le LFSR interne de longueur l et d'état initial $(s_0, \ldots, s_{l-1})$. Maintenant, en utilisant la forme polynomiale F de l'ANF de f, on voit que chaque bit connu de la suite chiffrante z_t nous donne une équation algébrique de degré d (le même que f) :

$$F(S_{t+t_1}, \ldots, S_{t+t_m}) = z_t \quad \forall t \geq 0 \ . \tag{8.1}$$

Comme on va évaluer ces polynômes multivariés sur $\mathbf{F}_2$, les degrés plus élevés que 1 en une variable sont inutiles. On peut donc faire le calcul sur l'anneau des polynômes à l variables quotienté par $X_1^2 - X_1, \ldots, X_l^2 - X_l$. Avec un grand nombre de bits de la suite chiffrante connus, on obtient donc un système algébrique fortement surdéterminé de degré d et dont la solution est la clef du système. La solution a de grandes chances d'être unique car deux états initiaux différents ont très peu de chance de donner les mêmes bits de la suite chiffrante utilisée pour écrire le système. Cela est même fortement à éviter pour un système de chiffrement.

Résoudre un système algébrique de degré élevé sur un corps fini est bien sûr un problème difficile. Les méthodes les plus efficaces à ce jour se ramènent à un calcul de bases de Gröbner. Ce calcul peut être fait de manière performante grâce aux algorithmes F4 et F5 de Jean-Charles Faugère [Fau99, Fau02]. Citons aussi l'algorithme FGLM [FGLM93] qui permet de calculer efficacement une base de Gröbner si l'on en dispose d'une pour un autre ordre sur les monômes.

Malheureusement, il est difficile d'évaluer la complexité des algorithmes de calcul de base de Gröbner. Il y a des résultats récents sur la complexité de F4 et F5 [BFSY04, BFSY05] mais ils ne s'appliquent que sous certaines hypothèses que l'on est bien loin de pouvoir vérifier pour les systèmes issus de la cryptographie.

D'autres approches qui sont en fait des raffinements de la méthode de linéarisation que l'on va voir plus loin ont été proposées. Il s'agit de l'algorithme XL décrit dans [CKPS00] mais dont la complexité s'avère moins bonne que celle de F4 [AFI+04, Die04] et également de l'algorithme XSL [CP02] mais dont la complexité est sujette à discussion [CL05].

Mentionnons également des approches récentes qui ne marchent que sur $\mathbf{F}_2$ et qui utilisent des algorithmes conçus pour résoudre des problèmes SAT de satisfaction d'ensemble d'expressions booléennes, ici utilisés sur celles que doivent satisfaire les bits de clefs [BCJ07]. En substance, ces algorithmes font une sorte de recherche exhaustive de manière intelligente et peuvent s'avérer efficaces selon le système à résoudre.

Finalement, si l'on veut avoir une borne précise de la complexité de résolution d'un système algébrique on a recours à la méthode de linéarisation

qui est beaucoup moins subtile que les précédentes. C'est une méthode qui ne fonctionne que quand on dispose de vraiment beaucoup d'équations (ce qui est notre cas) et qui consiste simplement à linéariser le système en rajoutant une nouvelle inconnue par monôme de degré au plus d. Comme le nombre de monômes de degré i est donné par $\binom{l}{i}$, on obtient un système linéaire de

$$1 + \binom{l}{1} + \cdots + \binom{l}{d}$$

inconnues. Remarquons que le nombre de tels monômes est exactement le même que la complexité linéaire L_d de la suite chiffrante. Si l'on veut avoir une chance de retrouver la clef il nous faut donc au moins L_d équations ce qui nous donne un système que l'on sait résoudre en $O(L_d^3)$. On voit que l'attaque décrite dans le chapitre précédent est donc beaucoup plus efficace.

Néanmoins, Faugère et Ars se sont aperçus qu'en faisant tourner leur algorithme de base de Gröbner sur certains registres filtrés, le système algébrique que nous venons de décrire pouvait se résoudre très rapidement ! Ces résultats sont décrits dans [FA03]. En fait, le système obtenu est très loin d'un système aléatoire qui serait complètement impossible à résoudre. Il existe une sorte de structure que les algorithmes basés sur les bases de Gröbner arrivent à exploiter.

C'est en 2003 également que Courtois et Meier dans [CM03] ont trouvé une astuce pour construire un système algébrique de plus faible degré que celui décrit ici. Cela apporte un certain éclairage sur la structure du système induit par les équations (8.1) et permet d'expliquer en partie les performances des bases de Gröbner. C'est leur approche que nous allons étudier dans la section suivante.

Mentionnons également un article récent de Fischer et Meier [FM07] qui tente de dégager d'autres caractéristiques qui peuvent rendre ces systèmes faciles à résoudre. Cet article explique ainsi pourquoi certains systèmes étudiés dans [FA03] ont pu être résolus rapidement alors que l'approche de Courtois et Meier ne s'applique pas dans ces cas.

8.2 L'attaque algébrique standard

Nous décrivons ici l'attaque algébrique exposée dans l'article de Courtois et Meier [CM03]. C'est aussi ici que nous définissons la notion d'immunité algébrique d'une fonction booléenne qui va occuper une grande partie de cette thèse.

L'attaque algébrique standard repose essentiellement sur la notion d'annulateur d'une fonction booléenne.

Définition 8.1 (Annulateur)**.** Soit f une fonction booléenne à m variables. Un annulateur de f est une autre fonction booléenne à m variables g telle

que le produit point par point de f et g soit nul :

$$f(\mathbf{x})g(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \mathbf{F}_2^m \ .$$

On notera G pour le polynôme qui correspond à l'ANF de la fonction g. L'idée de l'attaque est alors basée sur le fait qu'aux temps t où z_t vaut 1, on peut maintenant remplacer l'équation (8.1) qui s'écrit ici

$$F(S_{t+t_1}, \ldots, S_{t+t_m}) = 1$$

par

$$G(S_{t+t_1}, \ldots, S_{t+t_m}) = 0 \ .$$

S'il existe des annulateurs de f non nuls et de faible degré, on tombe donc sur un système algébrique de plus petit degré. Plus précisément, s'il existe un annulateur non nul de degré r alors on peut écrire un système algébrique de degré

$$1 + \binom{l}{1} + \cdots + \binom{l}{r} \ .$$

Il s'agit juste de la dimension de l'espace des fonctions de degré au plus r sur l variables, elle est égale à la complexité linéaire L_r d'un registre de longueur l filtré par une fonction de degré r. Pour retrouver l'état initial il suffit maintenant de résoudre ce système. En utilisant la méthode de linéarisation, on a besoin de $O(L_r)$ bits de suite chiffrante et cela nous donne une complexité en $O(L_r^3)$ pour retrouver l'état initial du registre.

En conclusion, si f admet des annulateurs de faible degré, le registre filtré peut être très vulnérable. C'est ce qui est arrivé pour les chiffrements à flot Toyocrypt et LILI-128 comme expliqué dans l'article de Courtois et Meier [CM03].

Remarquons qu'aux points où z_t vaut 0, on peut de manière tout à fait similaire utiliser un annulateur de la fonction $1 + f$ qui prend exactement les valeurs opposées à celles de f. C'est ce qui a conduit Willi Meier, Enes Pasalic et Claude Carlet à introduire la notion d'immunité algébrique d'une fonction booléenne dans l'article [MPC04].

Définition 8.2 (Immunité algébrique)**.** L'immunité algébrique d'une fonction booléenne f à m variables est définie comme le plus petit entier r tel que f ou $1 + f$ admette un annulateur non nul de degré r.

Le calcul de la valeur de l'immunité algébrique occupe une grande partie de cette thèse. Nous démontrerons plus loin au chapitre 10 que l'on a la propriété suivante :

Proposition 8.3 (Immunité maximale)**.** Une fonction booléenne sur m variables a une immunité algébrique d'au plus $\lceil \frac{m}{2} \rceil$.

Il apparaîtra au chapitre 11 qu'une fonction booléenne aléatoire a presque toujours une immunité algébrique proche de l'optimale. Comme souvent les fonctions aléatoires ont de bonnes propriétés, sauf qu'au delà d'un certain nombre de variables elles sont généralement trop complexes à évaluer ! En pratique, il sera donc important de vérifier qu'une fonction n'a pas une immunité algébrique trop faible comme cela a été le cas pour les deux chiffrements à flot cités plus haut qui utilisaient des fonctions avec une ANF creuse.

Du moment que m est assez petit devant la longueur l du registre, cette attaque algébrique peut devenir tout à fait compétitive avec celle de Rønjom et Helleseth du chapitre précédent. En effet, pour r et d petit devant l, on peut approcher L_r par l^r et L_d par l^d (voir l'annexe B). Si l'immunité algébrique est trois fois plus faible que le degré de la fonction, on obtient des attaques où l'ordre de grandeur de la complexité est semblable.

Il existe également un avantage indéniable de l'attaque algébrique que nous venons de décrire qui est bien sûr la longueur de suite chiffrante nécessaire. En effet, il est difficilement envisageable de donner à un attaquant accès à autant de bits que la complexité linéaire de la suite chiffrante. Cela est déjà beaucoup moins vrai avec les attaques algébriques où la longueur de suite chiffrante nécessaire est souvent beaucoup moins importante. On peut également remarquer que si une fonction admet un grand nombre d'annulateurs, on a besoin d'encore moins de bits de suite chiffrante.

8.3 L'attaque algébrique rapide

Nous allons voir maintenant que même si une fonction n'admet pas d'annulateur de faible degré, il est possible qu'elle admette d'autres types de relations exploitables. L'attaque que nous allons présenter ici porte le nom d'attaque algébrique rapide et a été présentée par Nicolas Courtois dans son article [Cou03]. On parle de version rapide car les idées utilisées sont semblables et sa complexité ne peut pas être pire que la version standard de l'attaque. Cette attaque a été plus tard confirmée et améliorée par Armknecht dans [Arm04] et par Hawkes et Rose dans [HR04].

L'attaque est basée sur l'existence de deux fonctions booléennes à m variables, g et h telle que l'on ait

$$f(\mathbf{x})g(\mathbf{x}) = h(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbf{F}_2^m \ . \tag{8.2}$$

On peut alors introduire une notion d'immunité pour les attaques rapides :

Définition 8.4 (immunité pour les attaques rapide)**.** Dans le cas des attaques algébriques rapides, l'immunité d'une fonction booléenne f dépend de l'existence d'un couple de degré (r, e) avec $r < e$ tel qu'il existe deux fonctions non triviales g et h de degré respectif r et e qui satisfont (8.2).

La relation $fg = h$ implique en fait que h est un annulateur de $1 + f$ car elle implique que quand f vaut 0, h vaut nécessairement 0. L'idée est de s'autoriser maintenant un degré un peu plus grand pour h que l'immunité algébrique de f en espérant qu'il existe des g de plus faible degré. On peut remarquer que faire le raisonnement avec $1 + f$ ne change rien car on a alors $(1 + f)g = h$ donc $fg = h + g$ et comme le degré de g est plus petit que celui de h, on retombe sur la même relation avec un h différent.

En utilisant la forme polynomiale de l'ANF de g et h (notée respectivement G et H) la relation (8.2) donne pour chaque valeur de z_t connue l'équation algébrique suivante (où les inconnues $X_1, \ldots, X_l$ correspondent aux bits de l'état initial à rechercher) :

$$H(S_{t+t_1}, \ldots, S_{t+t_m}) = z_t G(S_{t+t_1}, \ldots, S_{t+t_m}) \ . \tag{8.3}$$

L'idée est alors de combiner plusieurs relations de ce type de manière à faire disparaître les monômes de degré plus grand que r. Ces termes proviennent de H et l'on peut voir qu'il n'est pas nécessaire de connaître les z_t pour chercher de telle combinaisons, cette étape est donc du précalcul. Plus formellement, on va rechercher des coefficients p_i pour i allant de 0 à un entier T le plus petit possible tels que tous les termes de degré strictement plus grand que r disparaissent dans la somme :

$$H^*(X_1, \ldots, X_l) \stackrel{\text{def}}{=} \sum_{i=0}^{T} p_i H(S_{i+t_1}, \ldots, S_{i+t_m}) \ . \tag{8.4}$$

On en comprendra la raison plus loin, mais il est commode de voir ces coefficients comme ceux d'un polynôme $p(X)$ de degré T. Remarquons qu'un simple argument de dimension nous indique qu'il existe toujours un T plus petit que $L_e - L_r$. Une fois ces coefficients p_i calculés, la même combinaison linéaire va éliminer les termes de degré plus grand que r pour tous les décalages dans le temps. En effet, si l'on remplace pour un t fixé chaque X_i par $S_{t+i-1}(X_1, \ldots, X_l)$, alors on obtient un polynôme de même degré qui correspond à l'application de la formule (8.4) à partir du temps t. C'est-à-dire qu'après calcul de la somme des équations (8.3) selon les coefficients de $p(X)$, on obtient de nouvelles équations de degré r :

$$H^*(S_t, \ldots, S_{t+l-1}) = G_t^*(X_1, \ldots, X_l) \quad \forall t \geq 0 \ . \tag{8.5}$$

avec G_t^* défini par

$$G_t^*(X_1, \ldots, X_l) \stackrel{\text{def}}{=} \sum_{i=0}^{T} p_i z_{t+i} G(S_{t+i+t_1}, \ldots, S_{t+i+t_m}) \ .$$

Ce dernier polynôme dépend de la suite chiffrante et ne peut pas être précalculé. On obtient donc une nouvelle attaque qui s'effectue en quatre étapes :

1. Calculer les fonctions g et h.
2. Trouver un polynôme $p(X)$ tel que H^* soit de degré au plus r.
3. Calculer les G_t^*.
4. Résoudre le nouveau système algébrique (8.5) et retrouver l'état initial.

Nous verrons dans la section suivante qu'en utilisant de l'algorithmique assez évoluée, la complexité de cette attaque est en général dominée par la dernière étape, c'est-à-dire qu'elle est dominée par la résolution du système linéaire final de degré r. Cette complexité est de $O(L_r^3)$ en utilisant la méthode de linéarisation. Pour écrire ce système, on a besoin de $O(T + L_r)$ bits de suite chiffrante, ce qui est du même ordre que $O(L_e)$.

8.4 Aspect algorithmique de l'attaque rapide

Chacune des 4 étapes de l'attaque algébrique rapide nécessite en fait beaucoup de calcul. Nous verrons comment traiter le point 1 plus tard dans cette thèse à partir du chapitre 12.

Pour le point 2, Courtois a proposé deux idées dans son article [Cou03]. La première est d'utiliser simplement un pivot de Gauss pour retrouver les coefficients de la relation de degré minimal et donne une complexité en $(L_e - L_r)^3$ ce qui est équivalent à $O(L_e^3)$. D'après Courtois, cette méthode présente l'avantage de pouvoir s'appliquer même si l'on ne dispose pas de bits consécutifs de la suite chiffrante. En revanche, ce n'est alors plus du tout du précalcul, et l'on ne peut pas faire glisser la relation dans le temps non plus. La complexité est donc nettement plus grande que celle d'une attaque algébrique normale.

La deuxième idée, beaucoup plus efficace, se rapproche de ce que l'on a vu au chapitre précédent sur la complexité linéaire. Le problème est en effet parfaitement équivalent à trouver la complexité linéaire et le polynôme générateur de la suite $(h'(\mathbf{x}_t))_{t\geq 0}$ où h' est une fonction pour laquelle on n'a gardé que les monômes de h de degré strictement plus grand que r. On peut alors prendre comme $p(X)$ de la section précédente ce polynôme. En effet, si l'on utilise les coefficients de ce polynôme dans (8.4) pour construire H^* alors tous les termes de degré plus grand que r disparaîtront.

Courtois a proposé dans [CM03] d'appliquer l'algorithme de Berlekamp-Massey pour trouver ce polynôme générateur. Pour cela, il suffit de partir d'un état initial quelconque, de calculer la suite filtrée et d'appliquer Berlekamp-Massey. Cela fonctionne très bien quand le LFSR est de polynôme générateur primitif, mais dans le cas contraire, on ne peut pas être sûr que l'état initial choisi arbitrairement soit dans la même classe que celui de la clef secrète recherchée. Cela peut être un problème, car dans les systèmes réels il est possible qu'il y ait plusieurs LFSR derrière la fonction de filtrage, ce qui donne une relation linéaire qui n'est pas issue d'un po-

lynôme primitif. Ce problème est discuté en détail dans l'article d'Armknecht [Arm04] où il y apporte une solution efficace.

Enfin, toujours pour cette étape 2, Hawkes et Roses ont proposé une troisième approche dans [HR04]. En fait, si l'on se rappelle du contenu du chapitre précédent, on connaît une expression exacte d'un polynôme générateur de la suite $(h'(\mathbf{x}_t))_{t\geq 0}$, c'est $p_e(X)$. La complexité linéaire réelle de la suite est inférieure à L_e car bornée par $L_e - L_r$, mais cela ne pose pas de gros problèmes. L'attaque nécessite juste un peu plus de bits de la suite chiffrante mais en contrepartie on a $H^* = 0$. De plus, ce polynôme n'a aucune raison d'annuler G_t^* car la suite $(f(\mathbf{x}_t)g(\mathbf{x}_t))_{t\geq 0}$ a a priori une complexité linéaire bien supérieure.

Une fois le polynôme $p(X)$ trouvé, il reste à calculer les G_t^* ce qui peut être l'étape la plus coûteuse. Courtois avait estimé la complexité de cette phase à $O(TL_r)$ qui est en fait uniquement la complexité pour calculer un seul G_t^*. Il faut en fait en calculer L_r pour pouvoir ensuite appliquer la méthode de linéarisation au système algébrique final de l'étape 4. La complexité de l'étape 3 est donc en $O(TL_r^2)$ avec un algorithme trivial. Mais en y réfléchissant, on voit que dans cet algorithme, beaucoup de calculs sont effectués plusieurs fois. Toujours dans l'article [HR04], les auteurs ont montré comment faire ce calcul efficacement à l'aide d'une transformée de Fourier discrète. La complexité est alors ramenée à $O(TL_r \log L_r)$.

8.5 Résumé

Nous avons vu dans cette section comment ramener la recherche de l'état initial du registre filtré à la résolution d'un système algébrique.

Si la fonction de filtrage est de degré d, on peut ainsi retrouver l'état initial en résolvant un système algébrique de degré d. Pour cela, on a besoin d'environ L_d bits de suite chiffrante pour obtenir une solution par la méthode de linéarisation. La complexité est par contre en $O(L_d^3)$ ce qui est bien plus mauvais que l'attaque de Rønjom et Helleseth. Il existe des techniques plus efficaces pour résoudre de tels systèmes algébriques, mais leur complexité est mal connue.

Si la fonction de filtrage admet ce que l'on appelle des annulateurs de faible degré, il est par contre possible de réduire fortement cette complexité. Si de tels annulateurs ont un degré r, on peut alors remplacer L_d par L_r dans la discussion ci-dessus.

Pour mesurer la résistance du registre filtré face à cette attaque, la notion d'immunité algébrique d'une fonction booléenne f a été introduite. C'est le degré minimum pour lequel f ou $1 + f$ admet un annulateur non trivial.

Une version dite rapide qui améliore l'attaque algébrique standard peut s'appliquer s'il existe un couple (r, e) avec $r < e$ et deux fonctions g et h de degré respectif r et e qui satisfont $fg = h$.

Chapitre 9

Les codes de Reed-Muller

Les questions que nous allons traiter par la suite sont intimement liées à la famille des codes de Reed-Muller qui sont eux même intimement liés aux fonctions booléennes. Ces codes qui ont en fait été découverts par Muller [Mul54] furent la première classe non triviale de codes capables de corriger des erreurs multiples. Reed [Ree54] en a donné un algorithme de décodage très performant par vote majoritaire qui permet de les décoder jusqu'à la moitié de leur distance minimale sur le canal BSC. Il en a aussi donné une description très claire qui a été adoptée par la plupart des chercheurs par la suite.

Depuis, ces codes ont fait l'objet de nombreuses recherches et sont sans doute la famille de codes la mieux connue même si de nombreuses questions à leur sujet sont encore ouvertes. Et comme le dit si bien Massey dans "The ubiquity of Reed-Muller Codes" [Mas01] si l'on creuse profondément dans presque tous les problèmes algébriques de la théorie des codes ou de la cryptographie, on retrouve ces codes qui gisent au fond. Une bonne référence sur les codes de Reed-Muller est le livre de MacWilliams et Sloane [MS77] ou l'article d'Assmus [Ass92].

Nous donnons dans ce chapitre une description de ces codes et de leurs propriétés qui nous ont été utiles dans cette thèse. On mentionnera également quelques aspects algorithmiques de leur encodage et décodage qui nous seront utiles par la suite.

9.1 Définition et premières propriétés

Lorsque l'on a vu les fonctions booléennes, la définition des codes de Reed-Muller est des plus simples et tient en une phrase :

Définition 9.1 (Codes de Reed-Muller)**.** Le code de Reed-Muller d'ordre r sur m variables est l'espace vectoriel des vecteurs de valeurs des fonctions booléennes sur m variables et de degré au plus r.

Pour abréger, on notera un tel code RM(r, m). Si l'on se rappelle du chapitre 3, on voit qu'il s'agit d'un code de longueur 2^m, longueur que nous noterons n dans ce chapitre. Nous supposerons ici que les points de $\mathbf{F}_2^m$ sont énumérés par l'ordre usuel défini dans la première section du chapitre 3. La dimension de RM(r, m) est donc la même que celle des fonctions de degré au plus r sur m variables :

Lemme 9.2 (Dimension)**.** La dimension du code RM(r, m) est

$$k \stackrel{\text{def}}{=} \sum_{i=0}^{r} \binom{m}{i} .$$

Il suffit pour s'en convaincre de se rappeler qu'une base des fonctions booléennes de degré au plus r est formée par les fonctions monômes de degré au plus r. Une simple énumération de ces monômes par degré nous donne alors cette dimension.

Cette base est celle qui sert usuellement à définir la matrice génératrice du code RM(r, m) que nous supposerons toujours sous la forme suivante :

Définition 9.3 (Matrice génératrice)**.** La matrice génératrice du code RM(r, m) que nous noterons M est la matrice dont les lignes correspondent à des vecteurs de valeurs de fonctions monômes de degré au plus r. Ces monômes seront classés de haut en bas en utilisant l'ordre usuel sur $\mathbf{F}_2^m$ définie dans le chapitre 3.

0000	1	1111111111111111
0001	X_1	0101010101010101
0010	X_2	0011001100110011
0011	X_2X_1	0001000100010001
0100	X_3	0000111100001111
0101	X_3X_1	0000010100000101
0110	X_3X_2	0000001100000011
1000	X_4	0000000011111111
1001	X_4X_1	0000000001010101
1010	X_4X_2	0000000000110011
1100	X_4X_3	0000000000001111

FIG. 9.1 – Matrice génératrice de RM$(2, 4)$ donnée avec le monôme correspondant à chaque ligne. Ce dernier est donné d'abord sous la forme du nombre de 4 bits associé pour bien faire ressortir l'ordre, puis sous la forme polynomiale.

Un exemple de telle matrice est donnée sur la figure 9.1. Cette matrice génératrice n'est pas sous la forme systématique, mais elle présente

l'avantage que l'encodage d'un mot revient à calculer le vecteur des valeurs d'une fonction booléenne de degré au plus r représentée par ses k coefficients d'ANF non nuls. Rappelons que pour encoder k bits d'information d'un vecteur $\mathbf{x}$ en un mot de code de longueur 2^m il suffit de calculer $\mathbf{x}M$. Ici, la forte structure de la matrice génératrice facilite l'encodage et le décodage de $\mathrm{RM}(r, m)$ comme on le verra plus loin.

Une première propriété particulièrement importante des codes de Reed-Muller est leur structure récursive qui est connue sous le nom de décomposition $(u, u + v)$:

Proposition 9.4 (Décomposition (u,u+v))**.** Les codes de Reed-Muller vérifient la relation de récurrence suivante :

$$\mathrm{RM}(r+1, m+1) = \{u|u+v, \quad \text{avec } u \in \mathrm{RM}(r+1, m) \text{ et } v \in \mathrm{RM}(r, m)\} \ .$$

Le symbole | désigne la concaténation de séquences binaires.

Démonstration. La preuve provient directement de la représentation à l'aide de l'ANF d'une fonction booléenne. Pour une fonction f de $\mathrm{RM}(r+1, m+1)$ on peut ainsi écrire :

$$F(X_1, \ldots, X_{m+1}) = U(X_1, \ldots, X_m) + X_{m+1}V(X_1, \ldots, X_m)$$

où U et V sont des polynômes à m variables. Il suffit pour cela de regrouper les termes de l'ANF qui font intervenir X_{m+1}. On sait de plus que le degré de U est d'au plus $r+1$ et que celui de V est d'au plus r. Aux deux polynômes U et V correspondent bien des fonctions booléennes de m variables qui appartiennent aux codes de la proposition. De plus, le vecteur des valeurs de f s'écrit bien $u|u+v$ avec l'ordre choisi sur les monômes. □

Cette décomposition récursive est particulièrement utile pour démontrer des résultats sur les codes de Reed-Muller ou même pour les décoder comme dans l'algorithme de Dumer que l'on mentionnera plus loin ou dans l'algorithme du chapitre 13.

Une illustration de l'utilisation de cette décomposition récursive est donnée dans la démonstration du lemme suivant qui nous sera utile plus loin. Remarquons d'abord que $\mathrm{RM}(m, m)$ contient tous les vecteurs de longueur 2^m et que $\mathrm{RM}(0, m)$ est le code à répétition de longueur 2^m.

Lemme 9.5 (Le code $\mathrm{RM}(m-1, m)$)**.** Le code $\mathrm{RM}(m-1, m)$ est formé de tous les vecteurs de 2^m bits de poids pair.

Démonstration. Pour commencer, c'est vrai pour $\mathrm{RM}(0, 1)$ qui contient les deux mots 00 et 11. En utilisant maintenant la décomposition $(u, u+v)$ on voit d'abord que la partie (u, u) est de poids pair. Quand à v c'est un mot de $\mathrm{RM}(m-2, m-1)$ et l'on peut donc montrer le résultat par récurrence. □

Une autre application de la décomposition $(u, u+v)$ peut être le calcul de la distance minimale des codes de Reed-Muller que nous noterons dans toute cette thèse d_1 :

Proposition 9.6 (Distance minimale)**.** La distance minimale du code $\mathrm{RM}(r,m)$ est

$$d_1 \stackrel{\text{def}}{=} 2^{m-r} \ .$$

Démonstration. Le résultat peut se montrer par récurrence sur m en utilisant la décomposition $(u, u+v)$, voir [MS77], chapitre 13 théorème 3. On en verra également une démonstration à la section suivante grâce aux équations qui vont nous servir au décodage majoritaire. □

9.2 Autres propriétés

Après la longueur, la dimension et la distance minimale, la propriété la plus importante d'un code est peut être sa distribution de poids, c'est-à-dire le nombre de mot de codes pour tous les poids de Hamming possibles. Malheureusement, pour les codes de Reed-Muller cette distribution est inconnue dans le cas général et n'est en fait connue que pour très peu de paramètre.

On la connaît pour les codes d'ordre 0, 1, 2, leur codes duaux (voir la définition plus loin) obtenus pour $r = m-1$, $r = m-2$, $r = m-3$, pour le code $\mathrm{RM}(m,m)$, pour les petits codes avec m inférieur ou égal à 8, pour le code $\mathrm{RM}(3,9)$, le code $\mathrm{RM}(3,10)$ et les codes duaux de ces deux derniers. On connaît également la distribution de poids exacte jusqu'à 2.5 fois la distance minimale depuis les travaux de Kazumi, Tokura et Asumi [KT70] et [KTA74].

Au chapitre 11 on verra que l'on aurait bien aimé connaître cette distribution pour analyser le comportement de $\mathrm{RM}(r,m)$ sur le canal à effacements. Nous avons quand même pu trouver une solution en utilisant les distances généralisées d'un code linéaire qui sont connues pour les codes de Reed-Muller.

Définition 9.7 (Distances généralisées)**.** Les distances de Hamming généralisées d'un code linéaire $\mathcal{C}$ de dimension k sont définies pour $i \in [1,k]$ par

$$d_i \stackrel{\text{def}}{=} \min_{v \in V_i}(|\mathrm{supp}(v)|)$$

où V_i est l'ensemble de tous les sous-codes de dimension i de $\mathcal{C}$. On peut montrer que les distances généralisées sont strictement croissantes.

Proposition 9.8 (Distances généralisées des Reed-Muller)**.** Considérons le code $\mathrm{RM}(r,m)$ de dimension k et l'ordre usuel sur l'ensemble $\{\mathbf{x} \in \mathbf{F}_2^m, |\mathbf{x}| \leq r\}$, on numérote ainsi les éléments de cet ensemble de $\mathbf{x}_1$ à $\mathbf{x}_k$. À un $\mathbf{x}_i$ on rappelle que l'on peut associer un nombre n_i dans $[0, 2^m-1]$ comme expliqué

au début du chapitre 3. Les distances de Hamming généralisées de $\mathrm{RM}(r,m)$ s'expriment alors par :

$$d_i = n - n_{k-i} \ .$$

Démonstration. Voir l'article de Wei [Wei91]. □

Pour construire des fonctions booléennes d'immunité algébrique maximale, nous aurons également besoin de la notion de code dual.

Définition 9.9 (Code dual)**.** On appelle code dual d'un code $\mathcal{C}$ de longueur n l'espace linéaire des mots de longueur n dont le produit scalaire avec tous les mots de $\mathcal{C}$ est nul. Il s'agit donc de la définition classique du dual d'un espace vectoriel.

Proposition 9.10 (Dual des Reed-Muller)**.** Le code dual de $\mathrm{RM}(r,m)$ est le code $\mathrm{RM}(m-r-1,m)$. En particulier, les codes $\mathrm{RM}(r,2r+1)$ sont des codes auto-duaux.

Démonstration. Le produit scalaire de deux mots de codes de Reed-Muller n'est rien d'autre que la parité du produit des deux fonctions booléennes associées. Ici, le produit d'un mot de $\mathrm{RM}(r,m)$ avec un de $\mathrm{RM}(m-r-1,m)$ est de degré au plus $m-1$, il est donc de parité nulle avec le résultat 9.5. Le code $\mathrm{RM}(m-r-1,m)$ est donc inclut dans le dual de $\mathrm{RM}(r,m)$. Pour l'égalité, il suffit de regarder les dimensions, on a pour la dimension du second code

$$\sum_{i=0}^{m-r-1} \binom{m}{i} = \sum_{i=m}^{r+1} \binom{m}{i} = 2^m - k \tag{9.1}$$

d'où le résultat. □

Les codes de Reed-Muller vérifient de nombreuses autres propriétés dont on ne se servira pas ici. Nous mentionnons quand même deux d'entre elles pour montrer l'étendue et la complexité de la structure de ces codes. La première est particulièrement intéressante car elle fait ressortir certains liens avec les LFSR :

Proposition 9.11 (Sous-code cyclique)**.** Si on enlève le bit qui correspond à l'image de $\mathbf{0}$ alors les codes de Reed-Muller $\mathrm{RM}(r,m)$ sont équivalents à des codes cycliques.

Démonstration. Il existe plusieurs explications de ce résultat, mais pour nous, le moyen le plus naturel de le voir est de considérer la liste des images d'une fonction booléenne avec l'ordre donné par un LFSR primitif ! En effet en considérant un LFSR primitif de longueur m, celui-ci décrit de manière cyclique tous les éléments non nuls de $\mathbf{F}_2^m$. Si l'on regarde le décalé d'un mot de code associé à une fonction booléenne f, on peut voir que c'est aussi un mot de code donné par la fonction d'ANF $F(S_1,\ldots,S_m)$ qui est de même degré que f. □

Cette vision des choses est très intéressante, par exemple l'équation de parité minimale n'est rien d'autre que le polynôme $p_r(X)$ qui engendre linéairement la suite produite par le LFSR filtré par n'importe quelle fonction de degré r. C'est d'ailleurs un autre moyen de montrer que la complexité linéaire du registre de longueur m filtré par une fonction de degré r est k.

La deuxième propriété que nous voulons mentionner est le fait que les codes de Reed-Muller sont également des codes géométriques [MS77, Ass92]. Cette représentation permet de formuler certaines propriétés de manière particulièrement élégante. La géométrie euclidienne de dimension m sur $\mathbf{F}_2$ contient 2^m points dont les coordonnées correspondent aux vecteurs de m bits. Si on enlève le point tout à 0, on parle alors de géométrie projective.

Tout sous-ensemble S peut être associé à un vecteur d'incidence de longueur 2^m qui contient des 1 sur les points de S et des 0 ailleurs. Les codes de Reed-Muller peuvent être alors vus comme les vecteurs d'incidence de sous-ensembles de ces géométries.

Un des résultats les plus intéressants avec cette vision des choses est la caractérisation des mots de poids minimum de $\mathrm{RM}(r, m)$:

Proposition 9.12 (Mots de poids minimum)**.** Les mots de poids minimum dans $\mathrm{RM}(r, m)$ sont exactement les vecteurs d'incidence des sous-espaces affine de dimension $m - r$ de la géométrie euclidienne de dimension m sur $\mathbf{F}_2$. On peut également montrer que ces mots engendrent linéairement tout le code.

Démonstration. Voir le MacWilliams Sloane [MS77] chapitre 13, paragraphe 4, page 379. □

9.3 Quelques algorithmes utiles

Nous présentons maintenant quelques algorithmes autour des codes de Reed-Muller dont on s'est servi dans nos travaux. Le premier est utile pour encoder efficacement k bits d'information en un mot de code. Les autres concernent quant à eux le décodage des codes de Reed-Muller.

Génération efficace de mots de code

On a vu que sous la forme présentée ici, encoder un mot de $\mathrm{RM}(r, m)$ (c'est-à-dire passer d'un message de k bits à un mot de code de 2^m bits) revient à calculer le vecteur de valeurs d'une fonction booléenne de degré au plus r donnée comme la liste de ses k coefficients d'ANF non nuls.

Cette opération peut se faire efficacement grâce à une transformée de Möbius dite rapide (voir l'annexe A). Pouvoir encoder rapidement un code correcteur peut être très important selon l'application que l'on veut en faire. De plus, nous verrons au chapitre 14 qu'il existe un algorithme plus efficace

que l'algorithme trivial pour décoder sur le canal à effacements les codes qui s'encodent rapidement.

Vote majoritaire

Sur le canal BSC, il existe une procédure efficace de décodage des codes de Reed-Muller jusqu'à la moitié de leur distance minimale. Elle est connue sous le nom de décodage par vote majoritaire et est basée sur le lemme suivant :

Lemme 9.13 (Équations pour le décodage majoritaire)**.** Soit f une fonction booléenne et $\mathbf{u}$ un monôme de même degré que f. Alors pour tout $\mathbf{x}$ qui contient $\mathbf{u}$

$$\sum_{\mathbf{y} \subseteq \mathbf{u}} f(\mathbf{x} + \mathbf{y}) = f_{\mathbf{u}} \ .$$

Démonstration. Commençons par regarder ce qui se passe pour le terme $f_{\mathbf{u}}\mathbf{x}^{\mathbf{u}}$ de l'ANF de f. Dans la somme, seul le terme avec $\mathbf{y}$ égal à 0 peut être non nul, terme qui vaut exactement $f_{\mathbf{u}}$.

Ensuite, comme $\mathbf{u}$ est de même degré que f, tous les autres monômes présents dans l'ANF de f ont nécessairement une intersection avec $\mathbf{u}$ qui est strictement incluse dans $\mathbf{u}$. Le nombre d'éléments à 1 dans la somme est alors nécessairement pair, la somme est donc toujours nulle. □

Il est maintenant possible grâce aux équations du lemme 9.13 de décoder les codes de Reed-Muller sur le canal BSC jusqu'à strictement moins que $d_1/2$ erreurs. Pour cela, on note que le coefficient d'un monôme de plus haut degré r de l'ANF de la fonction associée au mot de code peut être obtenu par 2^{m-r} équations, une pour chaque valeur possible de $\mathbf{x}$ qui contient $\mathbf{u}$. Pour décider si un tel monôme apparaît dans l'ANF du mot émis, on choisit alors la valeur dominantes de ces équations, d'où le terme de vote majoritaire. Une fois tous les monômes de degré r retrouvés, on peut alors passer aux monômes de degré $r-1$ et ainsi de suite. S'il y a strictement moins de $2^{m-r}/2$ erreurs, on voit que le vote majoritaire nous donnera toujours la bonne réponse. Mais la distance minimale d_1 de $\mathrm{RM}(r, m)$ n'est rien d'autre que 2^{m-r}, d'où le résultat.

Remarquons que les équations du lemme 9.13 nous permettent d'avoir une borne sur la distance minimale de $\mathrm{RM}(r, m)$. En effet, si un mot de code est non nul, il existe un monôme non nul dans l'ANF de la fonction associée. Et si ce monôme est de degré d, on dispose alors de 2^{m-d} équations de support disjoint qui valent 1, la distance minimale est donc au moins de 2^{m-r}. Il est ensuite facile de vérifier que cette distance est atteinte par n'importe quel monôme de degré r par exemple.

Décodage des codes d'ordre 1

Pour les codes d'ordre 1, il existe un algorithme de décodage très efficace basé sur la transformée de Walsh. En fait, décoder les codes de Reed-Muller d'ordre 1 revient exactement à calculer la non-linéarité d'une fonction booléenne. On a déjà vu au chapitre 3 que ce calcul peut se faire de manière efficace grâce à la transformée de Walsh.

Ce qui est plus intéressant, c'est qu'un algorithme similaire est applicable pour tous les codes binaires et linéaires, nous reprenons plus ou moins ici la description qui en est faite dans [CJM02] et [LV04]. On suppose le code de dimension k, de longueur n et de matrice génératrice M qui est donc une matrice $k \times n$. On note M_i la i-ième colonne de M et l'on suppose que l'on a reçu le vecteur $y_1, \ldots, y_n$. L'idée est alors d'introduire la fonction W de $\mathbf{F}_2^k$ dans les entiers telle que :

$$W(\mathbf{x}) = \begin{cases} (-1)^{y_t} & \text{si } \exists i \mid M_i = \mathbf{x} \\ 0 & \text{sinon} \end{cases}$$

Remarquons que si le code est de dimension k, tous les M_i sont nécessairement différents et W est bien définie. La fonction W effectue ainsi une sorte de permutation sur le mot de code reçu et le plonge dans un vecteur de longueur 2^k. La transformée de Walsh de cette fonction au point u est alors

$$\widehat{W}(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbf{F}_2^k} W(\mathbf{x})(-1)^{\mathbf{u}\cdot\mathbf{x}} = \sum_{i=1}^{n} (-1)^{y_i + \mathbf{u}\cdot M_i} \ .$$

Or $\mathbf{u} \cdot M_i$ n'est rien d'autre que la valeur du i-ième bit du mot de code qui correspond aux bits d'informations codés dans $\mathbf{u}$. On voit alors que l'on a accès avec une seule transformée de Walsh à la distance du mot reçu à tous les mots de code possibles. Un code linéaire de dimension k peut donc toujours se décoder en $O(k2^k)$ via une transformée de Walsh rapide. L'algorithme trivial qui consiste à essayer tous les mots du code étant lui en $O(n2^k)$ mais utilise seulement une mémoire en $O(n)$ et non en $O(2^k)$.

En particulier, lors d'une attaque par corrélation où l'on recherche l'état interne d'un des registres internes, cela revient à décoder le code produit par le LFSR au maximum de vraisemblance ce qui peut se faire avec cette technique.

En très grande longueur mentionnons les travaux de Cedric Tavernier [KT04, KT06] qui prolongent ceux de Goldreich et Levin [GL89] avec un algorithme qui essaye de retrouver le mot le plus proche sans regarder toutes les positions. Cet algorithme est très utile en cryptographie pour trouver de bonnes approximations linéaires qui peuvent être exploitées par divers types d'attaques. Il peut par exemple servir pour effectuer une cryptanalyse de plusieurs tours du DES [Tav04] où le nombre de variables impliquées est beaucoup trop grand, ne serait-ce que pour calculer le vecteur des valeurs de la fonction.

Autres algorithmes

Pour finir ce rapide tour d'horizon sur les algorithmes de décodage des codes de Reed-Muller, mentionnons quelques travaux récents de théorie des codes sur le décodage efficace des codes d'ordre supérieur. Décoder de tels codes a également des applications en cryptographie, par exemple pour les codes d'ordre 2, on obtient la meilleure approximation quadratique d'une fonction booléenne.

Pour l'ordre 2, justement, il existe un algorithme de nature algébrique dû à Sidel'nikov et Pershakov [SP92] assez efficace. Il existe aussi une toute autre approche due à Ilya Dumer [Dum04, Dum06] qui fonctionne également pour les ordres supérieurs. L'idée est ici d'utiliser la décomposition récursive des Reed-Muller et des techniques qui appartiennent plutôt au monde du décodage à informations pondérées pour arriver à ses fins.

Chapitre 10

Immunité algébrique et code de Reed-Muller

Nous allons voir maintenant que la notion d'immunité algébrique est intimement liée aux codes de Reed-Muller et à leur comportement sur le canal à effacements. On commencera par décrire en détail ce canal avant d'explorer dans une seconde section le lien avec l'immunité algébrique. On finira par une section sur les fonctions d'immunité algébrique maximale.

10.1 Le canal à effacements

Le canal à effacements est un canal tel qu'après transmission d'un mot de code, certaines positions sont "effacées". C'est-à-dire que, contrairement au canal binaire symétrique, la position des erreurs (les effacements) est connue, on ne sait juste pas quel symbole il y avait à cette place.

Usuellement, on suppose qu'un symbole a une probabilité p donnée d'être effacé. Ici en revanche, nous supposerons fixé à $n-w$ le nombre de positions effacées, n étant la longueur du code et w le nombre de positions connues. On définit donc la probabilité d'erreur de la manière suivante :

Définition 10.1 (Probabilité d'erreur)**.** Nous désignerons par probabilité d'erreur et noterons P_{err} la probabilité que le décodage ne soit pas unique en supposant que les $n-w$ positions effacées soient choisies de manière uniforme.

On peut remarquer que la probabilité d'erreur $P_{\text{err}}(p)$ du cas classique de la théorie des codes où la probabilité d'effacements par symbole est p se déduit facilement de notre probabilité d'erreur $P_{\text{err}}(w)$ par :

$$P_{\text{err}}(p) = \sum_{w=0}^{n} \binom{n}{w} p^{n-w}(1-p)^{w} P_{\text{err}}(w) \ .$$

Dans tous les cas, pour un code linéaire sur le canal à effacements, cette probabilité d'erreur est indépendante du mot transmis et ne dépend que de la position des erreurs que l'on va désigner par *motif d'effacements* :

Définition 10.2 (Motif d'effacements)**.** On peut coder les positions effacées comme des 1 dans un mot binaire de longueur n, on parle alors de motif d'effacements. Pour un motif d'effacements fixé, on notera $\mathcal{I}$ l'ensemble des mots du code dont le support est inclus dans ce motif. $\mathcal{I}$ est un sous-espace vectoriel et l'ensemble des mots de code compatibles avec le mot reçu est égal à $c + \mathcal{I}$ où c est le mot de code envoyé.

La probabilité d'erreur est donc exactement la probabilité que $\mathcal{I}$ ne soit pas réduit au mot nul.

Décoder un code linéaire au maximum de vraisemblance sur le canal à effacements est beaucoup plus simple que dans le cas du canal binaire symétrique. Il suffit en effet de résoudre un simple système linéaire. Ainsi, chaque position connue nous donne une équation linéaire que doivent satisfaire les k bits d'information qui définissent le mot transmis. Cette équation est donnée, pour une position non effacée d'indice i, par la i-ème colonne M_i de la matrice génératrice M du code. Plus formellement, si la position i du mot reçu n'est pas effacée et vaut y_i, on a l'équation suivante sur les k bits d'information $b_1, \ldots, b_k$ du mot émis :

$$(b_1, \ldots, b_k).M_i = y_i \ . \tag{10.1}$$

On remarque en particulier que si l'on note le système à résoudre sous forme matricielle, ce n'est rien d'autre qu'une sous matrice de la matrice génératrice du code. Si l'on peut encoder un mot de code efficacement, alors en ne retenant que les positions non effacées, on peut également calculer un produit matrice-vecteur efficacement ce qui comme on le verra au chapitre 14 peut conduire à un algorithme de décodage performant.

Le décodage est unique si et seulement s'il y a en dehors du support du motif d'effacements k positions qui donnent des équations indépendantes. De tels ensembles de k positions forment ce que l'on appelle un *ensemble d'information* :

Définition 10.3 (Ensemble d'information)**.** Un ensemble d'information $\mathcal{B}$ est la donnée de k positions dans un mot de code qui conduisent à des équations (10.1) indépendantes.

Il existe un lien entre les ensembles d'information d'un code et de son dual dont nous nous servirons plus loin :

Proposition 10.4 (Ensemble d'information du dual)**.** Si $\mathcal{B}$ est un ensemble d'information d'un code linéaire $\mathcal{C}$, alors le complémentaire de $\mathcal{B}$ (c'est-à-dire toutes les positions qui ne sont pas dans $\mathcal{B}$) est un ensemble d'information pour le dual de $\mathcal{C}$.

Démonstration. Raisonnons par l'absurde, si ce n'était pas le cas il existerait un mot du dual c^* non nul qui vaut 0 en dehors de $\mathcal{B}$. Or c'est bien sûr impossible, car il est alors facile de construire un mot du code dont le produit scalaire avec c^* est non nul. □

Quand seulement w positions ne sont pas effacées, on ne dispose que de w équations linéaires et un décodage unique n'est possible que si la dimension k du code et plus petite que w. En fait, on verra plus loin (chapitre 11) que cette limite correspond asymptotiquement à la capacité du canal :

Théorème 10.5 (Capacité du canal à effacements)**.** La capacité du canal à effacements est $1 - p$ dans le cas classique et elle est égale à w/n dans le cas où exactement $n - w$ positions sont effacées.

10.2 Lien avec l'immunité algébrique

Le résultat essentiel de cette section montre que l'immunité algébrique d'une fonction booléenne est reliée au comportement sur le canal à effacements de $\mathrm{RM}(r, m)$ en présence du motif d'effacements f ou $1 + f$:

Lemme 10.6 (Lien avec l'immunité algébrique)**.** Chercher les annulateurs de degré au plus r d'une fonction booléenne f est la même chose que décoder $\mathrm{RM}(r, m)$ en présence d'un motif d'effacements égal au vecteur des valeurs de $1 + f$. Pour ce motif, l'ensemble des annulateurs est alors exactement l'ensemble $\mathcal{I}$ défini dans la section précédente.

Démonstration. Supposons que l'on cherche des annulateurs de degré au plus r d'une fonction booléenne f. On recherche donc une fonction g qui doit prendre la valeur 0 en tous les points où f vaut 1. Ce problème peut se voir comme la recherche d'un mot de $\mathrm{RM}(r, m)$ (associé à g) qui aurait pu être transmis lorsque l'on reçoit le mot tout à 0 avec les positions en dehors du support de f effacées. L'ensemble des annulateurs de f n'est donc rien d'autre que l'ensemble $\mathcal{I}$ défini plus haut lorsque le motif d'effacements est $1 + f$. □

Dans le cas des codes de Reed-Muller, l'équation linéaire (10.1) associée à une position $\mathbf{x}$ sur les k coefficients de l'ANF d'une fonction booléenne f de degré au plus r découle directement de la transformée de Möbius :

$$f(\mathbf{x}) = \sum_{\mathbf{u} \subset \mathbf{x}} f_{\mathbf{u}} \quad \mathbf{u} \in \mathbf{F}_2^m,\ |\mathbf{u}| \leq r\ . \tag{10.2}$$

En fait pour ces codes, le décodage sur le canal à effacements peut être vu comme un problème d'interpolation où l'on cherche des fonctions f de degré au plus r qui interpolent le mot reçu sur les positions non effacées. On reviendra sur le système qui permet de décoder $\mathrm{RM}(r, m)$ sur ce canal

au chapitre 12 où l'on rentrera dans les détails du calcul de l'immunité algébrique.

Pour les codes de Reed-Muller, il existe d'autres algorithmes de décodage qui peuvent être plus efficaces que la simple résolution du système linéaire induit par le motif d'effacements. C'est en particulier le cas lorsque le nombre d'effacements est assez éloigné de la capacité du canal. Un exemple est l'algorithme de Ilya Dumer sur le canal à effacements qui est de nature probabiliste, c'est-à-dire qu'il est possible que l'algorithme ne retrouve pas le mot émis alors que c'est réalisable. Remarquons que ces algorithmes ont de nombreux points communs avec l'algorithme du chapitre 13.

Nous n'insistons pas trop sur ce type d'algorithme ici car pour des applications en cryptographie on veut être sûr qu'une fonction n'admet pas d'annulateur, ou au moins avoir une très forte probabilité que cela soit le cas, ce que ces algorithmes ne nous garantissent pas. En plus, dans les applications, on est souvent soit très proche de la capacité, soit confronté à des tailles trop grandes pour envisager de regarder toutes les positions et d'utiliser des algorithmes classiques. C'est d'ailleurs ce que fait l'algorithme de Cédric Tavernier qui a des applications en cryptographie, il ne regarde absolument pas toutes les positions, le nombre de variables étant beaucoup trop grand. On verra aussi que c'est également le cas pour notre algorithme du chapitre 13.

10.3 Fonctions d'immunité algébrique maximale

Le lien avec les codes de Reed-Muller va nous permettre d'utiliser de nombreux résultats de la théorie des codes correcteurs pour en déduire des propriétés de l'immunité algébrique d'une fonction. Une illustration parfaite en sera le chapitre suivant où l'on démontre un des principaux résultats de cette thèse sur l'immunité algébrique d'une fonction aléatoire.

Mais avant cela, intéressons-nous à l'immunité algébrique maximale d'une fonction donnée et aux fonctions connues qui l'atteignent. Remarquons qu'une fonction booléenne f sur m variables est d'immunité algébrique plus grande que r si et seulement si le support de f et de $1 + f$ contient un ensemble d'information pour $\mathrm{RM}(r, m)$.

Proposition 10.7 (Immunité maximale[1])**.** L'immunité algébrique maximale d'une fonction à m variables est $\lceil m/2 \rceil$.

Démonstration. Nous avons vu que l'existence d'annulateurs dépend de l'inversibilité du système qui permet de décoder le motif d'effacements $1 + f$ sur le canal à effacements. En particulier, s'il y a plus d'inconnues que d'équations, la fonction f admettra toujours un annulateur non trivial. Nous

[1] Résultat démontré indépendamment dans [FA03] et dans [CM03]

avons vu qu'il s'agit d'un système $w \times k$ avec w le poids de f et k défini par

$$k \stackrel{\text{def}}{=} \sum_{i=0}^{r} \binom{m}{i} .$$

Comme l'immunité algébrique dépend des annulateurs de f et de $1+f$, on obtient une majoration de l'immunité algébrique qui dépend du poids de Hamming maximum entre ces deux fonctions. Ce poids est minimal dans le cas d'une fonction équilibrée et vaut exactement 2^{m-1}. Il est alors facile de voir que $k > 2^{m-1}$ dès que $r \geq \lceil m/2 \rceil$, d'où le résultat. □

L'immunité algébrique d'une fonction f dépend à la fois des annulateurs de f et de $1+f$, mais ces deux familles d'annulateurs ne sont pas complètement indépendantes. On a en particulier :

Proposition 10.8 (Dualité)**.** Soit f une fonction booléenne à m variables et de poids $2^m - k$, où k est la dimension de $\mathrm{RM}(r, m)$. Alors si f n'admet pas d'annulateur de degré r, $1+f$ n'admet pas d'annulateur de degré $m-r+1$.

Démonstration. Ce résultat découle de deux propositions que nous avons vu précédemment. En effet d'après 9.10, $\mathrm{RM}(m-r-1, m)$ est le code dual de $\mathrm{RM}(r, m)$ et la proposition 10.4 nous montre que le complémentaire d'un ensemble d'information pour le premier en est un pour le second. □

Un cas particulièrement intéressant est lorsque m est impair et r correspond à l'immunité algébrique maximale possible $(m-1)/2$. On a alors $k = 2^{m-1}$ et pour montrer qu'une fonction équilibrée est d'immunité algébrique maximale, il suffit de considérer uniquement les annulateurs de f. En fait on a le résultat suivant :

Proposition 10.9 (Cas m impair)**.** Dans le cas où m est impair, il y a une bijection entre les fonctions d'immunité algébrique maximale et les ensembles d'information du code de Reed-Muller auto-dual $\mathrm{RM}((m-1)/2, m)$.

Démonstration. Il suffit de remarquer que comme dans ce cas, k vaut 2^{m-1}, pour toute fonction f d'immunité algébrique maximale, $\sup(f)$ et $\sup(1+f)$ sont des ensembles d'information pour $\mathrm{RM}((m-1)/2, m)$. □

Dans le cas où m est pair, chaque ensemble d'information nous donne une fonction d'immunité algébrique maximale, mais il y en a d'autres.

L'exemple le plus connu de fonction d'immunité algébrique maximale est la fonction majorité :

Proposition 10.10 (Fonction majorité)**.** La fonction majorité sur m variables est une fonction symétrique qui vaut 1 si et seulement si son entrée est de poids de Hamming strictement supérieur à $m/2$. La fonction majorité est d'immunité algébrique maximale.

Démonstration. L'essentiel de la démonstration sera donné dans la proposition 12.2 où l'on verra que le système associé aux positions de poids au plus $r = \lfloor m/2 \rfloor$ est en fait involutif et a fortiori inversible : c'est un ensemble d'information pour $\mathrm{RM}(\lfloor m/2 \rfloor, m)$. Pour montrer qu'il n'existe pas d'annulateur pour $1+f$, il suffit alors d'appliquer la proposition précédente. Pour m pair, on montre en fait que la fonction majorité n'admet pas d'annulateur de degré $m/2$ et que $1+f$ n'en admet pas de degré $m/2-1$. Ce résultat est exposé pour m impair en termes d'ensemble d'information du code de Reed-Muller auto-dual dans [KMM05], il a été mentionné plus tard et indépendamment dans [DMS05, BP05]. □

De nombreux travaux ont été effectués pour trouver des fonctions d'immunité algébrique maximale avec en plus de bonnes propriétés cryptographiques. La plupart sont des fonctions dérivées de la fonction majorité que nous venons de voir [DMS05, Car07, BP05]. Il existe aussi une construction récursive de fonction d'immunité algébrique maximale [DGM05]. Nous attirons aussi l'attention sur le résultat suivant qui ne donne peut être pas des fonctions faciles à calculer ou avec de bonne propriétés cryptographiques mais qui est intéressant :

Proposition 10.11 (Ensembles d'information induit par un LFSR[2])**.** La fonction booléenne à m variables qui vaut 1 sur $L_{\lfloor m/2 \rfloor}$ états internes consécutifs d'un LFSR primitif de longueur m est d'immunité algébrique maximale.

Démonstration. La preuve découle directement de la proposition 9.11 où le code de Reed-Muller est vu comme un code cyclique. En effet, pour un code cyclique, k positions consécutives forment toujours un ensemble d'information et l'on finit la preuve comme pour la fonction majorité. On reviendra sur cette propriété d'ensemble d'information des codes cycliques dans la proposition 11.9 sur les codes à rendement cohérent. □

Le nombre de fonctions d'immunité algébrique maximale est un problème ouvert, la meilleure borne connue est la suivante et découle d'un résultat de la théorie des matroïdes (qui formalise de manière abstraite la notion de famille indépendante dans un espace vectoriel). Ce résultat est publié sous une forme similaire dans [QFL05] suite à une suggestion faite dans l'une de nos review. Nous donnerons au chapitre 11 une autre démonstration de cette borne. On verra également qu'elle semble assez éloignée de la réalité. Nous donnons la borne dans le cas impair, mais on obtient le même genre de résultat dans le cas pair.

Proposition 10.12 (Nombre d'ensembles d'information)**.** Pour m impair, il existe au moins $2^{2^{m-1}}$ fonctions d'immunité algébrique maximale, c'est-à-dire au moins la racine carrée du nombre de toutes les fonctions booléennes.

[2] Il s'agit d'un résultat qui n'a jamais été mentionné ailleurs.

Esquisse de preuve. La démonstration repose sur la propriété d'échange de base des matroïdes, voir [Oxl]. En substance, dans le cas particulier d'un espace vectoriel, si l'on a deux bases $\mathcal{B}_1$ et $\mathcal{B}_2$ pour tout sous-ensemble S_1 de la première base, il existe un sous-ensemble S_2 de la seconde base tels que $(\mathcal{B}_1 \backslash S_1) \cup S_2$ et $(\mathcal{B}_2 \backslash S_2) \cup S_1$ soient toutes les deux des bases. La preuve est alors immédiate car il existe au moins une fonction f d'immunité algébrique maximale et pour cette fonction son support et son complémentaire contiennent tous deux un ensemble d'information et donc une base. Comme ces deux bases sont disjointes, on obtient par échange des valeurs de f entre les bases 2^{2^k} fonctions différentes. □

10.4 Résumé

Le décodage sur le canal à effacements des codes de Reed-Muller et le calcul de l'immunité algébrique d'une fonction correspondent à deux visions différentes du même problème.

Une fois cela réalisé, il est alors possible d'utiliser l'important travail effectué sur les codes de Reed-Muller et en théorie des codes correcteurs pour obtenir des résultats sur l'immunité algébrique d'une fonction booléenne.

Il existe des fonctions dites d'immunité algébrique maximale qui sont les fonctions qui offrent la meilleure résistance face aux attaques algébriques standards. Ces fonctions sont intimement liées aux ensembles d'information des codes de Reed-Muller. Dans le cas où m est impair, il y en a exactement le même nombre que le nombre d'ensembles d'information du code de Reed-Muller auto-dual $\mathrm{RM}((m-1)/2, m)$.

Chapitre 11

Probabilité d'erreur sur le canal à effacements

De manière à avoir une idée de l'immunité algébrique d'une fonction booléenne choisie aléatoirement, il est naturel d'essayer d'analyser de manière théorique le comportement des codes de Reed-Muller sur le canal à effacements (voir chapitre 9). Pour cela, nous allons calculer une borne supérieure sur la probabilité que le décodage ne soit pas unique. La probabilité est prise pour un nombre d'effacements $n - w$ fixé sur tous les ensembles de positions effacées possibles. Cela nous donnera pour un code de Reed-Muller une borne supérieure sur la probabilité qu'une fonction booléenne aléatoire sur m variables et de poids w admette des annulateurs de degré au plus r. L'essentiel des résultats de ce chapitre a été publié dans [Did06a].

L'analyse est effectuée dans le cas d'un code linéaire général $\mathcal{C}$ de dimension k et de longueur n. Nous utiliserons dans tous nos exemples les codes de Reed-Muller et un nombre d'effacements égal à la moitié de la longueur ($w = 2^{m-1}$). Les exemples correspondent ainsi au calcul de l'immunité algébrique d'une fonction booléenne équilibrée. Ils nous permettront en particulier de suivre notre démarche scientifique car ce sont ces résultats là que nous avons cherché à améliorer.

11.1 Résultats expérimentaux

Nous allons commencer par regarder le comportement des codes de Reed-Muller sur le canal à effacements de manière expérimentale. Nous comparerons ces résultats au comportement moyen des codes de mêmes paramètres tirés de manière uniforme pour avoir une idée de ce qui se fait de mieux sur ce canal. En effet, un code aléatoire est souvent un très bon code [MC98] même s'il reste inutilisable en pratique. Sur le canal à effacements, un code aléatoire de rendement fixé atteint en particulier la capacité du canal comme nous allons le voir.

Nous rappelons que le nombre w de positions non effacées est aussi le nombre d'équations linéaires que les k bits d'information du mot de code émis doivent satisfaire. Quand la dimension k est strictement plus grande que w, un décodage unique est donc impossible car il y a plus d'inconnues que d'équations. Dans ce cas, P_{err} (qui est la probabilité de décodage non unique prise sur l'ensemble des w positions non effacées possibles) vaut toujours 1.

Le cas limite $k = w$ est intéressant car le nombre de motifs d'effacements décodables est alors exactement égal au nombre d'ensembles d'information du code. Par motif d'effacements, nous désignons une répartition possible des $n - w$ effacements parmi n. En utilisant l'algorithme que nous allons détailler au chapitre 14, nous avons fait des simulations pour les codes de Reed-Muller dans ce cas limite avec un motif d'effacements équilibré. Pour obtenir une dimension d'exactement $n/2$, seuls les codes de Reed-Muller auto-duaux ($m = 2r + 1$) conviennent.

r, m	2,5	3,7	4,9	5,11	6,13	7,15	8,17
P_{err}	0.67	0.84	0.71	0.71	0.71	0.72	0.72

FIG. 11.1 – Simulations pour les codes de Reed-Muller auto-duaux. Le nombre d'échantillons est respectivement pour les codes entre lignes verticales doubles de 10^6,10^5,10^4 et 10^3.

Ces résultats sont très intéressants et même surprenants car on retrouve presque la même proportion d'ensembles d'information que pour un code aléatoire. On a le résultat suivant bien connu pour une matrice aléatoire :

Théorème 11.1 (Matrice aléatoire inversible)**.** Pour $j \geq k$, une matrice binaire $k \times j$ tirée aléatoirement de façon uniforme est de rang plein (c'est-à-dire k) avec une probabilité

$$\prod_{i=j-k+1}^{j} \left(1 - \frac{1}{2^i}\right) .$$

Quand $j = k$, cette expression tend vers $0.289\cdots$ quand k tend vers l'infini.

Démonstration. Il suffit de raisonner par récurrence sur le nombre k de lignes. Pour $k = 1$, la matrice est de rang plein sauf si la ligne est nulle, ce qui nous donne $(1 - 1/2^j)$. Pour k plus grand, la probabilité est le produit de la probabilité qu'une matrice de $k - 1$ lignes soit de rang plein fois la probabilité de tirer une ligne en dehors de l'espace engendré par les $k - 1$ premières lignes choisies. L'espace étant de dimension $k - 1$ et donc de cardinal 2^{k-1} cela nous donne pour cette dernière probabilité $(1-1/2^{j-k+1})$ d'où la première partie du théorème.

Pour la limite dans le cas d'une matrice carrée, la démonstration est un peu plus délicate. Le lecteur est invité à lire [HKL97] pour une preuve ainsi que divers résultats sur le comportement d'un code aléatoire. □

Maintenant, comment passer de ce résultat sur les matrices binaires aléatoires au comportement d'un code linéaire ? Cela se fait de manière toute simple car en moyenne sur l'ensemble des matrices génératrices $k \times n$, un motif d'effacements de poids $n - w$ sera décodable de manière unique si et seulement si la sous-matrice associée de w colonnes est de rang k. Cette matrice étant aléatoire, la probabilité moyenne de décodage unique est donc donnée par le théorème 11.1 avec j égal à w. Finalement, en moyenne un code linéaire a une proportion d'ensembles d'information qui tend vers $0.289\cdots$ et donc une probabilité d'erreur de $0.711\cdots$ qui semble être la même que pour les codes de Reed-Muller.

Enfin, dans le cas où k est strictement inférieur à w, les simulations nous donnent pour les premiers codes de Reed-Muller avec un motif d'effacements équilibré et un k aussi proche que possible de 2^{m-1} les résultats suivants :

r, m	2,6	3,8	4,10	5,12	6,14
k	0.34	0.36	0.37	0.39	0.40
P_{err}	3.10^{-3}	2.10^{-5}	0	0	0

FIG. 11.2 – Simulations pour les codes de Reed-Muller tel que $k/n < 1/2$. Le k est choisi le plus grand possible, ce qui correspond aux codes pour lesquels $m = 2r + 2$. Le nombre d'échantillons est respectivement pour les codes entre lignes verticales doubles de 10^6 et 10^5.

Ici aussi on voit qu'il est facile de conjecturer que la probabilité d'erreur pour de tels codes va tendre vers 0 en présence d'un motif d'effacements équilibré. C'est ce qui se passe pour un code aléatoire de même rendement où la probabilité d'erreur nous est toujours donnée par 1 moins la formule du théorème 11.1 avec un j égal à $n/2$. On constate que cette probabilité décroît exponentiellement en $n - j - k$ par rapport au cas limite. Il est alors facile de montrer que la capacité du canal (qui vaut 1/2 ici) est atteinte.

Au final et au vu des expériences présentées ici, il est naturel de conjecturer que le comportement des codes de Reed-Muller sur le canal à effacements est très bon. Rappelons que cela revient à dire qu'une fonction booléenne aléatoire et équilibrée est presque tout le temps d'immunité algébrique maximale ou quasi-maximale. Cette conjecture a été formulée pour la première fois dans [CG05]. Démontrer cette conjecture est toujours un problème ouvert, notamment pour la proportion constante du nombre d'ensembles d'information. Nous avons néanmoins fait un pas vers sa démonstration avec les résultats que nous allons présenter dans ce chapitre.

11.2 Bornes standards de la théorie des codes

Nous allons maintenant voir ce que nous donnent les résultats classiques de théorie des codes appliqués au cas des codes de Reed-Muller. Malheureusement les résultats vont être assez éloignés des données expérimentales présentées dans la section précédente.

Quand on parle de probabilité d'erreur, il est d'usage d'utiliser ce que l'on appelle la borne de l'union pour en obtenir une borne supérieure. Sur le canal à effacements, sa formulation est la suivante

Théorème 11.2 (Borne de l'union)**.** Soit $\mathcal{C}$ un code linéaire de dimension k et de longueur n en présence de $n-w$ effacements choisis aléatoirement. Alors on a la borne suivante sur la probabilité de décodage non unique :

$$P_{\text{err}} \leq \sum_{i \leq n-w} A_i \frac{\binom{n-i}{w}}{\binom{n}{w}}$$

où A_i est le nombre de mots de code de $\mathcal{C}$ de poids de Hamming i.

Démonstration. Par linéarité on peut supposer que le mot émis est le mot nul. Le décodage n'est pas unique pour tous les motifs d'effacements de poids $n-w$ dont le support contient un mot de code non nul. En notant E_c l'ensemble des motifs qui contiennent le mot de code c, on peut alors écrire pour la probabilité d'erreur qui est prise sur l'ensemble des $\binom{n}{w}$ motifs d'effacements possibles :

$$P_{\text{err}} = \frac{1}{\binom{n}{w}} |\{\bigcup_{c \neq 0} E_c\}| \leq \frac{1}{\binom{n}{w}} \sum_{c \neq 0} |E_c| \ .$$

C'est la "borne de l'union". On finit la preuve en remarquant que pour un mot c de poids i, il y a exactement $\binom{n-i}{w}$ motifs d'erreurs dans E_c. En effet un tel motif contient c, il y a donc i positions fixées et il reste à en choisir $n-w-i$ parmi les $n-i$ restantes. □

Cette borne est assez fine pour les codes aléatoires (voir [MC98]) mais nécessite la connaissance de la distribution de poids du code, c'est-à-dire les A_i. Malheureusement, comme on l'a vu dans le chapitre 9, on ne la connaît pour les codes de Reed-Muller que pour de très petits paramètres. Dans ce cas, on n'a pas vraiment d'autres choix pour obtenir une borne que de supposer le pire, c'est-à-dire que tous les mots de code ont comme poids la distance minimale d_1. On obtient alors ce que l'on va appeler la borne de l'union avec la distance minimale :

$$P_{\text{err}} \leq (2^k - 1) \frac{\binom{n-d_1}{w}}{\binom{n}{w}} \ . \tag{11.1}$$

De cette borne, l'on déduit que :

Théorème 11.3 (Borne de l'union avec la distance minimale)**.** La borne de l'union basée sur la distance minimale nous donne

$$\log_2 P_{\text{err}} \leq k + d_1 \log_2 \left(1 - \frac{w}{n}\right) .$$

Démonstration. Considérons pour $i \geq 0$ la quantité :

$$\frac{\binom{n-i}{w}}{\binom{n}{w}} .$$

Pour $i = 0$ elle vaut 1, et si l'on fait le rapport entre deux i consécutifs on a

$$\frac{\binom{n-(i+1)}{w}}{\binom{n-i}{w}} = \frac{(n-(i+1))!(n-i-w)!}{(n-(i+1)-w)!(n-i)!} = \frac{n-w-i}{n-i} .$$

Or cette fraction est majorée par $\frac{n-w}{n}$ pour tout i. On en déduit que

$$\frac{\binom{n-d_1}{w}}{\binom{n}{w}} \leq \left(\frac{n-w}{n}\right)^{d_1} .$$

On termine alors la preuve en prenant le logarithme en base 2 de (11.1). □

Sur le tableau suivant, on voit pour un r fixé le premier m où cette borne nous donne un résultat plus petit que 1. Pour les r suivants, il est facile de voir que la probabilité d'erreur décroît très fortement.

r	2	3	4	5	6
m	7	11	15	19	24

On est donc bien loin des résultats expérimentaux.

Pour les codes de Reed-Muller, il est possible de faire un peu mieux car on connaît en fait la distribution de poids exacte jusqu'à $2.5d_1$ d'après les travaux de Kazumi, Tokura et Asumi [KT70] et [KTA74]. On a fait les simulations jusqu'à $2d_1$ car l'expression des poids est plus simple, cela nous donne :

r	2	3	4	5	6
m	6	9	13	18	22

C'est donc un peu mieux, mais reste très éloigné des résultats expérimentaux. D'ailleurs au niveau asymptotique on obtient exactement le même résultat :

Proposition 11.4 (Cas des Reed-Muller[1])**.** À l'aide de la borne de l'union basée sur la distance minimale (ou jusqu'à $2.5d_1$) on peut montrer que la probabilité d'erreur d'un code de Reed-Muller $\text{RM}(r, m)$ en présence de 2^{m-1} erreurs tend vers 0 quand m tend vers l'infini si $r = \lambda m$ avec $\lambda < 0.227\cdots$

[1]On peut trouver ce résultat dans [Did06a], il reprend en fait celui de [MPC04] où une borne moins précise était utilisée et où l'analyse dans le cas $2.5d_1$ était erronée.

Démonstration. On fait la preuve pour $w = 2^{m-1}$ mais les calculs sont tout à fait similaires si $w = \alpha n$. On suppose $r = \lambda m$ et l'on cherche un λ tel que la borne de l'union tende vers 0. Il est déjà nécessaire que $\lambda < 1/2$ et pour un tel λ on peut écrire pour la dimension du code

$$k \sim \binom{m}{\lambda m} \leq 2^{mh(\lambda)}$$

où h est la fonction d'entropie binaire

$$h(\lambda) \stackrel{\text{def}}{=} -\lambda \log_2(\lambda) - (1-\lambda)\log_2(1-\lambda) \ .$$

Une démonstration de cette équivalence est donnée dans l'annexe B. En utilisant maintenant le théorème 11.3 avec $w = n/2$ et la valeur de la distance minimale

$$d_1 = 2^{m-r} = 2^{m(1-\lambda)}$$

on obtient

$$\log_2 P_{\text{err}} \leq 2^{mh(\lambda)} - 2^{m(1-\lambda)} \ .$$

une condition nécessaire et suffisante pour que la borne tende vers 0 est alors donnée par l'équation suivante

$$h(\lambda) \leq (1-\lambda) \ .$$

Au passage, on voit qu'un facteur éventuel devant le d_1 utilisé ne change rien, car ce facteur est divisé par m dans l'équation précédente. Finalement, en résolvant cette équation on obtient la condition du théorème. □

Remarquons que ce résultat semble faible, car pour un r de cette forme le rendement k/n du code décroît exponentiellement vite vers 0 ce qui est vraiment très éloigné de ce que l'on essaye de montrer, à savoir un rendement proche de 1/2, ou qui ne s'en éloigne pas trop vite.

11.3 Utilisation des distances généralisées

Nous allons maintenant construire une nouvelle borne basée sur les distances de Hamming généralisées d'un code linéaire. Au terme de cette section, nous aurons montré le théorème suivant :

Théorème 11.5 (Nouvelle borne [Did06a])**.** Pour un code linéaire $\mathcal{C}$ de longueur n, dimension k et de distances généralisées $(d_i)_{i\in[0,k]}$ en présence de $n-w$ effacements, on a

$$P_{\text{err}} \leq \prod_{a=n-w+1}^{n} \left(\frac{a-d_1}{a}\right) \prod_{a=2}^{k} \left(\frac{d_a}{d_a - d_1}\right) \ .$$

Commençons par donner l'idée de la preuve. Étant donné un code linéaire $\mathcal{C}$ de longueur n et de dimension k ainsi qu'un motif d'effacements de poids $n-w$, on note $\mathcal{I}$ l'ensemble des mots du code dont le support est inclus dans le motif d'effacements. L'idée de départ derrière cette nouvelle borne va reposer sur l'algorithme suivant pour calculer $\mathcal{I}$.

On va considérer séquentiellement, et ce dans un ordre aléatoire, les w positions non effacées. Nous noterons $\mathcal{I}_j$ l'espace des mots de code en accord avec les j premières positions choisies. Nous dirons que nous sommes dans l'état (i,j) si $\mathcal{I}_j$ est de dimension i. Par définition, l'ensemble des mots de code inclus dans tout le motif sera $\mathcal{I}_w$ et le motif est décodable si l'on se retrouve dans l'état $(0,w)$ après avoir considéré toutes les positions non-effacées.

Soit maintenant la probabilité $p_{i,j}$ d'être dans l'état (i,j) prise sur tous les motifs d'effacements de poids $n-w$ et sur tous les ordres possibles de choix de positions non effacées. Cela revient en fait à considérer toute nouvelle position comme étant aléatoire parmi les positions restantes. La probabilité de décodage unique est $p_{0,w}$.

L'idée est alors d'obtenir des informations sur les probabilités $p_{i,j}$ en fonctions des probabilités de transitions entre les états. On peut d'ailleurs voir tout cela comme une chaîne de Markov. Pour cela, supposons que nous sommes dans l'état (i,j) et que nous considérons une nouvelle position, il y a alors que deux possibilités :

- $\dim(\mathcal{I}_{j+1}) = \dim(\mathcal{I}_j) - 1 = i-1$.
- $\dim(\mathcal{I}_{j+1}) = \dim(\mathcal{I}_j) = i$.

Soit $t_{i,j}$ la probabilité du premier événement, l'autre arrive alors avec probabilité $1-t_{i,j}$.

Maintenant, regardons un peu plus en détail ces probabilités de transitions, on va démontrer le résultat suivant

Lemme 11.6 (Borne inférieure sur les $t_{i,j}$)**.** On a toujours

$$t_{i,j} \geq t'_{i,j} \stackrel{\text{def}}{=} \frac{d_i}{n-j} \ .$$

De plus les états (i,j) pour lesquels $d_i > n-j$ sont inatteignables.

Démonstration. Si l'on se donne j positions et l'espace $\mathcal{I}_j$ associé, la dimension de l'espace suivant ne va dépendre que du choix de la position suivante par rapport au support de $\mathcal{I}_j$. Si cette position est choisie dans le support, alors la nouvelle contrainte qu'elle amène va forcement exclure des mots de $\mathcal{I}_j$ et la dimension va décroître de 1. Si au contraire la position choisie est en dehors du support, tous les mots de $\mathcal{I}_j$ vont satisfaire la nouvelle contrainte et l'espace va rester inchangé.

Le résultat du lemme découle directement de cette discussion. En effet, $n-j$ correspond au nombre de positions parmi lesquelles on peut choisir

la suivante et d_i est par définition une borne inférieure sur le cardinal de $\mathrm{supp}(\mathcal{I}_j)$. De plus, les j premières positions choisies sont bien en dehors du support par construction de $\mathcal{I}_j$, la probabilité de tomber dans le support est donc bien bornée par $t'_{i,j}$. Pour la même raison, les états pour lesquels $d_i > n - j$ sont bien inatteignables car on n'aurait pas pu prendre les j positions en dehors du support. □

Grâce à ces probabilités de transitions $t'_{i,j}$ on peut alors calculer des probabilités $p'_{i,j}$ en remplissant un tableau un peu de la même façon que dans un algorithme de programmation dynamique. On commence avec toutes les probabilités nulles sauf $p_{k,0}$ qui vaut 1. On remplit ensuite le tableau colonne par colonne avec les relations :

$$p'_{i,j+1} = p'_{i,j}(1 - t'_{i,j}) + p'_{i+1,j}t'_{i+1,j} \ . \tag{11.2}$$

Les $t'_{i,j}$ associés à des états inatteignables peuvent être plus grands que 1, mais cela n'a pas d'importance car la probabilité de ces états restera toujours à 0. Le processus est illustré sur la figure 11.3 pour le code RM$(1, 4)$. Les flèches noires illustrent le passage de masse d'une cellule à ses deux voisines possibles et les cases blanches correspondent aux états inatteignables.

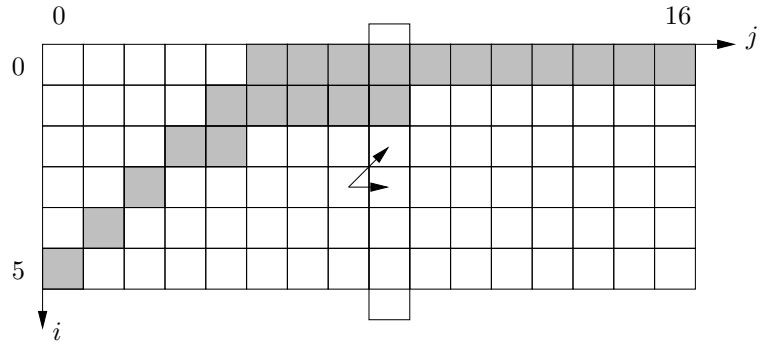

FIG. 11.3 – Exemple de calcul de $p'_{i,j}$ pour le code RM$(1, 4)$.

La question est maintenant de savoir quel est le lien entre les $p'_{i,j}$ et les $p_{i,j}$. Regardons ce qui se passe quand on remplace les probabilités de transition $t_{i,j}$ par les $t'_{i,j}$ une par une. Quand on remplace un $t_{i,j}$ par $t'_{i,j}$ la probabilité $p_{i-1,j+1}$ décroît alors que la probabilité $p_{i,j+1}$ croit. Néanmoins, toutes les sommes partielles de la colonnes $j+1$ seront plus petites ou égales aux sommes précédentes. On a en fait le lemme suivant :

Lemme 11.7 (Liens entre $p_{i,j}$ et $p'_{i,j}$)**.** En considérant les sommes partielles colonne par colonne

$$\sigma_{i,j} \stackrel{\text{def}}{=} p_{0,j} + \cdots + p_{i,j} \quad \text{et} \quad \sigma'_{i,j} \stackrel{\text{def}}{=} p'_{0,j} + \cdots + p'_{i,j}$$

on a pour tout $i \in [0,k]$ et tout $j \in [0,n]$

$$\sigma'_{i,j} \leq \sigma_{i,j} \ .$$

Démonstration. On a les relations suivantes entre sommes partielles :

$$\sigma_{i,j+1} = \sigma_{i,j} + t_{i+1,j} \left[\sigma_{i+1,j} - \sigma_{i,j}\right] \ .$$

Si maintenant on change uniquement ce $t_{i+1,j}$ en $t'_{i+1,j}$, comme $\sigma_{i+1,j}$ est plus grand que $\sigma_{i,j}$ on voit que $\sigma_{i,j+1}$ décroît. Les autres probabilités de transitions restant inchangées, on peut montrer par récurrence que toutes les autres sommes partielles sont également inférieures ou égales aux sommes précédentes. Finalement, on termine la preuve en changeant toutes les probabilités de transitions les unes après les autres. □

À ce stade, on est déjà capable de calculer une borne sur la probabilité d'erreur car on a :

$$P_{\text{err}} = 1 - p_{0,w} = 1 - \sigma_{0,w} \leq 1 - \sigma'_{0,w} \ .$$

Cette borne est calculable du moment que l'on connaît les distances de Hamming généralisées de notre code. Remarquons au passage que l'on peut également obtenir des bornes sur la probabilité que l'espace des mots que l'on pourrait avoir émis soit de dimension inférieure à une dimension fixée.

On a vu au chapitre 9 que les distances de Hamming généralisées sont connues pour les codes de Reed-Muller. On a fait des calculs numériques pour trouver les premiers codes pour lesquels cette borne est non triviale. On voit sur le tableau suivant que l'on a énormément gagné par rapport à la simple borne de l'union.

r	2	3	4	5	6
m	7	9	11	13	15

On vient de voir que l'on a un moyen numérique de calculer une nouvelle borne sur la probabilité d'erreur, mais on peut aller plus loin et obtenir une formule close. Pour cela on va étendre les relations (11.2) entre deux colonnes complètes du tableau. Soit C_j la colonne j, on a donc

$$C_j \stackrel{\text{def}}{=} \begin{pmatrix} p'_{0,j} \\ \vdots \\ p'_{k,j} \end{pmatrix} \ .$$

On peut écrire $C_{j+1} = A_j C_j$ où A_j est la matrice bidiagonale suivante :

$$A_j \stackrel{\text{def}}{=} \begin{pmatrix} 1-t'_{0,j} & t'_{1,j} & & \\ & 1-t'_{1,j} & \ddots & \\ & & \ddots & t'_{k,j} \\ & & & 1-t'_{k,j} \end{pmatrix} \ .$$

Cette matrice a $k+1$ valeurs propres distinctes (car les d_i sont toutes distinctes). Intéressons-nous à un vecteur propre à droite $\Sigma_{i,j}$ de A_j associé à la valeur propre $1-t'_{i,j}$. En résolvant le système d'équations que les coefficients d'un tel vecteur doivent satisfaire, on peut montrer que $\Sigma_{i,j}$ est défini à un facteur près par :

$$\Sigma_{i,j} \stackrel{\text{def}}{=} \left(\underbrace{0 \dots 0}_{i}, 1, \frac{t'_{i+1,j}}{t'_{i+1,j} - t'_{i,j}}, \dots, \prod_{a=i+1}^{k} \frac{t'_{a,j}}{t'_{a,j} - t'_{i,j}} \right)$$

où le premier 1 est en position i (en partant de 0). De plus, en regardant la définition de $t'_{i,j}$ on s'aperçoit que ce vecteur propre ne dépend pas de j. Nous écrirons donc simplement Σ_i avec :

$$\Sigma_i \stackrel{\text{def}}{=} \left(\underbrace{0 \dots 0}_{i}, 1, \frac{d_{i+1}}{d_{i+1} - di}, \dots, \prod_{a=i+1}^{k} \frac{d_a}{d_a - d_i} \right) .$$

L'intérêt de ces vecteurs propres est qu'il est facile de suivre l'évolution de $\Sigma_i.C_j$ d'une colonne sur l'autre. On a

$$\Sigma_i.C_j = \Sigma_i.A_{j-1} \dots A_0 C_0 = \left(\prod_{a=0}^{j-1} (1 - t'_{i,a}) \right) \Sigma_i.C_0 .$$

Ou encore en utilisant l'expression des $t'_{i,j}$ avec les distances de Hamming généralisées :

$$\Sigma_i.C_j = \left(\prod_{a=0}^{j-1} \frac{n - a - d_i}{n - a} \right) \Sigma_i.C_0 .$$

Pour $\Sigma_i C_0$, comme le seul coefficient non nul de C_0 est le dernier qui vaut 1, on a :

$$\Sigma_i.C_0 = \prod_{a=i+1}^{k} \frac{d_a}{d_a - d_i} .$$

À l'aide de tous ces vecteurs propres qui sont indépendants, on peut exprimer de manière close la probabilité d'erreur. Malheureusement, les calculs sont assez fastidieux et nous allons utiliser une approximation qui va déjà nous donner des résultats satisfaisants. Il faudrait certes essayer d'analyser l'expression exacte pour avoir une borne plus fine, ce que nous n'avons pas eu le temps de faire.

Comme tous les coefficients non nuls de Σ_i sont plus grands que 1, on va utiliser la majoration suivante :

$$1 - \sigma_{i,j} \leq \Sigma_i.C_j .$$

Cela nous donne une borne supérieure sur la probabilité que $\mathcal{I}_j$ soit de dimension plus grande ou égale à une valeur i. En l'appliquant pour i égal à 1, on obtient alors le théorème 11.5.

11.4 Les codes à rendement cohérent

Nous définissons maintenant ce que l'on a décidé d'appeler les codes à rendement cohérent. On verra que la plupart des codes linéaires connus rentrent dans cette catégorie pour laquelle on obtient des bornes intéressantes sur la probabilité d'erreur sur le canal à effacements. On commence par définir les codes à rendement cohérent avant de voir les deux principaux théorèmes (11.12 et 11.13) sur le comportement de ces codes sur le canal à effacements.

Nous rappelons que le rendement d'un code linéaire est défini comme sa dimension sur sa longueur (k/n). De la même manière, il est normal de définir le rendement d'un sous-code comme sa dimension sur la taille de son support. Or, comme la i-ème distance généralisée est une borne inférieure sur la taille d'un sous-code de dimension i, la quantité i/d_i correspond au rendement maximum d'un sous-code.

Définition 11.8 (Code à rendement cohérent [Did06a])**.** Un code linéaire $\mathcal{C}$ de longueur n, dimension k et distance de Hamming généralisées est dit à rendement cohérent si et seulement si

$$\forall i \in [1, k], \quad \frac{i}{d_i} \leq \frac{k}{n} \ .$$

Les codes à rendement cohérent sont donc des codes pour lequels n'importe quel sous-code a un rendement moins bon que le rendement du code complet. On va voir plusieurs lemmes qui montrent qu'il s'agit en fait d'une propriété qui semble "naturelle" pour les codes linéaires usuels. Un code qui n'est pas à rendement cohérent peut en effet sembler mal construit car pourquoi utiliser un code plus long si l'un de ses sous-codes a un meilleur rendement ? Par ailleurs, si un motif d'effacements rend le sous-code indécodable, ce sera aussi le cas du code complet.

Lemme 11.9 (Codes cycliques et auto-duaux [Did06a])**.** Tous les codes linéaires auto-duaux et tous les codes linéaires cycliques sont à rendement cohérent.

Démonstration. Soit $\mathcal{C}$ un code linéaire de dimension k et de longueur n. Soit $\mathcal{B}$ un ensemble d'information de $\mathcal{C}$. Soit V un sous-code linéaire de $\mathcal{C}$ de dimension i. Le point clef de la démonstration repose sur l'inégalité

$$|\mathcal{B} \cap \mathrm{supp}(V)| \geq i \ . \tag{11.3}$$

Cette inégalité est assez évidente, car le cardinal de l'intersection est clairement une borne supérieure sur la dimension de V.

Pour prouver le lemme dans le cas des codes auto-duaux, nous rappelons (voir proposition 10.4) que l'ensemble complémentaire de $\mathcal{B}$ est lui aussi une base. On a alors deux ensembles d'information disjoints, et avec (11.3)

on obtient que $|\text{supp}(V)| \geq 2i$. Finalement, comme un code auto-dual est nécessairement de rendement 1/2 on obtient le résultat.

Dans le cas des codes cycliques, en regardant la matrice génératrice engendrée par le polynôme minimal (voir [MS77]), on peut montrer que k positions consécutives forment toujours un ensemble d'information. C'est aussi le cas de k positions consécutives modulo la longueur du code. On a donc l'inégalité (11.3) pour chacun de ces n ensembles d'information. De plus chaque point dans $\text{supp}(V)$ ne contribue que pour k de ces ensembles. En dénombrant la contribution totale, on obtient

$$k|\text{supp}(V)| \geq ni$$

ce qui termine la preuve. □

Lemme 11.10 (Code dual [Did06a])**.** Le code dual d'un code à rendement cohérent est lui aussi à rendement cohérent.

Démonstration. On va utiliser les mêmes notations qu'au lemme précédent et l'on note en plus $(d'_i)_{i\in[1,n-k]}$ les distances de Hamming généralisées du dual de $\mathcal{C}$. Le résultat découle d'une relation entre les distances de Hamming généralisées d'un code et de son dual découverte par Wei [Wei91] :

$$\{d_i, i \in [0,k]\} \cup \{n+1-d'_i, i \in [1,n-k]\} = \{1,\ldots,n\} \ .$$

On peut réécrire la propriété de cohérence du taux par

$$\forall a \in [1,n] \quad |\{d_i \mid 1 \leq d_i \leq a\}| \leq a\frac{k}{n} \ .$$

Cela est en effet clairement vrai quand a est égal à l'un des d_i. Dans ce cas, les d_i étant croissants on obtient $i \leq d_i k/n$ et l'on retrouve la propriété de cohérence du taux. C'est donc vrai pour tout a, car quand a est entre deux d_i la valeur de la partie gauche ne change pas, et celle de la partie droite augmente. Maintenant, en utilisant la relation de Wei on obtient :

$$\forall a \in [1,n] \quad |\{d'_i | n+1-a \leq d'_i \leq n\}| \geq a - a\frac{k}{n} \ .$$

et pour les a de la forme $n - d'_i$ on retrouve la propriété de cohérence pour le dual. □

Lemme 11.11 (Reed-Muller [Did06a])**.** Les codes de Reed-Muller et les codes de Reed-Muller généralisés sont à rendement cohérent.

Démonstration. Depuis [Wei91] les distances généralisées des codes de Reed-Muller sont connues et il est possible de montrer qu'ils sont à rendement cohérent en effectuant des calculs sur ces distances (voir par exemple la démonstration dans [Did06a]).

Le moyen le plus simple reste néanmoins d'utiliser la cyclicité des codes de Reed-Muller, voir la proposition 9.11. On a $d_k = n$ et pour i dans $[1, k-1]$ les d_i sont les mêmes que pour le sous-code cyclique. En effet, un des espaces qui minimise le cardinal du support ne contient certainement pas le mot tout à 1. En utilisant maintenant la cohérence du sous-code cyclique on a $(k-1)d_i \geq i(n-1)$. Or les distances généralisées sont strictement croissantes (voir [Wei91]), donc d_i est supérieur ou égal à i, d'où le résultat. La démonstration est exactement la même pour les codes de Reed-Muller généralisés. □

Remarquons que pour les codes à rendement cohérent on retrouve et généralise la borne inférieure sur le nombre d'ensembles d'information d'un code obtenu pour les codes auto-duaux (voir chapitre 10) :

Théorème 11.12 (Ensemble d'information[2])**.** Une borne inférieure sur la proportion d'ensembles d'information d'un code linéaire $\mathcal{C}$ est donnée par

$$\prod_{i=1}^{k} \frac{d_i}{n-i+1} \ .$$

De plus si $\mathcal{C}$ est à rendement cohérent on obtient une proportion d'ensembles d'information de :

$$\left(\frac{n}{k}\right)^k / \binom{n}{k} \ .$$

Démonstration. En regardant comment on remplit le tableau de la section précédente, on voit que pour arriver sur un $\mathcal{I}_k$ de dimension 0 on est obligé de toujours suivre la transition de probabilité $t'_{i,j}$ qui fait diminuer la dimension. La proportion d'ensembles d'information est donc supérieure à

$$\prod_{i=0}^{k-1} t'_{k-i,i} = \prod_{i=1}^{k} \frac{d_i}{n-i+1} \ .$$

Avec la propriété de cohérence du taux cela nous donne un proportion plus grande que

$$\prod_{i=1}^{k} \frac{n}{k} \frac{i}{n-i+1} = \left(\frac{n}{k}\right)^k \frac{k!}{\frac{n!}{(n-k)!}}$$

d'où le résultat. □

Pour les codes à rendement cohérent, la borne du théorème 11.5 se simplifie en une borne ne dépendant que de la longueur, de la dimension et de la distance minimale du code. Ce résultat donné dans le théorème suivant

[2]Ce résultat est nouveau et n'a pas été publié dans [Did06a].

est à comparer avec celui obtenu avec la borne de l'union n'utilisant que la distance minimale, voir le théorème 11.3. On voit que quand la distance minimale est très bonne, on ne gagne pas grand chose, par contre l'effet du facteur entre crochets devient très important si la distance minimale du code se dégrade.

Théorème 11.13 (Probabilité d'erreur [Did06a])**.** Pour un code $\mathcal{C}$ à rendement cohérent de longueur n, dimension k et distance minimale d_1, on a :

$$\ln(P_{\text{err}}) \leq k \left[\frac{d_1}{n} \left(\ln \frac{n}{d_1} + 3 \right) \right] + d_1 \ln \left(1 - \frac{w}{n} \right) .$$

Démonstration. Commençons par prendre le logarithme de la borne du théorème 11.5 pour traiter les produits plus facilement, cela donne

$$\ln(P_{\text{err}}) \leq \sum_{i=n-w+1}^{n} \ln \left(1 - \frac{d_1}{i} \right) - \sum_{i=2}^{k} \ln \left(1 - \frac{d_1}{d_i} \right) .$$

La première somme est bornée supérieurement par

$$\sum_{i=n-w+1}^{n} \ln \left(1 - \frac{d_1}{i} \right) \leq - \sum_{i=n-w+1}^{n} -\frac{d_1}{i} \leq d_1 \ln \left(\frac{n-w}{n} \right) .$$

Pour la seconde somme, remarquons que pour tout i la fraction d_1/d_i est plus petite que d_1/d_2 par croissance stricte des distances généralisées. En utilisant la borne de Griesmer (voir [Wei91]) qui est une borne inférieure sur les distances de Hamming généralisées de n'importe quel code linéaire de distance minimale d_1, on a toujours $d_1/d_2 \leq 2/3$. Il existe donc une constante positive $a < 1$ telle que

$$- \sum_{i=2}^{k} \ln \left(1 - \frac{d_1}{d_i} \right) \leq \sum_{i=2}^{k} \left(\frac{d_1}{d_i} + a \frac{d_1^2}{d_i^2} \right) . \tag{11.4}$$

Le comportement asymptotique de cette somme va donc dépendre de la somme des inverses des distances de Hamming généralisées. En bornant inférieurement les $i_0 \stackrel{\text{def}}{=} \lfloor \frac{d_1 k}{n} \rfloor$ premières distances par d_1 et en utilisant la propriété de cohérence (définition 11.8) pour les autres, on obtient

$$\sum_{i=1}^{k} \frac{1}{d_i} \leq \frac{i_0}{d_1} + \frac{k}{n} \sum_{i=i_0+1}^{k} \frac{1}{i} .$$

Ce que l'on peut encore majorer par

$$\frac{i_0}{d_1} + \left(\frac{d_1 k}{n} - i_0 \right) \frac{1}{d_1} + \frac{k}{n} \int_{\frac{d_1 k}{n}}^{k} \frac{1}{x} \leq \frac{k}{n} \left(1 + \int_{\frac{d_1 k}{n}}^{k} \frac{1}{x} \right) = \frac{k}{n} \left(1 + \ln \frac{n}{d_1} \right) .$$

De la même manière nous avons pour la somme des inverses au carré

$$\sum_{i=1}^{k} \frac{1}{d_i^2} \leq \frac{i_0}{d_1^2} + \frac{k^2}{n^2} \sum_{i=i_0+1}^{k} \frac{1}{i^2} \leq \frac{k}{n}\left(\frac{1}{d_1} + \frac{k}{n}\int_{\frac{d_1 k}{n}}^{k} \frac{1}{x^2}\right) = \frac{k}{n}\left(\frac{1}{d_1} + \frac{1}{d_1} - \frac{1}{n}\right) .$$

Avec ces deux majorations, l'inégalité (11.4) devient

$$-\sum_{i=2}^{k} \ln\left(1 - \frac{d_1}{d_i}\right) \leq \frac{d_1 k}{n}\left(\ln\frac{n}{d_1} + 3\right) .$$

On en déduit finalement la majoration sur la probabilité d'erreur donnée par le théorème. □

11.5 Application aux Reed-Muller et à l'immunité algébrique

Comme on a vu que les codes de Reed-Muller étaient de rendement cohérent, il ne nous reste plus qu'à voir ce que les résultats précédents nous donnent. On peut se demander pourquoi on n'utilise pas directement l'expression exacte des distances généralisées des codes de Reed-Muller pour avoir des résultats plus précis. En fait, il apparaît que dans le cas des codes de Reed-Muller auto-duaux, la borne avec les distances généralisées exactes et la borne pour les codes à rendement cohérent sont très proches. Nous préférons utiliser la borne pour les codes à rendement cohérent de par sa simplicité.

Le résultat principal de cette section est le suivant :

Théorème 11.14 (Probabilité d'erreur pour RM(r, m) [Did06a])**.** En présence d'exactement 2^{m-1} effacements le code RM(r, m) a une probabilité d'erreur après décodage qui tend vers 0 du moment que r vérifie :

$$r \leq \frac{m}{2} - \sqrt{\frac{m}{2}\ln\left(\frac{m}{2}(1+\varepsilon)\right)}$$

pour tout $\varepsilon > 0$.

Démonstration. En utilisant la cohérence du rendement et le théorème 11.13 on obtient une probabilité d'erreur qui tend vers 0 pour

$$\frac{k}{n}\left(\ln\frac{n}{d_1} + 3\right) + \ln\left(1 - \frac{w}{n}\right) < -\varepsilon$$

avec $\varepsilon > 0$ fixé. Ici la longueur du code n est égale à 2^m. Nous pouvons supposer que $k < 2^{m-1}$ sinon on aura aucune chance d'avoir une probabilité

d'erreur qui tende vers 0. Cela implique que d_1 est plus grand que $2^{m/2}$. On obtient alors une probabilité qui tend vers 0 pour

$$\frac{k}{2^m} < \frac{\ln 2 - \varepsilon}{\ln(2^{m/2}) + 3} < \frac{1 - \varepsilon/\ln 2}{m/2} \ .$$

Quel est le r correspondant à un tel k ? En considérant une loi binomiale X de paramètre $p = 1/2$ sur m essais (voir l'annexe B) on a

$$k = 2^m \Pr\left(X \leq r\right) \ .$$

En utilisant maintenant la borne de Chernoff (voir l'annexe B) on obtient pour une variable $\lambda > 0$

$$\Pr\left(X - \frac{m}{2} \leq -\lambda \frac{\sqrt{m}}{2}\right) \leq \exp\left(-\frac{\lambda^2}{2}\right) \ .$$

Donc avec un r de la forme $\frac{m}{2} - \lambda\frac{\sqrt{m}}{2}$, une condition suffisante sur λ pour que k vérifie l'inégalité donnée plus haut est que

$$\lambda^2 \geq -2 \ln\left(\frac{1 - \varepsilon/\ln 2}{m/2}\right) \ .$$

Pour un tel λ, on obtient une probabilité qui tend vers 0 si

$$r \leq \frac{m}{2} - \sqrt{2 \ln\left(\frac{m/2}{1 - \varepsilon/\ln 2}\right)} \frac{\sqrt{m}}{2} \tag{11.5}$$

et avec un nouvel ε, on obtient notre résultat. □

Ce résultat est plutôt intéressant pour nous car on en déduit que asymptotiquement l'immunité algébrique d'une fonction booléenne équilibrée est presque optimale.

Proposition 11.15. L'immunité algébrique d'une fonction booléenne à m variables est de l'ordre de

$$\frac{m}{2}(1 - o(1))$$

avec une probabilité qui tend vers 1 quand m tend vers l'infini.

11.6 Résumé

Cette analyse nous a conduit à la découverte de nouveaux résultats assez généraux de la théorie des codes linéaires sur le canal à effacements. Du point de vue cryptographique, nous avons ainsi montré que l'immunité algébrique d'une fonction booléenne équilibrée choisie aléatoirement est très proche de l'optimale.

Pour analyser le comportement d'un code sur le canal à effacements, nous avons calculé une borne supérieure sur la probabilité que le décodage ne soit pas unique. La probabilité est prise pour un nombre d'effacements fixé sur tous les ensembles de positions effacées possibles. Un telle borne nous est donnée de manière classique en utilisant une borne de l'union. Malheureusement, la répartition des poids des codes de Reed-Muller n'étant pas connue pour la plupart d'entre eux, on en est réduit à une borne très générale qui n'utilise que leurs longueur, dimension et distance minimale. Nous obtenons ainsi des résultats très éloignés du comportement expérimental de ces codes.

Bien sûr, si on connaissait la distribution de poids exacte des codes de Reed-Muller, la borne de l'union aurait sûrement été très précise. Mais il s'agit là d'un problème très compliqué qui a résisté à la sagacité de nombreux chercheurs depuis maintenant près de 40 ans. Nous avons ainsi suivi une autre piste qui était d'essayer d'exploiter l'information contenue dans ce que l'on appelle les distances de Hamming généralisées qui sont connues pour les codes de Reed-Muller. Cette approche s'est révélée fructueuse et a conduit à une nouvelle borne utilisant ces distances et qui s'applique à tous les codes linéaires.

Au passage, le calcul de cette borne dans le cas des codes de Reed-Muller nous a amené à considérer une nouvelle propriété des codes linéaires que nous avons appelé la cohérence du rendement. Nous avons montré qu'elle est vérifiée par des classes de codes linéaires très importantes. Il s'agit d'une propriété qui nous semble plutôt naturelle car si un code avait un sous-code de meilleur rendement, on gagnerait à utiliser ce sous-code, surtout que tout motifs d'effacements non décodable le serait aussi pour le code original. Finalement, nous avons obtenu pour le comportement de ces codes sur le canal à effacements la meilleure borne connue utilisant uniquement la longueur, la dimension et la distance minimale.

Chapitre 12

Calcul de l'immunité algébrique

Nous allons nous intéresser maintenant, et ce pendant plusieurs chapitres (13 et 14), au calcul de l'immunité algébrique d'une fonction booléenne. Nous détaillons ici les algorithmes de base avant d'en voir des versions plus efficaces dans les prochains chapitres.

On se donne une fonction booléenne f à m variables et le problème est de trouver son immunité algébrique et éventuellement exhiber des annulateurs. L'intérêt est double, un calcul de l'immunité algébrique efficace permet de tester si telle ou telle fonction peut être utilisée dans un système de chiffrement. Quant au calcul explicite d'annulateurs, il est important de savoir s'il ne s'agit pas de l'étape limitante dans les attaques algébriques exposées au chapitre 8.

Nous supposerons que la fonction f est donnée sous la forme de son vecteur des valeurs et ne tiendrons pas compte de toute structure qu'elle pourrait avoir. Dans ce cas, l'approche la plus efficace actuellement repose sur de l'algèbre linéaire.

On commencera par traiter le cas de l'attaque algébrique standard dans la première section. Nous verrons ensuite comment trouver les relations utilisées dans la version rapide de l'attaque. On finira par un rapide état de l'art des divers algorithmes de la littérature et donnerons un aperçu des performances réelles de ces algorithmes sur un ordinateur personnel.

12.1 Utilisation de l'algèbre linéaire

Nous détaillons ici comment ramener le problème du calcul de l'immunité algébrique à un simple problème d'algèbre linéaire. En fait, nous nous intéresserons au problème de calculer les annulateurs de degré au plus r d'une fonction f donnée. En faisant varier r et en considérant aussi $1 + f$, l'immunité algébrique peut s'en déduire facilement. En effectuant une recherche

par dichotomie sur r, on ne perd ainsi qu'un facteur $\log_2 r$ par rapport à tous les algorithmes que nous allons voir.

Au premier abord, il n'est pas évident que l'approche basée sur de l'algèbre linéaire soit la meilleure. On pourrait raisonner directement au niveau algébrique en utilisant la forme polynomiale F de l'ANF de f. Calculer les annulateurs de f de petit degré est équivalent à trouver les polynômes de petit degré dans l'idéal

$$\left\langle 1 + F(X_1, \ldots, X_m), X_1^2 - X_1, \ldots, X_m^2 - X_m \right\rangle \ .$$

En effet, un g tel que $fg = 0$ vérifie $(1 + f)g = g$, le polynôme G qui correspond à l'ANF de g est donc un multiple de $1 + F$ modulo les polynômes $X_i^2 - X_i$. C'est un problème qui peut se résoudre avec des algorithmes utilisant les bases de Gröbner. Si la structure de l'ANF de f est très particulière, cette approche peut d'ailleurs se révéler efficace. En revanche, pour une fonction f peu structurée, le calcul de la base de Gröbner est en pratique beaucoup plus long que l'approche linéaire que nous allons maintenant détailler.

Soit une fonction f de m variables et de poids de Hamming w. On cherche des fonctions g de degré au plus r telles que pour tout $\mathbf{x}$ de $\mathbf{F}_2^m$, $f(\mathbf{x})g(\mathbf{x})$ est égal à 0. On a vu que de telles fonctions g sont entièrement représentées par

$$k \stackrel{\text{def}}{=} \sum_{i=0}^{r} \binom{m}{i}$$

coefficients dans $\mathbf{F}_2$, ce sont les k coefficients $(g_{\mathbf{u}})_{\mathbf{u} \in \mathbf{F}_2^m, |\mathbf{u}| \leq r}$ de l'ANF de g. Pour tout point $\mathbf{x}$ tel que $f(\mathbf{x})$ vaut 1, $g(\mathbf{x})$ doit nécessairement valoir 0. En utilisant la transformée de Möbius définie dans le théorème 3.11, pour un tel $\mathbf{x}$ on obtient une équation linéaire en les coefficients de g :

$$g(\mathbf{x}) = \sum_{\mathbf{u} \subseteq \mathbf{x}} g_{\mathbf{u}} = 0 \qquad \mathbf{u} \in \mathbf{F}_2^m, \quad |\mathbf{u}| \leq r \ . \tag{12.1}$$

En regroupant maintenant toutes ces équations dans une matrice M_1, on obtient un système linéaire de w équations à k inconnues que l'on peut mettre sous la forme

$$M_1 \overline{g} = 0 \ . \tag{12.2}$$

Le vecteur $\overline{g}$ contient les k coefficients de g qui sont ici des inconnues que l'on a arrangées de haut en bas en utilisant l'ordre usuel sur les éléments $\mathbf{u}$ de $\mathbf{F}_2^m$ de poids au plus r. Nous indexerons ces monômes de degré au plus r par $\mathbf{u}_1, \ldots, \mathbf{u}_k$ dans la suite. Chaque ligne de la matrice M_1 correspond à l'équation (12.1) associée à un $\mathbf{x}$ tel que $f(\mathbf{x})$ vaut 1. Les annulateurs de f sont alors les fonctions g de coefficients arrangés dans $\overline{g}$ qui vérifient (12.2).

Comme on l'a vu dans le chapitre 10 c'est exactement le problème du décodage des codes de Reed-Muller d'ordre r et à m variables sur le canal

à effacements. Le produit matrice-vecteur de M_1 par $\overline{g}$ correspond à une évaluation d'une fonction de degré au plus r de coefficients d'ANF donnés par $\overline{g}$ aux points tels que $f(x) = 1$. Nous rencontrerons par la suite plusieurs matrices d'évaluation de ce type et nous allons introduire une notation particulière.

Définition 12.1 (Matrice d'évaluation)**.** La matrice d'évaluation d'une fonction de degré r aux points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ de $\mathbf{F}_2^m$ est la matrice $N \times k$ que nous noterons $V^r_{\mathbf{x}_1,\ldots,\mathbf{x}_N}$ définie par

$$V^r_{\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}} = (\mathbf{x}_i^{\mathbf{u}_j})_{i\in[i,N],\ j\in[1,k]} \ .$$

Cette matrice est telle que

$$V^r_{\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}} \begin{pmatrix} g_{\mathbf{u}_1} \\ \vdots \\ g_{\mathbf{u}_k} \end{pmatrix} = \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_N) \end{pmatrix}$$

où g est la fonction booléenne de degré r et de coefficients de son ANF $g_{\mathbf{u}_1}, \ldots, g_{\mathbf{u}_k}$.

Le nom "matrice d'évaluation" provient du fait qu'un produit matrice-vecteur n'est rien d'autre que l'évaluation d'une fonction booléenne en certains points de $\mathbf{F}_2^m$. Cette propriété découle directement de la transformée de Möbius, et l'on voit que la ligne i de la matrice code en fait l'équation (12.1) associée à $\mathbf{x}_i$. Finalement, la matrice M_1 n'est rien d'autre que

$$M_1 \stackrel{\text{def}}{=} V^r_{\{\mathbf{x}\in\mathbf{F}_2^m, f(\mathbf{x})=1\}} \ .$$

Si la matrice $V^r_{\{\mathbf{x}\in\mathbf{F}_2^m, f(\mathbf{x})=1\}}$ est de rang plein alors f n'admet pas d'annulateur non trivial de degré au plus r. Dans le cas contraire, les annulateurs sont directement donnés par les éléments non nuls du noyau de cette matrice.

Le calcul de l'immunité algébrique se ramène donc à la résolution d'un système linéaire. C'est l'approche retenue par tous les algorithmes efficaces de calcul de l'immunité algébrique qui exploitent la structure particulière de ce système pour le résoudre plus rapidement qu'avec une simple élimination Gaussienne en $O(wk^2)$. Nous verrons comment dans les chapitres suivants.

12.2 Cas des attaques algébriques rapides

Le cas des attaques algébriques rapides peut se formuler quasiment de la même manière que le cas standard. On recherche cette fois deux fonctions g et h telles que

$$f(\mathbf{x})g(\mathbf{x}) + h(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \mathbf{F}_2^m \ . \tag{12.3}$$

On a fait passer le h de l'autre côté sans changement de signe car on travaille sur $\mathbf{F}_2$. On note r le degré de g et $\overline{g}$ le vecteur des k coefficients de son ANF. De manière similaire, e est le degré de h et l'on note $\overline{h}$ le vecteur de ses k' coefficients. On peut alors retranscrire pour un $\mathbf{x}$ donné l'équation (12.3) à l'aide de la transformée de Möbius par

$$f(\mathbf{x}) \sum_{\mathbf{u} \subseteq \mathbf{x}, |\mathbf{u}| \leq r} g_{\mathbf{u}} + \sum_{\mathbf{u} \subseteq \mathbf{x}, |\mathbf{u}| \leq e} h_{\mathbf{u}} = 0 \ .$$

Et en regroupant toutes ces équations sous forme matricielle on obtient

$$\mathrm{Diag}((f(\mathbf{x}))_{\mathbf{x} \in \mathbf{F}_2^m}) V^r_{\mathbf{F}_2^m} \overline{g} + V^e_{\mathbf{F}_2^m} \overline{h} = 0$$

avec le produit par f qui correspond au produit par la matrice diagonale. On peut regrouper tout ceci dans un système de taille $2^m \times (k + k')$ donné par :

$$\left(\mathrm{Diag}((f(\mathbf{x}))_{\mathbf{x} \in \mathbf{F}_2^m}) V^r_{\mathbf{F}_2^m} \mid V^e_{\mathbf{F}_2^m} \right) \begin{pmatrix} \overline{g} \\ \overline{h} \end{pmatrix} = 0 \ . \tag{12.4}$$

Si ce système est de rang plein, alors il n'existe pas de couple de fonctions (g, h), avec les contraintes de degré fixées, qui vérifie $fg = h$. Dans le cas contraire, les solutions nous donnent directement les fonctions g et h en interprétant la solution comme les coefficients de l'ANF de ces fonctions. Au final on obtient une complexité en $O(2^m(k+k')^2)$ pour trouver les fonctions g et h avec des contraintes de degré fixées.

Il est en fait possible d'obtenir un système plus compact comme cela a été montré dans [BLP06]. C'est aussi le même genre d'approche que l'on retrouve dans [ACG+06]. L'idée est de regarder ce qui se passe sur les points $\mathbf{x}$ de poids au plus e. On a alors le résultat suivant qui a été démontré dans [DM06] et que l'on a déjà utilisé pour prouver que la fonction majorité est d'immunité algébrique maximale.

Proposition 12.2. La matrice $V^e_{\{\mathbf{x} \in \mathbf{F}_2^m, |\mathbf{x}| \leq e\}}$ est une matrice involutive (c'est-à-dire sa propre inverse).

Démonstration. Effectuons directement le produit d'une ligne i par une colonne j. En indexant de 1 à k' les $\mathbf{x}$ de poids inférieur à e, ce produit vaut

$$\sum_{|\mathbf{u}| \leq e} \mathbf{u}^{\mathbf{x}_i} \mathbf{x}_j^{\mathbf{u}} \ .$$

Chaque élément ne vaut donc 1 que pour les $\mathbf{u}$ de poids au plus e tel que $\mathbf{x}_j \subseteq \mathbf{u} \subseteq \mathbf{x}_i$. La contrainte de poids n'est en fait pas importante, car comme $\mathbf{x}_i$ est aussi de poids au plus e, la deuxième inclusion implique que $\mathbf{u}$ a un poids au plus e. On peut donc récrire le produit par

$$\sum_{\mathbf{u} \in \mathbf{F}_2^m | \mathbf{x}_j \subseteq \mathbf{u} \subseteq \mathbf{x}_i} 1 \ .$$

Cette somme vaut 0 si $\mathbf{x}_j$ n'est pas inclue dans $\mathbf{x}_i$, elle vaut $2^{|\mathbf{x}_i+\mathbf{x}_j|}$ sinon. Finalement, elle ne vaut 1 sur $\mathbf{F}_2$ que quand $\mathbf{x}_i = \mathbf{x}_j$. Le produit de la matrice par elle même est donc la matrice identité. □

Maintenant, il est possible d'exploiter uniquement les valeurs de $f(\mathbf{x})$ aux points $\mathbf{x}$ de poids au plus e de manière à exprimer les k' coefficients de h en fonctions des k coefficients de g. On doit avoir sur ces points

$$\mathrm{Diag}(f(\mathbf{x})_{\{\mathbf{x},|\mathbf{x}|\leq e\}})V^r_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\overline{g} + V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\overline{h} = 0 \ .$$

Ce qui devient en multipliant à gauche par la matrice involutive $V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}}$ et en faisant passer h de l'autre coté :

$$\overline{h} = \left[V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\mathrm{Diag}(f(\mathbf{x})_{\{\mathbf{x},|\mathbf{x}|\leq e\}})V^r_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\right]\overline{g} \ .$$

Au final, il ne reste plus que les coefficients de g comme inconnues, et le système (12.4) devient $M_2\overline{g} = 0$ avec

$$M_2 = \mathrm{Diag}(f(\mathbf{x})_{\mathbf{F}_2^m})V^r_{\mathbf{F}_2^m} + V^e_{\mathbf{F}_2^m}\left[V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\mathrm{Diag}(f(\mathbf{x})_{\{\mathbf{x},|\mathbf{x}|\leq e\}})V^r_{\{\mathbf{x},|\mathbf{x}|\leq e\}}\right] \ .$$

Ici, la première partie de la somme correspond à l'évaluation de fg sur tous les points de $\mathbf{F}_2^m$ et la deuxième partie à l'évaluation de h sur ces mêmes points. Par construction, les lignes qui correspondent à des $\mathbf{x}$ de poids au plus e sont en fait nulles dans cette matrice. La matrice M_2 peut donc être réduite à une matrice $(2^m - k') \times k$.

Le calcul de cette matrice peut sembler assez lent, une méthode directe étant en $O(2^m kk')$. Nous verrons plus loin (chapitre 14) qu'en fait un produit matrice-vecteur avec la matrice M_2 peut se calculer rapidement. La matrice M_2 peut ainsi se calculer efficacement avec k produits matrice-vecteur qui prennent chacun $O(m \log m)$. En effet, pour un produit de M_2 avec un vecteur qui n'a que la coordonnée i non nulle, on obtient la colonne i de M_2. La complexité finale est alors de $O(m2^m k)$.

Au final, la complexité du calcul de f et g, pour des contraintes de degré fixées, peut se faire en $O(2^m k^2)$. C'est à peu près le même temps que pour les attaques algébriques standards modulo le calcul de la matrice M_2 qui est un peu plus compliqué. Par contre, même si les outils utilisés sont les mêmes, il est plus difficile dans ce cas là d'analyser la probabilité que cette matrice soit de rang plein comme on l'a fait au chapitre 11 pour M_1 et l'attaque algébrique standard. Il est ainsi difficile d'évaluer l'immunité d'une fonction aléatoire aux attaques algébriques rapides.

12.3 État de l'art

Nous venons de voir comment le calcul de l'immunité algébrique d'une fonction donnée peut se ramener à un problème d'algèbre linéaire. Quand la

fonction booléenne étudiée n'a pas de structure particulière, c'est la méthode de choix à utiliser. Une fois le système écrit, la méthode de base se ramène à une simple élimination gaussienne.

Il existe essentiellement trois autres algorithmes plus performants pour le calcul de l'immunité algébrique dont on a résumé les complexités dans les tables 12.1 et 12.2. On donne ici aussi quelques résultats de calculs réels effectués avec notre implémentation des algorithmes. Tous les calculs ont été effectués sur un P4 cadencé à $3.2Ghz$ et disposant de $2G$ de mémoire vive. Les programmes ont été écrits en C et sont disponibles sur ma page Web.

Le premier algorithme (chapitre 13) est une sorte d'élimination gaussienne améliorée qui tente d'utiliser la forte structure de la matrice M_1. Il est particulièrement utile pour vérifier qu'une fonction donnée n'admet pas d'annulateur de faible degré, voir la table 12.3. La complexité est alors linéaire en k si l'on fait la moyenne sur l'ensemble des fonctions booléennes.

Le deuxième (chapitre 14) est quant à lui le meilleur algorithme pour montrer qu'une fonction est d'immunité algébrique maximale, voir la table 12.4. Pour la plupart des paramètres, c'est aussi le meilleur algorithme dans le cas des attaques rapides. Son point fort est vraiment l'utilisation mémoire qui est de l'ordre de $O(2^m)$ par comparaison aux autres algorithmes qui utilisent une mémoire quadratique en k. Dépasser les 20 variables lorsque l'on veut montrer qu'une fonction est d'immunité maximale devient alors impossible pour ces derniers qui ont alors besoin d'environ 2^{38} bits de mémoire. Par comparaison, avec l'algorithme du chapitre 14 on a réussi à montrer qu'une fonction aléatoire de 25 variables était d'immunité algébrique maximale après 20 jours de calcul.

Le dernier algorithme est celui avec la meilleure complexité théorique en $O(k^2)$ en moyenne (le pire des cas étant en $O(2^m k)$), il utilise aussi l'algèbre linéaire et est décrit dans [ACG+06]. Il peut sembler surprenant que l'on ne détaille pas cet algorithme dans cette thèse, mais de nombreux points restent imprécis et nous ne sommes pas capable de le décrire de manière satisfaisante. Depuis, nous avons eu en notre possession une version longue, mais nous n'avons pas eu le temps de vérifier tous les détails de cet algorithme avant la rédaction de ce document.

Une autre approche pour connaître l'immunité algébrique d'une fonction est de considérer des fonctions très structurées et d'essayer de calculer directement leur immunité algébrique. C'est ce qui a conduit à plusieurs constructions de fonctions d'immunité algébrique maximale. De nombreux travaux ont également été effectués pour calculer ou borner l'immunité algébrique des familles de fonctions présentées au chapitre 3. Par exemple on peut voir, pour les fonctions puissances [NGG06] et pour les fonctions symétriques [BP05][1].

[1] Attention, une partie des résultats de cet article sont faux.

Algorithme	Temps	Mémoire
Élimination gaussienne	$O(wk^2)$	$O(k^2)$
[ACG+06]	$O(k^2)$	$O(k^2)$
Chapitre 13	$O(k)$ pour r fixé et $n \to \infty$	$O(k)$
Chapitre 14	$O(m2^m k)$	$O(m2^m)$

TAB. 12.1 – Résumé des complexités des différents algorithmes dans le cas des attaques algébriques standards. La complexité de [ACG+06] est une complexité en moyenne quand la fonction n'admet pas d'annulateur, sinon la complexité est de $O(k2^m)$. La complexité de l'algorithme du chapitre 13 est une complexité moyenne pour les fonctions aléatoires.

Algorithme	Temps	Mémoire
Élimination gaussienne	$O(2^m(k+k')^2)$	$O((k+k')^2)$
[ACG+06]	$O(k'k^2)$	$O(k^2)$
[BLP06]	$O(k'k^2 + k'^2)$	$O(k'k)$
Chapitre 13	–	–
Chapitre 14	$O(m2^m k)$	$O(m2^m)$

TAB. 12.2 – Résumé des complexités des différents algorithmes dans le cas des attaques algébriques rapides, on a toujours $k' > k$. Les algorithmes de [ACG+06] et [BLP06] se contentent en fait de travailler sur la matrice M_2 spécifiée dans la section précédente, seule leur façon de la manipuler change un peu. La complexité de ces deux algorithmes est une complexité moyenne, dans le pire des cas, il faut remplacer un k par 2^m. La complexité de l'algorithme du chapitre 13 devrait être plus ou moins la même que dans le cas standard, mais nous n'avons aucune preuve théorique de ce fait.

r, m	6,32	7,32	4,64	5,64	2,128	3,128	2,256
k	1.10^6	4.10^6	6.10^5	8.10^6	8.10^3	3.10^5	3.10^4
Chapitre 13	30s	2m40s	32s	8m	0.1s	32s	0.3s

TAB. 12.3 – Temps de calcul de l'algorithme du chapitre 13 pour prouver qu'il n'existe pas d'annulateurs de degrée au plus r. Toutes les fonctions ont été choisies aléatoirement de manière uniforme parmis l'ensemble des fonctions booléennes et se sont avérées sans annulateur du degré spécifié.

Algorithme	(r, m)	**Temps**
Élimination gaussienne	$(8, 17)$	quelques heures
[ACG+06]	$(9, 19)$	-
Chapitre 13	$(9, 19)$	6 heures
Chapitre 14	$(9, 19)$	102 secondes
Wiedemann (1 passe)	$(11, 23)$	11 heures
	$(12, 25)$	30 jours

TAB. 12.4 – Record de calcul du test de l'immunité algébrique maximale dans le cas d'une fonction équilibrée et aléatoire. Cela correspond au décodage du code de Reed-Muller auto-dual indiqué sur le canal à effacements et en présence de 2^{m-1} effacements.

Chapitre 13

Vérifier efficacement l'immunité algébrique

Nous allons voir dans ce chapitre une façon un peu plus efficace de résoudre le système linéaire induit par la recherche d'annulateurs d'une fonction f. Cette approche a fait l'objet d'une publication avec Jean-Pierre Tillich dans [DT06]. Elle est particulièrement efficace dans le cas assez utile en cryptographie où r est petit et m assez grand. On rappelle que f est une fonction de m variables et de poids w, que r est le degré maximum des annulateurs recherchés et que k est la dimension de $\mathrm{RM}(r, m)$.

Dans le cas où r est petit devant m, une fonction booléenne a très peu de chances d'admettre des annulateurs de degré au plus r et nous allons dans un premier temps nous consacrer au problème de prouver qu'il n'y a pas de tels annulateurs. Selon les paramètres, ce problème est plus simple que celui de calculer un annulateur. En effet, pour montrer l'absence d'annulateurs, il suffit d'exhiber k équations indépendantes alors que pour être sûr qu'un annulateur l'est vraiment, on doit prendre en compte toutes les w équations induites par les positions où f vaut un. Dans le cas où r est petit, cette différence est énorme. On a par exemple prouvé qu'une fonction de 64 variables n'admettait pas d'annulateur de degré au plus 5 en quelques minutes seulement ce qui est irréalisable si l'on regarde tous les 2^{64} points de l'espace.

Nous présentons ainsi dans la première section un algorithme qui pour r fixé et m grand a un temps de calcul moyen de $O(k)$ pour prouver qu'une fonction booléenne n'admet pas d'annulateur de degré au plus r. Cette complexité est optimale car il est nécessaire d'exhiber un ensemble d'information de k positions pour vérifier l'absence d'annulateur. Il est possible que l'algorithme ne réussisse pas à exhiber un tel ensemble pour une fonction qui n'admet pourtant pas d'annulateur. On montre que la proportion de telles fonctions est négligeable. L'analyse est donnée dans la deuxième section et repose sur les résultats théoriques du chapitre 11.

On présente ensuite un second algorithme qui lui va fonctionner tout le temps et renvoyer une base des annulateurs de f. Cet algorithme est en pratique beaucoup plus efficace qu'une simple élimination gaussienne. Nous conjecturons de plus que le temps de calcul moyen de cet algorithme pour r fixé est aussi bon que celui du premier algorithme, mais nous avons seulement réussi à prouver qu'il était majoré par un $O(k \log^2 m)$. Il est également possible d'adapter ce dernier algorithme au cas du calcul des relations pour les attaques algébriques rapides. On terminera ce chapitre en donnant les performances de notre implémentation.

13.1 Un premier algorithme

Il existe un algorithme qui est certainement optimal au niveau du nombre d'équations considérées pour prouver l'absence d'annulateur d'une fonction aléatoire f. Il suffit de prendre les équations associées à des points où f vaut 1 une par une et de manière aléatoire jusqu'à ce que l'on obtienne un système de rang plein. On peut implémenter un tel algorithme de la façon suivante :

Algorithme 13.1 (Élimination gaussienne paresseuse)**.** Étant donnée une fonction booléenne f de m variables et un paramètre r, cet algorithme répond à la question : existe-il des annulateurs de f non triviaux de degré au plus r ?

1. [Initialisation] On commence avec une matrice $k \times k$ que l'on note M toute à zéro.
2. [Tirer une équation] On choisit un nouveau point $\mathbf{x}$ aléatoire de $\mathbf{F}_2^m$ (qui n'a pas été choisi précédemment) tel que $f(\mathbf{x}) = 1$. S'il n'y en a plus, répondre `non`.
3. [Ajout de l'équation] À l'aide d'un pivot de Gauss partiel, on réduit par rapport à M l'équation sur les coefficients de l'ANF d'un annulateur recherché g associé à $g(\mathbf{x}) = 0$. Si cette équation est indépendante des précédentes, on la stocke sous une forme réduite dans une ligne de M, sinon M reste inchangée.
4. [Immunité ?] Si M est de rang k retourner `oui`, sinon aller en 2.

La complexité dans le pire des cas de cet algorithme est en $O(2^m k^2)$, mais si la conjecture du chapitre 11 sur le bon comportement des codes de Reed-Muller sur le canal à effacements est vérifiée, la complexité moyenne de cet algorithme est en $O(k^3)$. Avec notre théorème 11.13, nous sommes juste capable de prouver le lemme suivant qui ne s'en éloigne pas trop.

Théorème 13.2 (Complexité moyenne de l'algorithme 13.1 [DT06])**.** Si les points tels que $f(\mathbf{x}) = 1$ sont répartis aléatoirement et peuvent être tirés

en $O(1)$, la complexité moyenne de l'élimination gaussienne paresseuse est majorée par $O(rk^3)$.

Démonstration. Chaque équation ajoutée se traite en $O(k^2)$ et dans le pire des cas, on a besoin de traiter toutes les équations, c'est-à-dire 2^m équations. On va borner le temps moyen en bornant le nombre d'équations moyen au bout duquel l'algorithme se termine multiplié par $O(k^2)$.

Pour cela, calculons la probabilité que l'algorithme n'ait pas encore terminé après avoir ajouté $((r+4)\ln 2+3)k$ équations. C'est exactement la probabilité d'erreur sur le canal à effacements avec $w = 2^m-((r+4)\ln 2+3)k$ positions aléatoires effacées. En appliquant le théorème 11.13, on peut alors majorer cette probabilité par

$$\ln P_{\text{err}} \leq 2^{m-r}\left[\frac{k}{2^m}(r\ln 2+3)+\ln\left(1-\frac{((r+4)\ln 2+3)k}{2^m}\right)\right] .$$

Or $\ln(1-x)$ est toujours plus petit que $-x$ pour x positif ce qui nous montre que la probabilité que l'algorithme n'ait pas terminé après ce nombre d'étapes est bornée par

$$\exp\left(-\frac{4k\ln 2}{2^r}\right) .$$

On va donc séparer le temps d'exécution en deux, soit on a fini avec moins d'équations que $((r+4)\ln 2+3)k$, soit on va supposer que l'on les prend toutes. Le nombre d'équations moyen avant terminaison est donc majoré par

$$((r+4)\ln 2+3)k\left(1-\exp\left(-\frac{4k\ln 2}{2^r}\right)\right)+2^m\exp\left(-\frac{4k\ln 2}{2^r}\right) . \quad (13.1)$$

On va montrer que le terme de droite est un $o(1)$. Pour cela on peut déjà supposer $r < m/2$ sinon de toute manière $O(k^3)$ et $O(2^m k^2)$ sont la même chose. On a alors $k \geq \binom{m}{r} \geq m2^{r-1}$ et le terme de droite est majoré par

$$\exp\left[m\ln 2-\frac{4m2^{r-1}\ln 2}{2^r}\right] = 2^{-m}$$

qui est bien un $o(1)$. La majoration (13.1) est donc un $O(rk)$, ce qui termine la preuve. □

Dans le cas où $r \ll m$ on peut faire beaucoup mieux et en fait montrer qu'il n'y a pas d'annulateur en un temps moyen de $O(k)$. C'est optimal car il faut au moins regarder k points pour obtenir un système de rang plein et être sûr qu'une fonction n'admet pas d'annulateur de degré au plus r. L'idée est d'utiliser la décomposition $(u, u+v)$ des codes de Reed-Muller que nous rappelons ici.

Définition 13.3 (Décomposition $(u, u+v)$)**.** Soit f une fonction booléenne à m variables, la décomposition $(u, u+v)$ de f est donnée par deux fonctions booléennes a $m-1$ variables u et v telles que l'on puisse écrire f sous la forme :

$$f(x_1, \ldots, x_m) = u(x_1, \ldots, x_{m-1}) + x_m v(x_1, \ldots, x_{m-1}) \ .$$

De plus, si le degré de f est d alors u est de degré d et v de degré $d-1$.

On peut noter que quand $x_m = 0$ alors $f(x_1, \ldots, x_m) = u(x_1, \ldots, x_{m-1})$ et quand $x_m = 1$ alors $f(x_1, \ldots, x_m) = u(x_1, \ldots, x_{m-1}) + v(x_1, \ldots, x_{m-1})$. C'est-à-dire que u est la restriction de f sur l'espace $x_m = 0$ et $u+v$ et la restriction de f sur l'espace $x_m = 1$. On a alors le lemme suivant :

Lemme 13.4 (Caractérisation récursive de l'immunité[1].)**.** Soit f une fonction booléenne à m variables et sa décomposition $(u, u+v)$. Si u (la restriction de f sur $x_m = 0$) n'admet pas d'annulateur non trivial de degré au plus r et si $u+v$ (la restriction de f sur $x_m = 1$) n'admet pas d'annulateur non trivial de degré au plus $r-1$ alors f n'admet pas d'annulateur non trivial de degré au plus r.

Démonstration. Si f admet un annulateur non trivial g de degré au plus r, alors on peut lui aussi le décomposer en $(u', u'+v')$. Il y a alors 2 cas :

- soit $u' \neq 0$ et alors $uu' = 0$, ce qui nous donne un annulateur non trivial de u de degré au plus r.
- soit $u' = 0$ et alors $v' \neq 0$ par non trivialité de g. On a alors $(u+v)v' = 0$ et $(u+v)$ admet un annulateur non trivial v' de degré au plus $r-1$.

On vient de démontrer la contraposée du résultat. □

L'idée de notre algorithme repose entièrement sur le lemme 13.4. En effet pour vérifier l'immunité de f, il suffit de le faire sur deux fonctions d'une variable de moins et pour un paramètre r, soit égal, soit inférieur de 1. On va alors réappliquer ce lemme de façon récursive jusqu'à obtenir soit des sous-fonctions de $m' = 2d + 1 + \lceil \log_2 m \rceil$ variables, soit des sous-fonctions où le r' associé est égal à 0. On verra que ce m' est un bon compromis entre la facilité de tester la présence d'annulateurs pour de telles sous-fonctions et la probabilité qu'il en existe. L'algorithme, où l'on va tester l'immunité des sous-fonctions avec l'algorithme de Gauss paresseux, est donc le suivant :

Algorithme 13.5 (Vérification de l'immunité [DT06])**.** L'entrée de l'algorithme est une fonction booléenne f à m variables et un paramètre r. La sortie est `oui` si l'algorithme a prouvé que f n'admet pas d'annulateur de degré au plus r et `non` dans le cas contraire.

[1] Ce résultat tiré de [DT06] est déjà présent dans plusieurs articles comme [DGM04].

1. [décomposition] Décomposer de manière récursive l'espace des variables de f selon le paramètre r jusqu'à obtenir soit des sous-fonctions de $m' = 2r + 1 + \lceil \log_2 m \rceil$ variables, soit un degré associé r' de 0. Pour chaque sous-fonction, exécuter l'étape 2.
2. [Vérification de la sous-fonction] Utiliser l'algorithme d'élimination gaussienne paresseuse sur la sous-fonction avec le bon degré r'. Si elle admet un annulateur non trivial de degré au plus r', stopper l'exécution et renvoyer `non`.
3. [Immunité] Renvoyer `oui` : f n'a pas d'annulateur non trivial de degré au plus r.

On peut voir sur la figure 13.1 une illustration de la décomposition que nous allons utiliser. Le premier nombre entre parenthèses correspond au degré des annulateurs à chercher et le second au nombre de variables de la sous-fonctions.

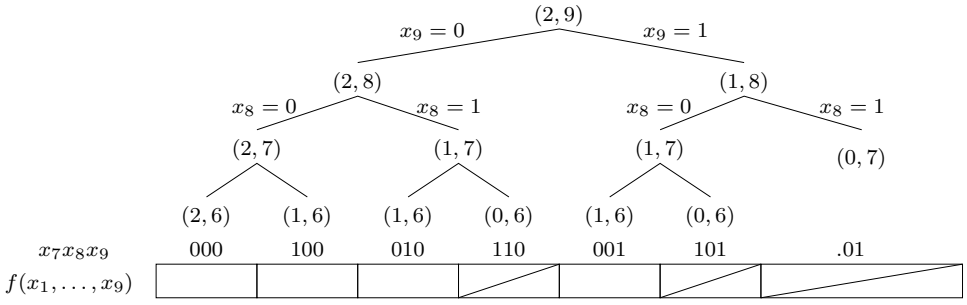

FIG. 13.1 – Décomposition récursive d'une fonction à 9 variables utilisée pour la recherche d'annulateurs de degré 2. La décomposition s'arrête aux sous-fonctions de degré 6, les carrés du bas indique les domaines des sous-fonctions par rapport au domaine de f qui est tout $\mathbf{F}_2^9$.

On a la proposition suivante pour une décomposition récursive de f :

Proposition 13.6 (Description des sous-fonctions)**.** Le premier niveau de décomposition d'une fonction $f(x_1, \ldots, x_m)$ correspond pour la partie gauche à la restriction de f pour $x_m = 0$ et pour la partie droite à la restriction pour $x_m = 1$. De manière plus générale, considérons une sous-fonction f' au i-ème niveau de décomposition. Elle est associée à un chemin $\mathbf{b} = (b_1, \ldots, b_i) \in \mathbf{F}_2^i$ qui part de la racine (i.e f) et qui suit, soit la sous-fonction de gauche, soit celle de droite à chaque niveau. La valeur de b_i vaut 0 si du niveau $i-1$ au niveau i on a pris la sous-fonction de droite et 1 sinon. La sous-fonction f' est donc la restriction de f pour $x_m = b_1, \ldots, x_{m+1-i} = b_i$. De plus , si l'on cherche un annulateur de degré au plus r pour f, il faudra regarder d'après le lemme 13.4 les annulateurs de degré au plus $r - |b|$ de f'.

L'algorithme 13.5 peut sembler assez simple, mais lorsque r est petit devant m on peut décomposer la fonction f assez loin en ayant toujours une très forte probabilité qu'aucune sous-fonction n'admet un annulateur. Bien sûr, plus la décomposition de f ira loin, plus il sera rapide de vérifier l'immunité de toutes les sous-fonctions et ainsi d'en déduire celle de f. En fait le vrai gain provient des sous-fonctions associées à un r' nul pour lesquelles il faudra juste vérifier qu'elles sont non nulles ce qui prendra un temps moyen de $O(1)$! D'un autre coté, plus on décompose f plus la probabilité qu'une sous-fonction admette un annulateur et donc que l'algorithme échoue est grande. La limite $2r + 1 + \lceil \log_2 m \rceil$ choisie dans l'algorithme est un bon compromis pour lequel nous avons le résultat suivant :

Théorème 13.7 (Complexité de l'algorithme [DT06])**.** Soit f une fonction booléenne équilibrée à m variables et choisie de manière aléatoire. Supposons r fixé et m tendant vers l'infini. L'algorithme 13.5 a une complexité moyenne de $O(k) = O(r^m)$ et prouve qu'il n'existe pas d'annulateur de f non trivial de degré au plus r, sauf pour une proportion de fonctions d'au plus $O\left(\exp\left(-2^r m(1 + o(1))\right)\right)$.

Démonstration. La preuve est longue et occupe toute la section suivante. □

13.2 Calcul théorique de la complexité

La preuve du théorème 13.7 repose sur les trois lemmes suivants et sera donnée en fin de section. Le premier lemme nous donne le nombre de sous-fonctions du type (r', m'). Pour prouver le théorème il faudra alors calculer pour une sous-fonction d'un type donné la probabilité qu'elle n'admette pas un annulateur non trivial de degré r' et la complexité moyenne d'un algorithme qui essaie de le prouver. Le deuxième lemme traite le cas des sous-fonctions qui ont un r' associé de 0 et le dernier lemme s'occupe des fonctions associées à un m' de la forme $2r + 1 + \lceil \log_2 m \rceil$.

Lemme 13.8 (Dénombrement des sous-fonctions)**.** Soit une fonction f de m variables pour laquelle on recherche des annulateurs de degré au plus r. Dans la décomposition de la section précédente, le nombre de sous-fonctions d'exactement m' variables associées à un $r' > 0$ est $\binom{m-m'}{r-r'}$ et le nombre de sous-fonctions d'au moins m' variables associé à un r' nul est $\binom{m-m'}{r}$.

Démonstration. Tout repose en fait sur la proposition 13.6 qui définit une bijection entre les sous-fonctions et les chemins dans l'arbre de décomposition. Il est ainsi possible d'associer à chaque sous-fonction un unique chemin codé par un vecteur de $\mathbf{F}_2^{m-m'}$ de poids au plus r. C'est immédiat pour les sous-fonctions d'exactement m' variables, et dans le cas où le chemin de la proposition 13.6 est de longueur plus petite que $m - m'$, alors il a nécessairement un poids r et l'on peut donc le compléter de manière unique en un vecteur

de longueur $m - m'$ en rajoutant des 0. Un vecteur de poids r' correspond alors exactement aux sous-fonctions associées à ce r', d'où le résultat. □

Lemme 13.9 (Sous-fonctions nulles)**.** Soit f une fonction booléenne aléatoire et équilibrée à m variables pour laquelle on recherche des annulateurs de degré au plus r. On considère des sous-fonctions f' de f obtenues en fixant la valeur de $m-m'$ variables de f. Alors f' est uniformément nulle (et admet donc un annulateur non trivial de degré 0) avec une probabilité majorée par $2^{-2^{m'}}$ et l'on peut vérifier que ce n'est pas le cas en un temps moyen de $O(1)$.

Démonstration. Soit $n = 2^m$, une fonction équilibrée de m variables est nulle sur un ensemble de i points avec une probabilité de

$$\binom{n-i}{n/2} / \binom{n}{n/2} .$$

Pour $i = 1$, la probabilité est d'exactement $1/2$, et si l'on fait le rapport entre deux termes consécutifs on a :

$$\binom{n-(i+1)}{n/2} / \binom{n-i}{n/2} = \frac{n-i-n/2}{n-i} \leq \frac{1}{2} .$$

Une sous-fonction de m' variables d'une fonction équilibrée de m variables est donc de poids 0 avec une probabilité inférieure à $2^{-2^{m'}}$.

Évaluons maintenant la complexité de trouver un point où f' vaut 1. La probabilité de tester i points avant d'en trouver un dont l'image vaut 1 est de

$$\frac{\binom{n-i}{n/2-1}}{\binom{n}{n/2}} . \tag{13.2}$$

Le temps moyen est alors donné par la formule

$$\sum_{i=1}^{2^{m'}} i \frac{\binom{n-i}{n/2-1}}{\binom{n}{n/2}} + 2^{m'} \binom{n-2^{m'}}{n/2} / \binom{n}{n/2} .$$

Le dernier terme correspond au cas de la fonction f' uniformément nulle avec la probabilité vue plus haut et est clairement borné par un $O(1)$. Pour la somme, on peut remarquer que le quotient entre deux termes consécutifs de la forme (13.2) vérifie :

$$\frac{\binom{n-(i+1)}{n/2-1}}{\binom{n-i}{n/2-1}} = \frac{n-i-(n/2-1)}{n-i} .$$

Il est donc inférieur à $1/2$ pour $i > 1$ et le temps moyen est donc majoré par

$$O\left(\sum_{i=1}^{2^{m'}} i \left(\frac{1}{2}\right)^i\right) + O(1) = O(1) .$$

□

Lemme 13.10 (Sous-fonctions de $2r+1+\lceil \log_2 m \rceil$ variables)**.** Soit f une fonction booléenne aléatoire et équilibrée à m variables pour laquelle on recherche des annulateurs de degré au plus r. On considère des sous-fonctions f' de f obtenues en fixant la valeur de $m-m'$ variables de f. Soit m' de la forme $2r+1+\lceil \log_2 m \rceil$, alors la probabilité que f' admette des annulateurs non triviaux de degré au plus r est bornée par une fonction en

$$O\left(\exp\left[-2^r m(1+o(1))\right]\right) \ .$$

Démonstration. Notons $k' = \binom{m'}{0} + \ldots \binom{m'}{r}$ et $n' = 2^{m'}$. On commence la preuve en calculant la probabilité que f' soit de poids i. Comme f est choisie de manière uniforme parmi toutes les fonctions équilibrées, cette probabilité vaut :

$$\frac{\binom{n-n'}{n/2-i}\binom{n'}{i}}{\binom{n}{n/2}} \ .$$

En utilisant l'encadrement de la fonction factorielle (voir l'annexe B) on peut montrer que cette probabilité est en

$$O\left(\exp\left[n'(h(i/n')-1)\right]\right)$$

où h est la fonction $h(x) = -x\ln x - (1-x)\ln(1-x)$. En utilisant le théorème 11.13, la probabilité que f ait un annulateur non trivial de degré au plus r est alors bornée supérieurement par une fonction en

$$\sum_{i=0}^{n'} O\left(\exp\left[n'\left(h\left(\frac{i}{n'}\right) - 1 + \frac{1}{2^r}\left(\frac{k'}{n'}(r\ln 2 + 3) + \ln\left(1-\frac{i}{n'}\right)\right)\right)\right]\right)$$

Il est facile de montrer que $k'/n' = o(1)$, on a donc une majoration de la forme

$$\sum_{i=0}^{n'} O\left(\exp\left[n'\left(h\left(\frac{i}{n'}\right) - 1 + \frac{1}{2^r}\ln\left(1-\frac{i}{n'}\right) + o(1)\right)\right]\right) \ .$$

On peut majorer le tout par n' fois le terme maximum de la somme qui dépend du maximum de la fonction $h(x) - 1 + \frac{1}{2^r}\ln(1-x)$ que l'on peut montrer inférieur à $-\frac{1}{2^{r+1}}$. On en déduit donc que la probabilité d'erreur est majorée par une fonction en

$$O\left(\exp\left[-\frac{n'}{2^{r+1}}(1+o(1))\right]\right) = O\left(\exp\left[-2^r m(1+o(1))\right]\right) \ ,$$

d'où le résultat. □

Nous sommes maintenant en mesure de prouver le théorème 13.7. Quel est le temps moyen d'exécution de l'algorithme 13.5 ? Pour les sous-fonctions associées à un $r' = 0$, on utilise le lemme 13.9 qui nous indique une complexité moyenne en $O(1)$. Pour les sous-fonctions associées à un $r' > 0$, on va appliquer l'élimination gaussienne paresseuse. Ce sont des fonctions d'exactement $m' = 2r + 1 + \lceil \log_2 m \rceil$ variables, la complexité de cet algorithme est donc bornée par $O(r(k')^3)$ avec $k' \leq O(m'^r)$, c'est le théorème 13.2. On obtient donc à l'aide du lemme 13.8 que la complexité moyenne de l'algorithme 13.5 est majorée par :

$$\sum_{i=1}^{r} \binom{m-m'}{r-i} O(rm'^{3r}) + \binom{m-m'}{r} O(1) \ .$$

et comme $\binom{m-m'}{r-i} = O(m^{r-i})$ on obtient bien une complexité finale en $O(m^r)$ ce qui est du $O(k)$ car r est fixé (voir l'annexe B). Remarquez que cette complexité est dominée par les sous-fonctions associées à un r' de 0.

À l'aide des trois lemmes précédents, on peut maintenant majorer la probabilité d'échec pour une fonction f équilibrée et aléatoire par :

$$\sum_{i=1}^{r} \binom{m-m'}{r-i} O\left(\exp\left[-2^r m(1+o(1))\right]\right) + \binom{m-m'}{r} 2^{2^{-m'}} \ .$$

Cela termine la preuve vu que l'on peut simplifier le tout en

$$O\left(m^{r-1} \exp\left[-2^r m(1+o(1))\right] + m^r 2^{-2^{2r+1}m}\right) = O\left(\exp\left[-2^r m(1+o(1))\right]\right) \ .$$

13.3 Une version pratique

Nous venons de voir dans ce début de chapitre un algorithme qui utilise une décomposition de f en sous-fonctions de manière à prouver que f n'admet pas d'annulateur de degré donné. La décomposition considérée était de profondeur fixe, mais il est possible de faire varier cette profondeur. Pour une fonction f donnée, la meilleure décomposition est celle la plus profonde possible telle qu'aucune sous-fonction n'admette d'annulateur du degré maximum requis.

On va utiliser l'ordre usuel sur les monômes qui va en fait nous permettre de travailler sur cette meilleure décomposition. Cela va nous conduire à un nouvel algorithme plutôt efficace en pratique et qui va même nous permettre de calculer les annulateurs s'ils existent. Malheureusement, une analyse précise de la complexité de cet algorithme est délicate, nous avons juste une borne supérieure de la complexité quand l'algorithme 13.5 aurait répondu `oui`, et même dans ce cas la borne est légèrement supérieure à $O(k)$.

L'algorithme est donné ci-dessous, ses relations avec l'algorithme 13.5 peuvent sembler obscures mais elles seront éclaircies plus loin.

Algorithme 13.11 (Algorithme incrémental [DT06])**.** L'entrée est une fonction booléenne en m variables et un paramètre r. La sortie est une base de l'espace des annulateurs de f de degré au plus r. Ces fonctions sont stockées sous la forme des coefficients de leur ANF. On utilise l'ordre usuel sur $\mathbf{F}_2^m$ et l'on numérote les monômes de poids au plus r de 1 à k.

1. [initialisation] Initialiser une pile S vide, cette pile va finir par contenir une base des annulateurs de f. Mettre l'indice du monôme courant i à 0. Choisir $\mathbf{x}$ comme la première valeur de $\mathbf{F}_2^m$ telle que $f(\mathbf{x}) = 1$.
2. [Empiler ?] Tant que le monôme d'indice $(i+1)$ est plus petit ou égal à $\mathbf{x}$, empiler dans S la fonction monôme correspondante et incrémenter i.
3. [Dépiler ?] S'il y en a, trouver la fonction la plus proche du sommet de S qui s'évalue en 1 au point $\mathbf{x}$. L'additionner à toutes les autres fonctions de S qui valent 1 au point $\mathbf{x}$ et la retirer de S.
4. [Saut ?] Si S est vide, mettre dans $\mathbf{x}$ la valeur du $(i+1)$ monôme moins 1.
5. [Boucle ?] Si possible, incrémenter $\mathbf{x}$ jusqu'à ce que $f(\mathbf{x}) = 1$ et aller en 2.
6. [Fin] Exécuter une dernière fois l'étape 2 et retourner la base courante stockée dans S.

La preuve que cet algorithme renvoie bien une base de l'espace des annulateurs de f découle directement du lemme suivant.

Lemme 13.12. Définissons $A_{<\mathbf{x}}$ (et respectivement $A_{\leq \mathbf{x}}$) comme l'ensemble des fonctions booléennes g à m variables engendrées par les monômes de degré au plus r et plus petits ou égaux à $\mathbf{x}$ qui vérifient $f(\mathbf{y})g(\mathbf{y}) = 0$ pour tous les points $\mathbf{y}$ strictement plus petits que $\mathbf{x}$ (respectivement inférieurs ou égaux à $\mathbf{x}$). Alors :

- [Étape 2] L'ensemble S après la fin de l'étape 2 est une base de $A_{<\mathbf{x}}$.
- [Étape 3] L'ensemble S après la fin de l'étape 3 est une base de $A_{\leq \mathbf{x}}$.
- [Étape 4] L'ensemble S après la fin de l'étape 4 est une base de $A_{\leq \mathbf{x}}$, même si $\mathbf{x}$ a changé durant l'étape 4.
- [Étape 5] Soit $\mathbf{x}^-$ la valeur de $\mathbf{x}$ à l'entrée de l'étape 5, après exécution de cette étape on a $f(\mathbf{x}^-) = f(\mathbf{x}) = 1$ et $f(\mathbf{y}) = 0$ pour $\mathbf{y}$ dans $]\mathbf{x}^-, \mathbf{x}[$.

Démonstration. Le point clef est que la valeur d'une fonction booléenne aux points $\mathbf{x}$ ne dépend que des monômes inférieurs ou égaux à $\mathbf{x}$ comme on l'a vu au chapitre 3 dans la proposition 3.10.

Tout d'abord les assertions pour les étapes 4 et 5 sont claires. Pour l'étape 4, avancer le $\mathbf{x}$ comme on le fait n'introduit aucun nouveau monôme dans la définition de $A_{\leq \mathbf{x}}$ qui reste donc l'espace réduit à la fonction nulle. Maintenant raisonnons par récurrence sur le nombre de fois que l'on exécute l'étape 2 pour montrer les assertions des étapes 2 et 3.

L'initialisation est claire, considérons donc une nouvelle étape 2 pour laquelle toutes les assertions du lemme sont valides avant son exécution. Dans l'étape 2, on empile dans S toutes les fonctions monômes $\mathbf{u}$ de poids au plus r comprises dans l'intervalle $]\mathbf{x}^-, \mathbf{x}]$. D'après l'hypothèse de récurrence, $f(\mathbf{y})$ vaut justement 0 pour tout $\mathbf{y}$ dans $]\mathbf{x}^-, \mathbf{x}[$, il est alors facile de vérifier que tous les éléments dans S appartiennent à $A_{<\mathbf{x}}$. Ils sont également clairement indépendants.

Considérons maintenant un élément g de $A_{<\mathbf{x}}$ et décomposons le par $g = g_{\leq \mathbf{x}^-} + g_{>\mathbf{x}^-}$, où les monômes de l'ANF de ces deux fonctions vérifient les inégalités données en indice. $g_{\leq \mathbf{x}^-}$ appartient nécessairement à $A_{\leq \mathbf{x}^-}$ (c'est le point clef rappelé en début de démonstration), il est donc généré par les éléments de S. La fonction $g_{>\mathbf{x}^-}$ est également engendrée par les éléments dans S que nous venons de rajouter. S est donc bien une base de $A_{<\mathbf{x}}$. L'assertion de l'étape 3 en découle, d'où le lemme. □

Remarque 1 L'étape 4 peut sembler surprenante car son omission ne change en rien le déroulement de l'algorithme. Néanmoins elle est essentielle pour obtenir une faible complexité car elle permet d'éviter de tester un grand nombre de points pour lesquels $f(\mathbf{x})$ vaut pourtant 1 mais qui n'apportent rien.

Remarque 2 Dans l'étape 3, prendre l'élément le plus proche du sommet de la pile est une heuristique qui permet d'améliorer le temps d'exécution de l'algorithme. En définissant le monôme de queue d'une fonction f comme le plus petit monôme apparaissant dans son ANF, plus on est près du sommet de la pile, plus grand est ce monôme de queue. Au final, prendre l'élément le plus proche du sommet aura tendance à réduire le nombre de monômes dans l'ANF de la fonction choisie qui sera donc plus facile à additionner avec les autres.

Remarque 3 Sur des applications pratiques, il arrive qu'une fonction admette un annulateur de faible degré, il existe alors plusieurs façons de modifier l'algorithme pour le trouver plus vite. Quand on sait qu'il y a un annulateur, on peut modifier l'étape 4 pour sauter au monôme suivant dès que S est de dimension 1 (où même un peu plus), cela nous fait vraiment gagner beaucoup de temps comme on le verra dans la dernière section. On peut aussi réduire artificiellement le poids de f, on aura alors à la fin un espace de candidats pour les annulateurs qui sera souvent de petite dimension, on pourra alors retrouver les vrais annulateurs en essayant les divers candidats de cet espace.

Intéressons-nous maintenant à la complexité de cet algorithme. Dans le pire des cas, il a la même complexité qu'une élimination gaussienne classique

en $O(|f|k^2)$. En effet, pour chaque point $\mathbf{x}$ tel que $f(\mathbf{x}) = 1$ il faut évaluer toutes les fonctions de S en $\mathbf{x}$, cette complexité est bornée par un $O(|S|k)$. Il arrive aussi que l'on doive ajouter certaines fonctions de S entre elles, ce qui se fait en $O(|S|k)$. Comme S contient au plus k fonctions, on obtient bien une complexité dans le pire des cas de $O(|f|k^2)$. Bien sûr en pratique, $|S|$ reste faible et l'algorithme se comporte beaucoup mieux.

On a d'ailleurs un résultat théorique sur ce comportement moyen grâce aux relations avec l'algorithme 13.5 lorsqu'il renvoie "`Oui j'ai prouvé que la fonction` f `n'avait pas d'annulateur de degré au plus` r". Appelons algorithme 13.5* une version modifiée de l'algorithme 13.5 qui utilise à l'étape 2 (c'est celle qui évalue l'immunité algébrique sur les sous-fonctions) l'algorithme 13.11 à la place de l'élimination gaussienne paresseuse. On a alors le résultat suivant :

Proposition 13.13 (Complexité de l'algorithme incrémental)**.** La complexité de l'algorithme 13.11 est bornée supérieurement par la plus petite complexité de l'algorithme 13.5* lorsque ce dernier renvoie `Oui`, le minimum sur la complexité étant pris sur toutes les décompositions de f pour lesquelles les sous-fonctions n'ont pas d'annulateur.

Démonstration. Supposons donc que l'on applique l'algorithme 13.5* sur une décomposition de f pour laquelle il renvoie `oui`. Considérons la première sous-fonction (la plus à gauche) pour laquelle on recherche des annulateurs de degré au plus r. Cette sous-fonction n'est rien d'autre que la restriction de f sur l'intervalle $[0, a[$ où a est de la forme $2^{m'}$ avec m' le nombre de variables de la sous-fonction. L'algorithme 13.5* coïncide avec l'algorithme 13.11 sur cette sous-fonction et trouve qu'il n'y a pas d'annulateur.

La seconde sous-fonction considérée par l'algorithme 13.5* est une restriction de f sur un intervalle de la forme $[a, b[$, et ce sera aussi le cas pour toutes les autres sous-fonctions considérées. Ces intervalles sont consécutifs et sont également examinés par l'algorithme 13.5* de manière consécutive. Comme ce dernier va prouver qu'il n'existe pas d'annulateur du degré voulu pour chacun de ces intervalles, l'algorithme 13.11 va de lui même s'appliquer indépendamment à chacun de ces intervalles. En fait, chaque fois que $\mathbf{x}$ passera l'une des bornes de ces intervalles, S ne contiendra aucun monôme et toutes les fonctions considérées par la suite ne dépendront plus des monômes avant $\mathbf{x}$. On en déduit la proposition. □

Malgré cette réduction, il reste le problème que l'algorithme 13.11 est moins efficace que l'algorithme d'élimination gaussienne paresseuse (algorithme 13.1) sur les sous-fonctions. La complexité moyenne de ce dernier est en effet de $O(rk^3)$ alors que la complexité dans le pire des cas de l'algorithme 13.11 est de $O(2^m k^2)$. En revanche sur les fonctions associées à un r' de 0 on a bien une complexité moyenne en $O(1)$ grâce à l'étape 4, on en déduit donc le résultat suivant :

Proposition 13.14. La complexité de l'algorithme 13.5* appliqué à la décomposition utilisée précédemment est de $O(k \log^2 m)$.

Démonstration. La démonstration est la même que pour le théorème 13.7 en remplaçant la complexité de l'algorithme de Gauss paresseux par $O(2^{m'} k'^2)$ à la place de $O(rk'^3)$. □

13.4 Cas de l'attaque algébrique rapide

Tout ce que nous avons dit jusqu'à présent ne fonctionne que pour la recherche d'un annulateur et donc pour l'attaque algébrique standard. On peut néanmoins adapter l'algorithme au cas des attaques rapides tout en conservant le même comportement. Par contre, l'absence de théorème sur la probabilité d'existence des relations utilisées dans ces attaques nous interdit tout résultat théorique sur la complexité.

On va procéder de la même façon, le point essentiel étant que la valeur de g et de h en un point $\mathbf{x}$ ne dépend que des monômes plus petits que $\mathbf{x}$. Pour chaque fonction g dans S on aura aussi la fonction correspondante h construite jusqu'au point $\mathbf{x}$. L'algorithme est le suivant :

Algorithme 13.15 (Algorithme incrémental pour les attaques rapides[2])**.** L'entrée est une fonction booléenne en m variables et deux paramètres r et e. La sortie est une base de l'espace des fonctions g, h telles que $fg = h$. Ces fonctions sont stockées sous la forme des coefficients de leur ANF. On utilise l'ordre lexicographique sur $\mathbf{F}_2^m$.

1. [initialisation] Commencer avec une pile vide S qui va finir par contenir une base des (g, h). Commencer avec $\mathbf{x}$ à 0.
2. [Empiler ?] Si $|\mathbf{x}| \leq r$ ajouter la fonction $(g, 0)$ dans S.
3. [Correction] Si $|\mathbf{x}| \leq e$, parcourir les couples (g, h) de S et ajouter $\mathbf{x}$ dans l'ANF de h si l'équation $fg = h$ n'est pas respectée. Si $|\mathbf{x}| \leq e$ on peut aussi sauter l'étape suivante.
4. [Dépiler ?] S'il y en a, trouver le couple de fonctions (g, h) le plus proche du sommet de S qui ne vérifie par $fg = h$ en $\mathbf{x}$. L'additionner à tous les autres couples de S qui ne vérifient pas $fg = h$ au point $\mathbf{x}$ et le retirer de S.
5. [Saut ?] Si S est vide, mettre dans $\mathbf{x}$ la valeur du prochain élément de $\mathbf{F}_2^m$ de poids au plus r, moins 1.
6. [Boucle ?] Si $\mathbf{x}$ n'est pas le dernier, l'incrémenter et retourner en 2.
7. [Fin] Retourner la base courante stockée dans S.

[2] Cet algorithme est nouveau et n'est pas mentionné dans [DT06].

La preuve de la correction de cet algorithme est quasiment la même que pour la version où l'on ne cherche que les annulateurs. Il suffit de remplacer les ensembles $A_{<\mathbf{x}}$ et $A_{\leq\mathbf{x}}$ par leur équivalent $R_{<\mathbf{x}}$ et $R_{\leq\mathbf{x}}$ qui contiennent les couples de fonctions (g, h) avec les bonnes contraintes de degré qui vérifient $fg = h$ jusqu'au point $\mathbf{x}$ (inclus ou non suivant la notation) et qui ne s'expriment qu'avec les monômes inférieurs ou égaux à $\mathbf{x}$. À la fin de l'étape 2 on a alors S qui est une base de $R_{<\mathbf{x}}$; une petite subtilité est que les fonctions h ne dépendent pas encore du monôme $\mathbf{x}$ si $|\mathbf{x}|$ est au plus e. Après l'étape 3 ou 4 selon le cas, on a cette fois S qui est une base de $R_{\leq\mathbf{x}}$.

La complexité dans le pire des cas est encore une fois presque comme l'élimination gaussienne standard en $O(2^m k(k + k'))$ où k' correspond à la dimension de l'espace des fonctions booléennes de degré au plus e. En effet pour chaque point $\mathbf{x}$ l'algorithme doit évaluer toutes les fonctions de S en $\mathbf{x}$, la complexité est majorée par $|S|(k+k')$. Il arrive que l'on doive aussi ajouter des couples de fonctions entre eux, la complexité est également majorée par $|S|(k + k')$. Or $|S|$ est bornée par k d'où le résultat.

Il est possible de ramener la complexité dans le pire des cas en $O(2^m k^2)$, pour cela on ne stocke pas les coefficients de h, juste ceux de g car on peut montrer :

Lemme 13.16. La valeur de $fg + h$ en un point $\mathbf{x}$ se calcule en fonction des coefficients de g et en supposant $fg + h = 0$ sur les points strictement plus petits que $\mathbf{x}$ en $O(k' \log k')$

Démonstration. Une fois connue la valeur de f au point $\mathbf{x}$, la valeur de $g(\mathbf{x})$ s'exprime via la transformée de Möbius en fonction des coefficients de g et celle de $h(\mathbf{x})$ en fonction des coefficients de h. On verra alors au chapitre 14 que cette dernière relation peut être transportée sur les coefficients de g via la transposée de la transformée de Möbius en $O(k' \log k')$. □

Une fois cette dépendance calculée, l'évaluation de chaque fonction dans S se fait alors en $O(k)$, la complexité pour un point $\mathbf{x}$ est donc bornée par $|S|k + k' \log k'$, ce qui se borne par un $O(k^2 + k' \log k')$.

En pratique bien sûr, $|S|$ n'est pas très grand donc on va encore plus vite. Et dans le cas où des sous-fonctions qui correspondent à des restrictions de f sur $[0, a[$ n'admettent pas de couple (g, h), on peut alors oublier tous les monômes jusqu'à a et appliquer l'étape 5.

13.5 Résumé et performances

Nous avons construit dans ce chapitre l'algorithme 13.11 qui permet de calculer les annulateurs éventuels d'une fonction booléenne plus rapidement qu'une simple élimination gaussienne. Ses performances s'expliquent essentiellement par une utilisation intensive de l'ordre usuel sur les monômes et ses propriétés, ce qui permet de simplifier un peu les calculs.

Nous avons implémenté ce dernier sur un Pentium 4 cadencé à 2.6Ghz et possédant 1Gb de mémoire vive. On a sur la table 13.1 une comparaison des performances avec l'algorithme de base qui est une simple élimination gaussienne. On peut voir que le gain de temps est important et que même l'utilisation mémoire est meilleure ce qui nous a permis de traiter des cas où une matrice $k \times k$ ne tient pas en mémoire.

r, m	4,10	5,12	6,14	7,16	8,18	9,20
k	386	1586	6476	26333	106762	431910
EG paresseuse	0s	0.1s	5s	2mn30s	oom	oom
Algo 13.11	0s	0.01s	0.5s	20s	15mn	12h

TAB. 13.1 – Temps de calcul pour tester l'immunité algébrique maximale de fonctions aléatoires dans le cas m impair. L'abréviation `oom` veut dire `Out Of Memory`, c'est-à-dire que l'algorithme utilise plus de 2Gb de mémoire vive.

Néanmoins, c'est dans le cas où m est grand devant r que notre algorithme est le plus efficace. La complexité moyenne théorique qui est alors en $O(k)$ apparaît clairement sur la table 13.2 que l'on avait déjà vue au chapitre précédent. Cet algorithme permet ainsi de vérifier que l'immunité algébrique d'une fonction booléenne n'est pas trop faible, même si cette fonction a un très grand nombre de variables. Il peut être utilisé pour construire une fonction cryptographique en choisissant une fonction qui a un bon comportement vis à vis des autres critères cryptographiques (non-linéarité, équilibre, résilience, ...) puis en testant ensuite son immunité algébrique.

r, m	6,32	7,32	4,64	5,64	2,128	3,128	2,256
k	1.10^6	4.10^6	6.10^5	8.10^6	8.10^3	3.10^5	3.10^4
Chapitre 13	30s	2mn40s	32s	8mn	0.1s	32s	0.3s

TAB. 13.2 – Temps de calcul de l'algorithme 13.11 pour prouver l'immunité algébrique d'une fonction aléatoire. Toutes les fonctions choisies se sont effectivement avérées sans annulateurs du degré spécifié.

Enfin, pour illustrer la remarque 3 faite sur l'algorithme incrémental 13.11 on peut trouver dans la table 3 les performances en présence d'un annulateur. On voit ainsi que dans sa version normale, l'algorithme est obligé de tester toutes les positions ce qui prend énormément de temps. Par contre dans sa version $(*)$ qui utilise les modifications décrites dans la remarque 3, on peut alors retrouver très vite l'annulateur de la fonction.

r, m	2,30	3,30	4,30	5,30	6,30
k	466	4526	3.10^4	2.10^5	8.10^5
Algo 13.11	13mn	1h	3h45	-	-
Algo 13.11 (*)	1s	1s	4s	31s	4mn34s

TAB. 13.3 – Temps de calcul pour $m = 30$ dans le cas où au moins un annulateur non trivial existe. Les vecteurs de valeurs des fonctions sont générés comme des vecteurs aléatoires choisis hors du support d'un annulateur lui aussi aléatoire.

Chapitre 14

Algèbre linéaire creuse et canal à effacements

Lorsque l'on est capable de générer un mot de code efficacement à partir des bits d'information à transmettre, il est possible de décoder sur le canal à effacements plus rapidement qu'avec un simple pivot de Gauss. On utilise pour cela des méthodes développées initialement pour résoudre des systèmes linéaires creux. Cette approche du décodage sur le canal à effacements a fait l'objet d'une publication dans [Did06b] où on l'applique au cas des Reed-Muller et au calcul de l'immunité algébrique. On remarquera que ce genre d'approche existait déjà dans le cas des codes LDPC où la matrice de parité est creuse, voir [BM04].

Pour résoudre un système linéaire $Mx = y$ où M est une matrice $n \times n$ ces méthodes ne modifient pas M comme dans un pivot de Gauss et passent l'essentiel de leur temps à calculer de l'ordre de n produits entre la matrice M et des vecteurs. Lorsque M est creuse, il est alors possible de faire un produit matrice-vecteur plus rapidement qu'en $O(n^2)$ ce qui débouche sur un algorithme final plus rapide que l'élimination gaussienne en $O(n^3)$.

Il existe essentiellement sur les corps finis deux familles d'algorithmes qui procèdent ainsi. La première est une adaptation des algorithmes du gradient conjugué et de Lanczos [COS86, Odl84] et la deuxième que nous utiliserons ici repose sur l'algorithme de Wiedemann [Wie86]. De nombreuses études ont été menées sur ce sujet car, il y a de nombreuses applications où l'on doit résoudre de gros systèmes linéaires creux. C'est notamment le cas en cryptographie à clef publique où ces algorithmes sont utilisés dans la dernière étape des algorithmes de factorisation ou de calcul de logarithme discret les plus modernes. Ils ont également été utilisés dans le contexte des attaques algébriques contre le cryptosystème HFE, voir [FJ03].

Mais il n'est pas nécessaire que la matrice soit creuse pour que cette approche fonctionne, il suffit juste de pouvoir calculer un produit matrice-vecteur efficacement pour que ces méthodes deviennent performantes. C'est

en particulier le cas si l'on peut générer un mot de code rapidement, car comme on l'a déjà remarqué, un produit matrice-vecteur pour la matrice impliquée dans le décodage sur le canal à effacements n'est rien d'autre qu'une génération d'un mot de code.

Un très bon exemple de matrices qui ne sont pas creuses mais qui possèdent une forte structure permettant un produit matrice-vecteur rapide est justement donné par les matrices génératrices des code de Reed-Muller. Notre approche utilise l'algorithme de Wiedemann pour résoudre efficacement de tels systèmes et nous verrons que cela conduit à un algorithme de calcul de l'immunité algébrique des plus efficaces.

L'organisation de ce chapitre est la suivante. On commence par décrire l'algorithme de Wiedemann dans le cas d'une matrice carrée avant de voir comment le généraliser au cas d'une matrice quelconque. On verra ensuite un peu plus en détail comment générer efficacement des mots de code pour les codes de Reed-Muller. Cela nous permettra de décoder ces codes et donc de calculer l'immunité algébrique d'une fonction efficacement comme nous le montrent les résultats de notre implémentation que nous commenterons à la fin du chapitre.

14.1 L'algorithme de Wiedemann

Nous présentons dans cette section l'algorithme de Wiedemann pour une matrice carrée A de taille $n \times n$ comme décrit dans [Wie86]. On s'occupera du cas non carré dans la section suivante.

L'approche utilisée par l'algorithme de Wiedemann est de prendre un vecteur b et de calculer ce que l'on appelle la séquence de Krylov

$$b, Ab, A^2b, \ldots, A^nb, \ldots \ .$$

Cette approche est en fait utilisée par tous les algorithmes qui font de l'algèbre linéaire en considérant le produit matrice-vecteur comme une boite noire. La séquence de Krylov satisfait une récurrence linéaire et admet un polynôme minimal $P_b(X)$ tel que $P_b(A)b = 0$. De plus, $P_b(X)$ divise le polynôme caractéristique de la matrice A, il est donc de degré au plus n.

L'idée derrière l'algorithme de Wiedemann est de retrouver ce polynôme $P_b(X)$ en utilisant l'algorithme de Berlekamp-Massey. Comme cet algorithme ne marche pas pour les séquences vectorielles, on va choisir un autre vecteur aléatoire u et calculer les produits scalaires

$$u.b, u.Ab, u.A^2b, \ldots, u.A^{2n}b \ .$$

La complexité de cette étape est en $O(2n)$ produits matrice-vecteur avec la matrice A. Une fois ces $2n$ termes connus, il est alors possible de retrouver le polynôme minimal $P_{u,b}(X)$ de cette nouvelle séquence en $O(n^2)$ avec

Berlekamp-Massey, voir le chapitre 7. On sait de plus que $P_{u,b}(X)$ divise $P_b(X)$ et Wiedemann a montré dans [Wie86] qu'ils sont égaux avec une probabilité plus grande que $1/(6 \log n)$. On peut remarquer que si X divise $P_{u,b}(X)$ on est sûr que A est une matrice singulière car elle admet alors 0 comme valeur propre.

Maintenant, supposons que l'on connaisse $P_b(X)$ que l'on peut alors écrire $P_b(X) = c_0 + XQ(X)$ où $Q(X)$ est un autre polynôme. Si c_0 est non nul, alors $AQ(A)b = b$ et $Q(A)b$ est une solution x du système $Ax = b$. Par contre, si c_0 est nul, ce qui implique que X divise $P_b(X)$, alors $AQ(A)b = 0$ et $Q(A)b$ est un élément du noyau de A qui est non nul par minimalité de $P_b(X)$. On peut donc soit trouver une solution de $Ax = b$, soit un élément non trivial du noyau avec cette méthode.

La complexité d'un calcul pour un b et un u donnés est celle nécessaire pour calculer $Q(A)$, ce qui nécessite n produits matrice-vecteur avec A. Mais ces produits ont déjà été calculés pour construire la séquence de Krylov. De plus, il est possible de vérifier la cohérence du résultat (si par exemple $P_{u,b}(X)$ est différent de $P_b(X)$) en testant si l'on obtient vraiment une solution ou un élément du noyau, ce qui demande seulement un produit par A.

Regardons maintenant le cas où l'on veut juste savoir si une matrice sur $\mathbf{F}_2$ est singulière ou non, ce qui va nous être utile pour savoir si une fonction booléenne admet ou non un annulateur de degré donné.

Proposition 14.1 (Preuve de la singularité)**.** Soit A une matrice $n \times n$ de $\mathbf{F}_2$, singulière, alors en appliquant une seule passe de l'algorithme de Wiedemann avec un choix aléatoire de b et de u on montrera que A est singulière avec une probabilité plus grande que $1/4$.

Démonstration. Déjà, quand A est une matrice singulière et que l'on connaît $P_b(X)$, on est sûr de trouver un élément non nul du noyau (et donc de prouver la singularité) du moment que b n'est pas dans l'image de A. Sur $\mathbf{F}_2$, cela arrive avec une probabilité d'au moins $1/2$.

Mais ici, il n'est pas nécessaire de connaître $P_b(X)$, il faut juste calculer la probabilité que X divise $P_{u,b}(X)$. Pour cela, décomposons $\mathbf{F}_2^n$ en utilisant les sous-espaces caractéristiques de A. On peut en particulier écrire $\mathbf{F}_2^n = E_0 \oplus E_1$ où E_0 est le sous-espace qui correspond à la valeur propre 0 et E_1 est tel que la restriction de A sur E_1 est inversible. Avec cette décomposition, écrivons $b = b_0 + b_1$. Soit P_0 et P_1 les polynômes minimaux associés respectivement aux séquences $(u.b_0, u.Ab_0, \dots)$ et $(u.b_1, u.Ab_1, \dots)$. On sait que P_0 est juste une puissance de X et que P_1 n'est pas divisible par X. Le polynôme minimal associé à la somme est donc le produit P_0P_1. On voit alors que X ne divise $P_{u,b}(X)$ que si $u.b_0$ est non nul. Ce qui arrive pour un choix aléatoire de u et de b avec une probabilité plus grande que $1/4$. □

Un algorithme pour prouver la singularité d'une matrice est alors le suivant :

Algorithme 14.2 (Test de singularité)**.** L'algorithme prend en entrée une matrice A carrée de taille $n \times n$ et essaie de savoir si elle est singulière ou pas. Il y a une chance que l'algorithme réponde non alors que A est singulière, cette probabilité dépend du nombre de passes i et est inférieure à $(3/4)^i$.

1. [Choix aléatoire] Choisir aléatoirement deux vecteurs de n bits, b et u.
2. [Berlekamp-Massey] Trouver le polynôme minimal qui engendre la séquence binaire $(u.b, u.Ab, \ldots, u.A^{2n}b)$.
3. [Singulière ?] Si le polynôme est divisible par X renvoyer `A singulière`.
4. [Inversible] Sinon et que $P_{u,b}(X)$ est de degré n renvoyer `A inversible`.
5. [Boucle] Tant que la probabilité d'échec est trop grande, retourner en 1.
6. [Probablement non] Renvoyer `A probablement inversible`.

Au niveau de la complexité, chaque passe requiert $2n$ produits matrice-vecteur avec A, $2n$ produits scalaires avec u et une exécution en $O(n^2)$ de l'algorithme de Berlekamp-Massey présenté au chapitre 7. L'implémentation de Berlekamp-Massey pour les séquences binaires est entièrement vectorisable et donc très efficace en pratique. La partie limitante est donc souvent le calcul des produits matrice-vecteurs. Il est possible, pour équilibrer un peu les diverses complexités, d'effectuer le calcul avec plusieurs vecteurs aléatoires u pour un même vecteur b. Sans beaucoup plus de temps de calcul on obtient ainsi un probabilité qui décroît en $(1/2)^i$ en fonction du nombre i de passes.

14.2 Conditionnement pour les matrices non carrées

Le cas d'une matrice carrée que nous avons vu dans la section précédente s'applique parfaitement dans le cas où l'on veut tester si une fonction avec un nombre impair de variables est d'immunité algébrique maximale ou non. Par contre, dans la plupart des autres cas, le système linéaire que l'on veut résoudre n'est pas carré et l'on est gêné car il est alors impossible de définir la séquence de Krylov.

Heureusement, il y a un moyen de contourner la difficulté en rendant la matrice carrée sans en changer les propriétés qui nous intéressent. Pour s'assurer qu'une fonction n'a pas d'annulateur par exemple, on doit conserver le rang de la matrice. Plus formellement, considérons une matrice A de taille $n \times k$ avec $k < n$. On va alors générer une matrice creuse Q de taille $k \times n$ de telle manière qu'avec forte probabilité le produit QA soit de rang k si A l'était. Une telle matrice est connue sous le nom de "préconditionneur" pour la matrice A.

D'après [Wie86] on peut construire Q de la manière suivante :

Proposition 14.3 (Préconditionnement)**.** Si A est une matrice binaire non singulière de taille $k \times n$ avec $k < n$, construisons une matrice binaire Q de la manière suivante. Un bit de la ligne i et d'une colonne entre 1 et k est mis à 1 avec une probabilité $w_i = \min(1/2, 2\log n/(k+1-i))$. Alors, avec une probabilité d'au moins 1/8, les deux propriétés suivantes sont vérifiées :

- La matrice QA de taille $k \times k$ est non singulière.
- Le poids de Hamming total des lignes de Q est d'au plus $2n(2+\log n)^2$.

Démonstration. Voir l'article de Wiedemann [Wie86]. □

Il existe aussi une autre construction de Q dans l'article [Wie86] telle que QA est non singulière avec une probabilité plus grande qu'un ε fixé et telle que le poids de Hamming total de Q est un $O(n \log n)$. C'est donc un résultat asymptotiquement meilleur, mais moins applicable en pratique car le ε est plus petit que le 1/8 du théorème précédent.

Une fois cette matrice Q construite, on retombe dans le cas d'une matrice carrée avec un peu plus de travail à chaque produit matrice-vecteur car on doit également calculer le produit avec la matrice Q. C'est pour cela que l'on cherche à générer une matrice Q la plus creuse possible de manière à rendre ce calcul efficace. Quand la matrice a un poids de Hamming de $O(n \log n)$ on peut ainsi faire un produit matrice-vecteur en $O(n \log n)$. Il faut également de l'ordre de $O(n \log n)$ mémoire en plus pour stocker cette matrice Q.

Pour déterminer si la matrice originale A est singulière ou non, l'algorithme est alors le suivant. On génère une matrice Q et l'on teste la singularité de QA avec l'algorithme de Wiedemann. Si la matrice est non singulière (avec la probabilité d'échec de l'algorithme de Wiedemann), alors A est non singulière aussi. Sinon, on peut recommencer i fois l'expérience avec différentes matrices Q. La probabilité de trouver i matrices QA singulières sachant que A ne l'est pas est alors plus petite que $(7/8)^i$.

On peut remarquer qu'avec peu de calculs supplémentaires, on peut calculer un élément non trivial du noyau de QA lorsque cette matrice est singulière. Si A est singulière, avec une probabilité d'au moins 1/8, cet élément sera en fait dans le noyau de A (c'est une conséquence de la proposition 14.3). On peut donc faire tourner l'algorithme jusqu'à ce que l'on soit sûr que A soit singulière (lorsque l'on trouve un élément dans le noyau de A, ce qui est facilement vérifiable). Si pour un calcul on trouve que QA est inversible, on peut aussi procéder à de nombreuses passes avec l'algorithme de Wiedemann pour avoir un probabilité très forte que A soit non singulière.

14.3 Application aux Reed-Muller et à l'immunité algébrique

Pour nous, l'intérêt essentiel de cette approche sur le décodage d'un code sur le canal à effacements est qu'elle s'applique parfaitement aux codes de

Reed-Muller et donc au calcul de l'immunité algébrique. En effet, rappelons l'expression donnée au chapitre 12 des matrices M_1 (de taille $|f| \times k$) et M_2 (de taille $2^m \times k$) qui interviennent dans le calcul des relations pour les attaques algébriques standards et rapides :

$$M_1 \stackrel{\text{def}}{=} V^r_{\{\mathbf{x} \in \mathbf{F}_2^m, f(\mathbf{x})=1\}},$$

$$M_2 \stackrel{\text{def}}{=} \text{Diag}(f(x)_{\mathbf{F}_2^m}) V^r_{\mathbf{F}_2^m} + V^e_{\mathbf{F}_2^m} \left[V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}} \text{Diag}(f(\mathbf{x})_{\{\mathbf{x},|\mathbf{x}|\leq e\}}) . V^r_{\{\mathbf{x},|\mathbf{x}|\leq e\}} \right]$$

On observe que ces deux définitions ne font intervenir pour calculer un produit matrice-vecteur que des produits par des matrices diagonales (qui se calculent linéairement) et par des matrices dites d'évaluation (les matrices V). Or, le calcul de ces dernières peut lui aussi se dérouler efficacement car ces matrices ne sont rien d'autre que des sous-matrices de matrices génératrices des codes de Reed-Muller. On obtient l'algorithme suivant :

Algorithme 14.4 (Produit matrice-vecteur rapide pour $V^r_{\mathcal{A}}$)**.** Étant donnés r, m et un sous-ensemble $\mathcal{A}$ de $\mathbf{F}_2^m$, cet algorithme calcule le produit matrice-vecteur de $V^r_{\mathcal{A}}$ par un vecteur $(b_1, \ldots, b_k)$ où k est la dimension de $\text{RM}(r, m)$.

1. [Mise en place] Construire un vecteur s de 2^m bits comme suit. Si $\mathbf{u}$ est le i-ème point de $\mathbf{F}_2^m$ de degré au plus r, alors $s_{\mathbf{u}} = b_i$, sinon $\mathbf{u}$ est de poids plus grand que r et l'on pose $s_i = 0$.
2. [Möbius] Calculer la transformée de Möbius rapide de s en place.
3. [Extraction] Le résultat est donné par les $s_{\mathbf{u}}$ avec $\mathbf{u}$ dans $\mathcal{A}$.

La complexité de cet algorithme est alors dominée par la transformée de Möbius qui se fait en $O(m2^m)$ comme expliqué dans l'annexe A. On notera cependant que les étapes 1 et 3 ne sont pas du tout négligeables car contrairement à la transformée de Möbius elles ne sont pas du tout vectorisables à l'aide des instructions bits à bits des processeurs modernes. On verra néanmoins un peu plus loin qu'il est possible de les effectuer efficacement dans le cas où on calcule la séquence de Krylov.

Dans le cas de la matrice $V^e_{\mathbf{x},|\mathbf{x}|\leq e}$ il est en fait possible de calculer la transformée de Möbius en $O(k' \log k')$ mais on perd alors l'aspect vectorisable du calcul qui est très important lorsque l'on implémente vraiment les algorithmes.

Le produit par une matrice transposée d'une matrice d'évaluation se calcule lui aussi très efficacement. Il suffit d'appliquer une sorte de transformée de Möbius transposée :

$$s_{\mathbf{x}} = \sum_{\mathbf{u} \supseteq \mathbf{x}} s_{\mathbf{u}} \ .$$

Ce qui revient dans l'étape de fusion de l'algorithme donné en annexe A à sommer la partie gauche sur la partie droite et non l'inverse. Le fait que

l'on puisse effectuer ce calcul rapidement rentre en fait dans le cadre d'un résultat général et assez surprenant :

Proposition 14.5 (Produit avec la matrice transposée)**.** La complexité du produit matrice-vecteur pour une matrice M est du même ordre de grandeur qu'avec la matrice transposée M^{T}.

Démonstration. La preuve est loin d'être triviale. C'est le principe de Tellegen [Tel52] encore appelé principe de transposition [KKB88]. □

Le calcul rapide d'un produit matrice-vecteur par la transposée présente deux applications assez utiles pour nous.

Enchaînement rapide des produits Lorsque l'on doit itérer plusieurs fois un produit matrice-vecteur comme dans le cas du calcul de la séquence de Krylov, la transposée de la transformation de Möbius peut nous être utile. En effet, si une matrice carrée (ou rendue carrée après conditionnement) est de rang plein, sa transposée également. Pour savoir si une fonction admet ou non des annulateurs d'un degré fixé, on peut alors effectuer les calculs avec $M_1 M_1^{\mathrm{T}}$ par exemple. L'intérêt est que les deux étapes 1 et 2 du calcul précédent ne sont alors plus nécessaires, il suffit juste d'appliquer un masque binaire pour mettre à 0 les positions hors des ensembles impliqués dans les matrices V. Tout le calcul du produit matrice-vecteur devient alors vectorisable ce qui nous permet d'améliorer considérablement les performances.

Utilisation dans l'algorithme du chapitre 13 On achève ici la preuve du lemme 13.16. Pour cela il suffit de montrer qu'étant donnée une équation sur les coefficients de la fonction h de la forme $b.\overline{h} = 0$, si $fg = h$ il est possible de la transférer sur les coefficients de g en $O(k' \log k')$.

Nous avons montré dans le chapitre 12 que si $fg = h$, il est possible d'exprimer les coefficients de h en fonction de ceux de g et des valeurs de f aux points de poids au plus e. La relation est donnée par $\overline{h} = M\overline{g}$ avec

$$M = V^e_{\{\mathbf{x},|\mathbf{x}|\leq e\}} \mathrm{Diag}(f(\mathbf{x})_{\{\mathbf{x},|\mathbf{x}|\leq e\}}) V^r_{\{\mathbf{x},|\mathbf{x}|\leq e\}} \ .$$

Grâce à la transposée de M, il est alors possible d'exprimer une relation que doivent satisfaire les coefficients de h en fonction de ceux de g. Par exemple si l'on veut $b.\overline{h} = 0$, cela revient à écrire $b.M\overline{g} = 0$ et $b.M$ (au sens du produit de M par un vecteur ligne à gauche) se calcule par $M^{\mathrm{T}}b$. Étant donnée la forme de la matrice M, le tout demande une complexité en $O(k' \log k')$ car on ne travaille que sur les points de poids au plus e.

14.4 Résumé et performances

Nous avons mis en évidence comment utiliser des algorithmes issus du monde de l'algèbre linéaire creuse pour décoder efficacement sur le canal

à effacements des codes qui s'encodent rapidement. En appliquant cette approche aux codes de Reed-Muller, on obtient ainsi un algorithme de calcul de l'immunité algébrique très performant qui fonctionne également dans le cas des attaques rapides.

Dans les deux cas, l'algorithme se ramène à l'utilisation de l'algorithme de Wiedemann sur des matrices $k \times k$ pour lesquelles un produit matrice-vecteur se calcule en $O(m2^m)$. La complexité finale est donc de $O(mk2^m)$ aussi bien pour les attaques algébriques normales que pour les versions rapides. Pour obtenir de très faibles probabilités d'échec, il convient d'effectuer plusieurs passes de l'algorithme de Wiedemann (notamment pour les attaques rapides où il faut en plus essayer plusieurs matrices Q de préconditionnement). Mais pour obtenir une probabilité fixée, le nombre de passes est plutôt faible et indépendant de n, il n'influe donc pas sur la complexité asymptotique.

Nous avons implémenté cette approche en C et donnons maintenant les performances obtenues sur un Pentium 4 cadencé à 3.2Ghz et possédant 2Gb de mémoire vive.

Lorsque l'on veut tester l'immunité algébrique maximale d'une fonction booléenne, on est alors dans le cas d'une matrice carrée et nous avons vu qu'il est possible d'obtenir une version entièrement vectorisable de l'algorithme. On peut voir dans la table 14.1 que l'on obtient alors un algorithme très efficace. Le temps est donné pour une seule passe de l'algorithme de Wiedemann avec un choix aléatoire de b et 4 choix pour u. La probabilité d'échec est donc relativement grande, mais les passes étant complètement parallélisables sur d'autres machines, on obtient quand même une bonne mesure des performances.

$\boldsymbol{r, m}$	4,9	5,11	6,13	7,15	8,17	9,19	10,21	11,23	12,25
Temps	$< 1s$	$< 1s$	0.01s	0.3s	5s	102s	30m	11h	20j

TAB. 14.1 – Temps de calcul dans le cas carré pour tester l'immunité algébrique maximale r d'une fonction équilibrée à $m = 2r + 1$ variables. Le temps est indépendant de la fonction et nous avons utilisé une version entièrement vectorisée du produit matrice-vecteur qui repose sur une transformée de Möbius rapide implémentée en 128bits avec le jeu d'instructions SSE2.

Dans le cas non carré, les résultats sont beaucoup moins impressionnants et sont donnés dans la table 14.2. Les temps sont donnés pour une seule passe avec une matrice Q aléatoire, un b et 4 u. Pour avoir une probabilité d'échec faible, il faut exécuter cet algorithme plusieurs fois (16 donne une probabilité d'erreur de 0.1 et 32 de 0.01).

La raison de la différence de performances entre le cas carré et le cas général provient de l'impossibilité d'effectuer la multiplication par Q de

façon vectorisable, on perd donc de l'ordre d'un facteur 32 voir plus dans le temps de calcul. Pour résoudre ce problème, il conviendrait d'utiliser la version par bloc de l'algorithme de Wiedemann ([Cop94, Tho03]) que nous mentionnons juste brièvement ici. L'idée est de calculer la séquence de Krylov pour de nombreux u et b d'un coup, ce qui peut se faire rapidement avec les instructions vectorielles des processeurs modernes. On applique alors une version plus évoluée de l'algorithme de Berlekamp-Massey qui va utiliser toutes ces informations et nous permettre de diminuer le nombre de produits matrice-vecteur nécessaires.

r, m	2,23	3,19	3,20	3,21	4,19	5,19
k	277	1160	1351	1562	5036	16664
Temps	264s	100s	252s	630s	640s	2706s
Mémoire	1397Mb	118Mb	255Mb	547Mb	160Mb	194Mb

TAB. 14.2 – Temps et mémoire requis pour le calcul de l'immunité contre les attaques algébriques standards.

Par contre, ce qui est intéressant c'est que le temps de calcul dans le cas des attaques algébriques rapides est pratiquement le même que pour les attaques standards comme nous le montrent les deux dernières tables. Le degré e a seulement une petite influence car seule la taille de Q en dépend. Mais quelle que soit sa valeur, le temps et la mémoire utilisés ne dépassent pas le cas où e vaut $n/2$ de plus d'un facteur 2.

$r/e, m$	3/8,17	3/9,19	3/10,21	4/8,17	5/8,17	6/8,17
k	834	1160	1562	3214	9402	21778
Temps	15s	101s	614s	82s	297s	801s
Mémoire	25Mb	118Mb	547Mb	33Mb	40Mb	45Mb

TAB. 14.3 – Temps et mémoire requis pour le calcul de l'immunité contre les attaques algébriques rapides. On a pris ici $m = 2e + 1$.

$r/e, m$	3/7,19	3/8,19	3/9,19	3/10,19	3/11,19
Temps	154s	130s	101s	70s	43s
Mémoire	192Mb	159Mb	118Mb	77Mb	43Mb

TAB. 14.4 – Dépendance du calcul de l'immunité contre les attaques rapides en fonction de e. Dans tous les cas, k vaut ici 1160.

Conclusion et perspectives

Nous avons décrit dans cette thèse la plupart des approches actuellement connues pour attaquer un LFSR filtré par une fonction non linéaire. Au cours de cette étude, nous avons vu plusieurs problèmes qui peuvent se formuler aussi bien en termes de cryptographie que de codes correcteurs. Cela n'est pas si étonnant étant donné que ces deux disciplines utilisent le même genre d'outils mathématiques et algorithmiques, mais l'éclairage légèrement différent apporté par chacune d'elles s'est révélé fructueux et nous a ainsi permis d'obtenir des résultats nouveaux.

Nous avons ainsi établi une nouvelle borne sur la probabilité d'erreur après décodage sur le canal à effacements et introduit une nouvelle propriété d'un code linéaire qui est la cohérence du rendement. Appliquée aux codes de Reed-Muller, cette borne nous a permis de montrer que l'immunité algébrique d'une fonction booléenne aléatoire était presque optimale. Nous avons enfin trouvé deux approches efficaces pour calculer cette immunité algébrique sur une fonction donnée.

Sans forcément utiliser le lien avec le domaine des codes correcteurs, on a aussi montré comment on peut utiliser un calcul de logarithme discret pour trouver des équations de parité de poids faible vérifiées par une récurrence linéaire. Enfin, nous avons décrit et analysé rigoureusement une nouvelle attaque sur le registre filtré qui utilise ces équations. Il s'agit d'une attaque assez générique et certainement l'une des plus efficaces connues.

Nous revenons maintenant sur trois points qui nous semblent particulièrement intéressants.

Le registre filtré

Malgré l'étendue des travaux menés sur le registre filtré et le nombre d'attaques connues, cette construction de chiffrement à flot pourtant assez simple offre une sécurité importante. Ainsi, s'il est correctement dimensionné, il peut offrir des performances honorables tout en étant à l'abri des attaques connues effectuées sur les machines actuelles.

Mentionnons que le registre filtré est certainement la construction la plus sûre parmi les générateurs aléatoires utilisant une fonction de mise à jour de l'état interne linéaire. Pour une même taille de mémoire, l'expérience

montre en effet que des constructions qui utilisent plusieurs registres sont moins bonnes.

À ce jour, l'attaque la plus efficace connue pour retrouver l'état initial est peut être celle du chapitre 6, si le nombre de variables de la fonction de filtrage reste raisonnable. Viennent ensuite les attaques par compromis "temps-mémoire-longueur de suite chiffrante" qui ne dépendent que de la taille de registre utilisé. Enfin, les attaques de nature algébrique peuvent s'avérer très efficaces sur certains systèmes mais il est possible de construire des fonctions de filtrage qui rendent leur complexité importante. Pour cela, les algorithmes de calcul de l'immunité algébrique présentés aux chapitres 13 et 14 peuvent être utiles.

Probabilité d'erreur sur le canal à effacements

L'analyse du comportement des codes de Reed-Muller sur le canal à effacements soulève de nombreux problèmes que nous trouvons très intéressants. Au chapitre 11 nous avons réussi à exploiter l'information contenue dans les distances généralisées de ces codes pour arriver à une majoration de la probabilité d'erreur, mais plusieurs questions restent ouvertes.

En particulier, on ne connaît pas de borne précise sur le nombre d'ensembles d'information de ces codes même s'il semblerait qu'il y en ait une proportion constante. En fait, à notre connaissance, peu de travaux portent sur le problème combinatoire de dénombrer le nombre d'ensembles d'information d'un code linéaire.

Calcul de l'immunité algébrique

Le problème de calculer l'immunité algébrique d'une fonction booléenne efficacement (ce qui correspond comme on l'a vu au chapitre 12 à inverser une matrice avec une structure particulière) a connu très récemment plusieurs avancées et nous semble maintenant résolu de manière presque optimale.

Au début de cette thèse, nous ne savions pas exploiter la structure particulière de ce système linéaire et les algorithmes génériques étaient de complexité cubique. Vu la structure du système, cela nous semblait néanmoins possible d'y arriver en un temps quadratique, ce qui a été ensuite confirmé dans [ACG+06] même si quelques points de leur algorithme restent flous.

Depuis, nous avons trouvé une autre approche qui donne un algorithme essentiellement quadratique présenté dans le chapitre 14. Cette approche est assez intéressante car elle montre que si un code est encodable efficacement, alors il est également possible de le décoder rapidement sur le canal à effacements.

Aujourd'hui, avec les algorithmes découverts ces dernières années, il est tout à fait possible de calculer l'immunité algébrique efficacement pour le

nombre de variables et le degré de sécurité requis dans les systèmes de chiffrement à flot.

Annexe A

Transformée de Möbius et de Walsh rapide

La transformée de Möbius et la transformée de Walsh peuvent être calculées rapidement par des algorithmes très similaires. Dans les deux cas, on utilise une propriété récursive de ces transformées pour réduire la complexité de calcul qui se ramène à $O(m2^m)$ pour des fonctions sur m variables. L'algorithme naïf étant lui quadratique en $O(2^{2m})$.

Algorithme A.1 (Transformée de Möebius rapide)**.** Étant donné le vecteur de 2^m éléments de $\mathbf{F}_2$ des valeurs d'une fonction booléenne f, cet algorithme renvoie le vecteur des coefficients de l'ANF de f, i.e. les $f_{\mathbf{u}}$ pour $\mathbf{u}$ décrivant $\mathbf{F}_2^m$.

1. [Stop ?] Si $m = 1$ alors renvoyer $(f(\mathbf{0}), f(\mathbf{0}) + f(\mathbf{1}))$.
2. [Récursion droite] Faire une transformée de Möbius de la fonction à $m-1$ variables $f^{(0)}$ dont les valeurs sont données par la première moitié du vecteur de valeurs de f, c'est-à-dire les $f(\mathbf{x})$ avec $\mathbf{x} = (x_1, \ldots, x_m)$ et $x_m = 0$.
3. [Récursion gauche] Faire une transformée de Möbius de la fonction à $m-1$ variables $f^{(1)}$ dont les valeurs sont données par la deuxième moitié du vecteur de valeurs de f, c'est-à-dire les $f(\mathbf{x})$ avec $\mathbf{x} = (x_1, \ldots, x_n)$ et $x_m = 1$.
4. [Fusion] On a

$$f_{(u_1,\ldots,u_m)} = f^{(0)}_{(u_1,\ldots,u_{m-1})} + u_m f^{(1)}_{(u_1,\ldots,u_{m-1})} \ .$$

La transformée de Möbius étant involutive, on peut appliquer exactement le même algorithme pour obtenir le vecteur des valeurs d'une fonction booléenne en fonction des coefficients de son ANF. C'est d'ailleurs plutôt ce sens que l'on utilise pour générer efficacement des mots de code du Reed-Muller et qui nous sert au chapitre 14 pour décoder efficacement ce code sur le canal à effacements.

Algorithme A.2 (Transformée de Walsh rapide)**.** Étant donné un vecteur de 2^m éléments de $\mathbf{R}$ représentant une fonction W de $\mathbf{F}_2^n$ dans $\mathbf{R}$, cet algorithme renvoie le vecteur des valeurs de $\widehat{W}$.

1. [Stop ?] Si $m = 1$ alors renvoyer $(W(\mathbf{0}) + W(\mathbf{1}), W(\mathbf{0}) - W(\mathbf{1}))$.
2. [Récursion droite] Faire une transformée de Walsh de la fonction à $m-1$ variables $W^{(0)}$ dont les valeurs sont données par la première moitié du vecteur de valeurs de W, c'est-à-dire les $W(\mathbf{x})$ avec $\mathbf{x} = (x_1, \ldots, x_m)$ et $x_m = 0$.
3. [Récursion gauche] Faire une transformée de Walsh de la fonction à $m-1$ variables $W^{(1)}$ dont les valeurs sont données par la deuxième moitié du vecteur de valeurs de W, c'est-à-dire les $W(\mathbf{x})$ avec $\mathbf{x} = (x_1, \ldots, x_m)$ et $x_m = 1$.
4. [Fusion] On a

$$\widehat{W}(u_1, \ldots, u_m) = \widehat{W^{(0)}}(u_1, \ldots, u_{m-1}) + (-1)^{u_m} \widehat{W^{(1)}}(u_1, \ldots, u_{m-1}) \ .$$

La complexité en $O(m2^m)$ pour chacun de ces algorithmes découle du fait qu'à chaque appel on applique l'algorithme sur deux problèmes de taille deux fois plus petite. La preuve est facile du moment que l'on montre que l'égalité à l'étape 4 de chacun des algorithmes est vérifiée.

Pour la transformée de Möebius, cela découle directement de la définition donnée au théorème 3.11 qui peut s'écrire :

$$f_{\mathbf{u}} = \sum_{\mathbf{x} \subseteq \mathbf{u}} f(\mathbf{x}) = \sum_{\mathbf{x} \subseteq \mathbf{u}, x_m = 0} f(\mathbf{x}) + \sum_{\mathbf{x} \subseteq \mathbf{u}, x_m = 1} f(\mathbf{x}) \ .$$

Comme la seconde somme vaut zero si u_m vaut 1, on peut réécrire tout ça :

$$f_{\mathbf{u}} = \sum_{(x_1, \ldots, x_{m-1}) \subseteq (u_1, \ldots, u_{m-1})} f((x_1, \ldots, x_{m-1}, 0)) \ + \ u_m \sum_{(x_1, \ldots, x_{m-1}) \subseteq (u_1, \ldots, u_{m-1})} f((x_1, \ldots, x_{m-1}, 1))$$

d'où le résultat.

On a pratiquement la même chose pour la transformée de Walsh en utilisant la définition 3.12 qui peut s'écrire :

$$\widehat{W}(\mathbf{u}) = \sum_{\mathbf{x} \in \mathbf{F}_2^m, x_m = 0} W(\mathbf{x})(-1)^{\mathbf{u} \cdot \mathbf{x}} + \sum_{\mathbf{x} \in \mathbf{F}_2^m, x_m = 1} W(\mathbf{x})(-1)^{\mathbf{u} \cdot \mathbf{x}} \ .$$

La contribution de u_m est alors uniquement un facteur 1 pour la première somme et un facteur $(-1)^{u_m}$ pour l'autre.

Annexe B

Quelques résultats sur la loi binomiale

Nous avons utilisé à plusieurs reprises, notamment dans les chapitres 6, 11 et 13, des résultats sur la loi binomiale. Nous effectuons dans cette annexe une rapide présentation de plusieurs résultats assez utiles sur le sujet.

Définition B.1 (Loi binomiale)**.** La loi binomiale de paramètre n et p correspond à la somme S_n de n variables aléatoires indépendantes $X_1, \ldots, X_n$ qui prennent la valeur 1 avec probabilité p et la valeur 0 avec probabilité $1-p$.

La probabilité d'obtenir chacune des valeurs possibles est parfaitement connue et est donnée par :

$$\Pr(S_n = i) = \binom{n}{i} p^i (1-p)^{n-i} \ .$$

Le symbole $\binom{n}{i}$ noté parfois C_n^i joue un rôle majeur en combinatoire et correspond au nombre de façons de choisir i éléments parmi n. On a

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \ .$$

Il est souvent utile de connaître le comportement asymptotique de cette quantité. Le résultat à utiliser est alors la formule de Stirling qui nous donne le comportement asymptotique de la fonction factorielle :

Proposition B.2 (Formule de Stirling)**.** Asymptotiquement quand n tend vers l'infini on a :

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \ .$$

Il existe en fait un encadrement précis de la factorielle ce qui permet d'écrire des formules exactes :

Proposition B.3 (Encadrement de $n!$ [Fel71])**.** On a l'encadrement suivant pour $n!$:

$$1 \leq \frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} \leq e^{\frac{1}{12n}} \ .$$

En utilisant ces propositions, il est alors possible d'en déduire le comportement d'un coefficient binomial $\binom{n}{k}$ pour k/n fixé :

Proposition B.4 (Comportement d'un coefficient binomial)**.** On a

$$\binom{n}{\lambda n} \sim \frac{2^{h(\lambda)n}}{\sqrt{\lambda(1-\lambda)2\pi n}}$$

où h est la fonction d'entropie binaire

$$h(\lambda) = -\lambda \log_2(\lambda) - (1-\lambda)\log_2(1-\lambda) \ .$$

Démonstration. On utilise la formule de Stirling :

$$\frac{n!}{(\lambda n)!((1-\lambda)n)!} \sim \frac{\sqrt{2\pi n}n^n}{e^n} \frac{e^{\lambda n}}{\sqrt{2\pi\lambda n}(\lambda n)^{\lambda n}} \frac{e^{(1-\lambda)n}}{\sqrt{2\pi(1-\lambda)n}((1-\lambda)n)^{(1-\lambda)n}} \ .$$

Les puissances de e s'annulent, et en écrivant que $a^b = 2^{b\log_2 a}$ il ne reste plus que

$$\binom{n}{\lambda n} \sim \sqrt{\frac{1}{\lambda(1-\lambda)2\pi n}} 2^{n\log_2 n - \lambda n \log_2(\lambda n) - (1-\lambda)n\log_2((1-\lambda)n)} \ .$$

Et en sortant le n des logarithmes on obtient finalement

$$\binom{n}{\lambda n} \sim \frac{1}{\sqrt{\lambda(1-\lambda)2\pi n}} 2^{-\lambda n \log_2(\lambda) - (1-\lambda)n\log_2(1-\lambda)} \ ,$$

d'où le résultat. □

Un cas assez intéressant pour nous est le cas d'une loi binomiale de m variables et de paramètre p égal à $1/2$. L'essentiel de la masse de la distribution de cette loi est concentré autour de la moyenne qui correspond à $m/2$. Pour $\lambda = 1/2$ on a ainsi :

$$\binom{m}{m/2} \sim \sqrt{\frac{2}{\pi m}}\ 2^m \ .$$

Pour une telle loi, la dimension k du code $\mathrm{RM}(r, m)$ est en fait donnée par

$$k = 2^m Pr(S_m \leq r) \ .$$

Lorsque r est fixé, k est un $O(m^r)$ (résultat dont on s'est servi au chapitre 13). Cela découle directement de la majoration suivante :

$$k = \sum_{i=0}^{r} \binom{m}{i} \leq \sum_{i=0}^{r} m^i = \frac{m^{r+1}-1}{m-1} = O(m^r)$$

De manière plus générale, on a le lemme suivant :

Lemme B.5 (majoration d'une somme de binômes)**.** Pour $r \leq m/2$ on a toujours

$$\sum_{i=0}^{r} \binom{m}{i} \leq 2^{mh(\frac{r}{m})} \ .$$

Démonstration. Notons $\lambda = r/m$. On a

$$\sum_{i=0}^{r} \binom{m}{i} \leq \left(\frac{1-\lambda}{\lambda}\right)^{\lambda m} \left[\sum_{i=0}^{m} \binom{m}{i} \left(\frac{\lambda}{1-\lambda}\right)^i\right]$$

Le terme entre crochets n'est rien d'autre que l'expension de $(1+\lambda/(1-\lambda))^m$, on en déduit que

$$\sum_{i=0}^{r} \binom{m}{i} \leq \left(\frac{1-\lambda}{\lambda}\right)^{\lambda m} \left(\frac{1}{1-\lambda}\right)^m = 2^{mh(\lambda)}$$

d'où le résultat. □

On termine cette annexe par un théorème de concentration de la loi binomiale autour de son espérance dont on s'est servi au chapitre 6 et au chapitre 11.

Proposition B.6 (Borne de Chernoff [1])**.** Soit $0 < p < 1$, soient $X_1, \ldots, X_n$ n variables aléatoires indépendantes qui prennent la valeur 1 avec probabilité p et 0 avec la probabilité $1-p$ et soit $S_n = \sum_{i=1}^{n} X_i$. Alors pour tout $t \geq 0$,

$$\Pr(\mid S_n - np \mid \geq nt) \leq 2e^{-2nt^2} \ .$$

Si l'on veut la borne d'un seul côté (sans la valeur absolue), on peut enlever le facteur 2 dans le membre de gauche.

[1] Voir par exemple [Gal68] section 5.4.

Bibliographie

[ABMV93] G.B. Agnew, T. Beth, R.C. Mullin, and S.A. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6(1) :3–13, 1993.

[ACG+06] F. Armknecht, C. Carlet, P. Gaborit, S. Künzli, W. Meier, and O. Ruatta. Efficient computation of algebraic immunity for algebraic and fast algebraic attacks. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 147–164. Springer-Verlag, 2006.

[AFI+04] G. Ars, J.-C. Faugère, H. Imai, M. Kawazoe, and M. Sugita. Comparison between XL and Gröbner basis algorithms. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer-Verlag, 2004.

[And94] Anderson. Searching for the optimum correlation attack. In *IWFSE : International Workshop on Fast Software Encryption, LNCS*, 1994.

[Arm04] Frederik Armknecht. Improving fast algebraic attacks. In *FSE*, volume 3017 of *LNCS*, pages 65–82, 2004.

[Ass92] Assmus. On the Reed-Muller codes. *DMATH : Discrete Mathematics*, 107, 1992.

[Bab95] S. Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, number 408 in IEEE Conference Publication, 1995.

[BCJ07] Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. Cryptology ePrint Archive, Report 2007/024, 2007. http ://eprint.iacr.org/.

[Ber68] E.R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[Ber82] Elwyn R. Berlekamp. Bit-serial Reed-Solomon encoders. *IEEE Transactions on Information Theory*, 28(6) :869–874, 1982.

[BFSY04] M. Bardet, J-C. Faugère, B. Salvy, and B-Y. Yang. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proc. International Conference on Polynomial System Solving (ICPSS'2004)*, 2004.

[BFSY05] M. Bardet, J-C. Faugère, B. Salvy, and B-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *MEGA 2005*, Porto Conte, Italy, May 2005.

[BLP06] An Braeken, Joseph Lano, and Bart Preneel. Evaluating the resistance of stream ciphers with linear feedback against fast algebraic attacks. *ACISP 06*, 2006.

[BM04] David Burshtein and Gadi Miller. An efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel. *IEEE Transactions on Information Theory*, 50(11) :2837–2844, 2004.

[BP05] An Braeken and Bart Preneel. On the algebraic immunity of symmetric Boolean functions. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *INDOCRYPT*, volume 3797 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2005.

[BS00] A. Biryukov and A. Shamir. Cryptanalytic time-memory-data trade-offs for stream ciphers. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2000.

[Can02] Anne Canteaut. Le chiffrement à la volée. *Pour la science*, 2002.

[Can06] Anne Canteaut. Analyse et conception de chiffrements à clef secrète. *Habilitation à diriger des recherches*, 2006.

[Car07] Claude Carlet. Constructing balanced functions with optimum algebraic immunity. *IEEE International Symposium on Information Theory*, June 2007.

[CF01] A. Canteaut and E. Filiol. Ciphertext only reconstruction of stream ciphers based on combination generators. In *Fast Software Encryption - FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2001.

[CF02] A. Canteaut and E. Filiol. On the influence of the filtering function on the performance of fast correlation attacks on filter generators. In *23rd Symposium on Information Theory in the Benelux*, Louvain-la-Neuve, Belgium, May 2002.

[CG05] Claude Carlet and Philippe Gaborit. On the construction of balanced Boolean functions with a good algebraic immunity. In *Proceedings of BFCA (First Workshop on Boolean Functions : Cryptography and Application)*, pages 1–14, Rouen, France, March 2005.

[CHJ02] D. Coppersmith, S. Halevi, and C. Jutla. Scream : a software-efficient stream cipher. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[CJM02] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks : an algorithmic point of view. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[CJS00] V. Chepyshov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2000.

[CL01] Anne Canteaut and Françoise Lévy-dit-Véhel. La cryptographie moderne. *Revue Armement*, 2001.

[CL05] C. Cid and G. Leurent. An analysis of the XSL algorithm. In *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 333–352. Springer, 2005.

[CM03] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[Cop84a] Don Coppersmith. Evaluating logarithms in GF(2n). In *STOC '84 : Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 201–207, New York, NY, USA, 1984. ACM Press.

[Cop84b] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4) :587–593, 1984.

[Cop94] Don Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.*, 62(205) :333–350, January 1994.

[COS86] D. Coppersmith, A. Odlyzko, and R. Schroeppel. Discrete logarithms in GF(p). *Algorithmitica*, 1 :1–15, 1986.

[Cou03] N. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - CRYPTO 2003*,

volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer-Verlag, 2003.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. *Asiacrypt 2002*, LNCS 2501, 2002. http ://eprint.iacr.org/2002/044.

[CT00] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[CV05] A. Canteaut and M. Videau. Symmetric Boolean functions. *IEEE Transactions on Information Theory*, 51(8) :2791–2811, August 2005.

[Den82] D.E. Denning. *Cryptography and data security*. Addison-Wesley, 1982.

[DGM04] Deepak Kumar Dalai, Kishan Chand Gupta, and Subhamoy Maitra. Results on algebraic immunity for cryptographically significant Boolean functions. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2004.

[DGM05] Deepak Kumar Dalai, Kishan Chand Gupta, and Subhamoy Maitra. Cryptographically significant Boolean functions : Construction and analysis in terms of algebraic immunity. In *Fast Software encryption (FSE)*, volume 3557 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[Did06a] Frédéric Didier. A new bound on the block error probability after decoding over the erasure channel. *IEEE Transactions on Information Theory*, 52(10) :4496–4503, October 2006.

[Did06b] Frédéric Didier. Using Wiedemann's algorithm to compute the immunity against algebraic and fast algebraic attacks. *Indocrypt 2006*, 4329 :236–250, December 2006.

[Did07] Frédéric Didier. Attacking the filter generator by finding zero inputs of the filtering function. *Indocrypt 2007*, 2007.

[Die04] C. Diem. The XL algorithm and a conjecture from commutative algebra. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337. Springer-Verlag, 2004.

[DLC07] Frédéric Didier and Yann Laigle-Chapuy. Finding low-weight polynomial multiples using discrete logarithm. *IEEE International Symposium on Information Theory - ISIT 2007*, 2007.

[DM06] Deepak Kumar Dalai and Subhamoy Maitra. Reducing the number of homogeneous linear equations in finding annihilators. *SETA 2006*, 2006.

[DMS05] Deepak Kumar Dalai, Subhamoy Maitra, and S. Sarkar. Basic theory in construction of Boolean functions with maximum possible annihilator immunity. In *Designs, Codes and Cryptography*, volume 40, number 1, pages 41–58, July 2005. Also available at Cryptology ePrint Archive, http ://eprint.iacr.org/, No. 2005/229, 15 july, 2005.

[Dor87] Jean Louis Dornstetter. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Transactions on Information Theory*, 33(3) :428–, 1987.

[DT06] Frédéric Didier and Jean-Pierre Tillich. Computing the algebraic immunity efficiently. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2006.

[Dum04] Ilya Dumer. Recursive decoding and its performance for low-rate Reed-Muller codes. *IEEE Trans. Info. Theory*, 50(5) :811–823, 2004.

[Dum06] Ilya Dumer. Soft decision decoding of Reed-Muller codes : a simplified algorithm. *IEEE Trans. Info. Theory*, 52(3) :954–963, 2006.

[EJ04] Hakan Englund and Thomas Johansson. A new simple technique to attack filter generators and related ciphers. *Selected Areas in Cryptography*, LNCS 3357 :39–53, 2004.

[FA03] J.-C. Faugère and G. Ars. An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases. Technical Report RR-4739, INRIA, 2003.

[Fau99] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1-3) :61–88, 1999.

[Fau02] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*. ACM, 2002.

[Fel71] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. John Wiley & Sons, 1971.

[FGLM93] J. C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Grobner bases by change of ordering. *J. Symb. Comput.*, 16(4) :329–344, 1993.

[FJ03] J.-C. Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In *Advances in Cryptology-CRYPTO 2003*, volume 2729 of *LNCS*, pages 44–60. Springer Verlag, 2003.

[FM07] Simon Fischer and Willi Meier. Algebraic immunity of S-boxes and augmented functions. In *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 366–381, 2007.

[Gal68] Robert G. Gallager. *Information Theory and Reliable Communication*. John Wiley & Sons, Inc., New York, NY, USA, 1968.

[GCD00] J. Golic, A. Clark, and E. Dawson. Generalized inversion attack on nonlinear filter generators. *IEEE Transactions on Computers*, 49(10) :1100–1109, 2000.

[GH05] Jovan Dj. Golic and Philip Hawkes. Vectorial approach to fast correlation attacks. *Des. Codes Cryptography*, 35(1) :5–19, 2005.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89 : Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, New York, NY, USA, 1989. ACM.

[Gol94] J. Golic. Linear cryptanalysis of stream ciphers. In *Fast Software Encryption - FSE'94*, volume 1008 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Gol96] J. Golic. On the security of nonlinear filter generators. In *Fast Software Encryption - FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 1996.

[Gol97] J. Golic. Cryptanalysis of alleged A5 stream cipher. In *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.

[Gou04] Aline Gouget. *Etude des propriétés cryptographiques des fonctions boolèennes et algorithme de confusion pour le chiffrement symétrique*. PhD thesis, Université de Caen Basse-Normandie, 2004.

[GRS95] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries : the highly noisy case. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 294–303. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[GY79] F. Gustavson and D. Yun. Fast algorithms for rational Hermite approximation and solution of Toeplitz systems. *IEEE Transactions on Circuits and Systems*, 26(9), 1979.

[Hel80] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4) :401–406, 1980.

[HKL97] Tor Helleseth, Torleiv Kløve, and Vladimir I. Levenshtein. On the information function of an error-correcting code. *IEEE Trans. Inform. Theory*, 43(2) :549–557, 1997.

[HR04] P. Hawkes and G. G. Rose. Rewriting variables : the complexity of fast algebraic attacks on stream ciphers. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer-Verlag, 2004.

[HS05] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 353–372. Springer, 2005.

[Hub90] Klaus Huber. Some comments on Zech's logarithms. *IEEE Transactions on Information Theory*, 36(4) :946–, 1990.

[JJ00] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *CRYPTO '00 : Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 300–315. Springer-Verlag, 2000.

[JJ02] F. Jönsson and T. Johansson. A fast correlation attack on LILI-128. *Information Processing Letters*, 81(3) :127–132, February 2002.

[Jön02] F. Jönsson. *Some results on fast correlation attacks*. PhD thesis, Lund University, Sweden, 2002.

[Key76] E. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transaction on information theory*, 22(6) :732–736, 1976.

[KKB88] Michael Kaminski, David G. Kirkpatrick, and Nader H. Bshouty. Addition requirements for matrix and transposed matrix products. *Journal of Algorithms*, 9 :354–364, 1988.

[KMM05] J.D. Key, T.P. McDonough, and V.C. Mavron. Information sets and partial permutation decoding for codes from finite geometries. *Finite Fields and their Applications*, 2005. A paraître.

[KT70] T. Kasami and N. Tokura. On the weight structure of Reed-Muller codes. *IEEE Trans. on Inform. Theory*, 16(6) :752–759, 1970.

[KT04] G. Kabatiansky and C. Tavernier. List decoding with Reed-Muller codes of order one. *nine International Workshop On Algebraic and Combinatorial Coding Theory, Proceedings ACCT-9*, pages 230–236, June 2004.

[KT06] G. Kabatiansky and C. Tavernier. List decoding of first order Reed-Muller codes ii. *tenth International Workshop on Algebraic and Combinatorial Coding Theory, Proceedings ACCT-10*, September 2006.

[KTA74] T. Kasami, N. Tokura, and S. Asumi. On the weight enumeration of weights less than 2.5d of Reed-Muller codes. *Information and control*, 30(4) :380–395, 1974.

[Lau07] C. Lauradoux. *Conception et synthèse en cryptographie symétrique*. Thèse de doctorat, École Polytechnique, Palaiseau, june 2007.

[LBGZ01] S. Leveiller, J. Boutros, P. Guillot, and G. Zémor. Cryptanalysis of nonlinear filter generators with $\{0,1\}$-metric Viterbi decoding. In *Cryptography and Coding - 8th IMA International Conference*, volume 2260 of *Lecture Notes in Computer Science*, pages 402–414. Springer-Verlag, 2001.

[Lev04a] Sabine Leveiller. A new algorithm for cryptanalysis of filtered LFSRs : the "probability-matching" algorithm. In *ISIT 2004*, page 234, 2004.

[Lev04b] Sabine Leveiller. *Quelques algorithmes de cryptanalyse du registre filtré*. PhD thesis, Ecole nationale supérieure des télécommunication, ENST, France, 2004.

[LN83] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its application*. Cambridge University press, 1983.

[Lup58] Oleg B. Lupanov. A method of circuit sythesis. *Izvesitya VUZ, Radiofiz*, 1 :120–140, 1958.

[LV04] Yi Lu and Serge Vaudenay. Faster correlation attack on bluetooth keystream generator E0. In *CRYPTO*, pages 407–425, 2004.

[LZGB03] S. Leveiller, G. Zémor, P. Guillot, and J. Boutros. A new cryptanalytic attack for PN-generators filtered by a Boolean function. In *Selected areas in cryptography - SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 2003.

[Mas69] J.L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15 :122–127, janvier 1969.

[Mas01] James L. Massey. The ubiquity of Reed-Muller codes. In *AAECC-14 : Proceedings of the 14th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 1–12, London, UK, 2001. Springer-Verlag.

[MC98] Samuel J. MacMullan and Oliver M. Collins. A comparison of known codes, random codes, and the best codes. *IEEE Trans. Inform. Theory*, 44(7) :3009–3022, 1998.

[McE87] R.J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.

[MFI01] Miodrag J. Mihaljevic, Marc P. C. Fossorier, and Hideki Imai. A low-complexity and high-performance algorithm for the fast correlation attack. *Lecture Notes in Computer Science*, 1978 :45–60, 2001.

[MH04] Havard Molland and Tor Helleseth. An improved correlation attack against irregular clocked and filtered keystream generators. In *CRYPTO*, pages 373–389, 2004.

[MPC04] W. Meier, E. Pasalic, and C. Carlet. Algebraic attacks and decomposition of Boolean functions. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer-Verlag, 2004.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The theory of error-correcting codes*, volume 16. North holland mathematical libray, 1977.

[MS88] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1988.

[MS02] Subhamoy Maitra and Palash Sarkar. Maximum nonlinearity of symmetric Boolean functions on odd number of variables. *IEEE Transactions on Information Theory*, 48(9) :2626–2630, 2002.

[Mul54] D. E. Muller. Application of Boolean algebra to switching circuit design and to error detecting. *IRE Trans. Electronic Computers*, EC-3, 1954.

[NGG06] Y. Nawaz, G. Gong, and K.C. Gupta. Upper bounds on algebraic immunity of Boolean power functions. *Fast Software encryption - FSE 2006*, 4047 :375–389, 2006.

[Odl84] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Theory and Application of Cryptographic Techniques*, pages 224–314, 1984.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO)*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[Oxl] James G. Oxley. *Matroid theory*, volume 3. Oxford graduate text in mathematics.

[PH78] S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24 :106–110, 1978.

[PK95] W.T. Penzhorn and G.J. Kühn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In *Cryptography and Coding - 5th IMA Conference*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer-Verlag, 1995.

[QFL05] Longjiang Qu, Guozhu Feng, and Chao Li. On the Boolean functions with maximum possible algebraic immunity : Construction and a lower bound of the count, 2005.

[Ree54] I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IRE Trans. Inform. Th.*, IT-4, 1954.

[RGH07] Sondre Rønjom, Guang Gong, and Tor Helleseth. On attacks on filtering generators using linear subspace structures. In Solomon W. Golomb, Guang Gong, Tor Helleseth, and Hong-Yeop Song, editors, *SSC*, volume 4893 of *Lecture Notes in Computer Science*, pages 204–217. Springer, 2007.

[RH07a] Sondre Ronjom and Tor Helleseth. Attacking the filter generator over $GF(2^m)$. *SASC*, 2007.

[RH07b] Sondre Ronjom and Tor Helleseth. A new attack on the filter generator. *IEEE Transactions on Information Theory*, 53(5) :1752–1758, 2007.

[Riz07] Panagiotis Rizomiliotis. Remarks on the new attack on the filter generator and the role of high order complexity. In Steven D. Galbraith, editor, *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 204–219. Springer, 2007.

[RSA77] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82, 1977.

[Sav94] Petr Savicky. On the bent functions that are symmetric. *European Journal of Combinatorics*, 15 :407–410, 1994.

[Sha49a] C.E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28 :656–715, 1949.

[Sha49b] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28 :59–98, 1949.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math.*, pages 415–440, 1971.

[Sie85] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Information Theory*, C-34(1) :81–84, 1985.

[SP92] V. Sidel'nokov and A. Pershakov. Decoding of Reed-Muller codes with a large number of errors. *Probl. Inform. Transmission*, 28 :80–94, 1992.

[Tav04] Cedric Tavernier. *Testeurs, problèmes de reconstruction univariés et multivariés, et application à la cryptanalyse du DES*. PhD thesis, Ecole doctorale de l'école polytechnique, January 2004.

[Tel52] B. D. H. Tellegen. A general network theorem, with applications. *Philips Res. Rept.*, 7 :168–175, 1952.

[Tho03] E. Thomé. *Algorithmes de Calcul de logarithmes discrets dans les corps finis*. PhD thesis, Ecole polytechnique, France, 2003.

[Vid04] Marion Videau. On some properties of symmetric Boolean functions. In *IEEE International Symposium on Information Theory - ISIT 2004*, page 500. IEEE, 2004.

[Vid05a] Marion Videau. *Critères de sécurité des algorithmes de chiffrement à clé secrète*. PhD thesis, Université Paris 6, 2005.

[Vid05b] Marion Videau. Symmetric Boolean functions with high nonlinearity. In *Conference records of the Western European Workshop on Research in Cryptology - WeWORC 2005*, pages 87–88, 2005.

[Wei91] Victor K. Wei. Generalized Hamming weights for linear codes. *IEEE Transaction on Information Theory*, 37 :1412–1418, September 1991.

[Wie86] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32 :54–62, January 1986.

[ZZ99] Yuliang Zheng and Xian-Mo Zhang. Plateaued functions. In *ICICS*, pages 284–300, 1999.

[Zém00] Gilles Zémor. *Cours de cryptographie*. Cassini, 2000.