Université de Limoges

École doctorale sciences et ingénierie pour l'information, mathématiques

Faculté des sciences et techniques

Année : 2015 Thèse N° X

Thèse

pour obtenir le grade de

docteur de l'université de Limoges

Discipline : Mathématiques et Applications

présentée et soutenue par

Gaël Thomas

le 2 juin 2015

Design et Analyse de sécurité pour les constructions en cryptographie symétrique

Thèse dirigée par Thierry Berger et Marine Minier

Jury :

Anne Canteaut	INRIA Paris-Rocquencourt	Présidente
Pierre-Alain Fouque	Université de Rennes 1	Rapporteur
Louis Goubin	Université de Versailles-Saint-Quentin	Rapporteur
Christophe Clavier	Université de Limoges	Examinateur
Henri Gilbert	ANSSI	Examinateur
Thierry Berger	Université de Limoges	Directeur
Marine Minier	INRIA, INSA-Lyon	Co-directrice

Remerciements

Pour commencer, je souhaite remercier mes directeurs de thèse, Thierry Berger et Marine Minier, pour m'avoir aidé durant toutes ces années depuis la lecture du premier article jusqu'à la soutenance.

Je remercie ensuite Pierre-Alain Fouque et Louis Goubin pour avoir accepté de rapporter mes travaux ainsi qu'Anne Canteaut, Christophe Clavier et Henri Gilbert pour avoir accepté d'être dans mon jury.

Je remercie également mes coauteurs, en particulier Hélène Le Bouder et Julien Francq.

Je remercie aussi particulièrement mon tuteur pédagogique François Arnault pour ses conseils et sa bonne humeur.

Je pense aussi à tous les membres du DMI, en particulier les doctorants, antérieurs ou actuels, pour les moments passés ensemble autour d'un tableau, d'une boisson ou d'un barbecue.

Enfin, je remercie mes amis, en particulier les joueurs du serveur Minecraft Jupiter pour m'avoir aidé à oublié un temps ma thèse, ainsi que ma famille, en particulier ma mère pour avoir relu et amélioré l'orthographe de ce manuscrit.

Et surtout Varda ma compagne féline pour avoir égayé mes soirées et mes nuits.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Les travaux réalisés au cours de cette thèse se situent au carrefour de la cryptographie symétrique et du monde des environnements contraints. Le but de cette cryptographie, dite cryptographie légère est de fournir et d'évaluer des algorithmes symétriques pouvant être implémentés sur des systèmes très limités en ressources. Les contributions de cette thèse à la cryptographie légère portent d'une part sur les registres à décalage à rétroaction avec retenue et sur les schémas de Feistel généralisés d'autre part. Ces travaux ont été réalisés dans le cadre du projet BLOC de l'agence nationale de la recherche (ANR-11-INS-011).

La première partie de ce document pose les bases nécessaires pour la compréhension de cette thèse. Après une introduction générale à la cryptographie, on se penche plus en détail sur les deux facettes complémentaires de la cryptographie symétrique légère explorées dans cette thèse : construire un algorithme cryptographique pouvant à la fois répondre aux exigences extrêmes et variées des systèmes embarqués et de l'informatique ubiquitaire en termes d'implémentation tout en offrant un niveau de sécurité raisonnable.

La deuxième partie s'intéresse aux registres à décalage à rétroaction avec retenue (Feedback with Carry Shift Register ou FCSR). Un premier chapitre présente l'état de l'art sur les FCSR, notamment les propriétés relatives à l'implémentation des FCSR en milieu matériel et en milieu logiciel ainsi que les principales attaques existantes contre les FCSR. Le deuxième chapitre porte sur deux nouvelles propriétés cryptanalytiques des FCSR. La première met en évidence certaines différentielles dont il est facile de calculer la probabilité. La seconde est une adaptation des propriétés linéaires de Tian et Qi [TQ09]. Aucune de ces deux propriétés ne remet en cause la sécurité des FCSR. Finalement, un troisième chapitre présente une proposition de fonction de hachage légère, appelée GLUON et publiée à Africacrypt 2012 [BDM+12]. Ce travail a été réalisé en début de thèse et concerne une évaluation pratique de la sécurité de GLUON via deux types d'attaques particulières.

La troisième et principale partie de cette thèse porte sur l'étude des schémas de Feistel généralisés. Un premier chapitre dresse l'état de l'art sur les généralisations du schéma de Feistel, notamment celles visant à accroître la diffusion au sens de Shannon. Ici la diffusion est mesurée à l'aide de ce que Suzaki et Minematsu [SM10] nomment *délai de diffusion*. Il s'agit du nombre de tours requis pour que toutes les branches du schéma aient été influencées par toutes les entrées. Le deuxième chapitre présente une représentation matricielle des schémas de Feistel généralisés développée au cours de cette thèse. Cette représentation matricielle permet une vision unifiée des schémas de Feistel généralisés en relation avec le délai de diffusion et permet aussi de raffiner la notion de délai de diffusion. Le troisième chapitre fait suite au deuxième chapitre. La représentation matricielle permet de généraliser plus avant le schéma de Feistel, en distinguant dans le schéma les fonctions apportant confusion de celles apportant diffusion. Nous nommons ces schémas des schémas de Feistel généralisés étendus. Ces schémas permettent de limiter le délai de diffusion sans trop augmenter le coût de l'implémentation. Un exemple particulier de famille d'EGFN fait ensuite l'objet d'une étude approfondie. On évalue leur sécurité par rapport aux attaques classiques. On observe alors que certaines attaques, comme les attaques intégrales ou celles par différentielles impossibles, dépendent directement du délai de diffusion et deviennent inefficaces contre les EGFN. En revanche, pour les attaques différentielles et linéaires, on ne constate pas d'amélioration par

rapport à la littérature. Pire si on choisit mal le schéma, celui-ci peut devenir très vulnérable à ces attaques. Ce chapitre ainsi que le précédent a fait l'objet d'une publication à SAC 2013 [BMT13]. Le quatrième chapitre est une application directe du chapitre précédent. On y présente une proposition de chiffrement par bloc léger, appelé LILLIPUT. La ligne directrice lors de la création de LILLIPUT est la diffusion. Celle-ci se manifeste d'une part par le choix des EGFN étudiés au chapitre précédent et d'autre part au niveau du cadencement de clés par un choix faisant aussi intervenir des schémas de Feistel généralisés. Ce chiffrement a été soumis au journal IEEE-TC. Le dernier chapitre de cette thèse est une collaboration avec Hélène Le Bouder, à l'époque doctorante en microélectronique à l'école des mines de Saint-Étienne, dont le sujet portait sur un formalisme unifiant les attaques physiques. Le travail décrit ici est à la confluence de son sujet et du mien. Il s'agit d'une tentative de vision globale de la résistance schémas de Feistel généralisés aux attaques différentielles en fautes (Differential Fault Analysis ou DFA) [BS91]. Son but est de prédire de manière générique le comportement d'un schéma de Feistel généralisé par rapport aux attaques en fautes. Ce travail a fait l'objet d'une publication à FDTC 2014 [BTLT14].

Première partie

Notions Générales

Chapitre 1

Introduction

1.1 Cryptologie

La *cryptologie*, littéralement "l'étude de ce qui est caché", désigne l'étude et la mise en pratique des techniques permettant la réalisation de communications sécurisées en présence de tiers. Cette science est composée de deux branches indissociables. D'une part, la *cryptographie*, littéralement "écriture des secrets", qui se consacre à créer des systèmes, appelés cryptosystèmes, visant à assurer les exigences suivantes :

Confidentialité : les données échangées ne sont disponibles qu'aux personnes autorisées ;

Authentification : les parties en présence sont bien qui elles prétendent être ;

Intégrité : les données échangées n'ont pas subi de modifications (volontaires ou non) ;

Non-répudiation : aucune partie ne peut a posteriori nier ses actes passés.

D'autre part, la *cryptanalyse*, littéralement "investigation des secrets", qui cherche à attaquer les cryptosystèmes, c'est-à-dire à mettre en défaut un ou plusieurs des points précédents.

1.1.1 Chiffrement

La confidentialité est historiquement le point principal parmi les objectifs de la cryptographie. La situation est traditionnellement modélisée de la façon suivante : On suppose qu'un individu, canoniquement appelé Alice, souhaite transmettre un message à un deuxième individu, appelé Bob, sans qu'un troisième larron, appelé Oscar, qui écouterait leurs communications ne puisse connaître le contenu des messages échangés. Pour cela, Alice utilise un *algorithme de chiffrement* (*cipher* en anglais) afin de transformer le message porteur de sens qu'elle souhaite envoyer, appelé *texte clair* (*plaintext*), en un autre message a priori vide de sens et appelé le *texte chiffré* (*ciphertext*) qu'elle transmet à Bob. Cette opération s'appelle le *chiffrement* (*encryption*). Bob de son côté réalise l'opération inverse, nommée *déchiffrement* (*decryption*), pour retrouver le message initial. Afin qu'Oscar ne puisse lui aussi déchiffrer le message, ces deux transformations, chiffrement et déchiffrement, dépendent d'un paramètre secret inconnu d'Oscar et appelé la *clé* (*key*). Le principe du chiffrement est résumé à la figure 1.1. Plus formellement, on a la définition suivante.

Définition 1.1. *Un système de chiffrement est la donnée de :*
- *un ensemble* $\mathbb{M}$ *des textes clairs possibles ;*
- *un ensemble* $\mathbb{C}$ *des textes chiffrés possibles ;*
- *un ensemble* $\mathbb{K}_P \times \mathbb{K}_S$ *des clés (où* $\mathbb{K}_P$ *contient les clés utilisées par Alice pour chiffrer et* $\mathbb{K}_S$ *celles utilisées par Bob pour déchiffrer) ;*
- *un ensemble de transformations de chiffrement* $\{E_{K_P} \in \mathbb{M} \to \mathbb{C} | K_P \in \mathbb{K}_P\}$ *;*
- *un ensemble de transformations de déchiffrement* $\{D_{K_S} \in \mathbb{C} \to \mathbb{M} | K_S \in \mathbb{K}_S\}$ *;*

tels que pour tout clair $m \in \mathbb{M}$ *et toute paire de clés* $(K_P, K_S) \in \mathbb{K}_P \times \mathbb{K}_S$*, on ait* $D_{K_P}(E_{K_S}(m)) = m$.

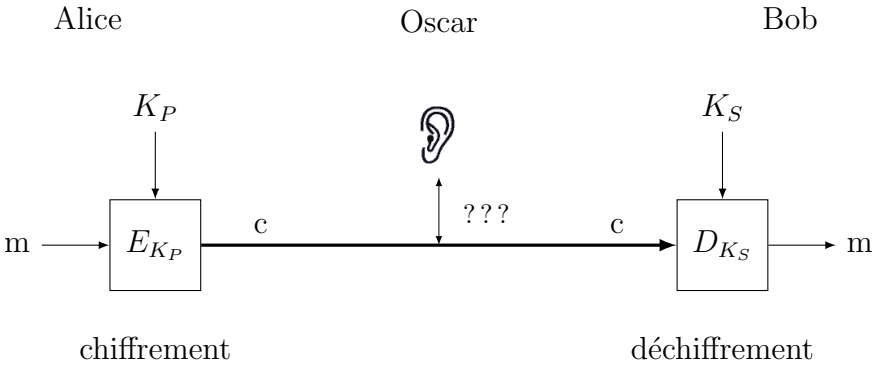


FIGURE 1.1 – Principe du chiffrement.

Il existe actuellement deux familles de chiffrements : les chiffrements asymétriques [DH76], aussi appelés chiffrements à clé publique, et les chiffrements symétriques, aussi appelés chiffrements à clé secrète.

Dans le cas d'un chiffrement asymétrique, la clé de chiffrement K_P est publique et se calcule à partir de la clé de Bob K_S qui est, elle, secrète. Ainsi n'importe qui est capable d'envoyer un message à Bob mais lui seul sera capable de le lire. Il faut cependant garantir que, connaissant la clé publique K_P, un attaquant soit incapable de retrouver la clé secrète K_S. Ces algorithmes se basent sur la difficulté de résoudre certains problèmes complexes tels que la factorisation d'entiers [RSA78], le calcul de logarithme discret dans un groupe [Gam85], le décodage d'un code correcteur aléatoire [McE78] ou la recherche d'un plus court vecteur dans un réseau euclidien [GGH97].

Dans le chiffrement symétrique, les clés de chiffrement et de déchiffrement sont égales. Cette unique clé $K = K_P = K_S$ est secrète, connue uniquement d'Alice et de Bob. Dans ce cas, la fonction de déchiffrement D_K est simplement l'inverse du chiffrement : $D_K = E_K^{-1}$, ce qui implique en particulier que l'ensemble des chiffrés $\mathbb{C}$ est de même cardinal que l'ensemble des clairs $\mathbb{M}$. La sécurité de ce schéma repose sur le fait qu'Oscar ne connaît pas la valeur de la clé K. En revanche, contrairement au chiffrement asymétrique, Alice et Bob doivent au préalable partager la clé K. De plus, Bob doit posséder une clé différente pour chaque personne avec qui il communique. Cependant ces algorithmes sont bien plus rapides que les algorithmes asymétriques – de l'ordre de plusieurs milliers de fois. C'est pourquoi ils sont encore très utilisés à l'heure actuelle.

En pratique les textes clairs, chiffrés ainsi que la clé sont des suites de symboles sur un même alphabet fini, en général $\mathbb{F}_2 = \{0, 1\}$ le corps fini à deux éléments muni de son addition (ou-exclusif binaire), notée "$\oplus$", et de sa multiplication (et binaire, aussi appelé xor), notée "$\cdot$".

Les chiffrements symétriques se subdivisent en deux familles : les chiffrements à flot et les chiffrements par bloc. Un chiffrement à flot fonctionne en générant, à partir de la clé K, une suite de symboles sur $\mathbb{F}_2$, appelée suite chiffrante, de la même longueur que le message à chiffrer. La suite chiffrante est alors combinée avec le texte clair au moyen de la loi de groupe, par exemple $\oplus$ sur $\mathbb{F}_2$, appliquée bit-à-bit. Notons le clair $m = m_0 m_1 \cdots m_{N-1}$,

le chiffré $c = c_0c_1 \cdots c_{N-1}$ et la suite chiffrante $s_0s_1, \cdots s_{N-1}$ (avec $m_i, c_i, s_i \in \mathbb{F}_2$) alors : $c_i = m_i \oplus s_i$. La sécurité d'un chiffrement à flot repose sur la qualité de la suite chiffrante générée.

La seconde famille de chiffrement symétrique est le chiffrement par bloc. Contrairement au chiffrement à flot qui traite les données un bit à la fois, un chiffrement par bloc divise le texte clair en blocs de longueur fixe n puis chiffre chacun de ces blocs séparément l'un après l'autre. Les différents blocs sont combinés entre eux via un mode opératoire. Par exemple, le mode ECB (*Electronique Code Book*) chiffre simplement successivement en parallèle chacun des blocs de clair. Un autre mode utilisé est le mode CBC (*Cipher Block Chaining*). Cette fois-ci, avant d'être chiffré, chaque bloc de clair est combiné via un $\oplus$ avec le chiffré du bloc précédent. Quel que soit le mode utilisé, un chiffrement par bloc se ramène essentiellement à une famille E_K de permutations paramétrées par la clé K qui transforment un bloc de taille fixe n en un autre ; d'où la définition suivante :

Définition 1.2. *Soit deux entiers n et κ où n est la taille de bloc et κ la taille de clé. Un chiffrement par bloc E est une fonction*

$$\begin{aligned} E : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n &\to \mathbb{F}_2^n \\ (K, m) &\mapsto c = E(K, m) \end{aligned}$$

telle que pour toute clé $K \in \mathbb{F}_2^\kappa$, l'application $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$ définie par $E_K(m) = E(K, m)$ est une bijection.

1.1.2 Hachage

Il existe un autre objet cryptographique que l'on associe souvent à la cryptographie symétrique ; ce sont les *fonctions de hachage*. Une fonction de hachage est une fonction dont l'entrée est une suite binaire de longueur arbitraire et qui retourne une sortie, appelé le haché, de longueur fixe n.

Définition 1.3. *Notons par $(\mathbb{F}_2)^*$ l'ensemble des suites binaires finies. Une fonction de hachage h est une fonction de $(\mathbb{F}_2)^* \to \mathbb{F}_2^n$.*

Les fonctions de hachage ne sont pas des algorithmes de chiffrement. En effet, aucun paramètre n'est inconnu de l'attaquant ; une fonction de hachage ne peut servir à garantir la confidentialité. Elles sont cependant un outil fort utile combiné à d'autres constructions : par exemple utilisées dans un MAC (*Message Authentication Code*) pour garantir intégrité et authentification. Enfin, elles sont souvent associées à la cryptographie symétrique car elles sont souvent construites à partir des mêmes éléments de base utilisés pour construire des chiffrements par bloc.

1.1.3 La Cryptanalyse ou «Que fait Oscar ?»

Un algorithme de chiffrement vise à garantir la confidentialité : un tiers non-autorisé ne peut connaître le contenu des données chiffrées. Le but d'un attaquant Oscar est de mettre en défaut ce principe. Précisons un peu les choses. Premier point, l'algorithme en lui-même

est connu de l'attaquant : la sécurité repose uniquement sur le caractère secret de la clé K choisie par Alice et Bob. C'est ce qu'on appelle le *principe de Kerckhoff*. Ce principe est motivé par deux choses. D'une part, si la sécurité d'un algorithme de chiffrement repose sur sa non-connaissance par un attaquant alors, dès que ce dernier en prend connaissance, il faut créer un nouvel algorithme de chiffrement robuste à partir de rien, ce qui demande beaucoup de temps. Au contraire, générer une nouvelle clé de chiffrement – aléatoire – est très rapide (à condition d'avoir une bonne source d'aléa). D'autre part, dans un monde où les télécommunications sont omniprésentes, il semble difficile en pratique d'avoir des algorithmes qui ne soient pas publics.

Étant donné un attaquant sur un algorithme de chiffrement, son but est d'être capable de retrouver les clairs correspondant à certains chiffrés, on parle de *décryptage*. Très souvent, cela consiste à retrouver la clé K utilisée. On distingue alors plusieurs types d'attaquants, classés ci-après en fonction du contrôle dont ils disposent sur les données auxquelles ils ont accès, du moins puissant au plus puissant.

Attaque à chiffré seul : l'attaquant ne dispose que d'un certain nombre de messages chiffrés.

Attaque à clair connu : l'attaquant dispose d'un certain nombre de couples clairs-chiffrés.

Attaque à clair choisi : l'attaquant *choisit* un certain nombre de clairs et dispose des chiffrés correspondants.

Attaque à chiffré choisi : l'attaquant choisit un certain nombre de clairs *ou* de chiffrés et dispose des chiffrés ou des clairs correspondants.

Une des mesures de l'efficacité d'une attaque est alors le nombre de clairs ou de chiffrés dont l'attaquant a besoin pour mener son attaque. Cela s'appelle la complexité en donnée de l'attaque. On s'intéresse aussi à la complexité calculatoire de l'attaque. On distingue la complexité en temps, généralement exprimée en nombre d'appels à la fonction de (dé-)chiffrement effectués par l'attaquant, et la complexité en mémoire, exprimée en nombre maximal de clairs/chiffrés stockés en mémoire à tout instant.

Pour retrouver la clé, un attaquant a toujours au moins une attaque à sa disposition : celle qui consiste à partir d'un couple clair-chiffré à chiffrer ce clair avec toutes les clés possibles et regarder celle qui donne le bon chiffré. Cette attaque se nomme recherche exhautive de la clé. Sa complexité en temps est de l'ordre de 2^κ appels à la fonction de chiffrement où κ est la taille de la clé. La clé doit donc être suffisamment longue de manière à rendre cette attaque trop coûteuse en pratique. De plus, comme cette attaque est toujours possible, une "vraie" attaque sur un chiffrement sera une attaque dont le coût sera inférieur au coût de la recherche exhaustive. Lorsqu'une telle attaque est découverte, l'algorithme attaqué est considéré comme "cassé" et on évitera de continuer à l'utiliser tel quel.

1.2 Chiffrement par bloc

En pratique, la construction d'algorithmes de chiffrement par bloc suit quelques principes. On en détaille ici les deux principaux : d'une part le diptyque confusion et diffusion introduit par Claude Shannon [Sha49], d'autre part la construction itérative de chiffrements par bloc.

1.2.1 Confusion & Diffusion

Dans son article fondateur de la cryptographie moderne [Sha49], Claude Shannon a discuté de deux propriétés que devrait vérifier un bon algorithme de chiffrement. Il s'agit de ce qu'il appelle la diffusion d'une part et de la confusion d'autre part. La propriété de diffusion demande que chaque partie du chiffré dépende de chaque partie du clair et de la clé, en d'autres termes, de petits changements en entrée doivent avoir un effet important en sortie. La confusion quant à elle doit servir à complexifier la dépendance qui existe entre le clair, la clé et le chiffré, ceci afin de complexifier le travail statistique d'un attaquant, celui-ci devant en effet obtenir le moins d'information possible sur la clé pour chaque couple clair–chiffré qu'il possède.

Pour ce qui est de la confusion, on dispose en pratique de deux moyens de base pour en obtenir. D'une part utiliser les opérateurs non-linéaires usuels, comme le ET binaire en matériel ou sa version bit-à-bit en logiciel ou encore utiliser l'addition modulaire (qui n'est pas linéaire par rapport au xor). On peut d'autre part utiliser des fonctions à l'expression algébrique plus complexe mais sous forme tabulée, on parle alors de boîte de substitution ou boîte-S (S-box en anglais), ainsi appelée car on substitue une autre séquence de bits à une séquence de bits. Une S-box est en général choisie de manière à maximiser la confusion qu'elle provoque. Comme évaluer cela, c'est-à-dire évaluer sa résistance aux attaques existantes, est en général exponentiel en sa taille (le nombre de bits qu'elle substitue), une S-box ne travaille que sur une partie du chiffrement, c'est ici qu'intervient la couche de diffusion pour mélanger les sorties de plusieurs S-box agissant sur des parties différentes du chiffrement. La vision opérateur logique et la vision S-box ne sont cependant pas exclusives comme on le verra à la section 2.1.

La diffusion, quant à elle, est en pratique linéaire. Ceci permet d'utiliser les outils d'algèbre linéaire et de la théorie des codes (linéaires) afin de construire de bonnes composantes de diffusion, permettant par exemple de bien mélanger entre elles les sorties de différentes S-box.

1.2.2 Chiffrement itératif

Lorsqu'on souhaite construire un algorithme de chiffrement par bloc, on doit le faire de manière à assurer sa sécurité, grâce aux concepts de confusion et diffusion, mais on doit aussi veiller à ce que son exécution et celle de son inverse soient peu coûteuses en ressources. En pratique, on utilise des constructions itératives. Dans celles-ci, l'algorithme de chiffrement E_K est la composée de r fonctions successives. Chacune de ces fonctions, appelée un tour, consiste à faire agir une même fonction f paramétrée par une sous-clé K^i, avec $i \in \{0, \cdots, r-1\}$. Les sous-clés $K^0, \cdots, K^{r-1}$ sont différentes à chaque tour et sont générées à partir de la clé secrète K (on parle aussi de clé maître) via un sous-algorithme appelé le *cadencement de clé* (*key schedule* en anglais). Ce principe du chiffrement itératif est résumé à la figure 1.2.

La fonction de tour f est choisie pour ses qualités à la fois vis-à-vis de la confusion et de la diffusion mais aussi pour sa simplicité d'évaluation et d'inversion. Le nombre r de tours, quant à lui, sera choisi suffisamment élevé de manière à garantir une diffusion et une confusion suffisantes.

Il existe actuellement deux types de chiffrement itératif : les réseaux substitution-permutation et les schémas de Feistel.

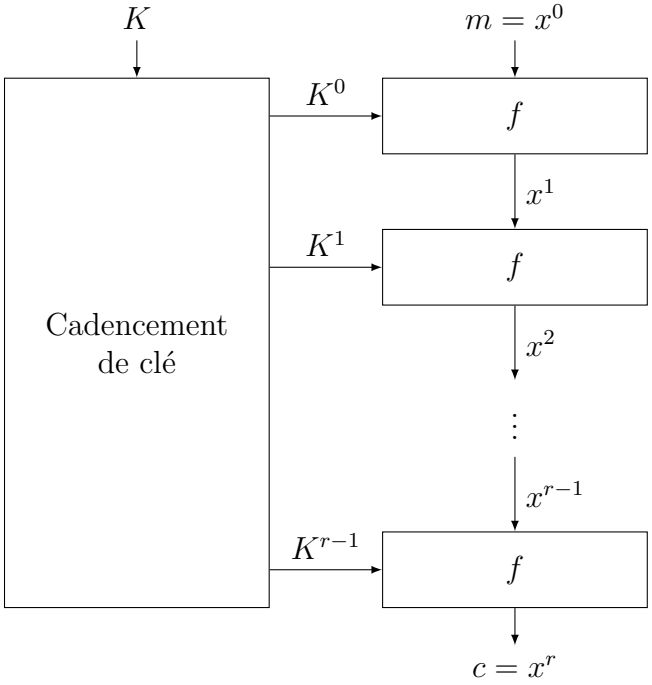


FIGURE 1.2 – Schéma générique d'un chiffrement itératif transformant un texte clair m en un texte chiffré c à l'aide de la clé K.

1.2.2.1 Réseaux Substitution-Permutation

Un *Réseau Substitution-Permutation* (*Substitution-Permutation Network* en anglais ou SPN) transcrit quasi-littéralement les concepts de diffusion et confusion de Shannon. Il s'agit d'une construction itérative dans laquelle chaque tour se compose successivement de trois opérations : une application dite de substitution qui assure la confusion, une application dite de permutation qui assure la diffusion et enfin une opération de mélange avec la clé du chiffrement.

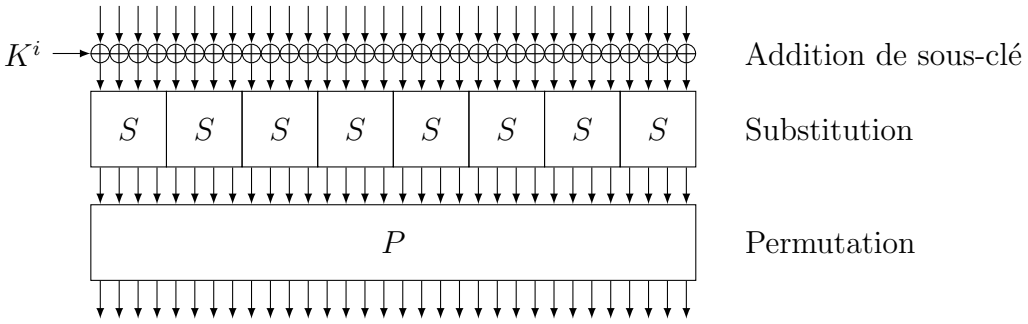


FIGURE 1.3 – Schéma générique d'un tour d'un réseau de substitution-permutation.

La couche de substitution est typiquement composée de petites applications réalisées en parallèle, appelées S-box, choisies pour être les moins linéaires possibles. La couche permutation, très souvent linéaire, a alors pour but de mélanger au mieux les sorties de ces S-box. Quant au mélange avec la clé, cela consiste à additionner (xorer) une sous-clé, obtenue à l'aide du cadencement de clé, à l'état interne de l'algorithme.

Les algorithmes tels que l'AES [FIP01], SERPENT [ABK] sont des SPN ainsi que ceux décrits à la section 2.2.1.

1.2.2.2 Schéma de Feistel

Le *schéma de Feistel* est une construction itérative permettant de transformer des fonctions de n bits vers n bits en une permutation (i.e. bijection) de $2n$ bits vers $2n$ bits. Pour cela, le schéma sépare le texte clair en deux blocs, x_0 et x_1, de taille identique. Un de ces deux blocs passe alors dans une fonction F, dépendante de la clé de tour et dite fonction de Feistel, puis le résultat est xoré à l'autre bloc. Cette construction est alors itérée plusieurs fois en inversant les rôles des deux blocs à chaque tour jusqu'à obtenir le texte chiffré (y_0, y_1). En général, on n'inverse pas les blocs lors du dernier tour. Cela rend en effet le schéma involutif à l'ordre des clés de tour près, comme le montre la figure 1.4.

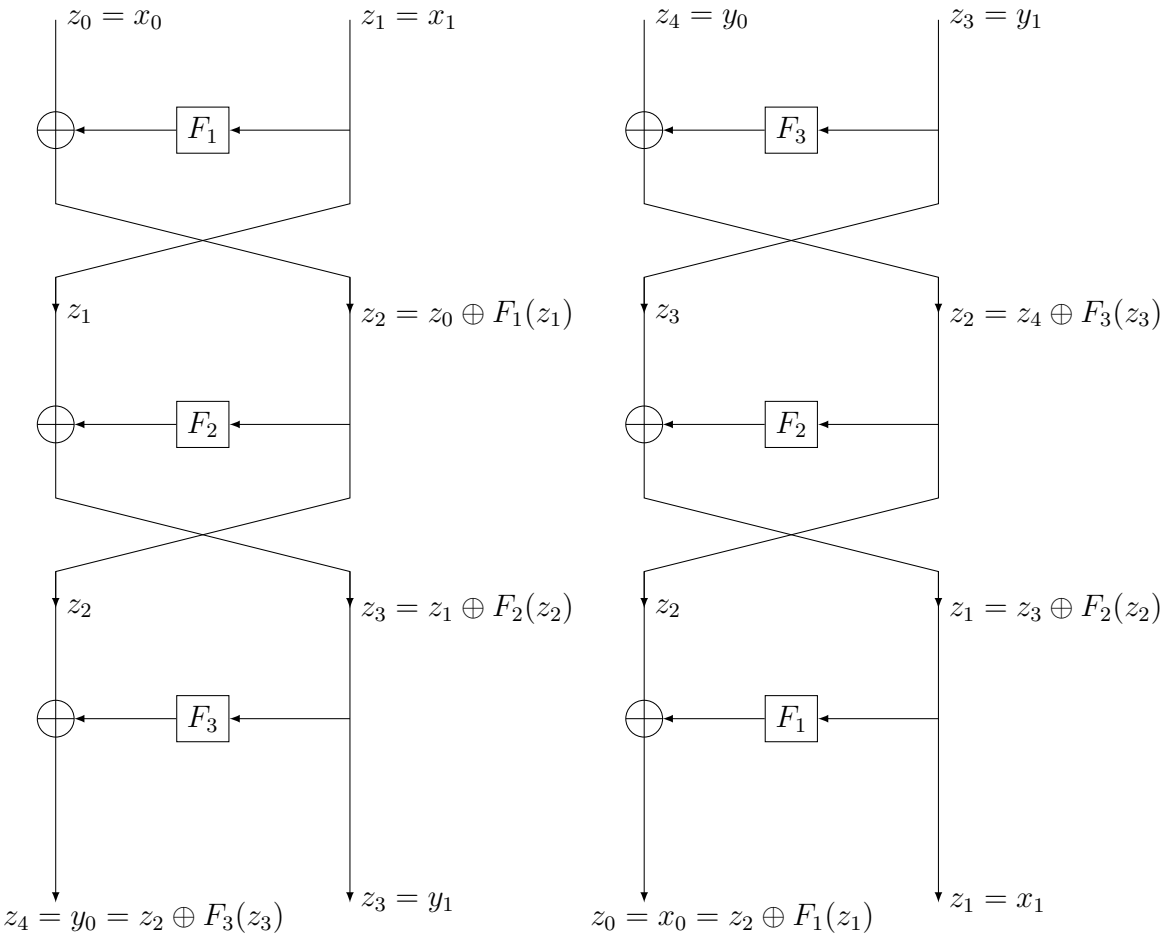


FIGURE 1.4 – Trois tours d'un schéma de Feistel (à gauche) et son inverse (à droite).

Les caractéristiques principales du schéma de Feistel sont les suivantes :
- à l'aide de la fonction de Feistel, seule une partie de l'état interne est modifiée à chaque tour ;
- le schéma est inversible même si la fonction de Feistel utilisée ne l'est pas ;
- la bijection réciproque suit le même schéma mais en inversant l'ordre des sous-clés.

Luby et Rackoff [LR88] ont étudié le caractère pseudo-aléatoire du schéma de Feistel. De nombreux chiffrements existants sont basés sur le schéma de Feistel. On peut citer, par exemple, le Data Encryption Standard (DES) [Nat77], Camellia [AIK+00], SEA [SPGQ06] ainsi que les algorithmes décrits à la section 2.2.2.

1.3 Hachage

Comme déjà dit, une fonction de hachage h est une fonction dont l'entrée est une suite binaire de longueur arbitraire et dont la sortie est une suite binaire de longueur fixe n. L'intérêt d'une fonction de hachage est leur caractère dit *à sens unique* : pour un message x, le haché $h(x)$ est très rapide à calculer, en revanche l'opération inverse est difficile. Plus précisément en cryptographie, on exige d'une fonction de hachage quelle soit résistante aux trois points suivants :

Préimage : étant donné un haché y, il est difficile de trouver un message x tel que $h(x) = y$;

Seconde Préimage : étant donné un message x, il est difficile de trouver un message $x' \neq x$ tel que $h(x') = h(x)$;

Collisions : il est difficile de trouver deux messages distincts x et x' tels que $h(x') = h(x)$.

Dans ces points, le mot "difficile" signifie qu'il n'existe pas de moyen de résoudre ces problèmes autre que des méthodes génériques, c'est-à-dire fonctionnant pour toute fonction de hachage h. Pour trouver une préimage d'un haché y donné (de longueur n), une méthode consiste à tirer de manière uniformément aléatoire un message x jusqu'à obtenir $h(x) = y$. La complexité de cette attaque est d'environ 2^n appels à la fonction h. De même pour la recherche de seconde préimage, on tire x' jusqu'à obtenir $h(x') = h(x)$. Enfin, pour la recherche de collision, on tire de manière uniformément aléatoire un message x et on stocke dans une table toutes les valeurs $h(x)$ déjà calculées. On s'arrête lorsqu'on a trouvé un x tel que $h(x)$ se trouve déjà dans la table. En utilisant le paradoxe des anniversaires, on estime la complexité de cette attaque à environ $2^{n/2}$. En somme, une bonne fonction de hachage est une fonction pour laquelle on ne connaît pas d'algorithme plus efficace que ces méthodes génériques ; soit 2^n pour la préimage et la seconde préimage, et $2^{n/2}$ pour les collisions.

On donne ensuite deux méthodes itératives pour construire des fonctions de hachage.

1.3.1 Construction de Merkle-Damgård

La construction de Merkle-Damgård est une construction itérative pour créer une fonction de hachage à partir d'une fonction dite fonction de compression dont l'entrée est de taille plus importante que celle de sa sortie mais finie fixe (contrairement à l'entrée de la fonction de hachage qui peut être de taille arbitraire). En d'autres termes, on appelle fonction de compression une fonction $f : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2^n$ avec $m \geq 1$. À partir d'une fonction de compression, on peut construire une fonction de hachage de la manière suivante (cf. aussi la figure 1.5) :

1. découper le message x en blocs de m bits $x_1, \cdots, x_\ell$, en prolongeant éventuellement x de manière à voir une taille multiple de m ;
2. poser $H_0 = IV$ (pour Initial Value) une valeur fixée de $\mathbb{F}_2^n$;
3. pour i entre 1 et ℓ, définir successivement $H_i = f(H_{i-1}, x_i)$;

4. retourner le haché $h(x) = H_\ell$.

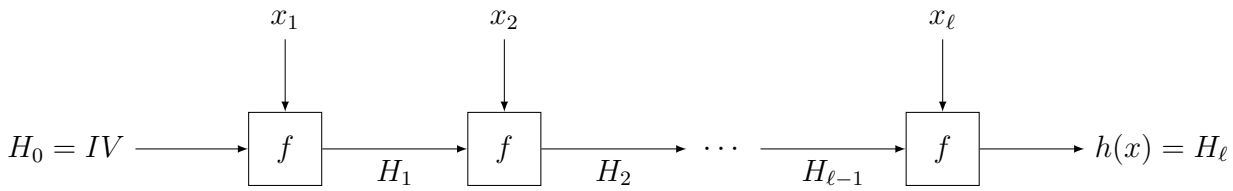


FIGURE 1.5 – Construction de Merkle-Damgård pour les fonctions de hachage.

Telle quelle, cette construction n'assure aucune des propriétés exigées des fonctions de hachage. Cependant, Merkle et Damgård [Mer89, Dam89] montrent que si le prolongement de x en un message de longueur multiple de m (opération qui se nomme *padding*) est fait d'une certaine manière et que la fonction de compression f est résistante aux collisions alors la fonction de hachage tout entière h l'est aussi. Un exemple de padding pour lequel leur résultat s'applique consiste à concaténer au message x la valeur $10 \dots 0|x|$ où $|x|$ désigne la longueur en bits de x et où le nombre de zéros ajoutés est tel que le résultat soit de longueur multiple de m.

1.3.2 Construction éponge

Proposée par Bertoni *et al.* [BDPA08], la construction "éponge" est une nouvelle méthode itérative pour construire une fonction de hachage à partir d'une fonction ou d'une permutation fixée f. Cette construction est notamment la base de la fonction de hachage Keccak [BDPA11], gagnante de la compétition SHA-3. Comme dans le cas de la construction de Merkle-Damgård, le message x, éventuellement paddé, est découpé en blocs $x_1, \cdots, x_\ell$ de longueur fixée r, appelé le *taux* de l'éponge. Une éponge, comme illustré à la figure 1.6, est un automate dont l'état interne se compose de deux parties : la première de la même taille r qu'un bloc de message et la seconde de taille notée cp, appelée la *capacité* de l'éponge. Étant donné un état initial et une fonction f de $\mathbb{F}_2^{r+cp}$ dans lui-même, une éponge hache le message x de la manière suivante :

1. **Initialisation** : le message x est paddé en suffixant un '1' puis suffisamment de zéros de manière à obtenir un message paddé de longueur multiple de m.
2. **Absorption** : les blocs de r bits du message sont xorés à r bits de l'état interne de l'éponge les uns après les autres et entrecoupés par une application de la fonction f sur l'état interne.
3. **Essorage** : r bits de l'état interne sont retournés en sortie, entrecoupés par une application de la fonction f, jusqu'à obtenir n bits de sortie.

L'intérêt de cette construction est que si la fonction de transition interne f est choisie uniformément parmi l'ensemble des fonctions de $\mathbb{F}_2^{r+cp}$ dans lui-même possibles et sous réserve que la capacité cp soit plus grande que la taille du haché n (en général, on prend $n = 2 \times cp$) alors la fonction éponge résiste à la préimage, à la seconde préimage et aux collisions.

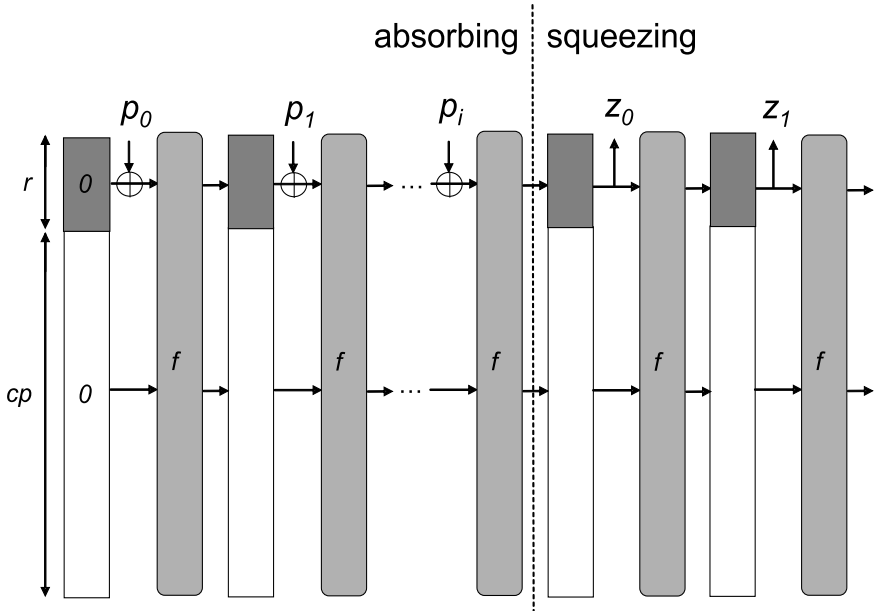


FIGURE 1.6 – La Construction éponge

Chapitre 2

Cryptographie légère

Les chiffrements par bloc classiques, tels que l'AES [FIP01] ou Camellia [AIK+00], ont été conçus pour des machines haut-de-gamme (ordinateur personnel, serveur web,...). Cependant, la dernière décennie a vu l'explosion de la présence d'appareils dans l'environnement de l'utilisateur. Ces appareils disposent de ressources plus limitées qu'un ordinateur de bureau et sont destinés à remplir une tâche spécifique de manière transparente pour l'utilisateur, on parle de *calcul omniprésent* (*pervasive computing* en anglais) ou encore d'*informatique ubiquitaire* (*ubiquitous computing*). On peut citer les smartphones, les microcontroleurs utilisés dans les voitures ou l'électroménager, mais aussi les radio-étiquettes (RFID tags) se trouvant sur les titres de transport ou les passeports, qui permettent de stocker et de récupérer des données attachées à un objet ou encore les réseaux de capteurs sans-fil utilisés pour surveiller un environnement particulier, par exemple la température d'un lieu ou la détection du mouvement.

Tous ces appareils varient fortement tant en termes d'architectures (matérielles et logicielles) qu'en termes de performances. Une telle différence entre tous ces appareils fait que les outils classiques de la cryptographie ne sont pas nécessairement bien adaptés pour le monde du calcul omniprésent. De plus, la sécurité est d'autant plus un enjeu ici que ces appareils communiquent en permanence, aussi bien entre eux qu'avec des machines plus puissantes. La cryptographie légère, aussi appelée cryptographie à bas coût (lightweight cryptography en anglais), a alors pour but de concevoir et d'analyser de nouveaux outils de sécurité adaptés à ces systèmes à fortes contraintes.

On s'attachera dans une première partie à cerner les différentes contraintes relatives à ces environnements particuliers, en distinguant notamment matériel et logiciel. Puis dans une seconde partie, on décrira quelques algorithmes de chiffrement par bloc existants prenant en compte ces contraintes lors de leur création.

2.1 Règles du jeu

Comme son nom le laisse entendre, la cryptographie légère cherche à fournir de la sécurité à un coût très réduit, ceci afin de pouvoir être utilisée même sur les plateformes les plus modestes. La plupart des contraintes seront spécifiques soit à l'implémentation matérielle soit à l'implémentation logicielle. Cependant il existe des points communs entre les deux, certains critères affectant le coût et les performances des algorithmes sur les deux plateformes.

Le premier point à discuter est la quantité de sécurité que l'on désire, ce qui se traduira en terme de longueur de clé pour les algorithmes de chiffrement. En effet, la sécurité fournie par les chiffrements classiques comme AES ou Camellia va de 128 à 256 bits. Pour les applications utilisant les plateformes les plus modestes un tel niveau de sécurité n'est pas forcément nécessaire. Ainsi, le niveau de sécurité attendu pour les algorithmes légers sera plus faible que dans le cas classique, en général entre 64 et 128 bits. Ceci permet de réduire la consommation de ressources de l'algorithme. Un autre moyen d'économiser des ressources est aussi de réduire la taille du bloc sur lequel travaille le chiffrement. Au lieu d'un état

interne de 128 comme pour AES, on utilisera un état plus petit, typiquement de 64 bits.

On donne par la suite les contraintes spécifiques aux plateformes matérielles puis logicielles tant en termes d'exigences mémoires que temporelles.

2.1.1 Point de vue matériel

Dans le cas matériel, le coût mémoire correspond à la taille du circuit ou au nombre de portes logiques requises pour implémenter l'algorithme. Celle-ci peut s'exprimer en μm^2 mais cette valeur est dépendante de la technologie utilisée. Pour faciliter la comparaison entre différentes implémentations, la taille du circuit est généralement exprimée en portes équivalentes (Gate Equivalent ou GE). Une GE est la taille d'une porte NAND à deux entrées dans la technologie utilisée. Ainsi la taille d'un circuit exprimée en GE sera l'aire de ce circuit exprimée en μm^2 divisée par la taille d'une porte NAND. Grâce à cette métrique, il est plus facile de comparer le coût mémoire de deux algorithmes différents même s'ils sont implémentés sur des technologies différentes.

Pour ce qui est du coût en temps, on s'intéresse à la latence, c'est-à-dire au temps nécessaire pour exécuter l'algorithme, généralement exprimé en nombre de cycles d'horloge, ainsi qu'au débit c'est-à-dire à la quantité de données traitées par unité de temps, généralement exprimé en bits/s pour une fréquence d'horloge donnée, typiquement 100kHz.

Une autre propriété à prendre en compte est la consommation électrique du circuit. Même s'il n'est pas aisé de comparer les consommations de deux implémentations différentes, on peut cependant donner un critère de conception qui influence la consommation électrique. Il s'agit de l'*arité sortante* (*fan-out* en anglais). L'arité sortante d'un bit est le nombre de portes logiques où le bit en question est une entrée. En vertu de la loi des nœuds, plus l'arité sortante sera élevée, plus il faudra de courant pour transmettre ce bit à toutes ses sorties. D'où un accroissement de la consommation électrique. Cependant trop limiter le nombre d'influences pour tous les bits diminue la diffusion de la fonction de tour du chiffrement, à moins d'utiliser plus de portes, ce qui augmente la taille du circuit. On privilégiera en général une taille de circuit faible.

Il faut aussi prendre en compte ce qu'on nomme *chemin critique* (*critical path* en anglais). Chaque traversée de porte logique par un signal demande un certain temps ; donc plus un bit doit traverser de portes, plus la fréquence d'horloge doit être basse afin de laisser le temps au signal de traverser toutes les portes sur son chemin en un seul cycle d'horloge. C'est le nombre maximal de portes que doit traverser un signal par cycle d'horloge que l'on nomme *chemin critique*. Cette valeur limite la fréquence du circuit ; on essaie donc d'avoir un chemin critique petit.

Finalement, on se penche sur les solutions utilisées actuellement pour concevoir des algorithmes de chiffrement efficaces en matériel. La première chose qu'on remarque est que les cellules mémoires stockant le bloc à chiffrer et la clé représentent la majorité de la taille du circuit. Ce constat est d'ailleurs une des motivations pour limiter la taille de bloc et de clé. Pour la même raison, on évite tout accès à des tables précalculées ou S-box. On implémente le calcul en lui-même à l'aide des équations algébriques liant les entrées et les sorties de la S-box. La partie confusion du chiffrement utilisera donc une S-box "simple" voire pas de S-box du tout. Pour la diffusion, les permutations des bits ne coûtent que du routage et sont donc quasiment gratuites. Pour les couches de diffusion plus complexes, à l'aide de matrices MDS par exemple, il s'agit alors d'en construire nécessitant un nombre

limité de portes XOR. Le chiffrement LED (cf. section 2.2.1.1,[GPPR11]) utilise par exemple une construction itérative basée sur un genre de schéma de Feistel généralisé à quatre tours.

2.1.2 Point de vue logiciel

En logiciel, la complexité mémoire est la quantité de RAM utilisée par l'algorithme. Mais il faut aussi prendre en compte l'espace nécessaire pour stocker le code de l'algorithme, par exemple sur une ROM.

La complexité en temps, quant à elle, est le nombre de cycles d'horloges nécessaires pour traiter un octet de données. Cela mesure la vitesse de l'algorithme. Cependant, obtenir une vitesse élevée au prix d'une surcharge, comme par exemple un cadencement de clé complexe, n'est peut-être pas acceptable dans tous les cas. Ainsi la latence, c'est-à-dire le nombre de cycles pour calculer cette surcharge doit aussi être prise en compte.

Une des différences essentielles entre matériel et logiciel est que le logiciel ne travaille pas au niveau du bit. Ceux-ci sont toujours en effet regroupés en paquets, aussi appelés mots. La longueur de ces mots, c'est-à-dire le nombre de bits qu'ils contiennent, est typiquement de 8, 16, 32 ou 64 bits. Par conséquent une opération travaillant au niveau du bit, comme on peut le voir en matériel, affichera souvent de très mauvaises performances en logiciel si elle casse cette structure de mot. Les opérations qui conservent cette structure sont dites *orientées-mot* (*word-oriented* en anglais) ou encore *software-friendly*. Ce sont en général les opérations de base disponibles au niveau du processeur sur les mots machines. Citons par exemple, les opérations arithmétiques élémentaires, les décalages des bits à l'intérieur d'un mot que ce soit avec perte (shift) ou bien sans perte (rotation). Ce sont aussi les opérations logiques (ET, OU, NOT, XOR) effectuées de manière bit-à-bit.

Regardons à présent les outils à notre disposition lorsqu'on souhaite concevoir un chiffrement efficace en logiciel. Pour la couche de diffusion, les opérations orientées mot sont à privilégier. Les matrices MDS sont ici un outil de choix puisqu'elles fonctionnent naturellement au niveau des mots et peuvent facilement s'écrire à l'aide du XOR bit-à-bit et des opérations de décalage. Pour la confusion, des S-box de la taille d'un mot (4 ou 8 bits) est un choix classique. On peut aussi s'inspirer de ce qui se fait en matériel. En matériel, les S-box ne sont pas précalculées mais implémentées à l'aide de portes logiques. Il est possible de faire la même chose en logiciel si on applique la même S-box de manière bit-à-bit. Par exemple, si on se donne une S-box n bits vers n bits, on peut l'appliquer en parallèle sur n mots, de n'importe quelle taille m, en l'appliquant m fois en parallèle successivement sur les n bits en position 0, puis sur les n bits en position 1 et ainsi de suite (cf. par exemple le chiffrement RECTANGLE section 2.2.1.3,[ZBL+14]) On parle alors d'implémentation par *couche de bits* ou implémentation *bit-slice*.

2.2 Quelques chiffrements par bloc légers

On présente ici différents algorithmes de chiffrement par bloc léger. Pour chacun, on précise la ou les stratégies utilisées par leurs auteurs pour s'adapter aux contraintes de la cryptograpie légère.

2.2.1 Réseaux Substitution-Permutation

2.2.1.1 LED

LED (Lightweight Encryption Device) [GPPR11] est un SPN présenté à CHES 2011 et chiffrant des messages de 64 bits avec des clés allant de 64 à 128 bits par pas de 4 bits. Sa construction est très proche de celle de l'AES. Son état interne est représenté par une matrice 4×4 de nybbles (mot de 4 bits, aussi appelé quartet). Un tour de LED se compose de quatres opérations : AddConstants qui xore une constante dépendante du tour et de la taille de clé, SubCells qui applique une S-box 4 bits appliquée en parallèle sur les nybbles, ShiftRows qui décale circulairement la ligne i de i positions vers la gauche ($i \in \{0, \cdots, 3\}$) et finalement MixColumnsSerial qui multiplie chaque colonne par une matrice de diffusion MDS. Quatre tours de LED constituent une étape. Le chiffrement complet se compose de s étapes entrecoupées d'additions avec une sous-clé. Le nombre d'étapes est $s = 8$ pour LED-64 et $s = 12$ pour les autres versions.

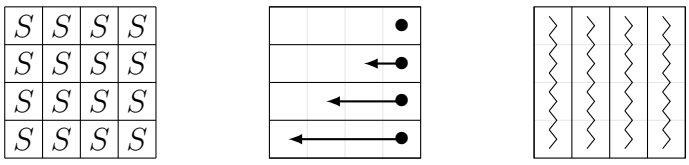


FIGURE 2.1 – Les trois opérations (de gauche à droite, SubCells, ShiftRows et MixColumnsSerial) qui, avec AddConstants, constituent un tour de LED.

Contrairement à l'AES, LED travaille au niveau des nybbles au lieu des octets. Les différentes opérations utilisées fonctionnent de manière similaire à celles de l'AES. La matrice MDS utilisée dans MixColumnsSerial est en revanche construite de manière particulière. En effet, elle peut s'exprimer comme puissance quatrième d'une matrice plus simple puisque sous forme compagnon et s'implémentant aisément en matériel. Ces matrices sont données à la figure 2.2. Ainsi l'opération MixColumnsSerial peut facilement s'implémenter en utilisant un genre de schéma de Feistel généralisé sur 4 tours.

$$\begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{pmatrix}^4$$

FIGURE 2.2 – En notation hexadécimale, matrice MDS utilisée dans MixColumnsSerial vue comme puissance quatrième d'une matrice compagnon orientée matériel.

2.2.1.2 PRESENT

Présenté à CHES 2007, PRESENT [BKL+07] est le plus connu des chiffrements légers et fait désormais parti du standard ISO/IEC 29192 [ISO11] sur la cryptographie légère, en même temps que CLEFIA (décrit section 2.2.2.1). Il chiffre des messages de 64 bits

avec des clés de 80 ou 128 bits. Il s'agit d'un SPN sur 31 tours, chaque tour se composant successivement d'une addition de sous-clé, d'une couche de S-box 4 bits en parallèle et enfin d'une permutation des bits. Un tour de PRESENT est donné à la figure 2.3.

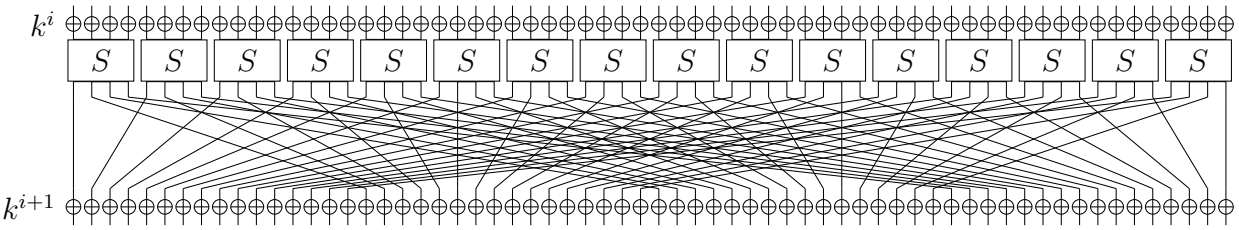
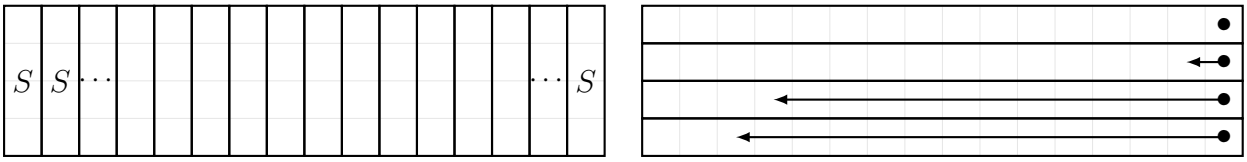


FIGURE 2.3 – Un tour de PRESENT.

À cause du choix d'une permutation des bits pour sa couche de diffusion, PRESENT est un chiffrement très orienté matériel. En effet, cette dernière ne coûte quasiment rien en matériel et la S-box 4 bits 28 GE.

2.2.1.3 RECTANGLE

RECTANGLE [ZBL+14] est un SPN chiffrant des messages de 64 bits avec des clés 80 ou 128 bits. Les bits du message sont arrangés en un tableau de 4 lignes et 16 colonnes. Un tour de RECTANGLE se compose de trois opérations successives : AddRoundkey qui xore une sous-clé à l'état courant, SubColumn qui applique une S-box 4 bits en parallèle sur les 16 colonnes et enfin ShiftRow qui décale circulairement le contenu de chacune des lignes d'une certaine valeur. La figure 2.4 illustre le fonctionnement de ces opérations. Le chiffrement complet se compose de 25 tours suivis d'une dernière addition de clé.



SubColumn ShiftRow

FIGURE 2.4 – Les deux opérations, SubColumn (à gauche) et ShiftRow (à droite), qui avec AddRoundkey constituent un tour de RECTANGLE.

RECTANGLE est construit de manière à permettre une implémentation bit-slice efficace. L'opération ShiftRow ne nécessite que du cablage en matériel et peut aussi être efficacement implémentée en logiciel. L'implémentation matérielle de la S-box de RECTANGLE coûte 20.75 GE et grâce à sa parallélisation en colonne, elle peut être facilement implémentée en logiciel via une technique bit-slice, ce faisant elle ne requiert que 12 opérations logiques.

2.2.2 Schémas de Feistel

2.2.2.1 CLEFIA

CLEFIA [SSA+07] est un chiffrement présenté à FSE 2007 et fait partie du standard ISO/IEC 29192 [ISO11] sur la cryptographie légère, en même temps que PRESENT (décrit

section 2.2.1.2). Tout comme l'AES, il chiffre des messages de 128 bits avec des clés de 128, 192 ou 256 bits. Il s'agit d'un schéma de Feistel généralisé avec 4 blocs et 2 fonctions internes 32 bits vers 32 bits F_0 et F_1 comme dépeint à la figure 2.5. Chacune de ces fonctions se compose successivement d'une addition de sous-clé, de quatre S-box 8 bits en parallèle et d'une multiplication matricielle de diffusion similaire au MixColumns de l'AES. CLEFIA utilise un jeu de 2 S-box S_0 et S_1, chacune utilisée à la fois dans F_0 et F_1. Chaque fonction utilise sa propre matrice de diffusion MDS. Le schéma est itéré 18, 22 ou 26 fois suivant la longueur de la clé.

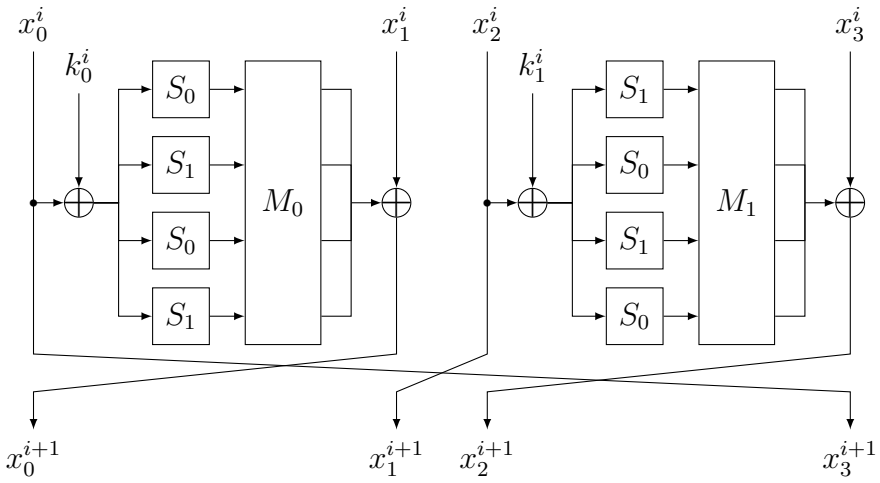


FIGURE 2.5 – Un tour de CLEFIA.

2.2.2.2 DESXL

Lorsque l'ancien standard de chiffrement américain DES [Nat77] fut conçu dans les années 1970, ses caractéristiques furent choisies dans l'optique d'une implémentation matérielle efficace sur la technologie de l'époque. En effet, observons le fonctionnement du DES. Il s'agit d'un schéma de Feistel à 16 tours chiffrant des messages de 64 bits avec des clés de 56 bits. Sa fonction de Feistel se compose d'abord d'une fonction dite d'expansion E qui duplique un bit sur deux, puis d'un xor avec une sous-clé de 48 bits, suivi par 8 S-box 6 bits vers 4 bits choisies notamment pour leurs bonnes propriétés différentielles ; les 32 bits en sortie des S-box subissent finalement une permutation des bits P. Cette permutation est telle que la sortie de chaque S-box soit envoyée sur au moins 5 S-box au tour suivant. La figure 2.6 illustre un tour de DES. Ainsi le coût d'une implémentation du DES, hormis les mémoires pour stocker le message à chiffrer, provient essentiellement des S-box puisque les fonctions E et P ne coûtent que des fils en matériel. Avec le regard que l'on peut aujourd'hui porter sur le DES, son design est toujours d'actualité et de manière surprenante peu de choses le rendent inutilisable en pratique. Ses points faibles sont essentiellement sa taille de clé (56 bits) et les cryptanalyses linéaires.

Ainsi, dans le contexte de la cryptographie légère, Leander *et al.* [LPPS07] proposent de remettre le DES au goût du jour afin de profiter de ses bonnes performances matérielles. Afin d'augmenter l'espace des clés, deux nouvelles clés de blanchiment sont xorées au clair (respectivement au chiffré) avant (resp. après) l'application du DES en lui-même, portant

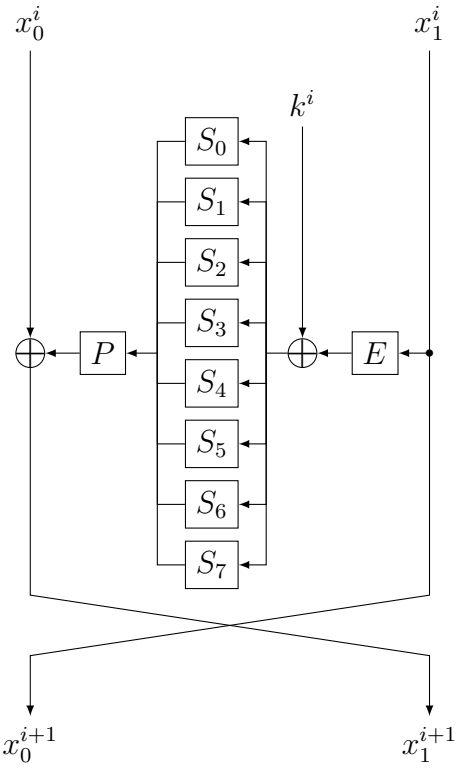


FIGURE 2.6 – Un tour de DES/DESXL.

ainsi la clé à $56+2\times64 = 184$ bits. Pour remédier à la vulnérabilité du DES face aux attaques linéaires, les 8 S-box du DES original sont remplacées par une unique S-box vérifiant toujours les critères originaux [BMP87] des S-box du DES mais aussi conçue pour sa résistance aux attaques linéaires et son faible coût d'implémentation. Cette version du DES renaissant de ses cendres se nomme DESXL [LPPS07] ; 'X' signifiant eXtended (étendu) à cause de l'augmentation de la taille des clés et 'L' signifiant Lightweight (léger) à cause de la nouvelle S-box.

2.2.2.3 LBlock

LBlock [WZ11] est un chiffrement léger présenté à ACNS 2011. Il chiffre des messages de 64 bits avec des clés de 80 bits en utilisant 32 tours d'un schéma de Feistel modifié : le bloc recevant la sortie de la fonction interne est décalé circulairement au début de chaque tour. La fonction interne se compose successivement d'une addition de clé, de 8 S-box 4 bits en parallèle et d'une permutation des mots de 4 bits. La figure 2.7 résume un tour de LBlock. Il est cependant à noter qu'un tour de LBlock est équivalent à un tour de déchiffrement de TWINE (cf. section 2.2.2.8), comme noté par les auteurs de TWINE dans le papier d'origine [SMMK12].

2.2.2.4 MIBS

Présenté en 2009, MIBS [ISSK09] est un schéma de Feistel à 32 tours chiffrant des messages de 64 bits avec des clés de 64 ou 80 bits. Sa fonction de Feistel se compose d'une

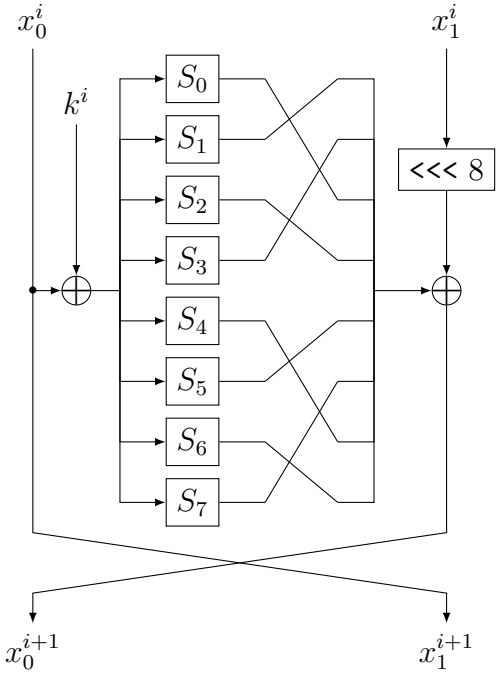


FIGURE 2.7 – Un tour de LBlock.

addition de sous-clé, de 8 S-box 4 bits en parallèle et enfin d'une application linéaire de diffusion agissant au niveau des nybbles. Ces choix font ressembler MIBS à une version lightweight du chiffrement Camellia [AIK+00]. La fonction de tour de MIBS est donnée à la figure 2.8.

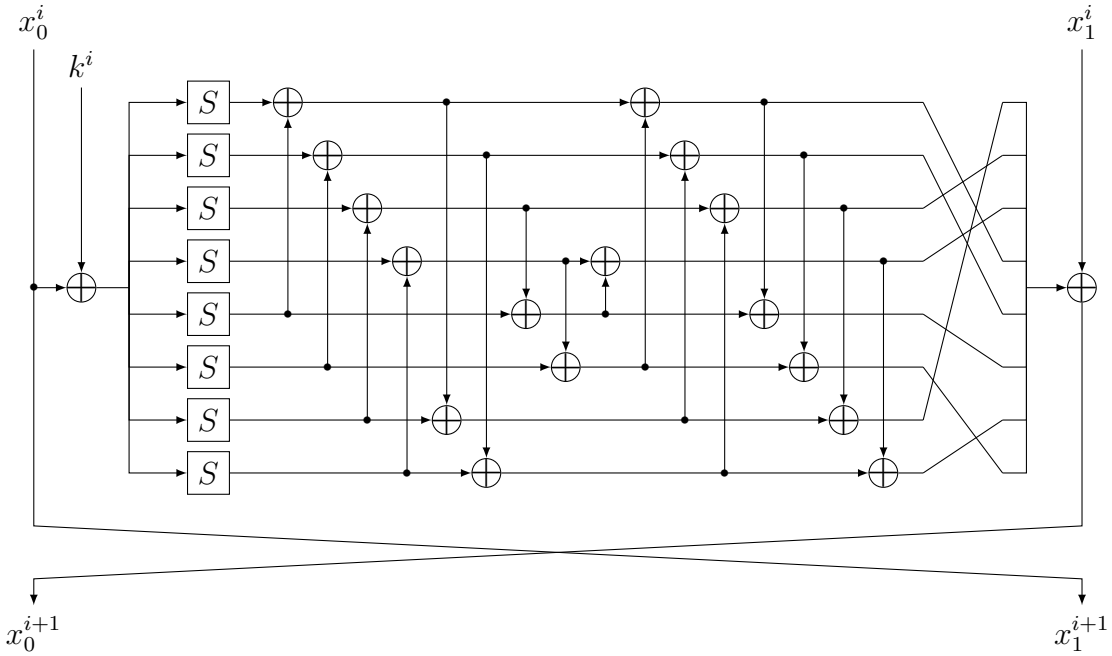


FIGURE 2.8 – Un tour de MIBS.

2.2.2.5 Piccolo

Présenté à CHES 2011, Piccolo [SIH+11] est un schéma de Feistel généralisé chiffrant des blocs de 64 bits avec des clés de 80 ou 128 bits. La vision Feistel est ici un peu modifiée : lors de l'application des fonctions internes du schéma, les 64 bits de message sont séparés en 4 blocs de 16 bits, tandis que lors de l'étape où les blocs sont permutés, la permutation agit au niveau des 8 octets et n'est pas un décalage circulaire. Piccolo utilise deux fonctions internes sur 16 bits. Celles-ci se composent de 4 S-box 4 bits en parallèle suivies d'une application linéaire de diffusion MDS puis d'une nouvelle étape de S-box en parallèle et enfin une addition de sous-clé. La figure 2.9 illustre un tour de Piccolo. Le schéma est répété 25 fois pour Piccolo-80 et 31 fois pour Piccolo-128.

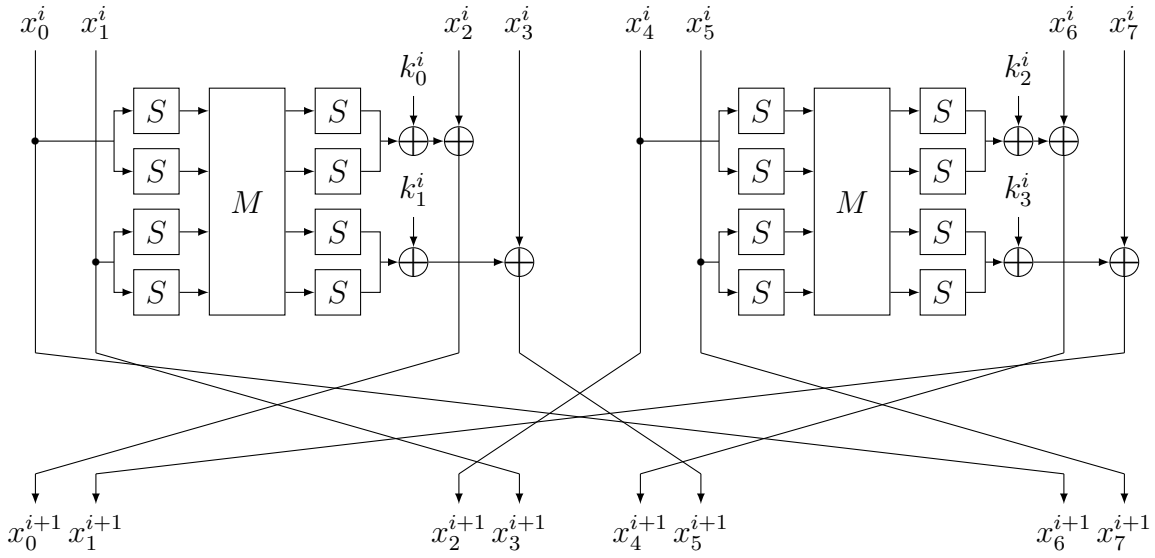


FIGURE 2.9 – Un tour de Piccolo, représenté au niveau des octets.

La S-box de Piccolo a été créée pour avoir une empreinte matérielle très petite. elle ne fait que 12 GE soit environ la moitié de la plupart des S-box 4 bits, tout en conservant de bonnes propriétés cryptographiques. Elle peut être implémentée avec 4 portes NOR, 3 portes XOR et 1 porte XNOR disposées suivant une structure Feistel, comme montré à la figure 2.10.

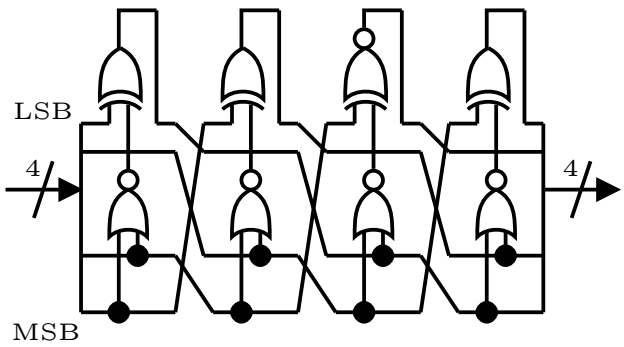


FIGURE 2.10 – S-box S de Piccolo. Extrait de [SIH+11].

2.2.2.6 SIMON

Proposé par la NSA en 2013 en même temps que son frère SPECK, SIMON [BSS+13] est une famille de chiffrement lightweight basé sur un schéma de Feistel. Cette famille se compose de 10 variantes avec différentes tailles de message et entre une et trois tailles de clé pour chaque taille de message. Les différentes combinaisons sont données par la table 2.1 en même temps que le nombre de tours pour chaque version de SIMON.

Taille de message	Taille de clé	Nombre de tours
32	64	32
48	72	36
48	96	36
64	96	42
64	128	44
96	96	52
96	144	54
128	128	68
128	192	69
128	256	72

Tableau 2.1 – Tailles de message et de clés possibles et nombre de tours correspondant pour la famille SIMON.

La fonction de Feistel de SIMON est choisie de manière à minimiser l'empreinte matérielle du chiffrement sans pour autant sacrifier les performances logicielles. Elle utilise pour cela des décalages circulaires (juste des cablages en matériel) et des opérations bit-à-bit, comme le montre la figure 2.11.

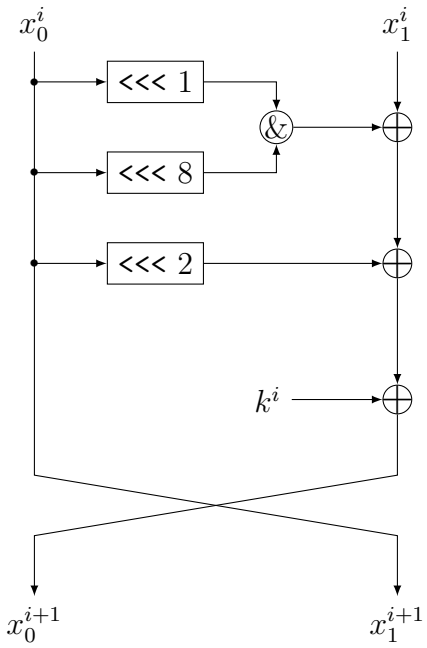


FIGURE 2.11 – Un tour de SIMON.

2.2.2.7 SPECK

Tout comme son frère SIMON, SPECK [BSS+13] est une famille de chiffrements lightweights présentés par la NSA en 2013. Il s'agit d'un schéma de Feistel modifié pouvant prendre différentes tailles de message et de clé, comme donné à la table 2.2.

Taille de message	Taille de clé	Nombre de tours	α	β
32	64	22	7	2
48	72	22	8	3
48	96	23	8	3
64	96	26	8	3
64	128	27	8	3
96	96	28	8	3
96	144	29	8	3
128	128	32	8	3
128	192	33	8	3
128	256	34	8	3

Tableau 2.2 – Tailles de message et de clés possibles ainsi que valeurs des rotations et nombre de tours correspondant pour la famille SPECK.

Contrairement à SIMON, SPECK vise l'efficacité logicielle particulièrement sur microcontroleur 8 bits, mais possède aussi de bonnes performances matérielles. Cela est possible grâce à l'utilisation de composantes de base efficaces dans les deux environnements : décalages circulaires, xors et addition modulaire. La fonction de tour de SPECK est décrite à la figure 2.12.

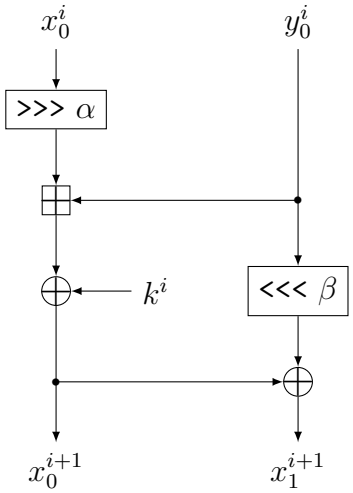


FIGURE 2.12 – Un tour de SPECK.

2.2.2.8 TWINE

TWINE est un chiffrement léger chiffrant des messages de 64 bits présenté à SAC 2012 par Suzaki *et al.* [SMMK12] et dont le but est d'offrir une empreinte matérielle réduite

tout en maintenant des performances logicielles décentes. Il s'agit d'un schéma de Feistel généralisé avec 16 blocs de 4 bits (nybbles) chacun qui accepte des clés de 80 ou 128 bits. La permutation π qui réarrange les blocs à la fin de chaque tour a été choisie de manière à maximiser la diffusion entre les blocs suivant un résultat de Suzaki et Minematsu [SM10]. Ce point sera développé en détail au chapitre 7 sur les schémas de Feistel généralisés. Il est cependant à noter qu'un tour de TWINE est équivalent à un tour de déchiffrement de LBlock (cf. section 2.2.2.3), comme noté par les auteurs de TWINE dans le papier d'origine [SMMK12]. En d'autres termes, si on écrit TWINE comme un Feistel classique à deux blocs en regroupant tous les nybbles en entrée de fonction dans un seul gros bloc de 32 bits et tous les nybbles en sortie de fonction dans un autre bloc de 32 bits, on obtient un schéma identique à celui de LBlock (à renumérotation des nybbles près). La permutation π utilisée est donnée par le tableau 2.3. La fonction interne utilisée 8 fois par tour se compose d'une addition avec une sous-clé suivie par une S-box 4 bits. Un tour de TWINE est donné par la figure 2.13. Le chiffrement complet se compose de 36 tours pour les deux longueurs de clés.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	5	0	1	4	7	12	3	8	13	6	9	2	15	10	11	14

Tableau 2.3 – Permutation π des blocs de TWINE .

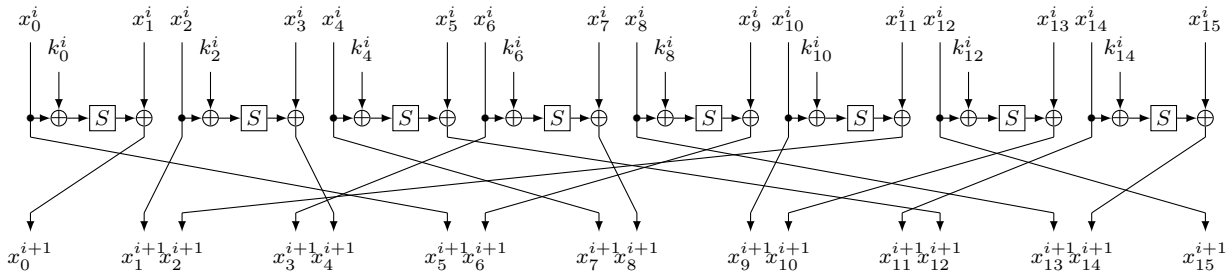


FIGURE 2.13 – Un tour de TWINE.

Chapitre 3

Cryptanalyse

Le chapitre précédent présentait des solutions utilisées par certains algorithmes de chiffrement pour respecter des contraintes d'implémentations fortes. Cependant le but premier de la cryptographie est la sécurité. C'est là qu'intervient la seconde facette de la cryptologie : la cryptanalyse.

Les attaques contre les chiffrements sont de deux types. Le premier type regroupe les attaques sur le chiffrement en tant qu'objet mathématique. Ce sont historiquement les attaques les plus anciennes, on parle d'ailleurs parfois de cryptanalyse *classique* pour les désigner. Dans ce contexte, l'attaquant exploite une faiblesse de l'algorithme de chiffrement pour retrouver la clé plus rapidement qu'avec une recherche exhaustive. Le second type d'attaques correspond aux attaques dites *physiques* qui en plus des faiblesses mathématiques du chiffrement exploitent les faiblesses physiques de l'implémentation de ce dernier.

On présente d'abord dans ce chapitre, le concept de distingueur qui formalise la notion de faiblesse d'un algorithme. On donne ensuite les principales classes d'attaques classiques (dans le sens de non-physique) réparties en deux catégories, les attaques dites statistiques qui exploitent plutôt une faiblesse de la partie confusion du chiffrement et les attaques dites structurelles qui exploitent au contraire plutôt une faiblesse de la partie diffusion. Dans une dernière section, on présente enfin brièvement la cryptanalyse physique.

3.1 Notion de distingueur

À l'heure actuelle, toutes les attaques existantes sur les algorithmes de chiffrement utilisent le concept de distingueur, il s'agit donc d'un outil fondamental en cryptanalyse. On donne d'abord la définition d'un distingueur sur un algorithme de chiffrement. Puis on explique comment transformer un distingueur en une attaque complète dans le cas d'un chiffrement itératif. Enfin on donne un résultat fondamental établi par Luby et Rackoff concernant une construction de Feistel idéalisée.

3.1.1 Définition

En termes simples, un distingueur est un algorithme A qui, comme son nom l'indique, cherche à distinguer deux situations via un jeu de questions/réponses. Plus précisément, un distingueur a accès à une fonction auxiliaire, appelée oracle, à qui il peut faire des requêtes : il donne une valeur à l'oracle et celui-ci lui renvoie une autre valeur. En interne, cet oracle est soit le chiffrement attaqué C soit une permutation aléatoire P_n tirée uniformément parmi toutes les permutations de $\mathbb{F}_2^n$. Sauf que le distingueur A ne sait pas de quoi est constitué l'oracle. Le but pour le distingueur est, à partir d'un certain nombre q de requêtes qu'il aura faites à l'oracle, de deviner s'il est face à C ou bien à P_n. Par convention, le distingueur retourne la valeur 1 s'il pense être en face de C ou la valeur 0 sinon. On note $p = \Pr[\mathsf{A}^{\mathsf{C}} = 1]$ la probabilité que le distingueur A renvoie la valeur 1 sachant que l'oracle est implémenté avec C. De manière analogue, on note $p^* = \Pr[\mathsf{A}^{\mathsf{P}_n} = 1]$ la probabilité

que le distingueur A renvoie la valeur 1 sachant que l'oracle est implémenté avec P_n. On mesure alors l'efficacité du distingueur via la quantité $|p - p^*|$ que l'on nomme avantage du distingueur. Plus l'avantage sera important, meilleur sera le distingueur. En d'autres termes, pour trouver un bon distingueur, on recherche une propriété sur le chiffrement liant clairs et chiffrés qui se produit avec une probabilité significativement différente de ce qu'on observerait sur une permutation aléatoire.

3.1.2 Attaque sur les derniers tours

Avoir un distingueur sur une partie du chiffrement indique une faiblesse de l'algorithme, mais cela n'est pas une attaque. Dans le cas d'un chiffrement itératif, il est cependant possible de transformer un distingueur sur un certain nombre de tours de chiffrement en une attaque.

Supposons que notre chiffrement E_K soit composé de r tours distincts. En d'autres termes,

$$E_K = F_{K_r} \circ F_{K_{r-1}} \circ \cdots \circ F_{K_1}.$$

Chaque fonction de tour F_{K_i} dépend d'une clé de tour K_i calculée à partir de la clé maître K. Supposons que l'on dispose d'un distingeur A sur le chiffrement réduit à ses $r-1$ premiers tours d'avantage $|p - p^*|$ où p est la probabilité d'observer la propriété dans le cas du chiffrement et p^* celle dans le cas d'une permutation aléatoire. On peut alors transformer ce distingueur en une attaque sur r tours, c'est-à-dire sur le chiffrement complet en procédant comme suit :

1. Collecter suffisamment de couples clairs-chiffrés (m_i, c_i) pour $i \in \{1, \cdots, q\}$;
2. Deviner la valeur de la clé du dernier tour K_r ;
3. Déchiffrer le dernier tour des messages c_i à l'aide de K_r, obtenir les chiffrés intermédiaires $c'_i = F^{-1}_{K_r}(c_i)$;
4. Compter le nombre de couples (m_i, c'_i) qui vérifient la propriété utilisée dans le distingueur. Si ce nombre est plus proche de $q \cdot p$ que de $q \cdot p^*$ alors on a retrouvé la bonne clé K_r, sinon retourner à l'étape 2 avec une nouvelle valeur de clé K_r.

L'attaque repose sur le principe suivant : à l'étape 2, si l'hypothèse sur la clé de tour K_r est bonne, alors lors de l'étape 4 on va observer la propriété du distingueur avec probabilité p, si elle est fausse, on suppose que l'on se retrouve dans le cas d'une permutation aléatoire et qu'on observe alors la propriété avec une probabilité p^*. Le nombre q de couples clairs-chiffrés doit être suffisamment grand pour que la valeur observée à l'étape 4 puisse être attribuée à l'un ou l'autre cas sans pouvoir être imputée à un écart statistique. En particulier, plus la propriété utilisée dans le distingueur sera inhabituelle pour une permutation aléatoire ou en d'autres termes plus l'avantage $|p-p^*|$ du distingueur sera grand, moins il faudra de couples clairs-chiffrés pour séparer les deux cas. Un bon cryptanalyste s'efforcera donc de trouver des distingueurs avec le meilleur avantage.

En pratique, on est souvent incapable d'attaquer avec succès un chiffrement complet. On attaque alors des versions affaiblies de ce chiffrement obtenues en réduisant le nombre de tours par rapport à la version "officielle". Le nombre maximal de tours que l'on peut attaquer avec une complexité en dessous de la recherche exhaustive fournit alors une nouvelle mesure de l'efficacité de l'attaque.

3.1.3 Indistinguabilité du schéma de Feistel

Grâce aux distingueurs, on est en mesure d'évaluer la faiblesse d'un algorithme en le comparant à ce qu'on observerait pour une permutation aléatoire. On peut alors se demander comment construire des algorithmes résistants aux attaques par distingueur. Un des résultats fondamentaux est le travail de Luby et Rackoff [LR88]. Leurs résultats concernent le schéma de Feistel, décrit à la section 1.2.2.2 de cette thèse, et bornent l'avantage de tout distingueur sur cette construction en supposant que les fonctions internes sont des fonctions aléatoires idéales. Leurs résultats sont donnés dans les théorèmes suivants. La démonstration, bien plus simple, donnée ici est due à Maurer [Mau02].

Théorème 3.1. *Soit f_1^*, f_2^*, f_3^* trois fonctions, choisies indépendamment et au hasard uniforme parmi les fonctions de $\mathbb{F}_2^n$ dans lui-même. Soit $\Psi = \Psi(f_1^*, f_2^*, f_3^*)$ la permutation de $\mathbb{F}_2^{2n}$ obtenue à l'aide de trois tours de la construction de Feistel utilisant successivement f_1^*, f_2^* puis f_3^*. Introduisons la quantité, appelée avantage prp (pseudorandom permutation) de Ψ, $\mathrm{Adv}_\Psi^{\mathsf{prp}}(q)$ définie par*

$$\mathrm{Adv}_\Psi^{\mathsf{prp}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CPA}} \left|\Pr[\mathsf{A}^\Psi = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n} = 1]\right|.$$

où P_n est une permutation aléatoire et où le maximum est pris sur tous les distingueurs A en modèle à clair choisi (chosen plaintext attack, CPA*), effectuant q requêtes à Ψ ou P_n et avec une puissance de calcul non-bornée.*

On a alors :

$$\mathrm{Adv}_\Psi^{\mathsf{prp}}(q) \leq \frac{q^2}{2^n}. \tag{3.1}$$

Démonstration. Pour chaque clair m_i avec $i \in \{1, \cdots, q\}$, notons L_i sa partie gauche et R_i sa partie droite, de sorte que $m_i = L_i||R_i$ où $||$ désigne la concaténation. Notons aussi $S_i = L_i \oplus f_1^*(R_i)$, $T_i = R_i \oplus f_2^*(S_i)$ et $V_i = S_i \oplus f_3^*(T_i)$ de sorte que le chiffré partiel soit successivement $m_i = L_i||R_i$ puis $R_i||S_i$ puis $S_i||T_i$ en enfin $T_i||V_i = c_i$, le chiffré (cf. la figure 3.1 pour une vision graphique) Sans perte de généralité, on suppose que les couples clairs-chiffrés (m_i, c_i) sont deux à deux distincts. En effet, demander une seconde fois le chiffré correspondant à un clair ne donne pas plus d'information que lors de la première requête.

Si les S_i sont tous distincts alors les $T_i = R_i \oplus f_2^*(S_i)$ sont complètement aléatoires puisque la fonction f_2^* est une fonction aléatoire. De même si les T_i sont distincts alors les $V_i = S_i \oplus f_3^*(T_i)$ sont aléatoires. Ainsi si ces deux événements se produisent, alors les chiffrés $c_i = T_i||V_i$ sont aléatoires et la fonction $\Psi(f_1^*, f_2^*, f_3^*)$ se comporte donc comme une fonction aléatoire pour ces couples clairs-chiffrés là.

Ainsi la possibilité de distinguer $\Psi(f_1^*, f_2^*, f_3^*)$ d'une fonction aléatoire ne peut se faire que lorsque certains S_i ou certains T_i collisionnent. Or pour $i \neq j$, si $R_i \neq R_j$ alors $\Pr[S_i = S_j] = 2^{-n}$; si $R_i = R_j$ alors $L_i \neq L_j$ puisque les clairs sont supposés deux à deux distincts et dans ce cas, $S_i \neq S_j$. D'où dans tous les cas, $\Pr[S_i = S_j] \leq 2^{-n}$. Il en va de même pour les T_i : $\Pr[T_i = T_j] \leq 2^{-n}$. En somme la probabilité de collision sur les S_i ou sur les T_i est majorée par :

$$\frac{q(q-1)}{2} \cdot 2^{-n} + \frac{q(q-1)}{2} \cdot 2^{-n} \leq \frac{q^2}{2^n}.$$

Et ainsi on obtient bien l'inégalité (3.1).

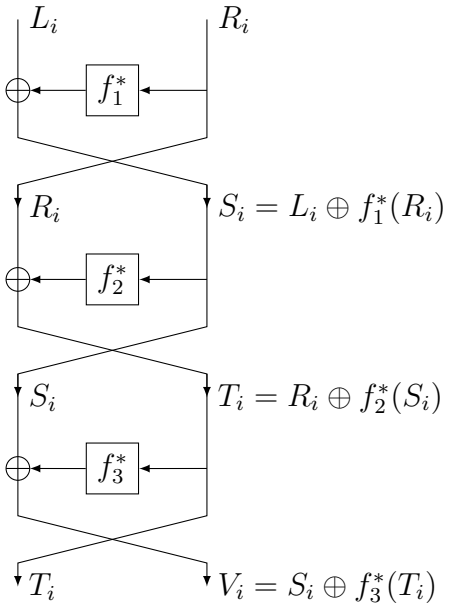


FIGURE 3.1 – Trois tours d'un schéma de Feistel.

□

Ce théorème montre que pour distinguer un schéma de Feistel à trois tours avec un avantage non-négligeable, le nombre de couples clairs-chiffrés doit être de l'ordre de $2^{\frac{n}{2}}$, dans le cas d'une attaque à clairs choisis. Lorsque l'attaquant est autorisé à faire également des requêtes à chiffrés choisis, trois tours de Feistel ne suffisent plus. En effet, avec les notations de la preuve précédente, il suffit de soumettre deux requêtes $L_1||R$ et $L_2||R$ avec la même partie droite. Notons $V_1||T_1$ et $V_2||T_2$ leurs chiffrés respectifs et déchiffrons le message $(V_2 \oplus L_1 \oplus L_2)||T_2$. Soit $L'||R'$ le clair obtenu. On a alors que la relation $R \oplus R' = T_1 \oplus T_2$ est toujours vérifiée, ce qui est rarement le cas pour une fonction aléatoire. En revanche, il suffit d'ajouter un quatrième tour au schéma pour retrouver la borne obtenue précédemment, comme précisé dans le théorème suivant.

Théorème 3.2. *Soit $f_1^*, f_2^*, f_3^*, f_4^*$ quatre fonctions, choisies indépendamment et au hasard uniforme parmi les fonctions de $\mathbb{F}_2^n$ dans lui-même. Soit $\Psi = \Psi(f_1^*, f_2^*, f_3^*, f_4^*)$ la permutation de $\mathbb{F}_2^{2n}$ obtenue à l'aide de quatre tours de la construction de Feistel utilisant successivement f_1^*, f_2^*, f_3^* puis f_4^*. Introduisons la quantité, appelée avantage sprp (strong pseudorandom permutation) de Ψ, $\mathrm{Adv}_\Psi^{\mathsf{sprp}}(q)$ définie par*

$$\mathrm{Adv}_\Psi^{\mathsf{sprp}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CCA}} \left| \Pr[\mathsf{A}^{\Psi,\Psi^{-1}} = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n,\mathsf{P}_n^{-1}} = 1] \right|$$

où P_n est une permutation aléatoire, P_n^{-1} son inverse et où le maximum est pris sur tous les distingueurs A en modèle à chiffré choisi (chosen ciphertext attack, CCA), effectuant q requêtes à (Ψ, Ψ^{-1}) ou à $(\mathsf{P}_n, \mathsf{P}_n^{-1})$ et avec une puissance de calcul non-bornée.

On a alors :

$$\mathrm{Adv}_\Psi^{\mathsf{sprp}}(q) \leq \frac{q^2}{2^n}. \tag{3.2}$$

Moralement, ces résulats signifient que le schéma de Feistel est une construction légitime : si les fonctions internes au schéma se comportent aléatoirement alors la construction totale se comporte aussi aléatoirement.

3.2 Attaques statistiques

Cette section ainsi que la suivante donne les principales classes d'attaques existant sur les chiffrements en bloc. Cette section se consacre aux attaques dites statistiques qui exploitent plutôt une faiblesse de la partie confusion du chiffrement.

3.2.1 Cryptanalyse différentielle

La cryptanalyse différentielle a été introduite par Biham et Shamir [BS90] et consiste à observer comment se propagent des différences au travers du chiffrement. Plus précisément, il s'agit d'une attaque à clair choisi dans laquelle on se donne une différence $\delta_0 \in \mathbb{F}_2^n$ entre des couples de clairs x et $x+\delta_0$ et où on s'intéresse à ce que peut devenir cette différence au niveau des chiffrés. Cependant, à chaque fois que l'on rencontre une application non-linéaire, plusieurs différences sont possibles en sortie. On s'intéresse alors à la différence en sortie la plus probable, c'est-à-dire celle qui sera vérifiée pour le plus de couples de clairs $(x, x+\delta_0)$. En d'autres termes pour une fonction F, on cherche une différence δ_1 telle que la probabilité

$$\Pr_{x \in \mathbb{F}_2^n}[F(x+\delta_0) - F(x) = \delta_1] \tag{3.3}$$

soit maximale. Mais on peut aussi choisir la valeur de la différence δ_0 que l'on applique sur les clairs afin de maximiser au mieux la probabilité (3.3). Ainsi lors d'une cryptanalyse différentielle, on cherchera deux valeurs de différences δ_0 et δ_1 telles que la probabilité (3.3) sur un certain nombre de tours de chiffrement soit la plus grande possible. Ceci nous donne un distingueur sur notre chiffrement que l'on transforme en une attaque sur les derniers tours via une hypothèse sur une partie de la clé.

3.2.2 Cryptanalyse linéaire

Introduite par Gilbert, Chassé et Tardy-Corfdir [GC90, TG91], la cryptanalyse linéaire est une attaque à clair connu qui exploite des relations linéaires entre les bits du clair et du chiffré qui se produisent de manière anormale. Plus précisément, on recherches deux n-uplets α et β, appelés masques, tels que la relation linéaire

$$\langle \alpha | x \rangle + \langle \beta | F(x) \rangle \tag{3.4}$$

n'ait pas une distribution uniforme, où $\langle \alpha | x \rangle$ désigne le produit scalaire usuel sur $\mathbb{F}_2^n$. Plus précisément, on recherche α et β tels que la probabilité $\Pr[\langle \alpha | x \rangle + \langle \beta | F(x) \rangle = 1]$ s'éloigne le plus possible de $1/2$.

La cryptanalyse linéaire est particulièrement célèbre pour avoir été la première attaque à casser le chiffrement DES [Mat93, Mat94].

3.3 Attaques structurelles

Contrairement aux attaques dites statistiques présentées précédemment qui exploitent les limites de la confusion procurées par les composantes non-linéaires d'un algorithme, les attaques dites structurelles se focalisent plus sur la manière dont s'effectue la diffusion entre les différentes parties du message en train d'être chiffré. On présente ici deux attaques structurelles ; les attaques par différentielle impossible et les attaques dites intégrales.

3.3.1 Attaque différentielle impossible

Décrites pour la première fois par Knudsen en 1998 lors de la soumission du chiffrement DEAL [Knu98] au concours AES, puis reprises en 1999 par Biham, Biryukov et Shamir [BBS99] à CRYPTO'99, les attaques différentielles impossibles sont une classe d'attaques exploitant la structure présente dans l'algorithme. Comme leur nom le suggère, les attaques différentielles impossibles utilisent des différentielles se produisant avec probabilité nulle à un endroit du chiffrement. Si pour une hypothèse sur la clé, on trouve un couple clair-chiffré vérifiant cette propriété différentielle, alors on sait que notre hypothèse sur la clé est fausse et on peut réduire ainsi l'ensemble des clés possibles.

Pour trouver une caractéristique différentielle impossible, Biham, Biryukov et Shamir [BBS99] utilisent une technique dite de "miss-in-the-middle". Elle consiste à trouver deux événements certains, c'est-à-dire qui arrivent avec probabilité 1, mais qui ne peuvent se produire simultanément. Dans le cas des différentielles impossibles, les deux événements certains sont des transitions différentielles, l'une sur les premiers tours du chiffrement et l'autre sur les derniers tours telles que la transition entre la sortie du premier et l'entrée du second soit impossible.

À Indocrypt 2003, Kim *et al.* [KHS+03] proposent une méthode générique, appelée la $\mathcal{U}$-méthode, pour trouver des différentielles impossibles en mode "miss-in-the-middle", lorsque les composantes non-linéaires (e.g. S-box) sont bijectives, en se focalisant sur des différences au niveau des blocs. En effet, la plupart des algorithmes disposent d'une notion de bloc, c'est-à-dire un n-uplet tel que "l'algorithme se décrive bien au niveau des blocs". Citons par exemple l'octet pour l'AES ou encore les deux branches d'un schéma de Feistel. Kim *et al.* notent alors que connaissant certaines propriétés sur les différences en entrée d'un tour, il est souvent possible de prédire ces mêmes propriétés au tour suivant. Plus précisément, ils considèrent cinq propriétés au niveau des différences sur les blocs :

- nulle (notée 0) ;
- non-nulle fixée (notée γ) ; il s'agit souvent de la différence que l'on introduit dans les clairs ou les chiffrés ;
- non-nulle non-fixée (notée δ) ; ce sont les différences dont on ne maîtrise pas la valeur mais dont on sait qu'elles sont non-nulles, par exemple parce qu'elles sont en sortie d'une S-box bijective dont l'entrée avait une différence non-nulle ;
- xor d'une différence non-nulle fixée et d'une différence non-nulle non-fixée (notée $\gamma \oplus \delta$) ;
- non-fixée (notée '?') ; cela correspond aux cas où l'on ne sait plus rien dire.

Ces propriétés sont alors modifiées au cours du chiffrement selon les fonctions rencontrées. Par exemple, lorsqu'une différence traverse une fonction non-linéaire (bijective), les différences γ deviennent des δ puisque à une différence en entrée donnée peuvent correspondre plusieurs différences en sortie. La bijectivité de la fonction garantit en revanche

que la différence observée est non-nulle, d'où le type δ. Si la différence en entrée de fonction est de type $\gamma \oplus \delta$, on ne sait en général rien dire en sortie, d'où un type '?' en sortie. Les autres types ne sont pas modifiés par le passage dans une fonction non-linéaire. La figure 3.2 contient un exemple de caractéristique différentielle utilisée dans la $\mathcal{U}$-méthode dans le cas d'un schéma de Feistel.

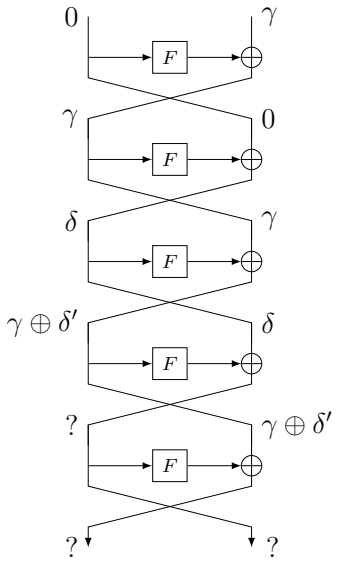


FIGURE 3.2 – Exemple sur schéma de Feistel de caractéristique différentielle utilisée dans la $\mathcal{U}$-méthode de Kim *et al.* [KHS+03].

Ces caractéristiques différentielles sont vraies avec probabilité 1. En effet, même si on ne connaît pas la valeur réelle de la différentielle en sortie, on connaît toujours son type. Cela correspond aux événements certains de la technique "miss-in-the-middle". Pour obtenir une caractéristique différentielle impossible, il suffit alors de combiner deux de ces caractéristiques différentielles de manière à obtenir une impossibilité sur au moins un bloc. Par exemple si la différence sur les premiers tours prédit un type $\gamma \oplus \delta$ sur un bloc au tour du milieu et que la différence sur les derniers tours prédit une différence γ (avec le même γ que sur les premiers tours) sur ce même bloc au même tour alors ces deux différences ne peuvent être égales puisqu'on devrait dans ce cas avoir une différence $\gamma = \gamma \oplus \delta$ ce qui donne $0 = \gamma \oplus \gamma \oplus \delta = \delta$; ce qui n'est pas possible. Les autres impossibilités sont nul (0) versus non-nul (γ ou δ) et γ versus γ' avec $\gamma \neq \gamma'$.

Cette méthode a été généralisée par Luo *et al.* [LWLG09] en gardant en plus en mémoire si deux différences à deux endroits du chiffrement sont égales.

3.3.2 Attaque intégrale

La cryptanalyse intégrale, a été introduite par Daemen, Knudsen et Rijmen avec le chiffrement Square [DKR97], ancêtre du gagnant de la compétition AES [FIP01] Rijndael. De ce fait, elle est aussi connue sous le nom de "square attack". La cryptanalyse intégrale est une attaque dite par saturation, c'est-à-dire que l'on se donne 2^n clairs qui prennent toutes les valeurs possibles sur n bits (correspondant généralement à un "bloc", typiquement branche de Feistel, octet, quartet,...) et qui sont tous égaux sur les autres bits. Le but de la

cryptanalyse intégrale est de prédire la valeur de la somme (ou intégrale) de ces 2^n valeurs sur certains blocs après quelques tours de chiffrements.

Comme dans le cas de la cryptanalyse différentielle impossible, on dispose d'un certain nombre de propriétés dont on cherche à savoir comment celles-ci se propagent à travers le chiffrement. Les propriétés considérées sont au nombre de quatre et sont :

- $\mathcal{P}$ (pour Permutation) ; le bloc considéré prend toutes les valeurs possibles ;
- $\mathcal{C}$ (pour Constant) ; le bloc considéré prend toujours la même valeur ; notons que pour deux blocs $\mathcal{C}$ différents la valeur de la constante n'est pas forcément la même ;
- $\mathcal{S}$ (pour Somme) ; la somme de tous les chiffrés est nulle ; notons que si on a $\mathcal{P}$ ou $\mathcal{C}$ alors $\mathcal{S}$ est vérifiée aussi, par conséquent on n'écrira $\mathcal{S}$ que lorsqu'on sera incapable de dire si on est $\mathcal{P}$ ou $\mathcal{C}$;
- '?' ; lorsqu'on ne sait plus rien dire.

Lorsqu'un bloc franchit une fonction (bijective), les propriétés $\mathcal{P}$ et $\mathcal{C}$ ne sont pas modifiées, en revanche $\mathcal{S}$ devient '?'. Du point de vue des combinaisons entre deux blocs (e.g. xors), $\mathcal{P} \oplus \mathcal{C}$ reste $\mathcal{P}$ puisque on applique la même fonction (xor de constante) à tous les éléments du bloc $\mathcal{P}$. En revanche, $\mathcal{P} \oplus \mathcal{P}$ ne reste en général pas $\mathcal{P}$ mais on a toujours la propriété $\mathcal{S}$ par linéarité de la somme. La figure 3.3 contient un exemple de caractéristique intégrale dans le cas d'un schéma de Feistel.

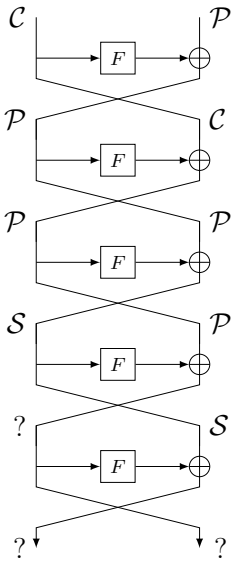


FIGURE 3.3 – Exemple sur schéma de Feistel de caractéristique intégrale.

La propriété intégrale se généralise à des intégrales d'ordre supérieur. Pour une intégrale d'ordre ℓ, on considère cette fois-ci $2^{n\ell}$ clairs qui prennent toutes les valeurs possibles sur ℓ blocs et tels que la somme de toutes ces valeurs soit prévisible après plusieurs tours.

De manière analogue à la $\mathcal{U}$-méthode de Kim *et al.* pour les différentielles impossibles, Zhang *et al.* [ZSW+12] proposent une méthode générique pour trouver des intégrales au niveau des blocs du chiffrement.

Une attaque intégrale sur la fonction de hachage, candidate SHA-3, Grøstl a été réalisée pendant cette thèse et est présentée en Annexe 11.3.2.4.

3.4 Attaques physiques

Un algorithme cryptographique est conçu pour être robuste mathématiquement, c'est-à-dire résistant à la cryptanalyse classique. Cependant, une fois implémenté dans un circuit, un attaquant peut utiliser les propriétés de ce dernier pour mener son attaque. Ces attaques utilisant les propriétés physiques de l'implémentation d'un algorithme cryptographique sont appelées attaques physiques. Il existe deux grandes familles d'attaques physiques : les *attaques par observation ou par canaux auxiliaires* et les *attaques par injection(s) de faute(s).*

Les attaques par observation ou par canaux auxiliaires, notées SCA (Side Channel Analysis), se basent sur l'observation du circuit pendant les calculs liés au chiffrement. Si un algorithme est sécurisé théoriquement, le circuit dans lequel il est implémenté ne l'est pas. C'est la frontière entre la théorie et la pratique. L'algorithme de chiffrement est théorique, le circuit lui est bien concret. Ainsi ce dernier a une consommation de courant, un temps de calcul, etc. Ce sont ces paramètres physiques qui vont nuire à la sécurité de l'algorithme de chiffrement. Ces paramètres physiques ne sont pas faciles à anticiper d'un point de vue théorique, on parle de *fuite d'information* par un canal auxiliaire. De plus, seules des mesures de ces paramètres physiques peuvent être obtenues, ces mesures dépendent donc à la fois de la fuite mais aussi de l'outil de mesure ; elles sont donc entachées de bruit.

Les attaques par injection(s) de faute(s), notées FIA (Fault Injection Analysis), analysent l'effet d'une perturbation intentionnelle sur le fonctionnement du circuit. Deux cas se distinguent alors, le comportement normal du circuit et le comportement perturbé du circuit. Les attaques FIA sont généralement puissantes et coûteuses. Ce sont également souvent des attaques invasives ; elles comportent un risque non négligeable qui est l'*endommagement* voire la *destruction* du circuit. En effet les outils d'injection tel que le laser peuvent détruire une partie du circuit. C'est pourquoi lors d'une attaque en faute, l'attaquant cherche toujours à minimiser le nombre de fautes injectées, afin de limiter les risques de destruction même partielle du circuit. Les moyens pour injecter une faute dans un circuit sont divers et variés : laser, modification de la tension, modification de l'horloge interne du circuit, etc.

Deuxième partie

Registres à décalage à rétroaction avec retenue

Chapitre 4

État de l'art

Ce chapitre a pour but d'introduire et de présenter les automates de type FCSR et leurs utilisations en cryptographie. On définit dans un premier temps les automates de type LFSR qui sont très similaires aux FCSR mais avec une structure plus simple. À partir des LFSR, on peut introduire les FCSR par analogie. On discute ensuite des propriétés liées à l'implémentation des FCSR, en exhibant notament deux familles de FCSR, une orientée matériel et une orientée logiciel. On introduit finalement quelques attaques existant sur ces algorithmes.

Le lecteur désirant plus de renseignements sur la théorie des LFSR et des FCSR pourra consulter le livre de Klapper et Goreski [GK12] sur le sujet.

4.1 LFSR : Automates linéaires

4.1.1 LFSR en mode Galois & Fibonacci

Les registres à décalage à rétroaction linéaire (linear feedback shift register en anglais ou LFSR) sont un type d'automates lié aux suites récurrentes linéaires sur un corps fini (en général $\mathbb{F}_2$). Un LFSR (binaire) en mode Fibonacci de longeur n est un automate dont l'état interne se compose de n mémoires $m_0(t), \cdots, m_{n-1}(t)$ contenant, à chaque instant $t \in \mathbb{N}$, chacune un bit. À chaque tic d'une horloge externe contrôlant l'automate, celui-ci est mis à jour de la manière suivante :

- le contenu de chaque mémoire est décalé d'une position vers la droite, *i.e.* $m_i(t+1) = m_{i+1}(t)$ pour $i \in \{0, \cdots, n-2\}$;
- la cellule la plus à gauche reçoit une nouvelle valeur $m_{n-1}(t+1)$ s'exprimant comme une combinaison linéaire fixe des valeurs des cellules à l'instant t :

$$m_{n-1}(t+1) = \sum_{i=1}^{n} q_i \cdot m_{n-i}(t). \tag{4.1}$$

Le contenu de la mémoire la plus à droite $m_0(t)$ constitue la sortie du LFSR. La figure 4.1 résume cette définition.

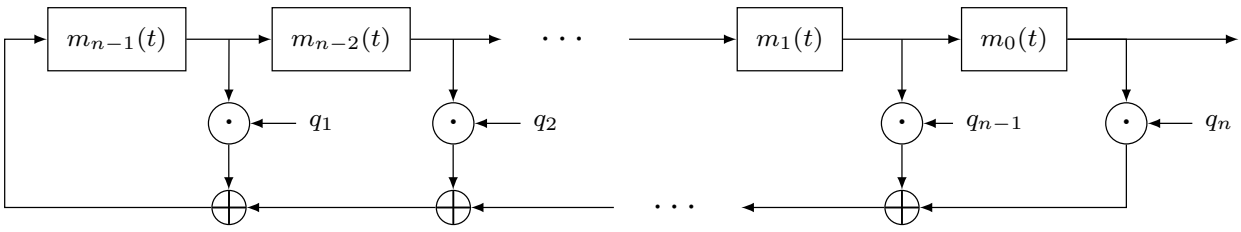


FIGURE 4.1 – LFSR en mode Fibonacci.

Ainsi, un LFSR permet d'engendrer une suite de bits $\big(m_0(t)\big)_{t\in\mathbb{N}}$ déterminée par ces premiers termes $m_0(0)$, $m_0(1) = m_1(0)$, …, $m_0(n-1) = m_{n-1}(0)$ et une relation de récurrence linéaire provenant de (4.1) :

$$m_0(t+n) = \sum_{i=1}^{n} q_i \cdot m_0(t+n-i).$$

À partir des coefficients de la relation (4.1), on peut définir un autre mode, le mode Galois, pour les LFSR. Un LFSR en mode Galois de longueur n se compose toujours de n mémoires binaires $m_0(t), \cdots, m_{n-1}(t)$ mais la manière dont celles-ci sont mises à jour à chaque tic d'horloge se fait différemment du cas Fibonacci. Dans le cas Fibonacci, une unique mémoire reçoit une combinaison linéaire de toutes les autres, tandis que dans le cas Galois, une unique mémoire influence toutes les autres. Plus précisément, comme indiqué à la figure 4.2, on a :

$$m_i(t+1) = m_{i+1}(t) \oplus q_{i+1} \cdot m_0(t) \text{ pour } i \in \{0, \cdots, n-1\}, \tag{4.2}$$

avec la convention que $m_n(t) = 0$, *i.e.* $m_{n-1}(t+1) = q_n \cdot m_0(t)$.

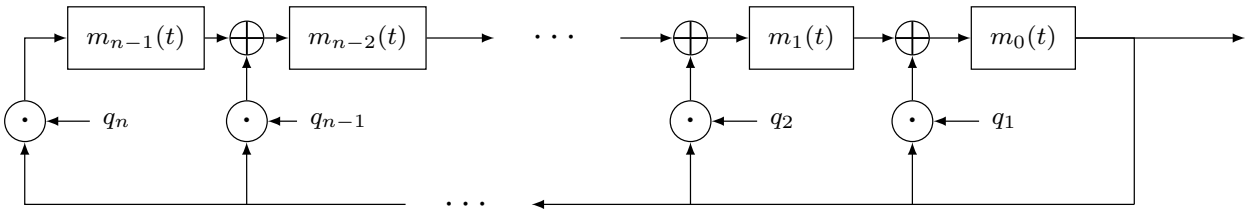


FIGURE 4.2 – LFSR en mode Galois.

4.1.2 Représentation matricielle

Comme on peut le voir sur les équations de mise à jour du registre d'un LFSR (équation (4.1) pour le mode Fibonacci et équation (4.2) pour le mode Galois), l'état du LFSR à un instant donné s'exprime linéairement en fonction de l'état à l'instant précédent. Notons $m(t) = (m_0(t), \cdots, m_{n-1}(t))^T \in \mathbb{F}_2^n$ le vecteur colonne représentant l'état du LFSR à l'instant t, il existe donc une matrice binaire T de taille $n \times n$ telle que

$$m(t+1) = T \cdot m(t). \tag{4.3}$$

On appelle cette matrice la matrice de transition du LFSR. Les matrices de transition des modes Galois et Fibonacci sont données à la figure 4.3.

On peut alors généraliser les LFSR en mode Galois et Fibonacci en autorisant n'importe quelle matrice binaire T dans l'équation (4.3).

Définition 4.1. *Un automate fini linéaire (linear finite state machine en anglais ou LFSM) de taille n se compose d'un vecteur colonne binaire de taille n, noté $m(t) = (m_0(t), \cdots, m_{n-1}(t))^T \in \mathbb{F}_2^n$, appelé état du LFSM, et d'une matrice $T \in \mathcal{M}_n(\mathbb{F}_2)$, appelée*

$$T_F = \begin{pmatrix} 0 & 1 & & (0) \\ \vdots & & \ddots & \\ 0 & (0) & & 1 \\ q_n & q_{n-1} & \cdots & q_1 \end{pmatrix} \qquad T_G = \begin{pmatrix} q_1 & 1 & & (0) \\ \vdots & & \vdots & \\ q_{n-1} & (0) & & 1 \\ q_n & 0 & \cdots & 0 \end{pmatrix}$$

FIGURE 4.3 – Matrices de transition d'un LFSR en mode Fibonacci T_F (gauche) et en mode Galois T_G (droite).

matrice de transition du LFSM. Le comportement du LFSM est le suivant : à l'instant $t = 0$, l'état est intialisé à une valeur $m(0) \in \mathbb{F}_2^n$; à chaque instant $t+1$ un nouvel état est calculé à partir du précédent à l'aide de l'équation (4.3) :

$$m(t+1) = T \cdot m(t).$$

On cherche alors à caractériser les suites binaires que l'on observe dans chacune des mémoires $\big(m_i(t)\big)_{t\in\mathbb{N}}$. On utilise pour cela une représentation sous forme de série formelle de ces suites.

Définition 4.2. *Soit $t_0 \in \mathbb{N}$ et $\big(s(t)\big)_{t\in\mathbb{N}}$ une suite binaire. On appelle série associée à la suite, la série formelle $S \in \mathbb{F}_2[[X]]$ définie par :*

$$S(X) = \sum_{t=0}^{\infty} s(t)X^t.$$

Dans le cas des LFSM, on note $M_i(t_0)$ la série associée à la suite $\big(m_i(t)\big)_{t\geq t_0}$ observée en position i de l'état du LFSM à partir de l'instant t_0, c'est-à-dire :

$$M_i(t_0)(X) = \sum_{t=t_0}^{\infty} m_i(t)X^{t-t_0}.$$

On note de plus $M(t_0) \in \mathbb{F}_2[[X]]^n$ le vecteur regroupant ces séries formelles :

$$M(t_0)(X) = (M_0(t_0)(X), \cdots, M_{n-1}(t_0)(X))^T = \sum_{t=t_0}^{\infty} m(t)X^{t-t_0}.$$

Comme l'état d'un LFSM ne peut prendre qu'un nombre fini de valeurs et que l'état suivant ne dépend que de l'état actuel, les suites générées par un LFSM sont (ultimement) pérodiques. Par conséquent, les séries $M_i(t_0)(X)$ vivent dans un sous-ensemble particulier de $\mathbb{F}_2[[X]]$. Le théorème suivant caractérise cet ensemble ; il s'agit des séries dites rationnelles, c'est-à-dire s'écrivant comme quotient de deux polynômes. Dans ce cas, on peut alors déterminer facilement la période de la suite et donner une expression du t-ième terme de la suite directement en fonction de t.

Proposition 4.1. *Soit une série $S \in \mathbb{F}_2[[X]]$ alors :*

1. *S est ultimement périodique si et seulement s'il existe deux polynômes $p, q \in \mathbb{F}_2[X]$ avec $Q(0) \neq 0$ tels que $S(X) = \dfrac{p(X)}{q(X)}$.*

2. *S est strictment périodique si et seulement si on a de plus* $\deg(p) < \deg(q)$.
3. *Si* $S(X) = \dfrac{p(X)}{q(X)}$ *avec* $\mathrm{pgcd}(p, q) = 1$ *alors la période de S est l'ordre multiplicatif de X dans* $\mathbb{F}_2[X]/(q)$.
4. *Si* $S(X) = \dfrac{p(X)}{q(X)} = \sum_{t=0}^{\infty} s_t X^t$ *est strictement périodique alors pour tout* $t \in \mathbb{N}$, *on a* $s_t = p(X)X^{-t} \bmod q \bmod X$,

où $x \bmod q \bmod X$ *signifie que x est un élément de* $\mathbb{F}_2[X]/(q)$, *vu comme un polynôme de degré au plus* $\deg(q) - 1$, $x \bmod q \bmod X$ *est alors son terme constant.*

Si on applique ce théorème aux séries générées par un LFSM $M_i(t_0)$, il existe des polynômes p_{i,t_0} et q_{i,t_0} tels que $M_i(t_0) = p_{i,t_0}/q_{i,t_0}$. Le théorème suivant montre cependant que les dénominateurs q_{i,t_0} sont en fait les mêmes pour tout i et tout t_0.

Théorème 4.2. *Soit un LFSM de matrice de transition* $T \in \mathcal{M}_n(\mathbb{F}_2)$. *Pour* $t_0 \in \mathbb{N}$, $m(t_0)$ *l'état du LFSM à l'instant* $t = t_0$ *et soit le vecteur de séries rationnelles associé* $M(t_0)$ *comme défini par la définition 4.2. Alors, on a :*

$$M(t_0)(X) = \frac{A(X) \cdot m(t_0)}{q(X)}.$$

avec $A = \mathrm{Adj}(I - XT) \in \mathcal{M}_n(\mathbb{F}_2[X])$ *et* $q(X) = \det(I - XT) \in \mathbb{F}_2[X]$, *où* Adj *désigne la matrice adjointe, c'est-à-dire la transposée de la matrice des cofacteurs.*

Démonstration. Par définition, la mise à jour de l'état du LFSM s'effectue grâce à l'équation (4.3) :

$$m(t + 1) = Tm(t).$$

En multipliant les deux membres de cette égalité par X^t et sommant pour $t \geq t_0$, on obtient :

$$M(t_0 + 1) = TM(t_0).$$

Or par définition de $M(t_0)$, on a $M(t_0) = m(t_0) + XM(t_0 + 1)$, d'où :

$$M(t_0) = m(t_0) + XTM(t_0).$$

Ce qui donne, en regroupant les termes en $M(t_0)$ à gauche de l'égalité :

$$(I - XT)M(t_0) = m(t_0).$$

La matrice $(I - XT)$ est inversible dans $\mathcal{M}_n(\mathbb{F}_2[[X]])$. En effet, son déterminant $q(X) = \det(I - XT)$ évalué en 0 est non-nul donc q est inversible dans $\mathbb{F}_2[[X]]$. En posant $A = \mathrm{Adj}(I - XT)$, on a $(I - XT)^{-1} = \dfrac{1}{q}A$ et on obtient le résultat cherché.

□

Ainsi les séries associées au LFSM écrites comme quotient de polynômes possèdent toutes le polynôme $q(X) = \det(I - XT)$ en dénominateur. Celui-ci ne dépend que de la matrice de transition du LFSM et pas de l'état dans lequel se trouve celui-ci. On donne un nom à ce polynôme particulier.

Définition 4.3. *Soit un LFSM de matrice de transition $T \in \mathcal{M}_n(\mathbb{F}_2)$, on appelle polynôme de connexion (ou polynôme de rétroaction) du LFSM, le polynôme $q(X) = \det(I - XT)$.*

Des propriétés de q découlent de nombreuses propriétés des LFSM. En particulier, grâce à la proposition 4.1, on connaît les périodes des suites générées. Si q est irréductible de degré n, cela garantit qu'il n'existe pas de LFSM avec moins de n mémoires, générant l'une de ces suites. De plus toutes ces suites auront la même période. En général, on cherche à produire des suites de période la plus grande possible. Il faut donc avoir q tel que l'ordre de X dans $\mathbb{F}_2[X]/(q)$ soit maximal. Un tel polynôme q est dit primitif. Dans ce cas, X engendre alors tout $\big(\mathbb{F}_2[X]/(q)\big)^*$ et la période des suites générées par ce LFSM sera donc $2^n - 1$. Ces suites sont appelées des m-séquences.

Dans le cas d'un LFSR en mode Galois ou Fibonacci, le polynôme de connexion s'exprime en fonction des coefficients q_i de la figure 4.3. Pour les deux modes, on a :

$$q(X) = 1 + \sum_{i=1}^{n} q_i X^i.$$

Si on suppose que $\deg(q) = n$, on peut associer un unique LFSR en mode Galois et un unique LFSR en mode Fibonacci à un polynôme q donné, grâce à cette formule. Il existe en revanche a priori pour un polynôme q donné, de nombreux LFSM dont le polynôme de connexion est q. Le théorème 4.2 garantit qu'ils auront tous les mêmes bonnes propriétés en terme de sécurité. Il ne dit en revanche rien sur les propriétés d'implémentation matérielles et logicielles de ces LFSM.

4.2 FCSR : Automates dyadiques

Dans la section précédente, on a vu les LFSR qui sont des automates à fonction de transition linéaire sur $\mathbb{F}_2$. On étudie ici les registres à décalage à rétroaction avec retenue (feedback with carry shift register en anglais ou FCSR). Introduits par Klapper et Goresky [KG93], ceux-ci sont l'analogue des LFSR en utilisant non plus l'addition modulo 2 mais l'addition dans $\mathbb{Z}$. De nombreux parallèles peuvent être dressés entre les LFSR et les FCSR.

4.2.1 Modes Galois & Fibonacci

Tout comme un LFSR en mode Fibonacci, un FCSR en mode Fibonacci de longueur n se compose de n mémoires $m_0(t), \cdots, m_{n-1}(t)$ contenant, à chaque instant $t \in \mathbb{N}$, chacune un bit. Contrairement au LFSR, celui-ci dispose en plus d'une mémoire supplémentaire entière $c(t) \in \mathbb{Z}$, appelée la retenue (carry en anglais). À chaque tic d'horloge, le FCSR est mis à jour comme indiqué par la figure 4.4, c'est-à-dire de la façon suivante :

- comme dans le cas des LFSR, le contenu des mémoires $m_0(t), \cdots, m_{n-1}(t)$ est décalé vers la droite, *i.e.* $m_i(t+1) = m_{i+1}(t)$ pour $i \in \{0, \cdots, n-2\}$;
- une somme intermédiaire $z \in \mathbb{Z}$ est calculée dans $\mathbb{Z}$ en fonction de l'état actuel :

$$z = c(t) + \sum_{i=1}^{n} q_i \cdot m_{n-i}(t). \tag{4.4}$$

où les $q_i \in \{0,1\}$, indépendants de t, indiquent quelles mémoires interviennent effectivement dans la somme.
- la nouvelle valeur de la mémoire la plus à gauche $m_{n-1}(t+1)$ est alors le bit de parité de z, *i.e.* $m_{n-1}(t+1) = z \bmod 2$;
- les autres bits de z constituent la nouvelle valeur de la retenue, *i.e.* $c(t+1) = z \text{ div } 2$.

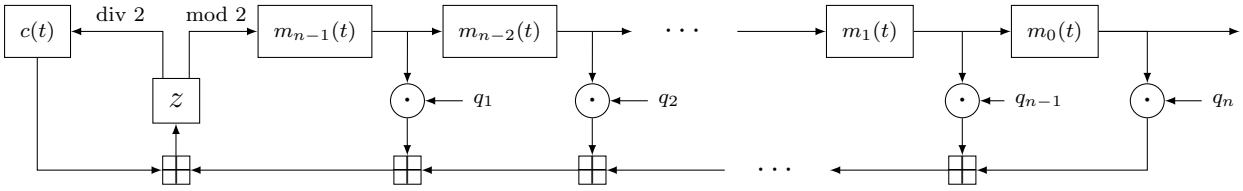


FIGURE 4.4 – FCSR en mode Fibonacci.

Pour $z \in \mathbb{Z}$, on note z div 2 le quotient de la division euclidienne de z par 2, de sorte que la décomposition modulo deux, $z = (z \bmod 2) + 2(z \text{ div } 2)$ avec $z \bmod 2 \in \{0,1\}$, soit unique.

Le mode Galois pour les FCSR se construit de manière analogue aux cas LFSR : une unique mémoire $m_0(t)$ sert de rétroaction à toutes les autres. Dans le cas FCSR, les additions sont faites avec retenue ; pour chaque mémoire $m_i(t)$ effectivement recevant une rétroaction on doit donc disposer d'une mémoire supplémentaire, notée $c_i(t)$, pour stocker la valeur de la retenue. Un FCSR en mode Galois se compose donc de n mémoires binaires $m_0(t), \cdots, m_{n-1}(t) \in \mathbb{F}_2$ et de $\ell \leq n$ mémoires entières $c_0(t), \cdots, c_{n-1}(t) \in \mathbb{Z}$, avec la convention que $c_i(t)$ est nulle pour tout $t \in \mathbb{N}$ lorsque $m_i(t)$ ne reçoit pas de rétroaction. La valeur de ℓ est le nombre de rétroactions, c'est le nombre de mémoires dont on a physiquement besoin pour stocker les retenues. Le FCSR est alors mis à jour de la manière suivante (cf. figure 4.5) :

$$\begin{cases} z_i = m_{i+1}(t) + c_i(t) + q_{i+1} \cdot m_0(t) \\ m_i(t+1) = z_i \bmod 2 \\ c_i(t+1) = z_i \text{ div } 2 \end{cases}$$

pour tout $i \in \{0, \cdots, n-1\}$ et avec la convention que $m_n(t) = 0$, *i.e.* $m_{n-1}(t+1) = c_{n-1}(t) + q_n \cdot m_0(t)$.

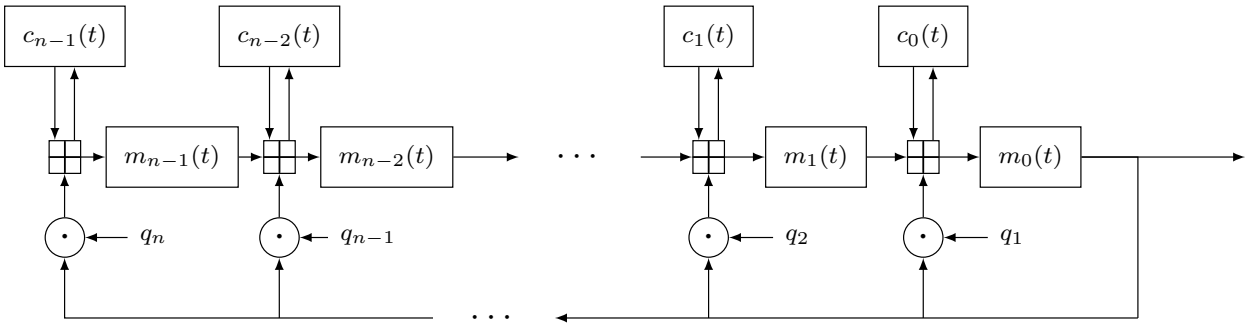


FIGURE 4.5 – FCSR en mode Galois.

4.2.2 Représentation matricielle

Comme pour les LFSR, on peut utiliser une représentation matricielle des FCSR en mode Galois ou Fibonacci : les opérations sont faites dans $\mathbb{Z}$ puis les résultats sont séparés entre les mémoires principales $m_i(t)$ et les retenues $c_i(t)$ via les opérations $\bmod 2$ et $\operatorname{div} 2$. Cette représentation est due à Arnault *et al.* [ABMP11].

Définition 4.4. *Un FCSR de taille n se compose d'un vecteur colonne binaire de taille n, noté $m(t) = (m_0(t), \cdots, m_{n-1}(t))^T \in \{0,1\}^n \subset \mathbb{Z}^n$, appelé registre principal du FCSR, d'un vecteur entier de taille n, noté $c(t) = (c_0(t), \cdots, c_{n-1}(t)) \in \mathbb{Z}^n$, appelé registre des retenues du FCSR et d'une matrice $T \in \mathcal{M}_n(\mathbb{Z})$, appelée matrice de transition du FCSR. Le comportement du FCSR est le suivant : à l'instant $t = 0$, l'état est intialisé à une valeur $(m(0), c(0)) \in \{0,1\}^n \times \mathbb{Z}^n$; à chaque instant $t+1$ un nouvel état est calculé à partir du précédent de la manière suivante :*

$$\begin{cases} z = c(t) + T \cdot m(t) \in \mathbb{Z}^n \\ m(t+1) = z \bmod 2 \\ c(t+1) = z \operatorname{div} 2 \end{cases}$$

où les opérations $\bmod 2$ et $\operatorname{div} 2$ sont étendues aux vecteurs en étant appliquées en parallèle sur chaque coefficient.

Comme pour les LFSM, on s'intéresse aux suite générées par un FCSR. Dans le cas LFSM, on avait utilisé les séries formelles $\mathbb{F}_2[[X]]$ et identifé les suites produites par un LFSM aux séries rationnelles, c'est-à-dire quotient de deux polynômes. Pour prendre en compte le fait que les additions sont faites avec retenues, on remplace les séries formelles par les entiers dyadiques (ou 2-adiques) $\mathbb{Z}_2$ et les polynômes par les entiers relatifs $\mathbb{Z}$.

Définition 4.5. *Soit $t_0 \in \mathbb{N}$ et $\big(s(t)\big)_{t\in\mathbb{N}}$ une suite binaire. On appelle série dyadique associée à la suite, l'entier dyadique $S \in \mathbb{Z}_2$ défini par :*

$$S = \sum_{t=0}^{\infty} s(t) 2^t.$$

Dans le cas des FCSR, on note $M_i(t_0)$ la série dyadique associée à la suite $\big(m_i(t)\big)_{t\geq t_0}$ observée en position i de l'état du FCSR à partir de l'instant t_0, c'est-à-dire :

$$M_i(t_0) = \sum_{t=t_0}^{\infty} m_i(t) 2^{t-t_0}.$$

On note de plus $M(t_0) \in \mathbb{Z}_2^n$ le vecteur regroupant ces séries dyadiques :

$$M(t_0) = (M_0(t_0), \cdots, M_{n-1}(t_0))^T = \sum_{t=t_0}^{\infty} m(t) 2^{t-t_0}.$$

Comme dans le cas des séries formelles (cf proposition 4.1), on caractérise les séries ultimement périodiques.

Proposition 4.3. *Soit S une série dyadique, alors*

1. *S est ultimement périodique si et seulement s'il existe $p \in \mathbb{Z}$ et $q \in \mathbb{Z}$ impair tels que* $S = \frac{p}{q}$;
2. *S est strictement périodique si et seulement s'il existe de tels p et q qui vérifient aussi* $pq < 0$ *et* $0 \leq |p| \leq |q|$;
3. *si* $S = \frac{p}{q}$ *avec* $\mathrm{pgcd}(p, q) = 1$, *la période de S est* $\mathrm{Ord}_q(2)$, *l'ordre multiplicatif de* 2 *dans* $(\mathbb{Z}/q\mathbb{Z})^*$;
4. *si* $S = \frac{p}{q} = \sum_{t=0}^{\infty} s_t 2^t$ *est strictement périodique alors pour tout* $t \in \mathbb{N}$, *on a* $s_t = p2^{-t} \mod q \mod 2$

où $x \mod q \mod 2$ *signifie que que x est d'abord réduit modulo q pour donner un nombre entre* 0 *et* $|q| - 1$ *puis que ce résultat est réduit modulo 2 pour donner soit* 0 *soit* 1.

On dispose aussi de l'analogue du théorème 4.2 pour les LFSM dans le cas des FCSR.

Théorème 4.4. *Soit un FCSR de matrice de transition* $T \in \mathcal{M}_n(\mathbb{Z})$. *Pour* $t_0 \in \mathbb{N}$, $(m(t_0), c(t_0))$ *l'état du FCSR à l'instant* $t = t_0$ *et soit le vecteur de séries dyadiques associé* $M(t_0)$ *comme défini par la définition 4.5. Alors, on a :*

$$M(t_0) = \frac{A \cdot \big(m(t_0) + 2c(t_0)\big)}{q}.$$

avec $A = \mathrm{Adj}(I - 2T) \in \mathcal{M}_n(\mathbb{Z})$ *et* $q = \det(I - 2T) \in \mathbb{Z}$, *où* Adj *désigne la matrice adjointe, c'est-à-dire la transposée de la matrice des cofacteurs.*

Démonstration. Par la définition 4.4 :

$$m(t + 1) + 2c(t + 1) = Tm(t) + c(t).$$

En multipliant les deux membres de cette égalité par 2^t, et en sommant pour $t \geq t_0$, on obtient :

$$M(t_0 + 1) + 2\sum_{t=t_0}^{\infty} c(t + 1)2^t = TM(t_0) + \sum_{t=t_0}^{\infty} c(t)2^t.$$

d'où en simplifiant :

$$M(t_0 + 1) = TM(t_0) + c(t_0).$$

Or par définition de $M(t_0)$, on a $M(t_0) = m(t_0) + 2M(t_0 + 1)$, d'où :

$$M(t_0) = m(t_0) + 2TM(t_0) + 2c(t_0).$$

Ce qui donne, en regroupant les termes en $M(t_0)$ à gauche de l'égalité :

$$(I - 2T)M(t_0) = m(t_0) + 2c(t_0).$$

On pose alors $q = \det(I - 2T)$ et $A = \mathrm{Adj}(I - 2T)$. q est inversible dans $\mathbb{Z}_2$, en effet $q \mod 2 = \det(I - 2T) \mod 2 = \det(I) = 1$. D'où le résultat, en multipliant la dernière équation par $(I - 2T)^{-1} \in \mathcal{M}_n(\mathbb{Z}_2)$. □

Comme dans le cas des LFSR, les séries observées dans les cellules du registre principal s'expriment comme un quotient. Dans le cas des FCSR, il s'agit du quotient dyadique d'entiers. Le dénominateur commum à ces fractions dyadiques $q = \det(I - 2T)$ ne dépend que de la matrice de transition du FCSR.

Définition 4.6. *Soit un FCSR de matrice de transition $T \in \mathcal{M}_n(\mathbb{Z})$, on appelle entier de connexion du FCSR, l'entier relatif $q = \det(I - 2T)$.*

L'entier de connexion permet de déterminer certaines propriétés des FCSR comme leur période ou encore la répartition statistique des sorties. Cependant la présence du registre à retenue vient compliquer les choses. En effet, dans le cas d'un LFSR dont le polynôme de connexion est primitif, l'ensemble des $2^n - 1$ états non-nuls possibles se trouvent tous sur un cycle de longueur $2^n - 1$. Pour un FCSR avec entier de connexion primitif q avec $\log_2 q \geq n$, l'ordre d'un cycle est $q - 1 \geq 2^n - 1$. Cependant, contrairement au LFSR, il y a $2^{(n+c)} - 2$ états possibles, si l'on exclut les états tout-à-zéro et tout-à-un qui sont des points fixes et où c est le nombre de cellules (binaires) du registre de retenues contenant effectivement une retenue. Par conséquent, l'entropie du FCSR lorsqu'il se situe sur un cycle est plus faible que l'entropie initiale. En particulier, deux états initiaux différents peuvent, après le même nombre d'itérations, arriver sur le même état.

Lorsqu'on regarde les séries dyadiques produites par les cellules du registre principal d'un FCSR, on sait qu'elles vont toutes avoir les mêmes propriétés statistiques puisqu'elles ont toutes le même entier de connection. Le lien entre ces séries est cependant beaucoup plus fort, comme le montre le théorème suivant.

Théorème 4.5. *Soit un FCSR avec un entier de connexion q* premier *et décrit par les séries dyadiques $M_i(t_0) = p_i(t_0)/q$ avec $0 < |p_i(t_0)| < q$ pour tout $i \in \{0, \cdots, n-1\}$. Si 2 est une racine primitive modulo q, i.e. $\mathrm{Ord}_q(2) = q - 1$ alors pour tout $i \in \{0, \cdots, n-1\}$, il existe un entier τ_i tel que $M_i(t_0) = M_0(t_0 + \tau_i)$. De plus, τ_i ne dépend pas du temps initial t_0 considéré.*

Pour démontrer ce théorème, on fait appel aux deux lemmes suivants :

Lemme 4.6. *Avec les notations du théorème ci-dessus et en notant $p(t)$ le vecteur $(p_0(t), \cdots, p_{n-1}(t))$, alors $\forall t \geq 0$, $p(t) \equiv 2p(t+1) \bmod q$.*

Démonstration. Posons $p^*(t) = m(t) + 2c(t)$. Par la définition 4.4 : $p^*(t+1) = Tm(t) + c(t)$. D'où

$$2p^*(t+1) = 2Tm(t) + 2c(t) = (2T - I)m(t) + p^*(t).$$

Par le théorème 4.4, on sait que $p(t) = \mathrm{Adj}(I - 2T)p^*(t)$, d'où en multipliant l'équation ci-dessus par $\mathrm{Adj}(I - 2T)$:

$$2p(t+1) = -qm(t) + p(t).$$

D'où le résultat en réduisant modulo q. □

Lemme 4.7. *Soit $B \in \mathcal{M}_n(\mathbb{Z})$ et q un nombre premier. Si le noyau de $B \bmod q$ est de dimension k alors $q^k \mid \det(B)$.*

Démonstration. On échelonne la matrice B en mettant des multiples de q là où on met des zéros habituellement. Le déterminant est inchangé. On obtient k lignes nulles modulo q. Le déterminant obtenu est donc divisible par q^k. □

On passe maintenant à la preuve du théorème proprement dit.

Démonstration. Par hypothèse $0 < |p_i(t_0)| < q$, ce qui signifie que $p(t_0) = p(t_0) \bmod q$. Ainsi $p(t_0)$ se trouve dans l'image de $A \bmod q$. Or $A = \mathrm{Adj}(I - 2T)$ et $q = \det(I - 2T)$ est premier. On montre dans un premier temps que $A \bmod q$ est de rang 1, en utilisant le lemme 4.7.

En effet, on sait que $\det(I - 2T)\det(A) = q^n$ donc $\det(A) = q^{n-1}$. Par le lemme 4.7, on déduit que la dimension du noyau de $A \bmod q$ est au plus $n-1$. Or $(I-2T) \times A = 0 \bmod q$ par conséquent la somme des dimensions des noyaux de $I - 2T \bmod q$ et de $A \bmod q$ est au moins n. À nouveau à l'aide du lemme 4.7, on sait que la dimension du noyau de $I-2T \bmod q$ est égale à 1. Le noyau de $A \bmod q$ est donc de dimension $n-1$ et la matrice $A \bmod q$ de rang 1.

Ainsi, on obtient la i-ème ligne de $A \bmod q$ en multipliant la première ligne par un certain entier que l'on note 2^{τ_i} puisque 2 est une racine primitive modulo q. Les valeurs τ_i sont uniquement déterminées par la donnée de la matrice de transition du FCSR. On a alors que $\forall t_0 \in \mathbb{N}, p_i(t_0) = 2^{-\tau_i} p_0(t_0)$.

En utilisant le lemme 4.6, on obtient que $p_i(t_0 + \tau_i) = p_0(t_0)$. En divisant par q, on obtient le résultat voulu.

□

Ce théorème signifie que, quitte à ne pas tenir compte des premiers bits correspondant à la prépériode, les suites produites par les différentes cellules du registre principal d'un FCSR sont les mêmes à un décalage temporel près.

4.3 Implémentations

Grâce à la représentation matricielle des FCSR, il est possible de construire des FCSR avec des bonnes propriétés au-delà des modes Fibonacci et Galois. On présente ici deux familles de FCSR issues de cette vision. L'une est orientée matériel tandis que l'autre est orientée logiciel. Précisons immédiatement que ces deux familles ne sont pas disjointes et que l'on peut ainsi créer des FCSR efficaces sur les deux plateformes. Notons aussi que tout ce qui est dit ici est aussi valable pour les LFSR.

4.3.1 FCSR annulaires

Introduite par Arnault *et al.* [ABL+09] en 2009, la famille des FCSR annulaires (Ring-FCSR en anglais) généralise les modes Galois et Fibonacci de la façon suivante. Dans le mode Fibonacci, une seule cellule reçoit des rétroactions de plusieurs autres ceullules. Dans le mode Galois au contraire, ce sont plusieurs cellules qui reçoivent une unique rétroaction. Les FCSR annulaires font fi de ces restrictions et permettent à n'importe quelle cellule d'influencer n'importe quelle autre. La seule chose qu'ils conservent des constructions précédentes est la structure de registre à décalage, d'où le nom de FCSR annulaire et la définition suivante.

Définition 4.7. *Un FCSR annulaire est un FCSR selon la définition 4.4 dont la matrice de transition $T \in \mathcal{M}_n(\mathbb{Z})$ possède une surdiagonale unitaire. En d'autres termes, on a :*

$$T = \begin{pmatrix} & 1 & & (*) \\ & & \ddots & \\ & (*) & & 1 \\ 1 & & & \end{pmatrix}.$$

Les FCSR en mode Fibonacci et Galois sont des cas particuliers de FCSR annulaires.

On donne à la figure 4.6 trois exemples de tels automates. Ces trois FCSR possèdent le même entier de connection $q = -347$. Le premier est en mode Fibonacci, le deuxième en mode Galois et le troisième est un FCSR annulaire mais ni en mode Fibonacci ni en mode Galois. Leurs matrices respectives sont données à la figure 4.7.

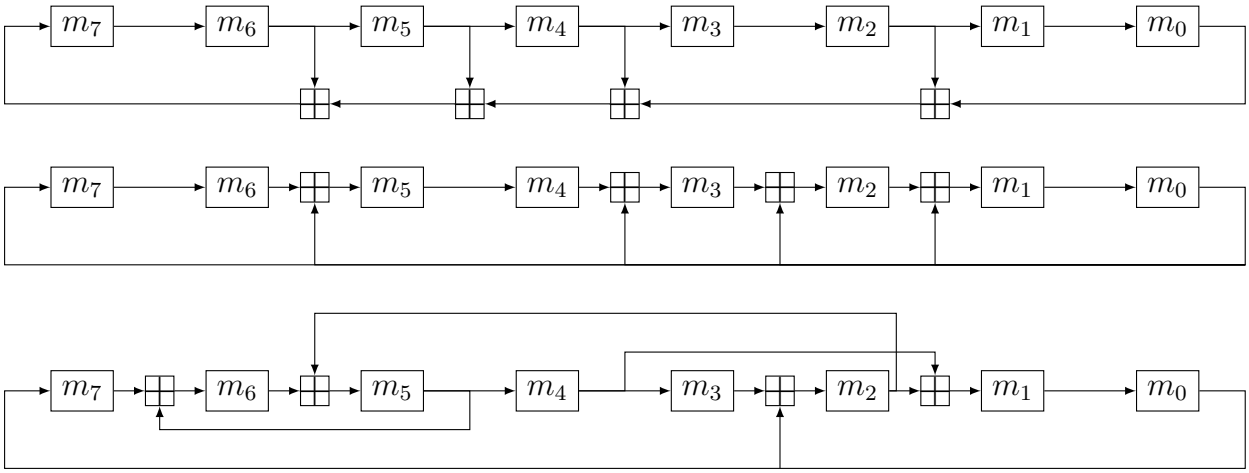


FIGURE 4.6 – Trois FCSR annulaires de même entier de connection $q = -347$. De haut en bas, il s'agit d'un FCSR en mode Fibonacci, puis d'un FCSR en mode Galois et enfin d'un FCSR annulaire.

$$\begin{pmatrix} & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & (0) & \\ & & & & 1 & & & \\ & & (0) & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ 1 & & 1 & & 1 & 1 & 1 & \end{pmatrix} \quad \begin{pmatrix} & 1 & & & & & & \\ 1 & & 1 & & & & (0) & \\ 1 & & & 1 & & & & \\ 1 & & & & 1 & & & \\ & & & & & 1 & & \\ 1 & & & (0) & & & 1 & \\ & & & & & & & 1 \\ 1 & & & & & & & \end{pmatrix} \quad \begin{pmatrix} & 1 & & & & & & \\ & & 1 & & 1 & & & \\ 1 & & & 1 & & & (0) & \\ & & & & 1 & & & \\ & & & (0) & & 1 & & \\ & & 1 & & & & 1 & \\ & & & & & 1 & & 1 \\ 1 & & & & & & & \end{pmatrix}$$

FIGURE 4.7 – Matrices de transition des trois FCSR donnés à la figure 4.6. De gauche à droite, le FCSR en mode Fibonacci, le FCSR en mode Galois et enfin le FCSR annulaire.

À condition de rajouter quelques contraintes supplémentaires sur la matrice de transition, il est possible de construire des FCSR annulaires orientés matériel. Observons les cas Fibonacci et Galois.

Pour le mode Fibonacci, chaque rétroaction doit traverser plusieurs additionneurs pour créer la nouvelle valeur du dernier bit. Comme déjà discuté à la section 2.1.1 sur les contraintes en matériel, chaque traversée de porte demande un certain temps. Donc plus un bit doit traverser de portes, plus la fréquence d'horloge doit être basse afin de laisser le temps au signal de traverser toutes les portes sur son chemin en un seul cycle d'horloge. Le chemin critique, qui est dans notre cas le nombre maximal d'additionneurs que doit traverser un signal par cycle d'horloge, vaut $\lceil \log_2((|q|-1)/2) \rceil$ si on implémente les additionneurs en parallèle.

Dans le mode Galois au contraire, l'unique bit de rétroaction ne passe que dans un seul additionneur, le chemin critique est alors ici optimal. Cependant, ce bit de rétroaction est utilisé à plusieurs endroits. Par conséquent son arité sortante (cf. section 2.1.1) est élevée, précisément elle vaut $(|q|+1)/2$. Au contraire, le mode Fibonacci est optimal de ce point de vue puisqu'un bit ne va que dans au plus deux endroits.

Le dernier critère d'efficacité pour les FCSR auquel on s'intéresse est la diffusion. Il s'agit de savoir combien de fois on doit itérer le FCSR avant que tous les bits en entrée aient une influence sur l'ensemble du FCSR. En d'autres termes, il s'agit du diamètre du graphe défini par le FCSR, dont la matrice d'adjacence n'est ni plus ni moins que la matrice de transition du FCSR. Pour un FCSR sur n bits en mode Galois ou Fibonacci, sa diffusion vaut $n-1$, c'est-à-dire la pire valeur possible.

La famille des FCSR annulaires, plus large que les modes Galois et Fibonacci, va alors permettre de construire des FCSR avec à la fois les avantages des deux modes mais sans leurs inconvénients et possiblement à un coût moindre ; le coût étant compté ici en nombre d'additionneurs binaires. Par exemple, le tableau 4.1 compare les trois FCSR, d'entier de connexion $q = -347$, donnés à la figure 4.6 sur ces quatre critères. On voit que le FCSR annulaire est optimal à la fois pour le chemin critique et l'arité sortante et possède une meilleure diffusion tout en ayant un coût similaire.

	Fibonacci	Galois	Annulaire
Chemin critique	3	1	1
Arité sortante	2	5	2
Coût	4	4	4
Diffusion	7	7	5

Tableau 4.1 – Chemin critique, arité sortante, coût et diffusion des FCSR donnés à la figure 4.6.

Plus généralement, si on se donne un FCSR de matrice de transition T, il est facile de voir que l'arité sortante est le maximum des poids de Hamming des colonnes de T. De la même manière, le chemin critique est le maximum des logarithmes en base 2 des poids de Hamming des lignes de T. Si on note L_i la i-ème ligne de T et C_j la j-ème colonne de T et w le poids de Hamming de $(|q|-1)/2$ alors on a le tableau 4.2.

Ainsi des critères pertinents pour générer de bons FCSR annulaires sont les suivants :

- N'avoir qu'au plus deux coefficients non-nuls par ligne et par colonne. Ceci garantit une arité sortante et un chemin critique optimaux.
- L'entier de connexion $q = \det(I - 2T)$ est premier et 2 est primitif modulo q.
- Le coût du FCSR sera voisin de $n/2$. En effet, il s'agit du nombre d'additonneurs

	Fibonacci	Galois	Annulaire
Chemin critique	$\lceil \log_2(w-1) \rceil$	1	$\max \lceil \log_2(w_H(L_i)) \rceil$
Arité sortante	2	$w-1$	$\max w_H(C_i)$
Coût	$w-2$	$w-2$	$w_H(T)-n$
Diffusion	$n-1$	$n-1$	$\leq n-1$

Tableau 4.2 – Comparaison entre le Chemin critique, l'arité sortante, le coût et la diffusion des FCSR selon différents modes.

utilisés, il s'agit donc aussi du nombre d'endroits où est introduit de la non-xor-linéarité. Afin de se prémunir de certaines attaques qui seront présentées dans la section 4.4, on préfère avoir le plus de points de ce genre.

– La diffusion est la plus rapide possible.

Les FCSR annulaires sont utilisés dans le chiffrement à flot orienté matériel F-FCSR-H v3 [ABL+09]. Les F-FCSR (F signifie filtré) sont des générateurs pseudo-aléatoires de type automate filtré. L'automate interne est ici un FCSR annulaire tandis que la fonction de filtrage f est une application xor-linéaire appliquée sur le registre principal du FCSR. Ce filtre xor-linéaire permet de casser la structure dyadique du FCSR tout en ayant un coût faible. Cette construction peut être vue comme l'analogue des LFSR filtrés. Dans le cas des LFSR filtrés, un filtre non-linéaire est appliqué à un LFSR (qui est linéaire). Dans le cas d'un FCSR filtré, on applique un filtre linéaire à un FCSR qui a une fonction de transition non-linéaire. La figure 4.8 donne un exemple de F-FCSR. Grâce à l'utilisation d'un FCSR annulaire et d'un filtre ne nécessitant que quelques portes XOR, les F-FCSR sont par construction efficaces sur plateformes matérielles.

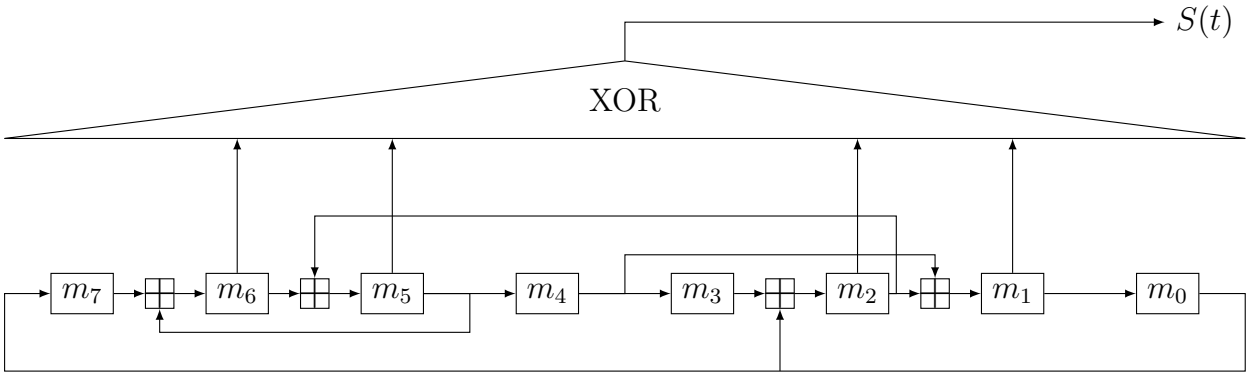


FIGURE 4.8 – Exemple de F-FCSR

4.3.2 FCSR sur les mots

Introduits par Berger *et al.* [BMP09], les FCSR sur les mots (*Word-FCSR* en anglais) sont une autre généralisation des FCSR mais ayant pour but une implémentation logicielle efficace. Comme discuté à la section 2.1.2, une des caractéristiques de l'environnement logiciel est de regrouper les bits par mot. Ainsi les FCSR sur les mots vont être définis à l'aide d'une matrice par bloc. La longueur d'un bloc correspond à la longueur d'un mot machine et chaque bloc de la matrice est une opération orientée logiciel, comme le décalage à perte

ou la rotation. De plus, la structure de registre à décalage est elle aussi transposée sur les mots : à chaque itération, le registre principal du FCSR est décalé d'un mot tout entier et non d'un bit. Par exemple, un FCSR sur les mots agissant sur des mots de w bits en mode Fibonacci aura une matrice de transition de la forme

$$T = \begin{pmatrix} 0 & I_w & & (0) \\ \vdots & & \ddots & \\ 0 & (0) & & I_w \\ A_r & A_{r-1} & \cdots & A_1 \end{pmatrix}$$

où I_w désigne la matrice identité de taille $w \times w$ et les $A_1, \cdots, A_r$ sont r matrices de taille $w \times w$ efficaces en logiciel. Ce FCSR peut être aussi bien vu comme étant composé de r mots de w bits chacun ou comme un FCSR binaire de longueur $n = w \times r$.

Similairement à ce qui a été fait pour le cas matériel, on peut autoriser les rétroactions en tout point du FCSR et s'affranchir ainsi des cas Galois et Fibonacci. On parle dans ce cas de *FCSR annulaire sur les mots* (*Word-Ring-FCSR*). La matrice de transition, définie sur les mots de w bits, est alors de la forme

$$T = \begin{pmatrix} & I_w & & (*) \\ & & \ddots & \\ & (*) & & I_w \\ I_w & & & \end{pmatrix}.$$

À condition d'utiliser des applications orientées logiciel dans les rétroactions comme les décalages ou les rotations, un tel FCSR possède une implémentation logicielle efficace. Mieux, en gardant les mêmes contraintes qu'en matériel (une seule rétroaction sur les mots par ligne et par colonne), il est possible d'obtenir un FCSR qui remplit aussi les contraintes matérielles et est donc efficace en logiciel et en matériel.

À titre d'exemple, un FCSR annulaire de taille $n = 40$ sur des mots de $w = 8$ bits est donné à la figure 4.9. Il utilise les opérations de décalage à perte à gauche et à droite notées respectivement L et R et définies par

$$\begin{aligned} L \cdot (x_0, x_1, \cdots, x_{w-2}, x_{w-1})^T &= (x_1, \cdots, x_{w-2}, x_{w-1}, 0)^T \\ R \cdot (x_0, x_1, \cdots, x_{w-2}, x_{w-1})^T &= (0, x_0, x_1, \cdots, x_{w-2})^T \end{aligned}$$

Son entier de connection est primitif et vaut -1497813390989, les séquences produites sont donc des ℓ-séquences. Il vérifie aussi les contraintes matérielles ; son arité sortante est de 2 et son chemin critique de 1. Il coûte 23 additionneurs et son diamètre est de 15.

4.4 Attaques sur les FCSR

4.4.1 LFSRisation des FCSR en mode Galois

À Asiacrypt 2008, Hell et Johannsson [HJ08] ont présenté une attaque sur le chiffrement à flot F-FCSR-H v2 [AB05a, AB05b] qui utilise un FCSR en mode Galois. Cette attaque se base sur une faiblesse de ce mode : une seule rétroaction contrôle l'ensemble des retenues.

$$\begin{pmatrix} . & I_8 & . & L^3 & . \\ . & . & I_8 & . & R^2 \\ . & . & L^1 & I_8 & . \\ R^3 & . & . & . & I_8 \\ I_8 & . & . & . & . \end{pmatrix}$$

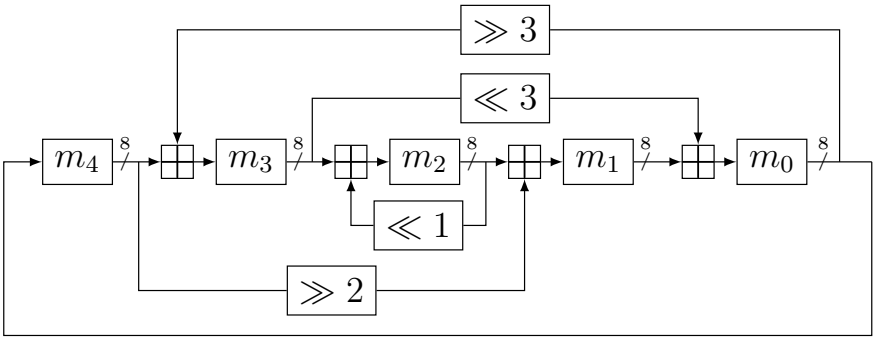


FIGURE 4.9 – Exemple de FCSR efficace à la fois en logiciel et en matériel.

Si celle-ci est nulle pendant suffisamment de tours consécutifs alors la fonction de transition devient affine et il est donc facile de retrouver l'état du FCSR à partir de la suite chiffrante générée. L'idée générale de cette attaque, appelée LFSRisation des FCSR, est présentée ici.

Rappelons en effet qu'un FCSR en mode Galois possède une unique rétroaction, à savoir la cellule m_0 du registre principal. Supposons $m_0(t) = 0$ à un instant t donné et observons le comportement de la retenue c_i. On sait que la nouvelle valeur de $c_i(t+1)$ est donnée par la relation

$$c_i(t+1) = m_0(t)c_i(t) \oplus m_0(t)m_{i+1}(t) \oplus c_i(t)m_{i+1}(t)$$

qui se simplifie ici en $c_i(t+1) = c_i(t)m_{i+1}(t)$. Ainsi on voit que :

- si une retenue est nulle alors elle reste nulle à l'instant d'après ;
- sinon, elle devient nulle avec probabilité 1/2 (en supposant $m_i(t+1)$ aléatoire).

D'où si la rétroaction est nulle pendant plusieurs itérations consécutives, toutes les retenues vont s'annuler avec une forte probabilité. Plus précisément, on voit que le nombre de retenues non-nulles va être environ divisé par deux à chaque itération. Comme F-FCSR-H v2 contient 82 retenues, il faut environ $\log_2 82 \approx 7$ rétroactions nulles consécutives pour obtenir un registre de retenues tout à zéro.

L'étape suivante consiste à maintenir cet état encore suffisamment de fois de manière à récupérer suffisamment de bits de la suite chiffrante pour inverser le système. F-FCSR-H v2 utilise un FCSR de 160 bits et extrait un octet à chaque itération via un filtre xor-linéaire donc si l'on maintient la rétroaction nulle pendant encore 19 tours (en plus des 7 nécessaires pour mettre toutes les retenues à zéro) alors on obtient 160 bits de suite chiffrante s'exprimant de manière linéaire en fonction du registre principal du FCSR. Il n'y a alors plus qu'à inverser ce système pour retrouver l'état interne du FCSR.

Cependant telle quelle, cette attaque ne fonctionne pas. En effet, Arnault, Berger et Minier [ABM08] montrent que sur le cycle principal la rétroaction et le vecteur de retenues ne peuvent être simultanément nuls.

Pour s'en convaincre, observons ce qui se passe dans la dernière (*i.e.* la plus à droite) retenue active c_1 du FCSR de F-FCSR-H v2. Supposons que la rétroaction soit non-nulle à l'instant $t-1$ puis nulle à partir de l'instant t pour un certain nombre de tours (dans le cas de F-FCSR-H v2 au moins deux tours). Or en remontant le début du registre jusqu'à la première retenue, on voit que $0 = m_0(t+1) = m_1(t)$. Or on a aussi que $m_1(t) = m_0(t-1) \oplus c_1(t-1) \oplus m_2(t-1)$ et par conséquent $c_1(t-1) \oplus m_2(t-1) = 1$ ou en d'autres termes, puisque les cellules ne contiennent qu'un seul bit, $c_1(t-1) \neq m_2(t-1)$. En utilisant

ces relations dans l'équation de la nouvelle retenue, on a

$$c_1(t) = m_0(t-1)c_1(t-1) \oplus m_0(t-1)m_2(t-1) \oplus c_1(t-1)m_2(t) = 1.$$

On voit ainsi que forcer la rétroaction à zéro entraîne que le bit de retenue c_1 se retrouve non-nul contrairement à ce qui été espéré. Hell et Johansson modifient alors leur attaque et s'intéressent désormais à l'évémement

$$c(t) = c(t+1) = \cdots = c(t+19) = (0, \cdots, 0, 1, 0).$$

Comme on vient de le voir, cela se produit avec une forte probabilité dès lors qu'on a 2 rétractions nulles consécutives. Le système à résoudre pour retrouver l'état interne du FCSR à partir des 160 bits de suite chiffrante générés est dans ce cas affine et non plus linéaire ; ce qui ne change pas fondamentalement les choses.

En somme, pour retrouver l'état interne du FCSR, l'attaque nécessite $\log_2 82 \approx 7$ rétroactions nulles consécutives pour mettre le registre des retenues dans l'état $(0, \cdots, 0, 1, 0)$ puis 19 rétroactions nulles consécutives supplémentaires pour le garder ainsi suffisamment longtemps afin d'extraire assez de suite chiffrante et réaliser l'attaque. En supposant une distribution uniforme du bit de rétroaction, cela se produit avec une probabilité de 2^{-26}. Il est ainsi possible à partir de 2^{26} octets de suite chiffrante de retrouver entièrement l'état interne du FCSR utilisé dans F-FCSR-H v2. En fait, les auteurs de l'attaque [HJ08] proposent une amélioration de cette attaque et font ainsi descendre la complexité en donnée à $2^{24.7}$ octets de suite chiffrante. De plus, à cause de la structure particulière choisie pour le filtre linéaire du F-FCSR-H v2, il est possible de subdiviser le système affine 160×160 en 8 systèmes de taille 20×20 indépendants et de résoudre le système en pré-calculant des tables de solutions pour ces sous-systèmes ; ce qui diminue la complexité en temps de l'attaque au prix d'une phase de précalculs plus importante.

Cette attaque a motivé les auteurs de F-FCSR-H v2 à proposer une nouvelle instance du chiffrement F-FCSR-H, nommé F-FCSR-H v3 [ABL+09] en utilisant cette fois-ci un FCSR annulaire (cf. section 4.3.1). Les retenues ne sont alors plus contrôlées par une seule rétroaction et la complexité de l'attaque précédente devient beaucoup plus grande que la recherche exhaustive.

4.4.2 Biais linéaire sur les séquences produites par un FSCR

En 2009, Tian et Qi [TQ09] ont montré qu'il existait des relations linéaires avec un avantage significatif à l'intérieur d'une ℓ-séquence qui donnent lieu à un distingueur linéaire sur les chiffrements de type F-FCSR. Cependant l'efficacité de ce distingueur dépend fortement de la capacité à trouver ces bonnes relations linéaires. Wang, Stankovski et Johansson [WSJ15] montrent alors en 2014 qu'il est possible de diminuer la recherche des ces relations en utilisant un paradoxe des anniversaires. Ils arrivent alors à trouver le premier distingueur sur F-FCSR-H v3 [ABL+09] dont la complexité est en dessous de la recherche exhaustive. Ce sont ces relations linéaires ainsi que la méthode pour les trouver que l'on présente dans cette section.

Soit une ℓ-séquence $\big(s(t)\big)_{t\in\mathbb{N}}$ d'entier de connexion q. Par la proposition 4.3, on sait qu'il existe un entier p tel que

$$s(t) = p \cdot 2^{-t} \bmod q \bmod 2.$$

On va s'intéresser à des relations linéaires faisant intervenir des termes de la forme 2^{-t} pour différentes valeurs de t.

Définition 4.8 ([WSJ15]). *Soit un entier de connexion q et des entiers $u_1, \cdots, u_i$ et $v_1, \cdots, v_j$, la relations linéaire*

$$2^{-u_1} + \cdots + 2^{-u_i} = 2^{-v_1} + \cdots + 2^{-v_j} \bmod q \tag{4.5}$$

est appelée un relation de type $(i+j)$ et est notée $\mathcal{R}_q(u_1, \cdots, u_i; v_1, \cdots, v_j)$.

Les relations de type $(m+1)$ avec $m \geq 3$ impair sont utilisées par Tian et Qi [TQ09] pour déduire des relations linéaires binaires sur une ℓ-séquence. À la fois par simplicité et parce que ce sont les relations qui permettent d'obtenir le biais le plus élevé, on se focalise sur le cas $m = 3$ mais tout ce qui est dit ici se généralise immédiatement à m quelconque. Une relation $\mathcal{R}_q(w, x, y; z)$ de type $(3+1)$ est donc de la forme :

$$2^{-w} + 2^{-x} + 2^{-y} = 2^{-z} \bmod q.$$

En multipliant les deux membres de l'égalité par $p \cdot 2^{-t}$, on obtient

$$p \cdot 2^{-(t+w)} + p \cdot 2^{-(t+x)} + p \cdot 2^{-(t+y)} = p \cdot 2^{-(t+z)} \bmod q.$$

Posons $p(t) = p \cdot 2^{-t}$, on a alors $s(t) = p(t) \bmod 2$ et

$$p(t+w) + p(t+x) + p(t+y) = p(t+z) \bmod q. \tag{4.6}$$

Tian et Qi [TQ09] s'intéressent alors à la probabilité d'observer la même relation mais sur les bits de parité uniquement, c'est-à-dire d'avoir pour tout $t \geq 0$,

$$s(t+w) \oplus s(t+x) \oplus s(t+y) = s(t+z). \tag{4.7}$$

En supposant que les termes $p(t+w)$, $p(t+x)$ et $p(t+y)$ sont des variables aléatoires mutuellement indépendantes, uniformément réparties sur $\mathbb{Z}/q\mathbb{Z} \setminus \{0\} = \{1, \cdots, q-1\}$, cette probabilité est donnée dans le théorème suivant. On s'intéressera cependant surtout au corollaire qui donne une approximation de cette probabilité pour les petites valeurs de m, car ce sont celles qui ont le plus grand biais.

Théorème 4.8 ([TQ09]). *Soit q un entier impair et $m \geq 2$. Soit la probabilité $P_m(q)$ définie par*

$$P_m(q) = \Pr\left[\left(\sum_{i=1}^{m} X_i \bmod q\right) \bmod 2 = \bigoplus_{i=1}^{m} (X_i \bmod 2) \,\middle|\, \sum_{i=1}^{m} X_i \neq 0 \bmod q\right] \tag{4.8}$$

où $X_1, \cdots, X_m$ sont des variables aléatoires mutuellement indépendantes et uniformes sur $\mathbb{Z}/q\mathbb{Z} \setminus \{0\} = \{1, \cdots, q-1\}$. Alors :

1. *si m est pair alors $P_m(q) = \dfrac{1}{2}$.*

2. *si m est impair alors*

$$P_m(q) = \frac{\sum_{i=0}^{(m-1)/2} \sum_{j=1}^{q-1} \Gamma_m(q, 2iq+j)}{(q-1)^m - \sum_{i=1}^{m-1} \Gamma_m(q, iq)}.$$

où $\Gamma_m(q,\ell)$ *est le nombre de* $1 \leq x_1, \cdots, x_m \leq q-1$ *tels que* $x_1 + \cdots + x_m = \ell$ *et est donné par*

$$\Gamma_m(q,\ell) = \sum_{i=0}^{m} (-1)^i \binom{m}{i} \binom{\ell - i(q-1) - 1}{m-1}.$$

Corollaire 4.9. *Si* $q > 7$*, alors*

$$P_3(q) = \frac{1}{3} + \mathcal{O}\left(\frac{1}{q}\right), \; P_5(q) = \frac{17}{30} + \mathcal{O}\left(\frac{1}{q}\right), \; P_7(q) = \frac{149}{315} + \mathcal{O}\left(\frac{1}{q}\right).$$

Les biais observés sur l'équation (4.7) sont donc respectivement de $\frac{1}{3}$, $\frac{2}{15}$ *et* $\frac{17}{315}$.

On remarque ainsi que l'équation (4.7) possède un biais élevé pour un cadre cryptanalytique. De plus, cette propriété est intrinsèque à la ℓ-séquence, c'est-à-dire qu'elle ne dépend que de son entier de connexion et non d'une propriété d'implémentation comme c'était le cas pour l'attaque par LFSRisation. Elle peut donc s'appliquer à n'importe quel FCSR, annulaire ou sur les mots, à condition de réussir à trouver une relation de type $(m+1)$.

Les auteurs de [TQ09] proposent une méthode pour trouver des relations de type $(m+1)$ mais celle-ci dépasse le coût de la recherche exhaustive pour F-FCSR-H v3 [ABL+09]. Wang, Stankovski et Johansson [WSJ15] montrent alors qu'il est possible de transformer une relation $(3+1)$ en une relation $(2+2)$. Ils utilisent ensuite une approche basée sur le paradoxe des anniversaires pour trouver efficacement des relations $(2+2)$. De plus, leurs résultats se généralisent aisément à des relations de type $(n+n)$.

Pour transformer une relation $(3+1)$ en une relation $(2+2)$, il suffit de remarquer deux choses :

- D'abord, comme 2 est une racine primitive modulo q, on a $2^{\frac{q-1}{2}} = -1 \bmod q$ et donc $p(t) = -p(t + \frac{q-1}{2}) \bmod q$.
- Ensuite, pour la ℓ-séquence correpondante, on a $s(t) = s(t + \frac{q-1}{2}) \oplus 1$.

Ainsi, on voit que, quitte à regarder $(q-1)/2$ instants plus tard (soit une demi-période), on peut passer n'importe quel terme de l'autre côté de l'égalité dans l'équation (4.6). Cela se traduit au niveau de la ℓ-séquence par une inversion de l'équation linéaire (4.7) ; ce qui n'en change pas le biais. Ainsi, on peut à partir d'une relation $(3+1)$ obtenir une relation $(2+2)$ et inversement.

Les auteurs [WSJ15] profitent de la symétrie des relations $(2+2)$ pour appliquer une méthode basée sur le paradoxe des anniversaires généralisé afin de trouver rapidement une telle relation. Comme cette relation ne dépend que de l'entier de connexion q, toutes les cellules d'un même FCSR vont la vérifier indépendamment. Puisque F-FCSR-H v3 utilise un filtre (xor-)linéaire pour obtenir sa sortie, on peut alors appliquer le *piling-up lemma* de Matsui [Mat93] pour en déduire le biais de la somme (xor) de toutes ces relations attendu en sortie. On obtient ainsi un distingueur linéaire sur F-FCSR-H v3. La phase hors-ligne de ce dernier (recherche de la relation linéaire) coûte $2^{56.2}$. La phase en-ligne a une complexité en temps de $2^{37.2}$ et nécessite $2^{39.2}$ bits de suite chiffrante.

Chapitre 5

Propriétés différentielles & linéaires des Registres à décalage à rétroaction avec retenue

Dans ce chapitre, on exhibe certaines propriétés différentielles et linéaires sur les automates de type FCSR auxquels on ajoute une fonction de filtrage xor-linéaire.

Soit $n \in \mathbb{N}^*$, on se donne le FCSR de taille n défini à l'aide de sa matrice de transition, notée $T \in \mathcal{M}_n(\mathbb{Z})$. On note $q = \det(I - 2T)$ l'entier de connexion de l'automate ainsi que $A = \mathrm{Adj}(I - 2T)$ où Adj désigne la matrice adjointe, c'est-à-dire la transposée de la matrice des cofacteurs. Pour $t_0 \in \mathbb{N}$ et $i \in \{0, \cdots, n-1\}$, on note $m_i(t_0)$ (respectivement $c_i(t_0)$) le contenu de la i-ème cellule du registre principal (respectivement du registre de retenues) à l'instant t_0, avec la convention $c_i(t_0) = 0$ pour tout t_0 s'il n'y a pas d'addition en position i. On note $m(t_0)$ et $c(t_0)$ les vecteurs colonnes associés :

$$m(t_0) = (m_0(t_0)), \cdots, m_{n-1}(t_0))^T \in \{0,1\}^n \text{ et } c(t_0) = (c_0(t_0), \cdots, c_{n-1}(t_0))^T \in \mathbb{Z}^n.$$

On rappelle que le FCSR est mis à jour par la fonction de transition suivante :

$$\begin{cases} m(t_0+1) = Tm(t_0) + c(t_0) \bmod 2 \\ c(t_0+1) = Tm(t_0) + c(t_0) \operatorname{div} 2 \end{cases} \tag{5.1}$$

où mod 2 et div 2 sont définis par $\forall x \in \mathbb{Z}_2,\ x = (x \bmod 2) + 2(x \operatorname{div} 2)$ et $(x \bmod 2) \in \{0,1\}$. Pour $t_0 \in \mathbb{N}$ et $i \in \{0, \cdots, n-1\}$, on définit aussi les séries dyadiques $M_i(t_0)$ associées aux cellules $m_i(t_0)$ par

$$M_i(t_0) = \sum_{t=t_0}^{\infty} m_i(t) 2^{t-t_0} \in \mathbb{Z}_2$$

ainsi que le vecteur colonne associé

$$M(t_0) = \sum_{t=t_0}^{\infty} m(t) 2^{t-t_0} = (M_0(t_0), \cdots, M_{n-1}(t_0))^T \in \mathbb{Z}_2^n.$$

On rappelle le théorème fondamental des FCSR [ABL+09]. Pour tout $t_0 \in \mathbb{N}$, on a :

$$M(t_0) = \frac{p(t_0)}{q} \text{ avec } p(t_0) = A \cdot (m(t_0) + 2c(t_0)) \in \mathbb{Z}^n. \tag{5.2}$$

Dans la suite de ce chapitre, on adopte les notations suivantes :

- les xors sont notés $\oplus$ ou $\bigoplus$,
- les additions à retenue seront notées $+$ ou $\sum$.

5.1 Propriétés différentielles

5.1.1 FCSR seul

Considérons la fonction de $\mathbb{F}_2^n$ dans lui-même suivante :

Pour chaque $m \in \mathbb{F}_2^n$, on initialise le FCSR avec $m(0) = m$ et $c(0) = 0$. Puis on itère le FCSR un certain nombre de fois, noté r. On extrait alors $y = m(r)$ qui constitue la sortie de la fonction.

Pour un état initial $m \in \mathbb{F}_2^n$, on considère M la série dyadique associée

$$M = M(0) = \sum_{t=0}^{\infty} m(t)2^t.$$

Le théorème fondamental des FCSR affirme que

$$M = \frac{Am}{q}.$$

Soit maintenant une différence $\delta \in \mathbb{F}_2^n$, notons $m' = m + \delta$. Remarquons que si l'on note m_i et δ_i les i-èmes bits de m et δ respectivement alors $m_i \oplus \delta_i \neq m_i + \delta_i$ si et seulement si $m_i = \delta_i = 1$. Par conséquent, si on se restreint aux m tels que $m_i = 0$ dès que $\delta_i = 1$ alors tout ce qui est défini à l'aide de $\oplus$ peut être immédiatement obtenu à l'aide de $+$, en particulier les différentielles qui nous intéressent.

Notons M' la série dyadique associée à l'état $m' = m + \delta$, on a alors :

$$M' = \frac{Am'}{q} = \frac{A(m+\delta)}{q} = M + \frac{p_\delta}{q}, \text{ en posant } p_\delta = A\delta. \tag{5.3}$$

Remarquons alors que la série p_δ/q est en fait la série obtenue à l'aide du FCSR initialisé avec $m(0) = \delta$ et $c(0) = 0$. Notons :

$$\frac{p_\delta}{q} = \sum_{t=0}^{\infty} \delta(t)2^t$$

c'est à dire que $\delta(t)$ est le t-ième terme du dévelopement dyadique de p_δ/q. De sorte qu'à chaque instant $t \geq 0$ et pour tout $0 \leq i \leq n-1$, on calcule $m'_i(t)$ comme somme de $m_i(t)$ et de $\delta_i(t)$ à l'aide d'un additionneur à retenue comme en figure 5.1. Si on note $\gamma_i(t)$ la retenue associée, on a :

$$\begin{cases} m'_i(t) = m_i(t) \oplus \delta_i(t) \oplus \gamma_i(t-1) \\ \gamma_i(t) = m_i(t)\delta_i(t) \oplus m_i(t)\gamma_i(t-1) \oplus \delta_i(t)\gamma_i(t-1) \end{cases} \tag{5.4}$$

avec la convention $\gamma_i(-1) = 0$.

Intéressons-nous alors à la probabilité d'avoir $m'_i(t+1) = m_i(t+1) \oplus \delta_i(t+1)$. Par la première égalité de (5.4) on a :

$$\Pr[m'_i(t+1) = m_i(t+1) \oplus \delta_i(t+1)] = \Pr[\gamma_i(t) = 0]$$

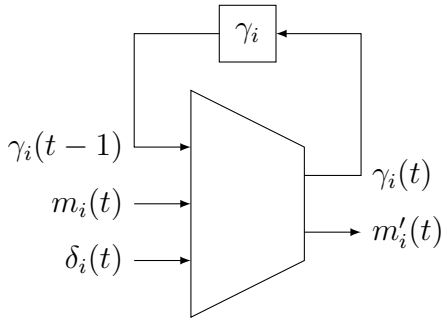


FIGURE 5.1 – Schéma d'un additionneur à retenue.

Nous allons donc regarder comment évolue la probabilité $\Pr[\gamma_i(t) = 0]$ en fonction du temps. On sait, par la seconde égalité de (5.4), que :

$$\gamma_i(t) = m_i(t)\delta_i(t) \oplus m_i(t)\gamma_i(t-1) \oplus \delta_i(t)\gamma_i(t-1) \tag{5.5}$$

Si $\delta_i(t) = 1$, cette égalité devient :

$$\gamma_i(t) = m_i(t) \oplus \gamma_i(t) \oplus m_i(t)\gamma_i(t-1) = m_i(t) \mid \gamma_i(t-1)$$

en notant $\mid$ l'opérateur booléen OU. De sorte que $\gamma_i(t) = 0$ si et seulement si $m_i(t) = \gamma_i(t-1) = 0$. Ainsi, on a alors

$$\Pr[\gamma_i(t) = 0] = \Pr[\gamma_i(t-1) = 0, m_i(t) = 0]$$

Hypothèse 5.1. *Pour tout $t \geq 0$ et tout $0 \leq i \leq n-1$, les $m_i(t)$ sont uniformément distribués et indépendants des $\gamma_i(t-1)$.*

En utilisant l'hypothèse 5.1, on a alors :

$$\Pr[\gamma_i(t) = 0] = \Pr[m_i(t) = 0]\Pr[\gamma_i(t-1) = 0] = \frac{1}{2}\Pr[\gamma_i(t-1) = 0]. \tag{5.6}$$

Si maintenant $\delta_i(t) = 0$, l'égalité (5.5) devient :

$$\gamma_i(t) = m_i(t)\gamma_i(t-1)$$

De sorte que $\gamma_i(t) = 1$ si et seulement si $m_i(t) = \gamma_i(t-1) = 1$. En utilisant l'hypothèse 5.1, on a alors

$$\Pr[\gamma_i(t) = 1] = \frac{1}{2}\Pr[\gamma_i(t-1) = 1]$$

D'où :

$$\Pr[\gamma_i(t) = 0] = 1 - \frac{1}{2}\big(1 - \Pr[\gamma_i(t-1) = 0]\big) = \frac{1}{2} + \frac{1}{2}\Pr[\gamma_i(t-1) = 0] \tag{5.7}$$

En combinant les équations (5.6) et (5.7), on peut écrire :

$$\Pr[m_i'(t) \oplus m_i(t) \oplus \delta_i(t) = 0] = \Pr[\gamma_i(t-1) = 0] = \frac{\overline{\delta_i(t-1)} + \Pr[\gamma_i(t-2) = 0]}{2} \tag{5.8}$$

où $\overline{\delta_i(t)}$ désigne le complémentaire de $\delta_i(t)$. Ce qui est équivalent à :

$$\Pr[m'_i(t) \oplus m_i(t) \oplus \delta_i(t) = 1] = \Pr[\gamma_i(t-1) = 1] = \frac{\delta_i(t-1) + \Pr[\gamma_i(t-2) = 1]}{2} \tag{5.9}$$

Notons $P_i^\delta(t) = \Pr[m'_i(t) \oplus m_i(t) \oplus \delta_i(t) = 1]$. L'équation (5.9) s'écrit alors

$$P_i^\delta(t) = \frac{\delta_i(t-1)}{2} + \frac{P_i^\delta(t-1)}{2}$$

d'où en itérant :

$$P_i^\delta(t) = \frac{\delta_i(t-1)}{2} + \frac{\delta_i(t-2)}{4} + \frac{P_i^\delta(t-2)}{4} = \frac{\delta_i(t-1)}{2} + \frac{\delta_i(t-2)}{4} + \frac{\delta_i(t-3)}{8} + \cdots$$

soit au final :

$$P_i^\delta(t) = \sum_{s=1}^{t} \frac{\delta_i(t-s)}{2^s} = \frac{1}{2^t} \sum_{u=0}^{t-1} \delta_i(u) 2^u.$$

De sorte que si on note $P^\delta(t) = (P_0^\delta(t), \cdots, P_{n-1}^\delta(t))^T$, on a :

$$P^\delta(t) = \frac{1}{2^t} \left(\frac{A\delta}{q} \bmod 2^t \right) \tag{5.10}$$

On dispose ainsi d'un moyen facile de calculer la probabilité de la différentielle $\delta \to \delta(r)$ entre le temps 0 et le temps r restreinte à un bit i pour un sous-ensemble des messages $\mathcal{M}_\delta = \{m \in \{0,1\}^n / \ \mathrm{Supp}(m) \cap \mathrm{Supp}(\delta) = \emptyset\}$ où $\mathrm{Supp}(m)$ désigne le support du vecteur m, c'est-à-dire l'ensemble des positions i telles que $m_i \neq 0$. Le cardinal de $\mathcal{M}_\delta$ vaut alors $|\mathcal{M}_\delta| = 2^{n - w_H(\delta)}$, où $w_H(\delta)$ est le poids de Hamming de δ.

5.1.2 FCSR filtré

On considère maintenant que l'extraction de la sortie du FCSR se fait à l'aide d'un filtre linéaire $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$. C'est-à-dire qu'à l'entrée $m(0) = m$ (et $c(0)$ initialisé à 0), on associe la sortie $y = f(m(r))$. Comme f est linéaire, on peut écrire :

$$f(\mu) = \bigoplus_{i=0}^{n-1} \lambda_i \mu_i$$

où les $\lambda_i \in \mathbb{F}_2$ sont les coefficients du filtre f. Soit I l'ensemble des positions i telles que $\lambda_i = 1$; alors f s'écrit :

$$f(\mu) = \bigoplus_{i \in I} \mu_i.$$

On se demande alors ce qui se passe au niveau des différentielles précédentes.

Soit $\delta \in \{0,1\}^n$, $m \in \mathcal{M}_\delta$ et $m' = m + \delta$. Par la section précédente, on sait calculer pour $i \in I$, $\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$. Dans le cas d'une attaque différentielle, on s'intéresse à la probabilité

$$\Pi^\delta = \Pr[f(m'(r)) \oplus f(m(r)) \oplus f(\delta(r)) = 1] \tag{5.11}$$

En effet, pour une fonction g de $\mathbb{F}_2^n$ dans $\mathbb{F}_2$ et un couple $(\delta, \top)$ on s'intéresse à la probabilité $\Pr[g(m \oplus \delta) \oplus g(m) = \top]$. Dans notre cas, g est la composée de r itérations du FCSR suivie de l'application du filtre f et $\top = g(\delta) \oplus 1$.

Comme f est linéaire, on peut récrire (5.11) :

$$\Pi^\delta = \Pr[f(m'(r) \oplus m(r) \oplus \delta(r)) = 1] = \Pr\left[\bigoplus_{i \in I} \left(m'_i(r) \oplus m_i(r) \oplus \delta_i(r)\right) = 1\right] \tag{5.12}$$

de sorte que Π^δ est la probabilité d'avoir un nombre impair de termes $m'_i(r) \oplus m_i(r) \oplus \delta_i(r)$ égaux à 1.

Hypothèse 5.2. *Pour $i \in I$, les probabilités $P_i^\delta(r) = \Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ sont indépendantes.*

Cette hypothèse est raisonnable si les indices $i \in I$ ne sont pas trop proches les uns des autres, par exemple si à chaque indice $i \in I$ se trouve un additionneur à retenue. Dans ce cas, on peut utiliser le lemme suivant, dont la preuve se trouve dans [Gal63] mais qui n'est rien d'autre que le *piling-up lemma* de [Mat93] :

Lemme 5.1. *Soit $(x_1, \cdots, x_\ell) \in \{0,1\}^\ell$, ℓ bits statistiquement indépendants. Notons $p_i = \Pr[x_i = 1]$ la probabilité que le i-ème bit soit égal à 1. Alors la probabilité que $(x_1, \cdots, x_\ell)$ contienne un nombre impair de 1 est donnée par :*

$$\frac{1}{2}\left[1 - \prod_{i=1}^{\ell}(1 - 2p_i)\right]$$

Sous réserve que l'hypothèse 5.2 soit vraie, la probabilité Π^δ devient grâce au lemme :

$$\Pi^\delta = \Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1] = \frac{1}{2}\left[1 - \prod_{i \in I}\left(1 - 2P_i^\delta(r)\right)\right] \tag{5.13}$$

Au final, on dispose d'un moyen pour calculer la probabilité de la différentielle $(\delta, f(\delta(r)) \oplus 1)$ pour un sous-ensemble des messages $\mathcal{M}_\delta$. Pour chaque indice i en entrée du filtre f, on calcule la probabilité $\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ à l'aide de l'équation (5.10), puis on utilise le lemme (5.1) pour en déduire la probabilité $\Pi^\delta = \Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$.

5.1.3 Résultats expérimentaux

5.1.3.1 FCSR seul

On regarde dans un premier temps ce qui se passe sur le FCSR donné par la figure 5.2. Si $d = 5$ désigne le diamètre du FCSR, on itère le FCSR $r = d + 4 = 9$ fois. On s'intéresse en particulier aux différentielles $\delta = \delta(0)$ de poids de Hamming 1 car ce sont celles pour lesquelles on peut utiliser (5.10) sur le plus de messages m. On rappelle que les retenues sont initialisées à $c(0) = 0$.

Pour chaque δ de poids 1 et chaque bit i en sortie, on compte le nombre de $m \in \mathcal{M}_\delta$ tels que, avec les notations de la section précédente, $m'_i(r) \oplus m_i(r) \oplus \delta_i(r) \neq 0$ que l'on compare

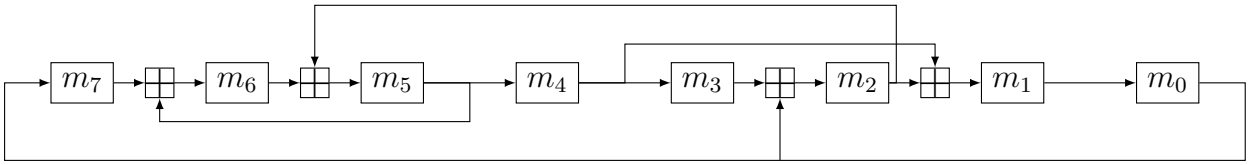


FIGURE 5.2 – Ring FCSR de taille $n = 8$, d'entier de connexion $q = -347$ et de diamètre $d = 5$.

avec la valeur obtenue grâce à (5.10). Le tableau 5.1 donne la probabilité théorique, le tableau 5.2 la fréquence observée et le tableau 5.3 donne le biais (valeur absolue de la différence) entre les deux.

	$\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ théorique							
Différentielle δ	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
$(1,0,0,0,0,0,0,0)$	0.830	0.414	0.0977	0.219	0.109	0.0547	0.430	0.660
$(0,1,0,0,0,0,0,0)$	0.660	0.830	0.195	0.438	0.219	0.109	0.859	0.320
$(0,0,1,0,0,0,0,0)$	0.352	0.176	0.611	0.953	0.977	0.988	0.383	0.703
$(0,0,0,1,0,0,0,0)$	0.703	0.352	0.223	0.908	0.953	0.977	0.766	0.406
$(0,0,0,0,1,0,0,0)$	0.727	0.363	0.836	0.691	0.346	0.172	0.250	0.453
$(0,0,0,0,0,1,0,0)$	0.516	0.758	0.109	0.539	0.770	0.885	0.832	0.0312
$(0,0,0,0,0,0,1,0)$	0.0312	0.516	0.219	0.0781	0.539	0.770	0.666	0.0625
$(0,0,0,0,0,0,0,1)$	0.0625	0.0312	0.438	0.156	0.0781	0.539	0.332	0.127

Tableau 5.1 – Valeurs théoriques pour le FCSR -347

	$\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ observé							
Différentielle δ	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
$(1,0,0,0,0,0,0,0)$	0,828	0,406	0,0938	0,0938	0,117	0,05z7	0,430	0,656
$(0,1,0,0,0,0,0,0)$	0,656	0,828	0,203	0,461	0,227	0,117	0,867	0,313
$(0,0,1,0,0,0,0,0)$	0,344	0,172	0,625	0,961	0,977	0,992	0,367	0,703
$(0,0,0,1,0,0,0,0)$	0,688	0,344	0,203	0,914	0,977	0,992	0,773	0,375
$(0,0,0,0,1,0,0,0)$	0,703	0,344	0,828	0,664	0,367	0,0938	0,242	0,406
$(0,0,0,0,0,1,0,0)$	0,547	0,781	0,0938	0,477	0,727	0,867	0,805	0,0156
$(0,0,0,0,0,0,1,0)$	0,0156	0,531	0,203	0,0703	0,367	0,680	0,648	0,0469
$(0,0,0,0,0,0,0,1)$	0,0469	0,0156	0,375	0,164	0,102	0,555	0,336	0,0938

Tableau 5.2 – Valeurs observées pour le FCSR -347

On refait le même test avec le FCSR de la fonction de hachage GLUON-64. La famille GLUON est décrite au chapitre 6 ; la matrice de GLUON-64 est donnée par la figure 6.2. Pour chaque δ de poids 1, on teste $n_{test} = 2048$ messages $m \in \mathcal{M}_\delta$ et on regarde si $m'_i(r) \oplus m_i(r) \oplus \delta_i(r) \neq 0$ que l'on compare avec la valeur obtenue grâce à (5.10). On itère toujours $r = d + 4$ fois avec cette fois-ci le diamètre qui vaut $d = 29$. Le tableau 5.4 donne la probabilité théorique pour quelques valeurs de δ et de i et les tableaux 5.5 et 5.6 donnent respectivement la fréquence observée et la valeur absolue de la différence entre les deux pour ces mêmes valeurs de δ et i. De plus, lors de ces tests, le biais maximal obtenu entre la

	Biais entre $\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ théorique et observé							
Différentielle δ	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
$(1,0,0,0,0,0,0,0)$	0.00195	0.00781	0.00391	0.0391	0.00781	0.000	0.000	0.00391
$(0,1,0,0,0,0,0,0)$	0.00391	0.00195	0.00781	0.0234	0.00781	0.00781	0.00781	0.00781
$(0,0,1,0,0,0,0,0)$	0.00781	0.00391	0.0137	0.00781	0.000	0.00391	0.0156	0.000
$(0,0,0,1,0,0,0,0)$	0.0156	0.00781	0.0195	0.00586	0.0234	0.0156	0.00781	0.0312
$(0,0,0,0,1,0,0,0)$	0.0234	0.0195	0.00781	0.0273	0.0215	0.00781	0.00781	0.0469
$(0,0,0,0,0,1,0,0)$	0.0312	0.0234	0.0156	0.0625	0.0430	0.0176	0.0273	0.0156
$(0,0,0,0,0,0,1,0)$	0.0156	0.0156	0.0156	0.00781	0.172	0.0898	0.0176	0.0156
$(0,0,0,0,0,0,0,1)$	0.0156	0.0156	0.0625	0.00781	0.0234	0.0156	0.00391	0.0332

Tableau 5.3 – Biais entre les valeurs théoriques et observées pour le FCSR -347

fréquence mesurée et la probabilité calculée (où le maximum est pris sur tous les indices i de sortie et toutes les différentielles δ de poids 1) est 0.0651.

	$\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ théorique							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 120	bit 121	bit 122	bit 123
0	0.623	0.868	0.329	0.402	0.817	0.308	0.214	0.402
1	0.190	0.568	0.868	0.671	0.290	0.313	0.808	0.221
30	0.956	0.983	0.237	0.507	0.631	0.474	0.566	0.419
31	0.635	0.327	0.304	0.102	0.326	0.770	0.912	0.592

Tableau 5.4 – Quelques valeurs théoriques pour le FCSR de GLUON-64

	$\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1]$ observé							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 120	bit 121	bit 122	bit 123
0	0.632	0.875	0.330	0.392	0.815	0.323	0.219	0.399
1	0.214	0.564	0.870	0.663	0.301	0.316	0.819	0.214
30	0.957	0.981	0.245	0.515	0.618	0.481	0.576	0.428
31	0.623	0.319	0.300	0.109	0.334	0.763	0.928	0.602

Tableau 5.5 – Quelques valeurs observées pour le FCSR de GLUON-64

Dans les deux FCSR, on constate un biais entre la valeur calculée et celle obtenue lors des tests. Cela veut donc dire que l'hypothèse 5.1 est fausse puisque ce biais ne peut être imputé à l'échantillonnage dans le cas du -347 (le test étant exhaustif). Cependant hormis quelques cas comme le bit 4 avec $\delta = (0,0,0,0,0,0,1,0)$ dans le tableau 5.3, la valeur de ce biais reste faible.

	Biais entre $\Pr[m'_i(r) \oplus m_i(r) \oplus \delta_i(r) = 1$ théorique et observé							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 120	bit 121	bit 122	bit 123
0	0.00928	0.00671	0.000794	0.0107	0.00232	0.0158	0.00476	0.00281
1	0.0237	0.00440	0.00183	0.00739	0.0116	0.00305	0.0118	0.00696
30	0.000488	0.00171	0.00826	0.00794	0.0131	0.00784	0.00958	0.00867
31	0.0116	0.00787	0.00345	0.00674	0.00812	0.00696	0.0162	0.00995

Tableau 5.6 – Biais en valeur absolue entre les valeurs théoriques et observées pour le FCSR de GLUON-64

5.1.3.2 FCSR filtré

On regarde maintenant ce qui se passe sur le FCSR -347 (figure 5.2) dont on filtre la sortie à l'aide du filtre linéaire f défini par $f(m_0, \cdots, m_7) = m_1 \oplus m_2 \oplus m_5 \oplus m_6$ (On xore les bits ayant une cellule de retenue). On utilise l'équation (5.13) pour calculer le probabilité théorique de $\Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$ que l'on compare à la valeur observée. Le tableau 5.7 résume les résultats.

	$\Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$		
Différentielle δ	Probabilité théorique	Fréquence observé	Biais
$(1,0,0,0,0,0,0,0)$	0.491	0.516	0.0243
$(0,1,0,0,0,0,0,0)$	0.387	0.391	0.00357
$(0,0,1,0,0,0,0,0)$	0.483	0.547	0.0634
$(0,0,0,1,0,0,0,0)$	0.458	0.438	0.0208
$(0,0,0,0,1,0,0,0)$	0.530	0.516	0.0145
$(0,0,0,0,0,1,0,0)$	0.603	0.625	0.0221
$(0,0,0,0,0,0,1,0)$	0.502	0.531	0.0297
$(0,0,0,0,0,0,0,1)$	0.502	0.453	0.0484

Tableau 5.7 – Valeurs théoriques et observées pour le FCSR -347 filtré.

On refait le même test sur le FCSR de GLUON-64 filtré par la fonction $f : \mathbb{F}_2^{144} \to \mathbb{F}_2^8$ donnée à la figure 6.2 La valeur observée est cette fois calculée sur un échantillon de $n_{test} = 2048$ messages $m \in \mathcal{M}_\delta$. Le tableau 5.8 donne la probabilité théorique pour quelques valeurs de δ et les 8 bits de sortie et les tableaux 5.9 et 5.10 donnent respectivement la fréquence observée et le biais entre les deux pour ces mêmes valeurs de δ.

	$\Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$ théorique							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
0	0.499961	0.504082	0.496059	0.500014	0.500464	0.499662	0.500602	0.500015)
1	0.499435	0.499750	0.498971	0.500087	0.498767	0.500295	0.501494	0.499893
30	0.500027	0.500703	0.506707	0.500000	0.498593	0.499943	0.500680	0.500086
31	0.500160	0.498428	0.500378	0.499994	0.500047	0.503195	0.500012	0.499744

Tableau 5.8 – Valeurs théoriques pour le FCSR de GLUON-64 filtré.

	$\Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$ observé							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
0	0.500000	0.498047	0.483887	0.496094	0.502930	0.495117	0.493164	0.503906)
1	0.498047	0.486816	0.516113	0.512207	0.489746	0.507324	0.493652	0.501953
30	0.515137	0.505859	0.496094	0.504883	0.509766	0.517090	0.498047	0.491211
31	0.500000	0.492676	0.504395	0.495605	0.503906	0.506348	0.507324	0.518555

Tableau 5.9 – Valeurs observées pour le FCSR de GLUON-64 filtré.

	Biais entre $\Pr[f(m'(r)) \oplus f(m(r)) = f(\delta(r)) \oplus 1]$ théorique et observé ($\times 10^{-2}$)							
i tel que $\delta_i = 1$	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
0	0.00386238	0.603485	1.21722	0.392056	0.246525	0.454521	0.743771	0.389099)
1	0.138855	1.29337	1.71423	1.21202	0.902081	0.702953	0.784206	0.205994
30	1.51100	0.515652	1.06134	0.488329	1.11728	1.71466	0.263309	0.887489
31	0.0160217	0.575209	0.401688	0.438833	0.385952	0.315285	0.731182	1.88107

Tableau 5.10 – Biais entre les valeurs théoriques et observées pour le FCSR de GLUON-64 filtré.

Comme dans le cas non filtré, on observe un biais entre la valeur théorique et la valeur mesurée, l'hypothèse 5.1 étant toujours fausse et l'hypothèse 5.2 pas forcément vraie non plus. La valeur du biais reste du même ordre de grandeur que dans le cas non filtré

5.2 Nouvelles propriétés linéaires des FCSR

Dans cette section, on adapte les résultats de Tian et Qi [TQ09] (cf. 4.4.2). Dans leurs travaux, ils établissent un lien entre l'existence d'une relation linéaire entre différents termes de la forme 2^{-t_i} modulo l'entier de connexion q d'un FCSR et l'existence de biais linéaire en regardant ces termes modulo 2. Chaque valeur de t_i correspond à un instant différent dans la suite chiffrante générée par le FCSR. Or on sait que le contenu des différentes cellules d'un même FCSR peut aussi être vu comme étant des décalages les unes des autres, on a donc aussi un biais linéaire qui va cette fois-ci se faire entre les différentes cellules et au même instant.

Rappelons brièvement le théorème 4.5. Quitte à ne pas tenir compte des premiers bits correspondant à la prépériode, les suites produites par les différentes cellules du registre principal d'un FCSR sont les mêmes à un décalage temporel près. De plus les valeurs de ces décalages sont indépendantes de l'initialisation du FCSR.

En somme, si on dispose d'un FCSR avec entier de connexion q premier et se trouvant dans l'état $p = (p_0, \cdots, p_{n-1}) \in \mathbb{Z}^n$ avec $p_i q < 0$ et $0 < |p_i| < |q|$ pour tout $i \in \{0, \cdots, n-1\}$ alors le contenu de la i-ème cellule à l'instant t est donné par

$$m_i(t) = 2^{-(t+\tau_i)} \bmod q \bmod 2 \tag{5.14}$$

avec τ_i tel que $p_i = 2^{-\tau_i} \bmod q$.

Pour $k \geq 2$ et un entier impair q, on introduit la probabilité $P_k(q)$ définie par

$$P_m(q) = \Pr\left[\left(\sum_{i=1}^{m} X_i \bmod q\right) \bmod 2 = \bigoplus_{i=1}^{m} (X_i \bmod 2) \,\middle|\, \sum_{i=1}^{m} X_i \neq 0 \bmod q\right] \tag{5.15}$$

où $X_1, \cdots, X_k$ sont des variables aléatoires indépendantes et uniformes sur $\mathbb{Z}/q\mathbb{Z} \setminus \{0\} = \{1, \cdots, q-1\}$.

On aimerait appliquer cela au cas où q est l'entier de connexion d'un FCSR et où les X_i sont certains des $p_i(t)$. On s'intéresse en particulier aux valeurs de k petites, puisque les relations ont alors un plus grand biais. La valeur de $P_k(q)$ est donnée dans le théorème 4.8.

Soit $q' = hq$ un multiple impair de q avec

$$q' + 1 = 2^{u_0} + 2^{u_1} + \cdots + 2^{u_r} \text{ avec } 0 < u_0 < u_1 < \cdots < u_r.$$

Pour $t \in \mathbb{N}$, en réduisant modulo q et en multipliant l'équation ci-dessus par $2^{-(t+u_0+u_r)}$, on a

$$2^{-(t+u_r)} + 2^{-(t+u_0+u_r-u_1)} + \cdots + 2^{-(t+u_0)} \equiv 2^{-(t+u_0+u_r)} \bmod q.$$

Si chacun de u_r, $u_0 + u_r - u_1$,..., u_0 et $u_0 + u_r$ correspond modulo $q-1$ à un τ_i alors l'équation devient

$$2^{-(t+\tau_{i_0})} + 2^{-(t+\tau_{i_1})} + \cdots + 2^{-(t+\tau_{i_r})} \equiv 2^{-(t+\tau_{i_{r+1}})} \bmod q \tag{5.16}$$

et peut être réécrite avec les $p_i(t)$ en

$$p_{i_0}(t) + p_{i_1}(t) + \cdots + p_{i_r}(t) \equiv p_{i_{r+1}}(t) \bmod q.$$

Ainsi

$$m_{i_0}(t) \oplus m_{i_1}(t) \oplus \cdots \oplus m_{i_r}(t) = m_{i_{r+1}}(t) \tag{5.17}$$

si et seulement si

$$(p_{i_0}(t) \bmod 2) \oplus \cdots \oplus (p_{i_r}(t) \bmod 2) = (p_{i_0}(t) + \cdots + p_{i_r}(t) \bmod q) \bmod 2 \tag{5.18}$$

Si on suppose que $p_{i_0}(t)$,..., $p_{i_r}(t)$ sont des variables aléatoires indépendantes et uniformes sur $(\mathbb{Z}/q\mathbb{Z}) \setminus \{0\} = \{1, \cdots, q-1\}$, alors la probabilité que l'équation (5.18) soit vérifiée est exactement la probabilité $P_{r+1}(q)$ comme définie par (5.15).

Par exemple, si on trouvait un multiple impair q' de q avec $q'+1 = 2^{u_0} + 2^{u_1} + 2^{u_2}$ tel qu'il existe des indices $0 \leq i_0, i_1, i_2, i_3 \leq n-1$ vérifiant :

$$\begin{cases} u_2 \equiv \tau_{i_0} \mod q-1 \\ u_0 + u_2 - u_1 \equiv \tau_{i_1} \mod q-1 \\ u_0 \equiv \tau_{i_2} \mod q-1 \\ u_0 + u_2 \equiv \tau_{i_3} \mod q-1 \end{cases} \tag{5.19}$$

alors l'équation (5.17) devrait être vérifiée avec probabilité $\frac{1}{3}$.

Contrairement à Tian et Qi [TQ09] qui n'ont pas de restrictions sur les valeurs des décalages τ_i (autres que la complexité de l'attaque), nous devons avoir que les décalages τ_i correspondent bien tous à des valeurs possibles pour les cellules du FCSR. Étant donné un FCSR, il est facile de déterminer un ensemble Θ de décalages temporels $\Theta = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$ en calculant un vecteur non-nul de $\mathrm{Adj}(I - 2T) \bmod q$. Pour trouver un biais linéaire de poids $r+1$, on recherche des décalages licites $\tau_{i_0}, \tau_{i_1}, \cdots, \tau_{i_r} \in \Theta$, deux à deux différents, qui vérifient :

$$\begin{cases} \tau_{i_0} + \tau_{i_r} \in \Theta \\ \displaystyle\sum_{j=1}^{r-1} 2^{-\tau_{i_j}} \equiv 2^{-(\tau_{i_0} + \tau_{i_r})} \bmod q \end{cases} \tag{5.20}$$

La condition $\tau_{i_0} + \tau_{i_r} \in \Theta$ correspond à la dernière équation du système (5.19) tandis que l'autre condition représente le multiple de q de petit poids.

Puisque Θ est de cardinal n, une recherche exhaustive sur Θ^{r+1} coûte $\mathcal{O}(n^{r+1})$ pour voir si le système (5.20) possède des solutions. Cependant la première partie de l'équation (5.20) ne concerne que 2 variables parmi les n, on peut donc d'abord rechercher les solutions de cette équation avant de passer à la seconde. Le coût de cette recherche est $\mathcal{O}(n^2 + n^{r-1})$.

On a cherché par cette méthode s'il existait de tels biais dans les FCSR de la fonction de hachage GLUON, présentée au chapitre suivant. La recherche s'est avérée infructueuse.

Chapitre 6

La Famille de fonctions de hachage GLUON

Ce travail se situe à la confluence de deux thèmes de l'actualité cryptographique : la cryptographie légère et la compétition SHA-3. De nombreux algorithmes de chiffrement ont déjà relevé le défi posé par la cryptographie légère qui est d'offrir de la sécurité même dans les environnements extrêmement contraints. Certains de ces algorithmes sont décrits à la section 2.2 du chapitre 2 consacré à la cryptographie légère. En revanche, peu de fonctions de hachage satisfaisant ces exigences existaient avant la publication de GLUON. En particulier, aucun finaliste SHA-3 ne peut prétendre au qualificatif de léger. Les fonctions de hachage légères précédant GLUON sont QUARK [AHMNP10] et PHOTON [GPP11] toutes deux basées sur la construction éponge (cf. section 1.3.2). C'est dans ce contexte que la famille GLUON [BDM+12] a été proposée comme nouvelle instance lightweight de fonction de hachage. Il s'agit d'un travail en collaboration avec Thierry Berger, Joffrey D'Hayer, Kevin Marquet et Marine Minier présenté et publié à AFRICACRYPT en 2012.

Une première section est dédiée à la présentation de la famille GLUON. La majeure partie de la conception de GLUON s'est déroulée avant mon arrivée en thèse fin 2011. Une deuxième et une troisième section présentent ma contribution à GLUON : l'évaluation de sa sécurité face aux attaques différentielles et aux "cubes testeurs" respectivement. Enfin une dernière section décrit une attaque sur l'une des instances de GLUON publiée par Perrin et Khovratovich [PK15, PK14] en 2014.

6.1 Description de GLUON

La famille de fonctions de hachage GLUON se base sur la construction éponge décrite à la section 1.3.2 où la fonction f appliquée à l'état interne entre chaque opération d'absorption ou d'essorage utilise un FCSR filtré. Ce FCSR filtré est directement inspiré par le chiffrement à flot orienté matériel F-FCSR-v3 [ABL+09] et par le chiffrement à flot orienté logiciel X-FCSR-v2 [BMP09]. Les instances de GLUON, bien que différentes les unes des autres, sont toutes basées sur des Word-Ring-FCSR (cf. section 10.5) pour être efficaces à la fois en matériel et en logiciel. La structure générale de la famille GLUON est décrite en figure 6.1. Les blocs élémentaires à partir desquels elle est faite sont les suivants :

- le contenu du Word-Ring-FCSR, composé de w mots de r bits chacun, est noté $m(t) = (m_0(t), \cdots, m_{w-1}(t))$ pour le registre principal et $c(t) = (c_0(t), \cdots, c_{w-1}(t))$ pour le registre des retenues avec a mémoires actives (*i.e.* a rétroactions internes). Le FCSR est donné par sa matrice associée T telle que définie à la section 4.2.
- un filtre FI est aussi défini pour filtrer le contenu du registre principal $m(t)$. Il est xor-linéaire pour casser la structure dyadique du FCSR. Comme fait dans [ABL+09], il consiste à xorer ensemble des versions shiftées des mots du registre principal avec une retenue active. Plus précisément, si $\mathcal{F} = \{m_{f_0}, \cdots, m_{f_{\ell-1}}\}$ désigne l'ensemble des

mots m_i possédant une rétroaction, alors r bits de sortie sont donnés par

$$FI(m) = \bigoplus_{j=0}^{\ell-1} (m_{f_j} <<< k_j) \text{ avec } k_j \in \{-r/2, \cdots, r/2\}.$$

Ce filtre linéaire est utilisé seulement à la fin du calcul de la fonction f pour extraire la sortie de l'état interne du FCSR.

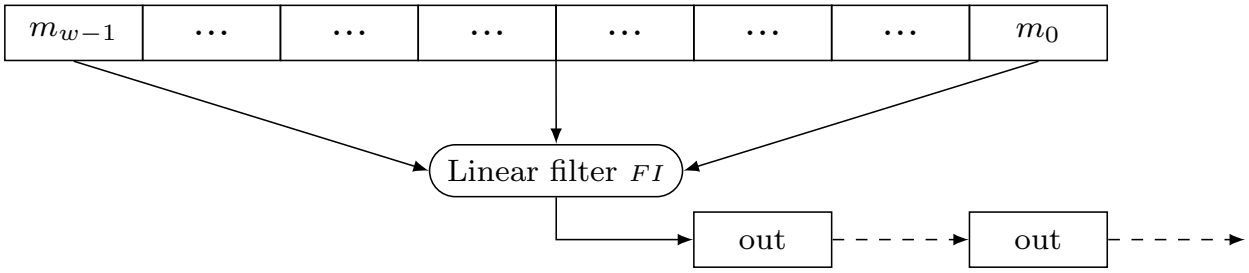


FIGURE 6.1 – Vue générale de la fonction f de GLUON.

La taille de l'entrée et de la sortie de f est $b = (w-1) \times r$ au lieu du $w \times r$ attendu parce qu'un mot en particulier (le mot m_{w-1}) du registre principal ne reçoit pas d'entrée de la fonction.

6.1.1 Détails de la fonction f

La fonction f de l'éponge processe une entrée de $b = (w-1) \times r$ bits de la manière suivante :

Initialisation : f initialise l'état interne du FCSR comme suit :
- les $w-1$ premiers mots de m sont initialisés avec les $(w-1) \times r$ bits d'entrée.
- le dernier mot m_{w-1} est initialisé avec la chaine tout à '1' de longueur r.
- le registre des retenues est initialisé à zéro.

Mise à jour : l'état interne $(m(t), c(t))$ du FCSR est mis à jour $d+4$ fois, où d désigne le diamètre du FCSR, en utilisant la fonction de transition du FCSR :

$$\begin{aligned} m(t+1) &= Tm(t) + c(t) \bmod 2 \\ c(t+1) &= Tm(t) + c(t) \operatorname{div} 2 \end{aligned}$$

Extraction : le FCSR est alors mis à jour $w-1$ fois. À chaque itération, un mot de r bits est extrait du registre principal m à l'aide du filtre FI de manière à obtenir les $b = (w-1) \times r$ de sortie.

6.1.2 La construction éponge déduite de la fonction f

Le taux associé à l'éponge est la taille r d'un mot du FCSR, sa capacité est $cp = (w-2) \times r$. La taille est $b = r + cp = (w-1) \times r$. Analysons en quelques mots comment cette fonction f rentre dans le modèle de l'éponge. La partie externe se compose du premier mot de r bits de l'entrée de f tandis que la partie interne est faite des $w-2$ mots restants. En

d'autres termes, durant l'absorption, à chaque appel à f, un mot de r bits p_i du message M à hacher est xoré avec le premier mot m_0 du registre principal du FCSR. Durant l'essorage, le premier mot obtenu après extraction du FCSR via le filtre FI sert de sortie.

6.1.3 Instances proposées

Comme pour QUARK [AHMNP10] et PHOTON [GPP11], on propose trois instances de la famille GLUON. On les note GLUON-N/r, comme fait pour QUARK. À partir des paramètres de taille, on utilise l'algorithme proposé dans [BMP09] pour calculer les différentes matrices T des FCSR : on tire au hasard des matrices T telles que

- $q = |\det(I - 2T)|$ est premier,
- $N \leq \log_2(q)$,
- l'ordre de 2 modulo q est $q - 1$ et est maximal pour s'assurer que les sorties produites sont des ℓ-séquences,
- le diamètre d est suffisamment petit pour assurer une diffusion rapide,
- le nombre de retenues est proche de $wr/2$.

Les filtres FI sont choisis au hasard avec la contrainte que les mots choisis doivent avoir une rétroaction.

On définit les deux opérations shift à gauche L et shift à droite R en convention big-endian pour un mot de r bits $(a_0, \cdots, a_{r-1})$ par $L(a) = (0, a_0, \cdots, a_{r-2})$ et $R(a) = (a_1, \cdots, a_{r-1}, 0)$.

Les instances de GLUON proposées sont :

- *niveau de sécurité* 64 *bits* : $r = 8$, $cp = 128$, $b = 136$ et $N = 128$, ce qui conduit à un FCSR composé de 18 mots de 8 bits et environ 70 retenues. La matrice T du FCSR et le filtre FI sont donnés par la figure 6.2. La valeur de q est

  $$q = -27\ 013\ 336\ 179\ 990\ 468\ 777\ 742\ 546\ 164\ 977\ 981\ 767\ 038\ 829.$$

  Les mots avec une rétroaction sont m_0, m_2, m_3, m_5, m_6, m_8, m_{11}, m_{12}, m_{13}, m_{14}, m_{16}, m_{17}. Le diamètre est $d = 29$.

- *niveau de sécurité* 80 *bits* : $r = 16$, $cp = 160$, $b = 176$ et $N = 160$, ce qui conduit à un FCSR composé de 12 mots de 16 bits et environ 90 retenues. La matrice T du FCSR et le filtre FI sont donnés par la figure 6.3. La valeur de q est

  $$q = -5\ 984\ 312\ 555\ 124\ 450\ 134\ 138\ 507\ 630\ 316\ 972\ 109\ 814\ 194\ 460\ 492\ 048\ 685\ 101.$$

  Les mots avec une rétroaction sont m_0, m_2, m_3, m_4, m_6, m_7, m_8, m_9. Le diamètre est $d = 58$.

- *niveau de sécurité* 112 *bits* : $r = 32$, $cp = 224$, $b = 256$ et $N = 224$, ce qui conduit à un FCSR composé de 9 mots de 32 bits et environ 130 retenues. La matrice T du FCSR et le filtre FI sont donnés par la figure 6.4. La valeur de q est

  $$q = -4\ 228\ 841\ 923\ 935\ 277\ 938\ 113\ 658\ 149\ 538\ 688\ 438\ 595\ 997$$
  $$356\ 223\ 832\ 672\ 089\ 412\ 465\ 139\ 266\ 477\ 362\ 728\ 972\ 043\ 613.$$

$$T = \begin{pmatrix}
0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & I & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & R^5 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L^3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & R^5 \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L^1 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & R^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \\
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L^6 & 0 & 0 & 0
\end{pmatrix}$$

$$\begin{aligned}
FI(m(t)) = {} & m_0(t) \oplus (m_2(t) >>> 4) \oplus (m_3(t) >>> 5) \oplus (m_5(t) >>> 3) \oplus \\
& (m_6(t) <<< 1) \oplus (m_8(t) >>> 2) \oplus (m_{11}(t) <<< 4) \oplus (m_{12}(t) >>> 1) \oplus \\
& (m_{13}(t) >>> 5) \oplus (m_{14}(t) >>> 3) \oplus (m_{16}(t) >>> 1) \oplus (m_{17}(t) <<< 2)
\end{aligned}$$

FIGURE 6.2 – Matrice T du FCSR et filtre FI de GLUON-64/8.

Les mots avec une rétroaction sont m_0, m_3, m_5, m_6, m_8. Le diamètre est $d = 42$.

Comme dans le cas de `PHOTON`, on pourrait aussi ajouter des instances avec des sorties de taille 80 et 256. Les valeurs de r correspondantes sont alors respectivement 8 et 32.

6.2 Tests différentiels

On cherche à évaluer expérimentalement la résistance de la fonction f de GLUON par rapport aux distingueurs différentiels. Pour rappel, un distingueur différentiel recherche des couples (a, b) de différences tels que le nombre de solutions x de l'équation $f(x \oplus a) \oplus f(x) = b$ soit le plus grand possible.

Étant donné la taille de la fonction f de GLUON (136 bits pour la plus petite instance), une approche exhaustive n'est pas envisageable. On réduit donc le spectre de notre étude. À cause de la structure éponge, un attaquant ne peut modifier que les r premiers bits de l'état interne de l'éponge. Par conséquent, on choisit d'étudier la fonction f restreinte à ces r premiers bits et on fixe le reste à une constante. On réitèrera alors l'expérience pour plusieurs valeurs de la constante. Ensuite, le filtre linéaire sur le FCSR extrait r bits à la fois, puis itère le FCSR et recommence. De par cette structure itérative, on choisit de ne

$$T = \begin{pmatrix}
0 & I & 0 & 0 & L^{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & L^3 & 0 & 0 \\
0 & L & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & I & L^2 & 0 & 0 & 0 & 0 & 0 \\
0 & I & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\
I & 0 & 0 & 0 & 0 & 0 & 0 & I & L^{12} & 0 & 0 & 0 \\
L^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & L^2 & 0 & I & 0 & 0 \\
0 & 0 & R^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \\
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$\begin{aligned} FI(m(t)) = m_0(t) \oplus (m_2(t) >>> 3) \oplus (m_3(t) >>> 7) \oplus (m_4(t) >>> 2) \oplus \\ (m_6(t) >>> 2) \oplus (m_7(t) >>> 4) \oplus (m_8(t) <<< 1) \oplus (m_9(t) >>> 2) \end{aligned}$$

FIGURE 6.3 – Matrice T du FCSR et filtre FI de GLUON-80/16.

$$T = \begin{pmatrix}
R^{12} & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & I & 0 & R^7 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & I & R^9 & 0 \\
0 & 0 & 0 & L^7 & 0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \\
I & 0 & L^6 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$FI(m(t)) = m_0(t) \oplus (m_3(t) >>> 15) \oplus (m_5(t) >>> 3) \oplus (m_6(t) <<< 5) \oplus (m_8(t) >>> 13)$$

FIGURE 6.4 – Matrice T du FCSR et filtre FI de GLUON-112/32.

considérer que les r premiers bits extraits mais on fera varier le nombre d'itérations du FCSR pour voir comment vont évoluer les meilleures différentielles. Enfin, on ne regarde que des différences en entrée avec un seul bit actif.

En somme, on réduit la fonction interne de GLUON à une fonction de $\mathbb{F}_2^r$ dans lui-même en fixant tout l'état à une constante en dehors du premier bloc. On itère alors le FCSR un certain nombre t de fois puis on extrait en utilisant une fois le filtre linéaire. On s'intéresse alors à la probabilité de la meilleure différentielle en fonction du nombre d'itérations $t \in \{1, \cdots, d + 4 + (w - 1)\}$ effectuées. Cette expérience est réitérée 128 fois avec différentes valeurs de constantes pour le reste de l'état interne initial. Les résultats sont donnés par la figure 6.5 pour GLUON-64 et par la figure 6.6 pour GLUON-80. Dans les deux

cas, on peut voir qu'après quelques tours où on arrive à avoir des différentielles qui passent avec probabilité un, la probabilité décroît rapidement à mesure que l'on augmente le nombre d'itérations du FCSR. La fonction finit par se comporter comme une fonction aléatoire et ce bien avant le nombre d'itérations réellement effectuées dans GLUON, respectivement $d+4 = 33$ pour GL-80 et $d+4 = 62$ pour GLUON-80. Par conséquent, GLUON-64 et GLUON-80 semblent bien protégés contre les attaques différentielles.

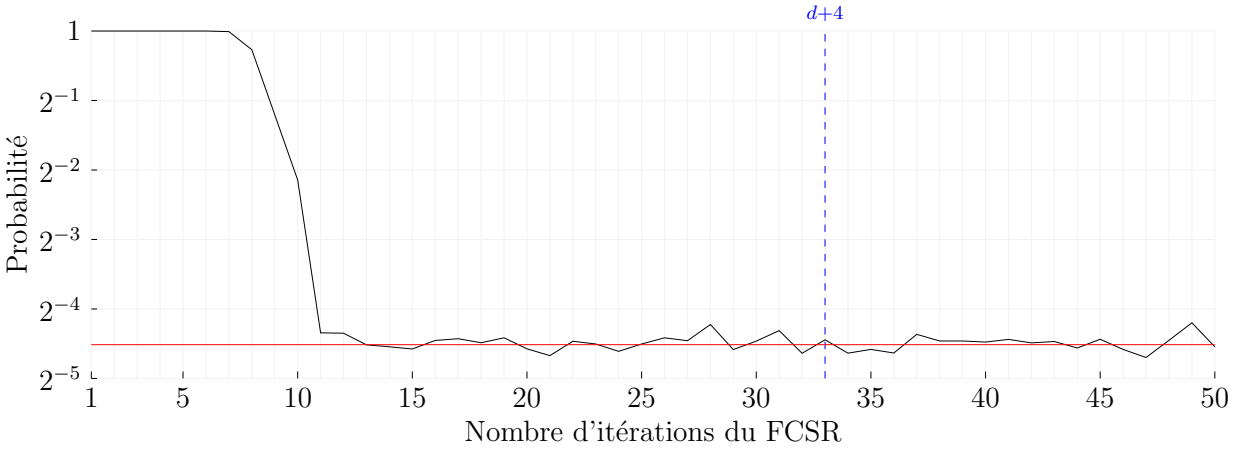


FIGURE 6.5 – Probabilité de la meilleure différentielle trouvée sur la version réduite de GLUON-64 en fonction du nombre de tours du FCSR. En rouge, la probabilité pour une fonction aléatoire.

Pour GLUON-112, il est trop coûteux de déterminer en pratique la probabilité d'une différentielle de manière exhaustive : 2^{32} appels à la fonction interne de GLUON par différence à tester et ceci doit être fait pour 32 différences possibles et répété 128 fois avec des constantes différentes ainsi que pour différentes valeurs de $r \in \{1, \cdots, 54\}$. On a donc opté pour une évaluation non-exhaustive. Ce faisant, lorsque la probabilité devient trop petite, on n'arrive pas à déterminer la vraie valeur à cause de l'échantillonage trop réduit. On a tout de même testé en prenant 2^{16} points sur les 2^{32}. On observe la même chose sur la figure 6.7 que pour les deux autres versions de GLUON : d'abord la probabilité un qui se maintient quelques tours puis une rapide chute et enfin une stagnation. Dans notre cas, la courbe stagne autour de 2^{-15} à cause de l'échantillonage. On conjecture que GLUON-112 se comporte aussi comme une fonction aléatoire vis-à-vis des attaques différentielles.

6.3 Cubes testeurs

Cette section résume l'étude effectuée sur la fonction interne de GLUON vis-à-vis des "cubes testeurs". Une première partie présente ce que sont ces cubes testeurs, tandis qu'une seconde partie présente le travail réalisé sur GLUON en lui-même.

6.3.1 Principe

Les "cubes attaques" sont une attaque sur les algorithmes de chiffrement introduite par Dinur et Shamir [DS09] en 2009 particulièrement efficace contre les fonctions construites

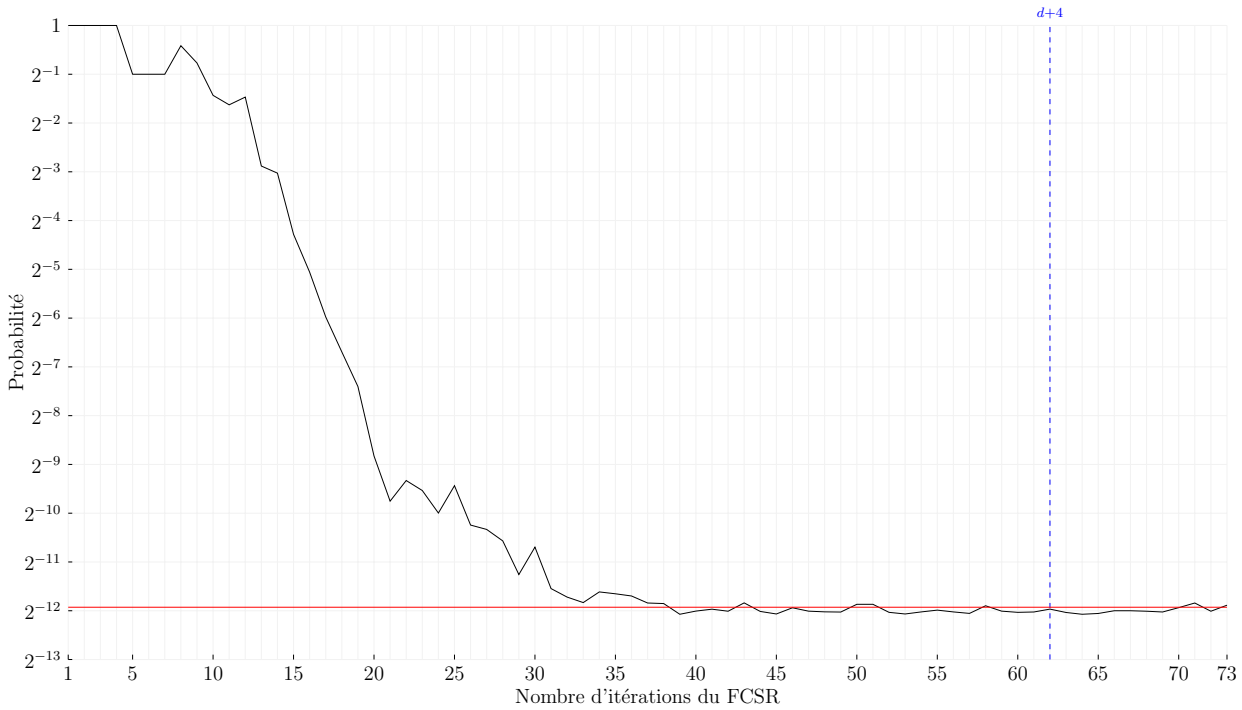


FIGURE 6.6 – Probabilité de la meilleure différentielle trouvée sur la version réduite de GLUON-80 en fonction du nombre de tours du FCSR. En rouge, la probabilité pour une fonction aléatoire.

à partir de composantes de degré algébrique faible. Les "cubes testeurs" sont une famille de distingueurs bâtis sur le même principe que les cubes attaques mais plus générale, au détriment de n'être plus qu'un distingueur.

Soit une fonction booléenne, $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Celle-ci peut être vue comme un polynôme en n indéterminées $x_1, \cdots, x_n$ sur $\mathbb{F}_2$ avec $x_i^2 = x_i$ et de degré au plus n. C'est ce qu'on nomme *forme algébrique normale* (ou ANF) de la fonction f. Si cette fonction fait partie d'un algorithme de chiffrement, il n'est en général pas possible de calculer $f(x_1, \cdots, x_n)$ si certains x_i dépendent de la clé. Il est toutefois toujours possible d'évaluer certaines parties internes de l'ANF de la fonction en observant le fait suivant.

Soit $I \subset \{x_1, \cdots, x_n\}$, notons t_I le monôme contenant tous les $x_i \in I$. En d'autres termes :

$$t_I = \prod_I x_i.$$

Factorisons alors dans f les monômes multiples de t_I : on a $f(x_1, \cdots, x_n) = p_I \cdot t_I + q_I$ où le polynôme p_I ne contient que des variables $x_i \notin I$ et où q_I ne contient pas de multiple de t_I. Il est alors possible, via uniquement des requêtes à f, d'évaluer le polynôme p_I même si celui-ci dépend de la clé dès lors que l'on peut modifier les variables $x_i \in I$ à sa guise (c'est donc une attaque à clair choisi). Il suffit pour cela de sommer $f(x_1, \cdots, x_n)$ sur les $2^{|I|}$ valeurs que peuvent prendre les $x_i \in I$, rappelant en cela les attaques intégrales (cf. section 3.3.2).

Considérons par exemple, la fonction $f(x_1, x_2, x_3, x_4) = x_1 + x_3 + x_1x_2x_3 + x_1x_2x_4 = x_1 + x_3 + x_1x_2(x_3 + x_4)$ et $I = \{x_1, x_2\}$. Avec les notations précédentes, on a $t_I = x_1x_2$,

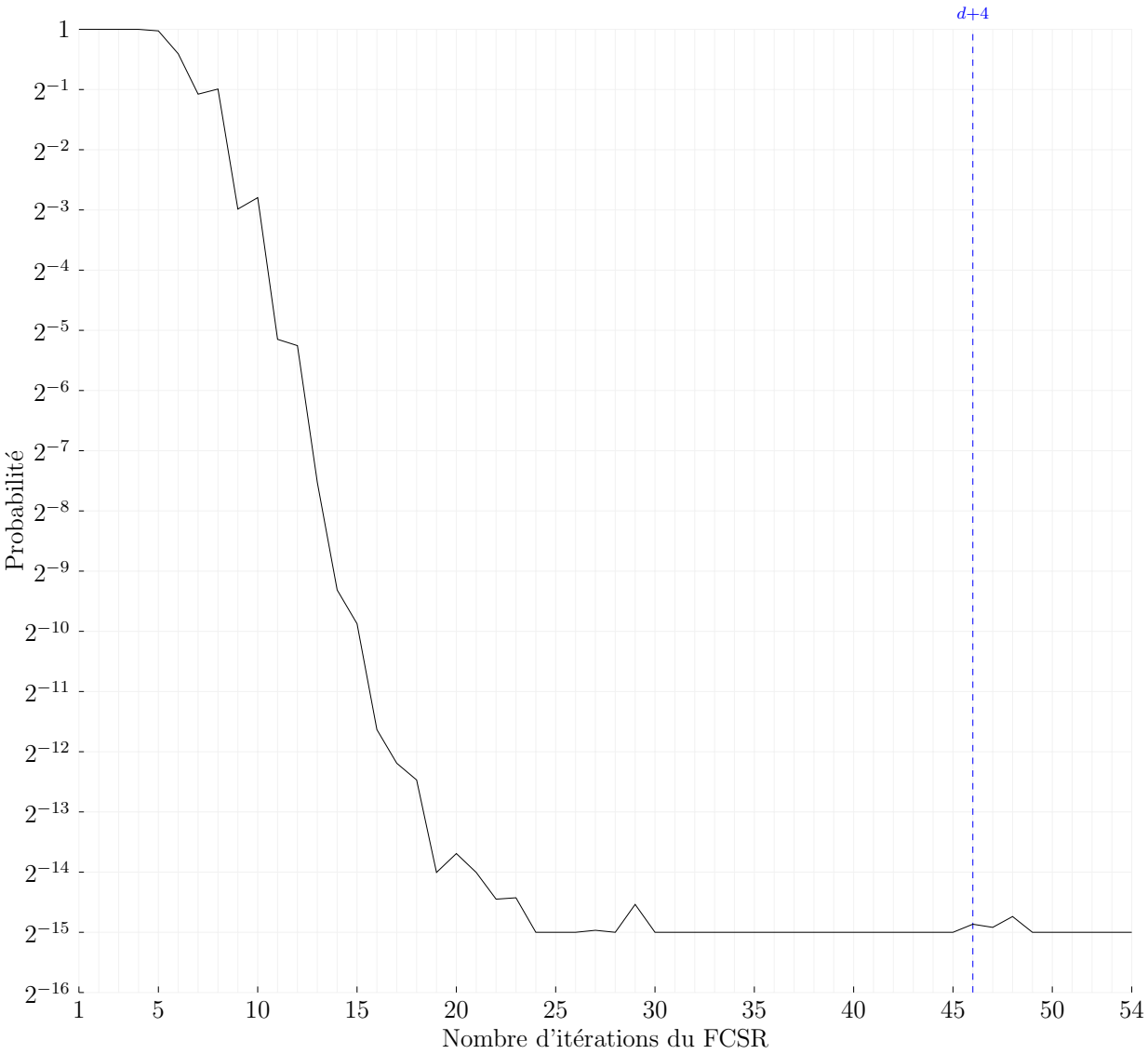


FIGURE 6.7 – Probabilité de la meilleure différentielle trouvée sur la version réduite de GLUON-112 en fonction du nombre de tours du FCSR. Estimation en évaluant sur 2^{16} points.

$p_I = (x_3 + x_4)$ et $q_I = x_1 + x_3$ et on a la propriété suivante :

$$\sum_{(x_1,x_2)\in\mathbb{F}_2^2} f(x_1, x_2, x_3, x_4) = 4(x_1 + x_3) + x_3 + x_4 = p_I.$$

De manière générale, l'ensemble I définit un hypercube de dimension $|I|$ contenant toutes les combinaisons possibles de $|I|$ bits – d'où le nom de cube attaque. En sommant sur ces $2^{|I|}$ valeurs, on obtient la valeur de p_I au point défini par les valeurs de $x_i \notin I$. Par abus de langage, le polynôme t_I s'appelle le *cube* tandis que le polynôme p_I se nomme *superpoly* associé à I.

Dans le cas des cubes attaques, on demande à ce que le superpoly p_I soit affine. Si on en trouve suffisament, on peut utiliser la méthode de la somme sur le cube pour évaluer ces polynômes et obtenir ainsi un système affine en les bits de clé qu'il ne reste plus qu'à

résoudre.

Dans le cas des cubes testeurs, on ne demande plus d'hypothèse de linéarité sur les superpolys. On utilise juste le fait qu'on peut voir f comme une boîte noire dont on peut modifier certaines entrées et d'autres non – dans le cas d'un fonction de hachage, on a accès à tous les bits. On fait alors appel à cette boîte noire pour calculer une partie de l'ANF de f. Si f est une fonction aléatoire, les superpolys vont avoir un comportement idéal. Dans le cas où f est une primitive cryptographique réelle, comme c'est le cas de la fonction interne de GLUON, certaines faiblesses risquent d'apparaître. On détecte alors ces faiblesses via un test de propriété algébrique. Les tests les plus courants incluent l'équilibre (le superpoly prend autant de fois la valeur 0 que la valeur 1), la linéarité, un degré faible, etc.

Au final, les cubes testeurs fournissent tout un panel de distingueurs sur la fonction f. Par exemple, Aumasson *et al.* [ADMS09] présentent des cubes attaques et des cubes testeurs sur les chiffrements MD6 et Trivium. Dans notre cas, c'est sur la fonction interne de GLUON que nous allons nous pencher. En effet, les cubes testeurs se révèlent particulièrement efficaces lorsque la fonction f possède une structure algébrique faible. On peut alors se demander si GLUON entre dans ce cas car les FCSR à partir desquels est bâtie GLUON possèdent une forte structure, nommément une structure dyadique. En particulier la fonction de transition d'un FCSR, bien que non-linéaire, n'est que quadratique.

6.3.2 Résultats

Pour rendre l'analyse praticable, on se place dans le même contexte que pour les tests différentiels. Seuls les r premiers bits varient, le reste est constant. De plus on ne regarde que les r premiers bits extraits mais on fait varier le nombre t de tours du FCSR : $t \in \{1, \cdots, d+4+(w-1)\}$. On s'intéresse à l'équilibre des superpolys obtenus à l'aide des cubes testeurs. Les tests sont exhaustifs lorsque la complexité le permet ; pour GLUON-112, on se restreint à des cubes de taille inférieure à 12 que l'on évalue sur 2^{12} points. Cette expérience est réitérée 128 fois avec différentes valeurs de constantes pour le reste de l'état interne initial. La métrique utilisée pour mesurer l'efficacité d'un cube I de dimension w et dont le superpoly est noté p_I, est :

$$2^{-w} \cdot \left(\Pr[p_I(x) = 1] - \frac{1}{2} \right)^2 .$$

Ce choix est motivé par le fait qu'il faut évaluer le superpoly $\left(\Pr[p_I(x) = 1] - \frac{1}{2}\right)^{-2}$ fois pour le distinguer d'une fonction aléatoire et que chaque évaluation du superpoly coûte 2^w appels à la fonction interne de GLUON. On recherche alors le cube testeur ayant la meilleure complexité. De plus, comme trouver un bon cube testeur n'est pas chose aisée pour un attaquant, on regarde aussi la complexité moyenne pour des cubes testeurs choisis au hasard. Les résultats sont donnés par la figure 6.8 pour GLUON-64 et par la figure 6.9 pour GLUON-80. Comme dans le cas des différentielles, on observe que la complexité se rapproche et rejoint le cas aléatoire après une quinzaine d'itérations.

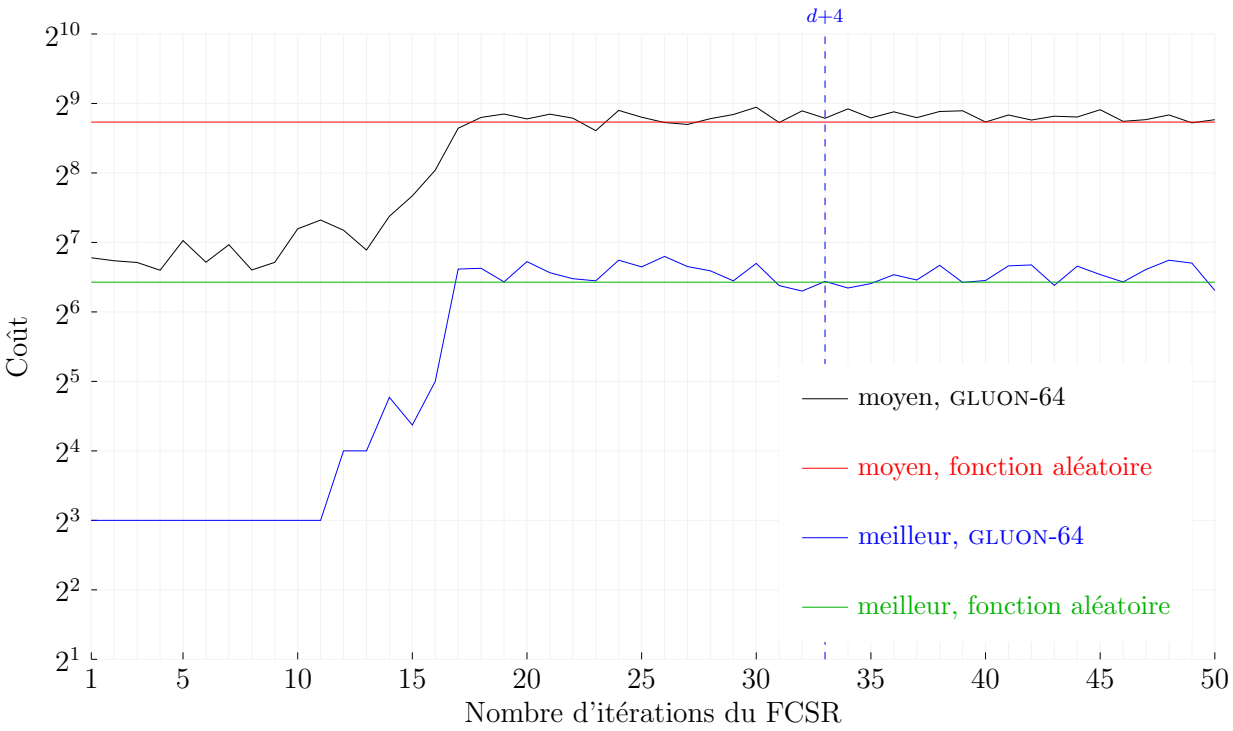


FIGURE 6.8 – Coût moyen et coût minimal des cubes testeurs pour GLUON-64.

6.4 Attaque sur GLUON-64

On présente ici une attaque sur GLUON-64 de Perrin et Khovratovich [PK15, PK14] présentée à FSE 2014. Il s'agit d'une recherche de préimage qui exploite une faiblesse de la construction éponge lorsque son taux r est petit et que la probabilité de collision de la fonction de transition utilisée s'éloigne trop de celui d'une fonction aléatoire.

Soit $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$, on définit le spectre de probabilité de collision comme étant l'ensemble $\{c_k\}_{k\geq 0}$ où c_k est la probabilité que l'équation $f(x+a) = f(a)$ (d'inconnue x) possède exactement k solutions lorsque a est tiré uniformément dans $\mathbb{F}_2^n$. En d'autres termes, on a

$$c_k = \Pr\left[\,|\{x \in \mathbb{F}_2^n | f(x+a) = f(a)\}| = k\right].$$

Le nombre moyen de solutions non-nulles de l'équation (d'inconnue x) est noté κ et vaut :

$$\kappa = \left(\sum_{k=1}^{\infty} k \cdot c_k\right) - 1.$$

À partir de cette valeur, il est possible de quantifier la perte d'information que l'on observe lorsqu'on itère successivement la fonction f, tant qu'on n'a pas atteint un cycle :

$$\frac{|\mathbb{F}_2^n|}{|f^i(\mathbb{F}_2^n)|} = \frac{\kappa}{2} \cdot i.$$

Il est aussi possible de raffiner les arguments de sécurité de la construction éponge en prenant en compte ce nombre moyen de collisions κ. Les auteurs de la construction

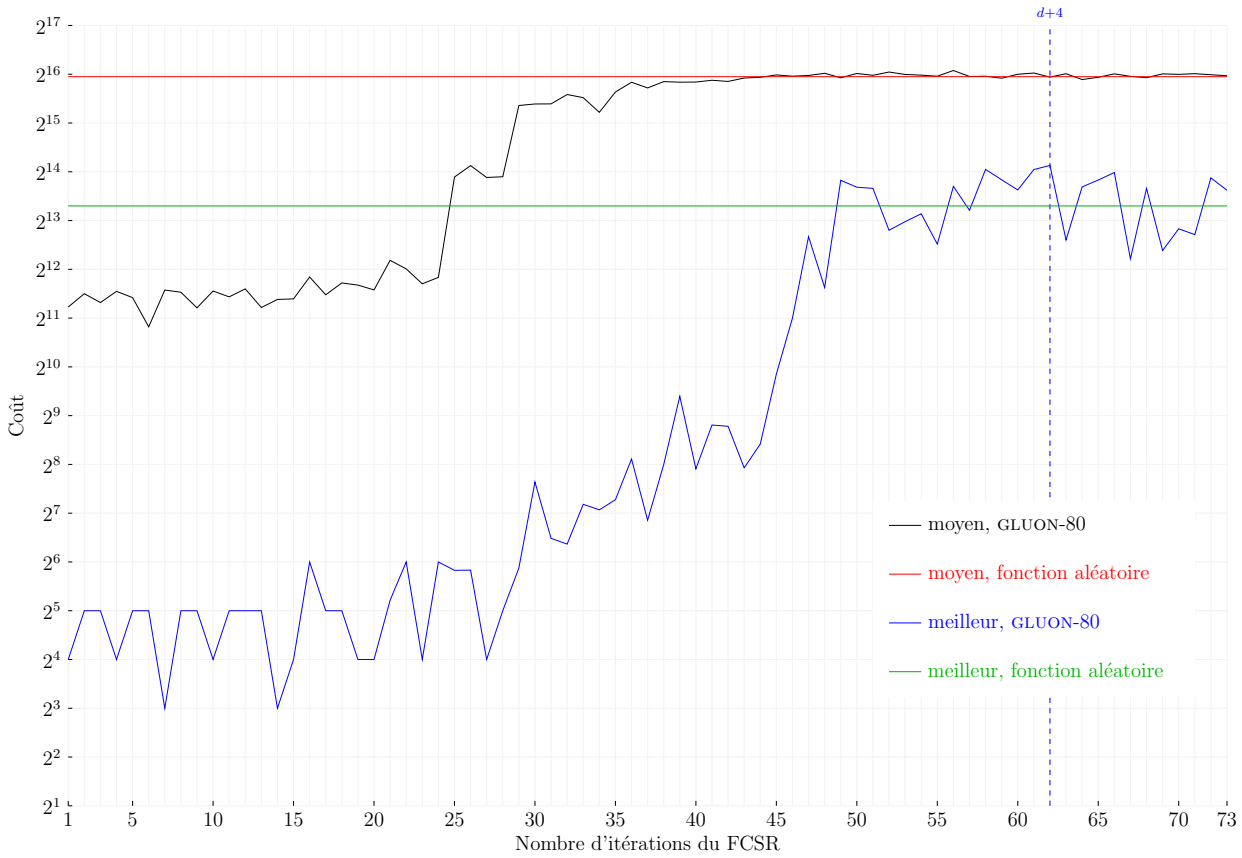


FIGURE 6.9 – Coût moyen et coût minimal des cubes testeurs pour GLUON-80.

éponge [BDPA08] montrent que pour une éponge de taux r, de capacité c et utilisant une fonction de transition aléatoire, la probabilité P de succès d'une recherche de collision est $P = q^2/2^{c+1}$ où q est le nombre de requêtes à la fonction de transition f de l'éponge. Perrin et Khovratovich [PK15, PK14] montrent alors que l'on peut la raffiner lorsque f est une fonction aléatoire dont le nombre moyen de collision κ est fixé :

$$P = \frac{q^2}{2^{c+1}} \cdot \left(1 + \frac{\kappa - 1}{2^r}\right) \tag{6.1}$$

Si f est totalement aléatoire, on a $\kappa = 1$ et on retombe bien sur la borne originale. Cependant, si le κ de la fonction choisie en pratique est proche de 2^r, la recherche de collision par paradoxe des anniversaires coûte un peu moins que les $c/2$ bits de sécurité attendus. En particulier, pour pouvoir répondre aux exigences de la cryptographie légère, le taux r de la fonction de hachage GLUON-64 est très petit puisqu'on a $r = 8$. De plus, la valeur de κ pour la fonction de transition de GLUON-64 est beaucoup plus élevée que pour une fonction aléatoire. Perrin et Khovratovich [PK15, PK14] ont estimé expérimentalement cette valeur en tirant au hasard 24000 éléments dans $\mathbb{F}_2^{c+r}$ et pour chaque élément, en calculant via un solveur SAT, à quelle valeur c_k il contribue. Ils trouvent que $\kappa = 6.982$ pour GLUON-64 au lieu de $\kappa = 1$ pour une fonction aléatoire. La figure 6.10 donne les valeurs du spectre de probabilité de collision $\{c_k\}_{k \geq 0}$ observé pour GLUON-64. Les valeurs sont très différentes d'une application aléatoire pour laquelle on a $c_k = e^{-1}/(k-1)!$ [FO89].

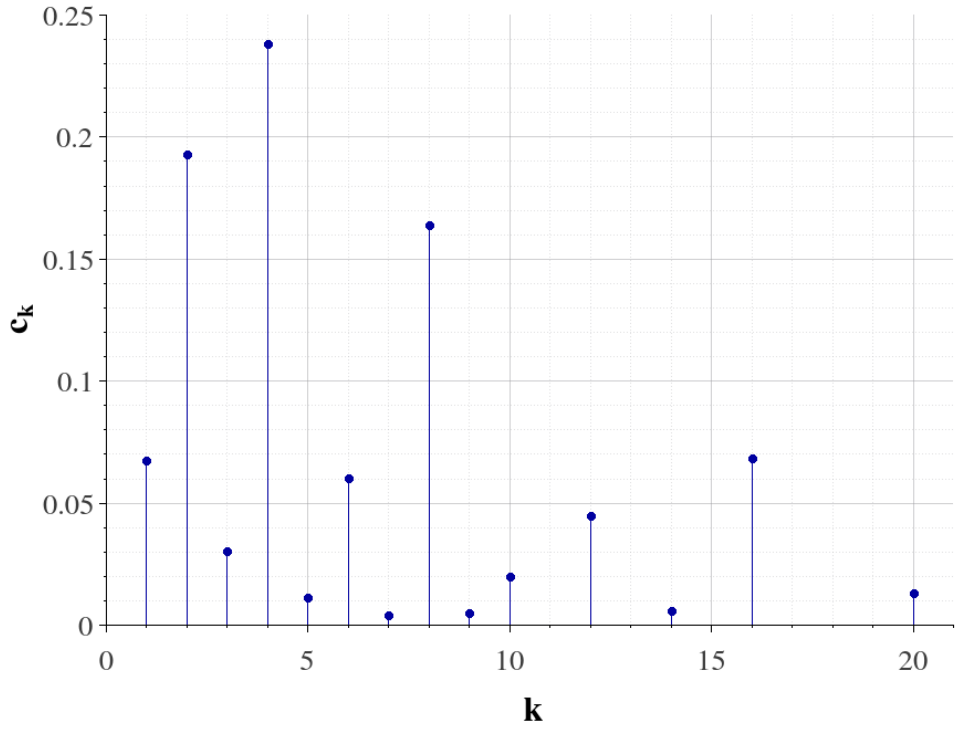


FIGURE 6.10 – Spectre de probabilité de collision de GLUON-64 observé [PK15, PK14].

Il est possible de transformer ces constatations en une recherche de préimage sur les longs messages. En effet, si un message m se termine par z fois le même bloc de taille r alors les z dernières injections sont identiques. L'ensemble des valeurs prises par l'état interne de l'éponge se réduit alors de plus en plus en fonction de z. Il est alors possible de trouver une préimage à $h(m)$ avec une complexité

$$2^c \cdot \frac{2^{r+2}}{\kappa \cdot z}.$$

Pour GLUON-64, on a $r = 8$ et $c = 128$. La complexité de l'attaque est alors de $2^{136+2}/(6.982 \cdot z) = 2^{128} \cdot (146.7/z)$. En somme, dès que le message m (inconnu) se termine par $z > 147$ blocs identiques alors cette attaque est plus efficace que la recherche exhaustive. La table 6.1 donne la complexité de l'attaque pour différentes valeurs de z.

Cette attaque ne fonctionne pas pour les deux autres variantes de GLUON. En effet, ces deux fonctions n'exhibent pas le comportement décrit à la figure 6.10. Expérimentalement, l'équation $f(x + a) = f(a)$ pour les fonctions de transition f de GLUON-80 et GLUON-112 n'a en général que $x = 0$ comme solution.

Les conclusions de cette attaque sont multiples. Si le nombre moyen de collisions κ de la fonction de transition de l'éponge est connu alors un terme en $(\kappa - 1)/2^r$ apparaît dans la complexité du paradoxe des anniversaires. Si le taux r de l'éponge est grand, comme dans le cas de SHA-3 où $r = 576$, cela ne pose pas de problème. Dans le cas de la cryptographie légère, un taux plus petit permet d'économiser de la place lors de l'implémentation. Il faut donc s'assurer dans ce cas que κ est petit. Une solution à cela est d'utiliser une fonction de

z	$\log_2(C)$
147	127.99
500	126.23
10^3	125.23
10^6	115.27
10^9	105.30
10^{18}	75.19

Tableau 6.1 – Complexité C de la recherche de préimage du haché $h(m)$ de [PK15, PK14] sur GLUON-64 en supposant que m se termine par z octets identiques.

transition bijective.

Dans le cas des FCSR, on voit que la taille de l'image des itérées diminue plus rapidement que pour une fonction aléatoire. Avoir une application bijective signifie se trouver sur le cycle principal de longueur $|q|-1$ dès le début. Cela semble difficile à réaliser pour un FCSR annulaire sans autre structure particulière. Pour un design à la GLUON, il faut donc s'assurer que le FCSR n'exhibe pas le comportement décrit à la figure 6.10. Quant à savoir pourquoi seule la version 64 bits de GLUON est touchée par cela et pas les deux autres, cela reste à l'heure actuellement un problème ouvert.

Troisième partie

Schémas de Feistel généralisés

Chapitre 7

État de l'art

On rappelle que le schéma de Feistel tel qu'introduit au chapitre 1 sépare son état interne en deux blocs. Un de ces deux blocs passe alors dans une fonction dépendante de la clé, dite fonction de Feistel, et le résultat est xoré à l'autre bloc. Au tour d'après, les rôles des deux blocs sont inversés. Les caractéristiques principales de ce schéma sont les suivantes :

- à chaque tour, une moitié de l'état agit sur l'autre moitié de l'état via la fonction de Feistel,
- seule une moitié de l'état est modifiée à chaque tour,
- le schéma en déchiffrement est le même que le schéma en chiffrement à l'ordre des sous-clés près,
- le schéma est inversible même si la fonction de Feistel utilisée ne l'est pas.

Les schémas de Feistel généralisés (*Generalized Feistel Networks* en anglais ou GFN) sont une classe plus générale de schémas reprenant les idées principales du schéma de Feistel mais en subdivisant l'état interne en plus de deux blocs. On se propose dans ce chapitre d'en dresser un état de l'art.

7.1 Différents types de GFN

7.1.1 Définitions et notations

Un schéma de Feistel généralisé divise son état interne en k blocs de n bits chacun. On note $x_0, \cdots, x_{k-1}$ les blocs en entrée du schéma et $y_0, \cdots, y_{k-1}$ les blocs en sortie. Plus généralement, on note y_i^r le contenu du i-ème bloc après r tours, avec par convention $y_i^0 = x_i$.

On divise un GFN en deux transformations successives : la couche non-linéaire et la couche permutation, comme sur la figure 7.1. La couche non-linéaire se compose de fonctions dépendantes de la clé, appelées fonctions de Feistel. Elles prennent certains blocs en entrée et leur sortie est xorée avec d'autres blocs. La couche permutation, quant à elle, consiste à permuter les k blocs du schéma de Feistel généralisé.

La manière dont les différentes fonctions sont disposées dépend du type de schéma de Feistel généralisé considéré, tandis que la permutation des blocs est en général le décalage circulaire. Dans le reste de cette section, nous supposerons que la permutation est le décalage circulaire à gauche (cf. Figure 7.1), c'est-à-dire que, pour $0 \leq i \leq k-1$, le bloc y_i reçoit le bloc $x_{i+1 \bmod k}$.

7.1.2 Différents types de schémas

Depuis leur introduction par Zheng, Matsumoto et Imai à CRYPTO'89 [ZMI89], les schémas de Feistel généralisés ont reçu beaucoup d'attention de la communauté [NR99, MV00, HR10, SM10, BS13]. Dans cette section, on se propose de lister les principaux schémas existants.

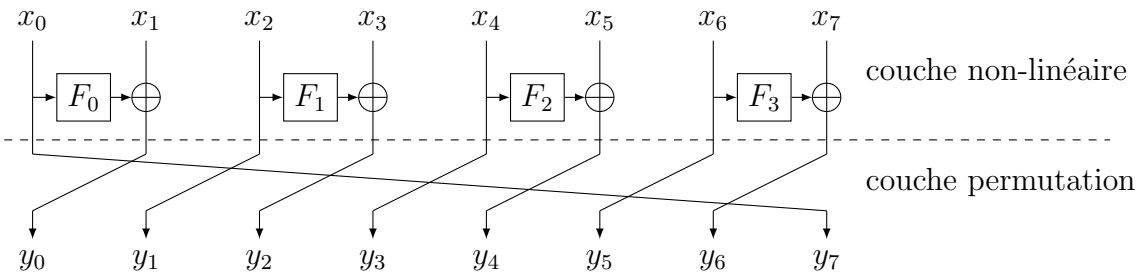


FIGURE 7.1 – Exemple de schéma de Feistel généralisé avec $k = 8$ blocs.

7.1.2.1 Feistel multi-source

Dans un schéma de Feistel généralisé multi-source (*source heavy* en anglais), un bloc parmi les k blocs joue un rôle particulier. Sans perte de généralité, on suppose qu'il s'agit du dernier bloc, x_{k-1}. La fonction de Feistel est une fonction de $\mathbb{F}_2^{(k-1)n}$ dans $\mathbb{F}_2^n$ et prend en entrée les blocs x_0 à x_{k-2} tandis que sa sortie est xorée avec x_{k-1}. Un exemple de GFN multi-source est donné en figure 7.2.

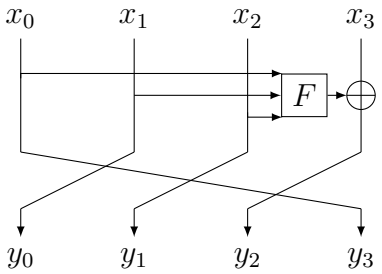


FIGURE 7.2 – Exemple de GFN multi-source avec $k = 4$ blocs.

Le GFN multi-source est utilisé par exemple dans les algorithmes de chiffrement RC2 [Riv98] et SPEED [Zhe97] ainsi que dans les familles de fonctions de hachage SHA-1 et SHA-2 [SHS12]. Son caractère pseudo-aléatoire est étudié dans [NR99, HR10].

7.1.2.2 Feistel multi-cible

Contrairement au Feistel multi-source dont la fonction de Feistel va de $\mathbb{F}_2^{(k-1)n}$ vers $\mathbb{F}_2^n$, le Feistel multi-cible utilise une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^{(k-1)n}$. Par convention, l'entrée de la fonction de Feistel est le bloc x_0 comme sur la figure 7.3 et la sortie est xorée aux $k-1$ autres blocs.

Le GFN multi-cible est par exemple utilisé dans le chiffrement MARS [BCD+99], finaliste de la compétition AES. Son caractère pseudo-aléatoire est étudié dans [MV00, HR10].

7.1.2.3 Feistel de type 1

Tout comme le Feistel classique à deux blocs, le Feistel de type 1 utilise une fonction de Feistel de $\mathbb{F}_2^n$ dans lui-même. À chaque tour, l'image par cette fonction de x_0 est xorée à x_1, comme sur la figure 7.4.

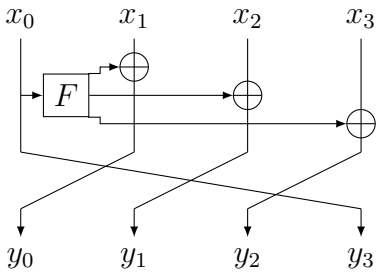


FIGURE 7.3 – Exemple de GFN multi-cible avec $k = 4$ blocs.

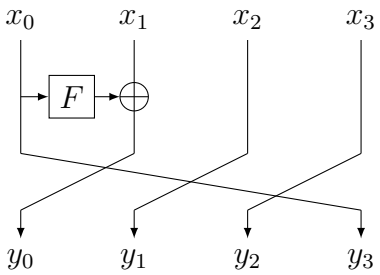


FIGURE 7.4 – Exemple de GFN de type 1 avec $k = 4$ blocs.

Le Feistel de type 1 est utilisé dans le candidat AES CAST-256 [AG99] et dans la fonction de hachage Lesamnta [HKY08], candidate à la compétition SHA-3. Son caractère pseudo-aléatoire est étudié dans [ZMI89, MV00, HR10].

7.1.2.4 Feistel de type 2

Tout comme le Feistel de type 1, le Feistel de type 2 utilise des fonctions de $\mathbb{F}_2^n$ dans lui-même. Plus précisément, lorsque le nombre de blocs k est pair, la couche non-linéaire d'un Feistel de type 2 est constituée de $k/2$ fonctions de Feistel $F_0, \cdots, F_{k/2-1}$. Chaque fonction F_i a pour entrée le bloc x_{2i} et xore sa sortie avec le bloc x_{2i+1}, comme sur les figures 7.1 et 7.5.

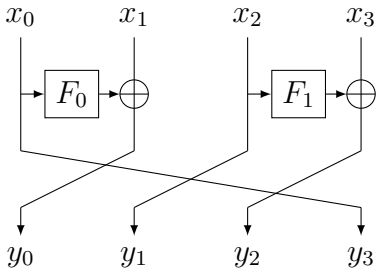


FIGURE 7.5 – Exemple de GFN de type 2 avec $k = 4$ blocs.

Le Feistel de type 2 est utilisé dans les chiffrements RC6 [RRSY98], HIGHT [HSH+06] et CLEFIA [SSA+07]. Son caractère pseudo-aléatoire est étudié dans [ZMI89, MV00, HR10, SM10]. De plus, des Feistels de type 2 dont la couche permutation a été modifiée sont utilisés dans les chiffrements légers Piccolo [SIH+11] et TWINE [SMMK12]. Ceci sera discuté plus en détail dans la section 7.3 ci-après.

7.1.2.5 Feistel de type 3

Un Feistel de type 3 nécessite $k-1$ fonctions de Feistel de $\mathbb{F}_2^n$ dans lui-même $F_0, \cdots, F_{k-2}$. Chaque fonction F_i a pour entrée le bloc x_i et xore sa sortie avec le bloc x_{i+1}. Il est à noter que le contenu du $(i+1)$-ème bloc en entrée de la fonction F_{i+1} est habituellement le contenu *avant* d'être xoré avec $F_i(x_i)$, comme sur la figure 7.6.

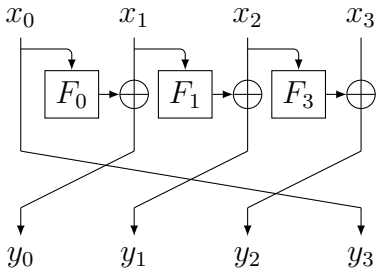


FIGURE 7.6 – Exemple de GFN de type 3 avec $k = 4$ blocs.

De plus, à cause de sa structure, le Feistel de type 3 en mode déchiffrement ne permet pas l'évaluation parallèle des différentes fonctions de Feistel. En effet, la fonction F_i a besoin de l'entrée $x_i = y_{i-1} \oplus F_{i-1}(x_{i-1})$, donc $F_{i-1}(x_{i-1})$ doit être évalué auparavant. Le caractère pseudo-aléatoire du Feistel de type 3 est étudié dans [ZMI89, MV00, HR10].

7.1.2.6 Feistel de Nyberg

Tout comme le Feistel de type 2, le Feistel de Nyberg [Nyb96] utilise $k/2$ fonctions de Feistel de $\mathbb{F}_2^n$ dans lui-même $F_0, \cdots, F_{k/2-1}$ mais disposées autrement. La fonction F_i a pour entrée le bloc $x_{k/2-1-i}$ et xore sa sortie avec le bloc $x_{k/2+i}$, comme sur la figure 7.7.

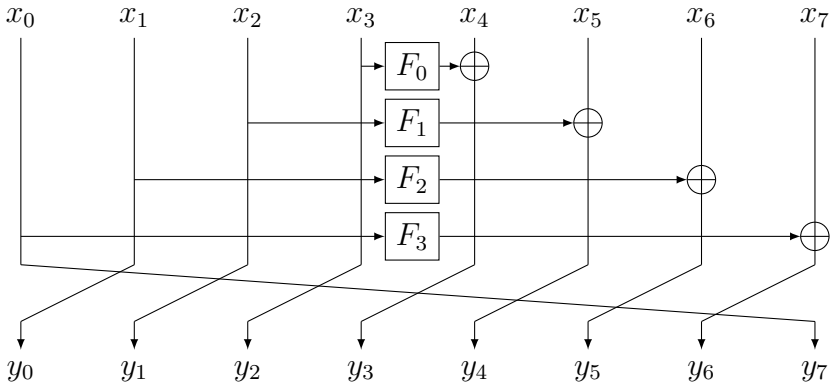


FIGURE 7.7 – Exemple de GFN de Nyberg avec $k = 8$ blocs.

Son caractère pseudo-aléatoire est étudié dans [SM10].

7.2 Délai de diffusion

À mesure que l'on augmente le nombre k de blocs d'un schéma de Feistel généralisé, on peut se demander, en fonction de k et du type de Feistel utilisé, combien de tours il

faut faire pour que le schéma soit sûr. C'est ce qu'essaye de capturer la notion de délai de diffusion (*full diffusion delay* en anglais) [SM10]. De manière informelle, il s'agit du nombre minimal de tours à effectuer pour que tous les blocs en sortie $y_0, \cdots, y_{k-1}$ dépendent de tous les blocs en entrée $x_0, \cdots, x_{k-1}$. Plus formellement, on définit dans un premier temps un graphe orienté associé au Feistel. Ce dernier permettra de donner une définition précise du délai de diffusion.

Définition 7.1. *Pour un GFN avec k blocs, le* graphe associé à ce GFN *est le graphe orienté dont l'ensemble des sommets est $\{0, \cdots, k-1\}$ et tel qu'il y a un arc du sommet i au sommet j si le bloc y_j dépend du bloc x_i (directement ou via une fonction de Feistel). Dans le cas où la dépendence se fait par une fonction de Feistel, on étiquette l'arc correspondant avec un symbole F pour le distinguer des arcs provenant de la couche permutation.*

De manière équivalente, ce graphe s'obtient en repliant les sorties d'un tour de Feistel sur les entrées ayant même indice.

Par exemple, le graphe associé au GFN défini à la figure 7.1 est donné à la figure 7.8.

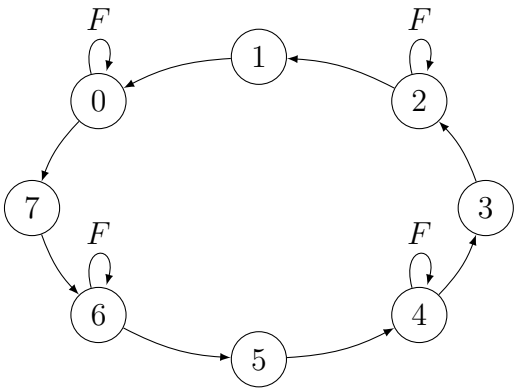


FIGURE 7.8 – Graphe associé au schéma de Feistel généralisé de la figure 7.1.

De manière informelle, on dit que l'entrée x_i influence la sortie y_j^r après r tours si x_i apparaît dans l'expression de y_j^r en tant que fonction de $x_0, \cdots, x_{k-1}$. On dit ensuite que le bloc x_i a diffusé au tour r si x_i influence tous les y_j pour $0 \leq j \leq k-1$. Lorsque tous les blocs x_i pour $0 \leq i \leq k-1$ ont diffusé au tour r, on dit que le GFN a atteint l'état de pleine diffusion (*full diffusion* en anglais). On appelle délai de diffusion le nombre minimal de tours nécessaires pour atteindre la pleine diffusion. On le note d^+. Plus formellement, on a la définition suivante du délai de diffusion.

Définition 7.2. *On dit que l'entrée x_i influence la sortie y_j^r après r tours s'il existe un chemin de longueur exactement r allant du sommet i au sommet j dans le graphe associé au GFN.*

Le délai de diffusion *d^+ est alors le plus petit entier r tel que pour tout couple de sommets (i, j), il existe un chemin de longueur exactement r allant de i à j.*

Si un tel entier n'existe pas, on pose $d^+ = +\infty$.

Deux choses importantes sont à noter. D'une part, si un GFN est en état de pleine diffusion au tour r alors il le restera les tours d'après. D'autre part, la notion de délai de diffusion est une propriété dite structurelle du schéma de Feistel généralisé car elle ne

dépend que de la manière dont sont réparties les fonctions de Feistel et non du contenu de ces fonctions.

De manière équivalente à ce qui vient d'être fait, on peut définir le graphe associé au Feistel en mode déchiffrement et définir ainsi le délai de diffusion inverse noté d^-. On considère dans cette thèse que le chiffrement *et* le déchiffrement sont d'égale importance, ainsi on se focalisera sur le délai de diffusion dans les deux sens d défini par $d = \max(d^+, d^-)$. Le délai de diffusion d des principaux types de GFN est donné par le tableau 7.1.

Pour des raisons de sécurité, il est nécessaire que d soit fini. En effet dans le cas contraire, quel que soit le nombre de tours r, il existe un bloc en entrée x_i et un bloc en sortie y_j^r tel que y_j^r ne dépende pas de x_i. Ce qui donne un distingueur trivial pour le schéma, puisqu'une telle propriété est improbable pour une fonction aléatoire.

Plus généralement, un schéma de Feistel généralisé avec un délai de diffusion petit aura tendance à mélanger les blocs plus rapidement, ainsi lors de la conception d'un algorithme utilisant un schéma de Feistel généralisé, on essaiera de minimiser ce délai de diffusion d.

Type	Multi-source	Multi-cible	Type 1	Type 2	Type 3	Nyberg
d	k	k	$(k-1)^2+1$	k	k	k

Tableau 7.1 – Délai de diffusion d (dans les deux sens) des différents types de GFN en fonction du nombre de blocs k.

7.3 Amélioration du délai de diffusion

Cette section résume les travaux de Suzaki et Minematsu à FSE 2010 [SM10] visant à améliorer la diffusion du Feistel de type 2 ainsi que ceux de Yanagihara et Iwata [YI13] concernant les autres types de Feistel.

7.3.1 Cas des Feistels de type 2

Le schéma de Feistel de type 2 tel que décrit à la section 7.1.2.4 se compose d'une couche non-linéaire dont les fonctions de Feistel sont réparties "un bloc sur deux" (i.e. du bloc i vers le bloc $i+1$ pour i pair) suivie d'un décalage circulaire des blocs comme sur la figure 7.5.

L'idée de Suzaki et Minematsu [SM10] est de relâcher cette définition en permettant n'importe quelle permutation des blocs en lieu et place du décalage circulaire. En choisissant bien cette dernière, ils montrent qu'il est possible d'améliorer la diffusion du Feistel de type 2.

Plus précisément, pour un nombre de blocs $2 \leq k \leq 16$, ils ont cherché les permutations qui minimisaient le délai de diffusion. Leurs résultats sont résumés au Tableau 7.2. La liste des permutations optimales se trouvent en Annexe A de [SM10].

De plus, lorsque k est une puissance de 2, ce qui est souvent le cas en pratique, ils donnent une construction générique dont le délai de diffusion est $2\log_2 k$, à comparer avec le type 2 classique (i.e. avec décalage circulaire) dont le délai de diffusion est k. À noter que cette valeur de $2\log_2 k$ est le minimum atteint par les permutations optimales trouvées expérimentalement pour $k \in \{2, 4, 8, 16\}$.

Finalement, ils font le lien entre le délai de diffusion et la résistance à certaines attaques.

k	2	4	6	8	10	12	14	16
classique	2	4	6	8	10	12	14	16
amélioré	2	4	5	6	7	8	8	8

Tableau 7.2 – Délai de diffusion des schémas de Feistel généralisés de type 2 améliorés par Suzaki et Minematsu [SM10] et comparaison avec le type 2 classique

7.3.2 Autres types de GFN

L'idée de Suzaki et Minematsu d'utiliser n'importe quelle permutation dans le cas du schéma de Feistel généralisé de type 2 est reprise par Yanagihara et Iwata [YI13] dans le cas des autres types de schémas de Feistel généralisés.

Dans les cas multi-source (cf. section 7.1.2.1) et multi-cible (cf. section 7.1.2.2), ils montrent que changer la permutation ne change pas le délai de diffusion dès lors que celui-ci est fini. En effet dans ces deux cas, un bloc joue un rôle particulier : récepteur unique dans le cas multi-source, émetteur unique dans le cas multi-cible. Ainsi aussi bien en chiffrement qu'en déchiffrement, toutes les entrées $x_0, \cdots, x_{k-1}$ doivent passer par ce bloc au moins une fois avant que le schéma n'atteigne la pleine diffusion, ce qui implique que la permutation des blocs doit être un cycle de longueur k. Par symétrie, ce schéma est équivalent au schéma classique avec permutation circulaire et donc $d = k$. Dans le cas contraire (permutation différente d'un cycle de longueur k), le schéma n'atteint jamais la pleine diffusion : $d = +\infty$.

Dans le cas du type 1 (cf. section 7.1.2.3) et pour $2 \leq k \leq 16$, Yanagihara et Iwata ont examiné les permutations qui minimisaient le délai de diffusion. Leurs résultats sont résumés au Tableau 7.3.

k	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
classique	2	5	10	17	26	37	50	65	82	101	122	145	170	197	226
amélioré	2	5	10	14	26	27	38	44	66	65	82	90	122	119	142

Tableau 7.3 – Délai de diffusion des schémas de Feistel généralisés de type 1 améliorés [YI13] et comparaison avec le type 1 classique.

Ils donnent aussi des constructions génériques de Feistel de type 1 minimisant le délai de diffusion suivant la valeur de k mod 4. La valeur du délai de diffusion dans chacun des cas est donnée au tableau 7.4. En comparant avec le tableau 7.3, on voit que les constructions génériques données sont optimales au moins jusqu'à $k = 16$. De plus, la dépendance du délai de diffusion reste quadratique en le nombre de blocs mais ces constructions gagnent asymptotiquement un facteur 2 par rapport au cas classique.

Finalement, dans le cas des Feistels de type 3 (cf. section 7.1.2.5) et pour $2 \leq k \leq 8$, ils ont examiné les permutations qui minimisaient le délai de diffusion et ont trouvé les résultats résumés au tableau 7.5. On voit ainsi que dans le cas du type 3 aussi, il est possible de faire mieux que le cas classique du décalage circulaire. On peut cependant se demander si la valeur $d = 4$ que l'on observe à partir de $k = 4$ est valable pour tout $k \geq 4$ ou bien s'il s'agit d'une croissance lente de d en fonction de k (e.g. logarithmique, voire plus lente).

amélioré				classique
$k \bmod 4$	0	$1, 3$	2	
d	$k(k+2)/2-2$	$k(k+1)/2-1$	$k(k+4)/2-4$	$(k-1)^2+1$

Tableau 7.4 – Délai de diffusion des constructions génériques des Feistels de type 1 donnés dans [YI13] et comparaison avec le type 1 classique.

Nombre de blocs	2	3	4	5	6	7	8
classique	2	3	4	5	6	7	8
amélioré	2	3	4	4	4	4	4

Tableau 7.5 – Délai de diffusion des schémas de Feistel généralisés de type 3 améliorés [YI13] et comparaison avec le type 3 classique

Chapitre 8

Approche matricielle

Ce chapitre présente les travaux effectués au cours de cette thèse concernant les schémas de Feistel généralisés : une vision générale de ces schémas en lien avec la notion de diffusion est présentée. Ce travail [BMT13], ainsi que le chapitre suivant, a été présenté et publié à SAC en 2013 en collaboration avec Thierry Berger et Marine Minier.

8.1 Représentation matricielle des GFN

Dans cette section, on donne d'abord une vision matricielle d'un schéma de Feistel généralisé puis dans un second temps, on caractérise les matrices qui correspondent à des schémas de Feistel généralisés.

8.1.1 Matrices d'un GFN

On rappelle que l'on note x_i les entrées d'un tour de schéma et y_j les sorties et qu'une itération d'un GFN se compose de deux étapes successives : la couche non-linéaire qui contient les fonctions de Feistel suivie de la couche permutation.

On sait représenter la couche permutation par une matrice : il s'agit de la matrice, que l'on note $\mathcal{P}$, de taille $k \times k$ à coefficients dans $\{0, 1\}$ associée à π^{-1} l'inverse de la permutation des blocs π. En d'autres termes $\mathcal{P}$ contient des 1 en position $(\pi(i), i)$.

Pour la couche non-linéaire, on définit sa matrice représentative de la manière suivante :

Définition 8.1. *Soit un GFN avec k blocs, la* matrice représentative *de la couche non-linéaire est la matrice $\mathcal{F}$ de taille $k \times k$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$ définie par :*

- *sa diagonale est tout à un,*
- *il y a un coefficient F en position (i, j) lorsqu'il y a une fonction de Feistel dont x_j est une entrée et dont la sortie est xorée avec (au moins) x_i.*

Le paramètre F dans la matrice représentative $\mathcal{F}$ de la couche non-linéaire correspond à l'étiquette F sur les arcs du graphe associé au GFN et sert à indiquer la présence d'une fonction de Feistel. La matrice associée au GFN peut alors être définie de la manière suivante :

Définition 8.2. *Soit un GFN dont la couche non-linéaire a pour matrice représentative $\mathcal{F}$ et la couche permutation a pour matrice $\mathcal{P}$, la matrice associée au GFN est la matrice $\mathcal{M} = \mathcal{P} \times \mathcal{F}$.*

Les matrices $\mathcal{P}$, $\mathcal{F}$ et $\mathcal{M}$ associées au GFN de la figure 7.1 sont données à la figure 8.1.

Le théorème suivant permet de faire le lien avec le graphe associé au GFN.

Théorème 8.1. *Soit un GFN de matrice $\mathcal{M}$, alors $\mathcal{M}$ est la transposée de la matrice d'adjacence du graphe associé au GFN.*

$$\mathcal{M} = \begin{pmatrix} F & 1 & & & & & & \\ & 1 & & & & & & \\ & F & 1 & & & & & \\ & & & 1 & & & & \\ & & & F & 1 & & & \\ & & & & & 1 & & \\ & & & & & F & 1 & \\ 1 & & & & & & & \end{pmatrix} \quad \mathcal{P} = \begin{pmatrix} & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ 1 & & & & & & & \end{pmatrix} \quad \mathcal{F} = \begin{pmatrix} 1 & & & & & & & \\ F & 1 & & & & & & \\ & & 1 & & & & & \\ & & F & 1 & & & & \\ & & & & 1 & & & \\ & & & & F & 1 & & \\ & & & & & & 1 & \\ & & & & & & F & 1 \end{pmatrix}$$

FIGURE 8.1 – Respectivement de gauche à droite, les matrices $\mathcal{M}$, $\mathcal{F}$ et $\mathcal{P}$ associées au GFN de type 2 de la figure 7.1

Démonstration. Comme $\mathcal{P}$ est une matrice de permutation, $\mathcal{M}$ n'est autre que la matice $\mathcal{F}$ dont on a réagencé les lignes. Les coefficients sur la diagonale de $\mathcal{F}$ vont en position $(\pi(i), i)$ ce qui correspond à une influence directe du bloc x_i sur le bloc $y_{\pi(i)}$. De même les coefficients F en position (i, j) dans $\mathcal{F}$ sont déplacés en position $(\pi(i), j)$ dans $\mathcal{M}$.

Ainsi si le coefficient en position (i, j) dans $\mathcal{M}$ vaut

- 1 lorsque $i = \pi(j)$, l'influence de x_j sur y_i se fait alors de manière directe via la couche permutation uniquement.
- F lorsqu'il y a une fonction de Feistel de x_j vers $x_{\pi^{-1}(i)}$, d'où une influence via une fonction de x_j vers y_i.
- 0 sinon, c'est-à-dire lorsque x_i n'influence pas y_j.

Pour conclure il suffit de remarquer que la transposée de la matrice d'adjacence d'un graphe orienté est la matrice d'adjacence du graphe dont on a inversé le sens des arcs. □

Ce théorème permet de voir que la vision graphique et la vision matricielle ne sont qu'une seule et même chose. Ainsi, les puissances de $\mathcal{M}$ ont une signification en terme de chemins dans le graphe donc en terme de diffusion de l'information dans le schéma. On peut alors caractériser le délai de diffusion d^+ en terme matriciel :

Théorème 8.2. *Soit un GFN de délai de diffusion (dans le sens direct) d^+ et de matrice associée $\mathcal{M}$ alors d^+ est le plus petit entier non nul tel que $\mathcal{M}^{d^+}$ n'ait que des coefficients non-nuls.*

Démonstration. Immédiat par définition de d^+ en terme de graphe et grâce aux propriétés des puissances des matrices d'adjacences. □

La vision matricielle est cependant plus puissante que la vision graphique. En effet, on peut multiplier des matrices différentes entre elles plutôt que de faire juste des élévations de puissance, ce qui revient à dire que les tours du schéma de Feistel peuvent ne pas tous être égaux. De plus, pour une matrice, cela a du sens de parler de l'inverse alors que ce n'est pas naturel dans le cas d'un graphe.

Dans tous les cas, cette représentation n'est pas une représentation exacte des schémas de Feistel généralisés. Cependant, elle contient assez d'information pour évaluer la diffusion, à savoir quelles entrées x_i influencent quelles sorties y_j et si cela est fait directement ou via une fonction de Feistel.

8.1.2 Équivalence de deux GFN

À présent que l'on dispose d'une représentation matricielle des GFN, on définit une notion d'équivalence en terme de schéma.

Définition 8.3. *Deux GFN sont équivalents si leurs graphes associés sont isomorphes. Matriciellement parlant, deux matrices de GFN $\mathcal{M}$ et $\mathcal{M}'$ sont équivalentes s'il existe une (matrice de) permutation π telle que $\pi\mathcal{M}\pi^{-1} = \mathcal{M}'$.*

En d'autres termes, deux GFN sont équivalents s'ils sont identiques à renumérotation des blocs près. Du point de vue de la diffusion, ils partagent donc les mêmes propriétés. De plus, on dispose de la propriété d'"équivalence de décompositions", qui dresse un lien entre l'équivalence de deux GFNs et l'équivalence de leurs deux couches respectives :

Théorème 8.3. *Soit $\mathcal{M} = \mathcal{P}\mathcal{F}$ et $\mathcal{M}' = \mathcal{P}'\mathcal{F}'$ deux GFN selon la définition 8.2 et équivalents. Soit π une permutation de la définition 8.3 qui lie $\mathcal{M}$ à $\mathcal{M}'$ par la relation $\pi\mathcal{M}\pi^{-1} = \mathcal{M}'$. Alors $\pi\mathcal{P}\pi^{-1} = \mathcal{P}'$ et $\pi\mathcal{F}\pi^{-1} = \mathcal{F}'$.*

Démonstration. Par hypothèse, on a $\pi\mathcal{P}\mathcal{F}\pi^{-1} = \mathcal{P}'\mathcal{F}'$. Par définition, $\mathcal{F}$ et $\mathcal{F}'$ ont une diagonale tout à un et des zéros ou des F ailleurs. Donc si on évalue $\mathcal{F}$ et $\mathcal{F}'$ en 0, on obtient la matrice identité. En spécifiant l'équation ci-dessus en 0, on a alors $\pi\mathcal{P}\pi^{-1} = \mathcal{P}'$. Ce qui implique $\pi\mathcal{F}\pi^{-1} = \mathcal{F}'$. □

En d'autres termes, deux GFN sont équivalents si et seulement si les deux couches qui les composent sont équivalentes entre elles et avec un élément conjugant π commum. Par exemple, si on étudie une famille de schémas ayant tous la même couche non-linéaire, comme c'est le cas dans [SM10, YI13], alors le théorème 8.3 permet de définir une relation d'équivalence sur la couche permutation.

8.1.3 Caractérisation des GFN

Parmi les matrices à coefficients dans $\{0, 1, F\}$, toutes ne correspondent pas forcément à un GFN. On aimerait donc caractériser les matrices qui correspondent à un GFN.

Une caractéristique importante des GFN est de transformer un ensemble de fonctions de Feistel en une permutation. Ainsi la matrice $\mathcal{M}$ doit être inversible et de plus son inverse ne doit pas contenir d'inverses d'expressions en F. En d'autres termes, on doit avoir $\det(\mathcal{M}) = \pm 1$, d'où $\det(\mathcal{F}) = \pm 1$. C'est une propriété que l'on peut observer pour toutes les classes de Feistels données en section 7.1.2 y compris leurs versions améliorées [SM10, YI13] car leurs matrices $\mathcal{F}$ sont triangulaires inférieurs avec une diagonale toute à un.

On se concentre dans un premier temps sur les Feistel qui vérifient la propriété supplémentaire que la couche non-linéaire est la même en chiffrement et en déchiffrement.

8.1.3.1 GFN quasi-involutifs

On définit ici un sous-ensemble des GFN caractérisés par la propriété suivante :

Définition 8.4. *Un GFN est dit quasi-involutif lorsque sa couche non-linéaire est une involution.*

Le terme quasi-involutif vient du fait que même si la couche non-linéaire est involutive, la couche permutation ne l'est en général pas. Tous les Feistel définis dans le chapitre 7 à l'exception des GFN de type 3 sont quasi-involutifs. En effet pour ces derniers, lors du déchiffrement les fonctions de Feistel doivent être évaluées séquentiellement, cf. section 7.1.2.5.

On donne la définition matricielle des GFN quasi-involutifs :

Définition 8.5. *Une matrice $\mathcal{M}$ de taille $k \times k$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$ est une matrice d'un schéma de Feistel généralisé quasi-involutif si elle peut s'écrire comme $\mathcal{M} = \mathcal{P}\mathcal{F}$ où $\mathcal{P}$ est une matrice de permutation et $\mathcal{F}$ vérifie les propriétés suivantes :*

1. *la diagonale est remplie de un,*
2. *les coefficients en dehors de la diagonale sont soit 0, soit F,*
3. *pour tout $0 \leq i \leq k-1$, la ligne i et la colonne i ne peuvent pas toutes deux contenir un coefficient F.*

Les points 1 et 2 signifient que la matrice $\mathcal{F}$ correspond à la couche non-linéaire du GFN en question. Le point 3 signifie que les blocs du GFN peuvent être partitionnés en trois catégories :

- les émetteurs dont le contenu est en entrée d'une fonction de Feistel,
- les récepteurs, qui reçoivent la sortie d'une fonction de Feistel,
- ceux qui n'émettent ni ne reçoivent.

Le caractère quasi-involutif de ces schémas de Feistel généralisés est montré dans le théorème suivant.

Théorème 8.4. *Soit $\mathcal{M} = \mathcal{P}\mathcal{F}$ un GFN quasi-involutif, alors $\mathcal{F}$ est inversible et $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$ où $\mathcal{I}$ désigne la matrice identité.*

Démonstration. Pour montrer que $\mathcal{F}$ est inversible, on calcule $\det(\mathcal{F})$. À cause du point 3 de la définition 8.5, pour chaque indice i, la ligne i ou la colonne i est entièrement nulle en-dehors du un sur la diagonale. De sorte qu'en développant le déterminant suivant soit la ligne i, soit la colonne i (suivant celle qui ne contient pas de F), et ce pour chaque $0 \leq i \leq k-1$, on obtient que $\det(\mathcal{F}) = 1$.

Pour prouver que $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$, on montre de manière équivalente que $(\mathcal{F} - \mathcal{I})^2 = 0$. Notons $f_{i,j}$, respectivement $f'_{i,j}$, le coefficient de $\mathcal{F} - \mathcal{I}$, respectivement $(\mathcal{F} - \mathcal{I})^2$, à la ligne i et colonne j. Par définition du produit matriciel, on a pour tout $0 \leq i, j \leq k-1$,

$$f'_{i,j} = f_{i,i}f_{i,j} + f_{i,j}f_{j,j} + \sum_{\substack{\ell \neq i \\ \ell \neq j}} f_{i,\ell}f_{\ell,j} = \sum_{\substack{\ell \neq i \\ \ell \neq j}} f_{i,\ell}f_{\ell,j}.$$

Considérons un terme $f_{i,\ell}f_{\ell,j}$ de la somme. Puisque $\ell \neq i$, $f_{i,\ell}$ ne peut être que zéro ou F. Mais si $f_{i,\ell} = F$ alors la colonne ℓ de $\mathcal{F}$ contient un F donc le point 3 implique que la ligne ℓ ne contient pas de F donc $f_{\ell,j} = 0$ pour tous les $j \neq \ell$. Ainsi tous les termes $f_{i,\ell}f_{\ell,j}$ sont nuls, d'où $f'_{i,j} = 0$ pour tout i et j. Ce qui implique que $(\mathcal{F} - \mathcal{I})^2 = 0$. □

Ce théorème montre bien que ces GFN sont quasi-involutifs. En effet, les matrices considérées sont à coefficients dans un anneau de caractéristique nulle pour qu'on puisse calculer des distances dans le graphe de Feistel. Ainsi lorsqu'on inverse l'addition des sorties

de fonctions de Feistel, on effectue la soustraction correspondante. D'où le terme $-\mathcal{F}$. Quant au terme $2\mathcal{I}$, il ne sert qu'à s'assurer que l'on a bien des uns sur la diagonale. Et si on revient en caractéristique 2, on retrouve bien $\mathcal{F}^{-1} = \mathcal{F}$.

Inversement, on sait caractériser les matrices $\mathcal{F}$ telles que $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$:

Théorème 8.5. *Soit $\mathcal{F}$ une matrice vérifiant les points 1 et 2 de la définition 8.5. Si $(\mathcal{F} - \mathcal{I})^2 = 0$ alors $\mathcal{F}$ vérifie aussi le point 3 de cette même définition.*

Démonstration. Notons $f_{i,j}$, le coefficient de $\mathcal{F} - \mathcal{I}$ en ligne i et colonne j. Pour tout $0 \leq i, j \leq k-1$, on a :

$$0 = \sum_{\ell=0}^{k-1} f_{i,\ell} f_{\ell,j} = \sum_{\substack{\ell \neq i \\ \ell \neq j}} f_{i,\ell} f_{\ell,j}$$

Tous les coefficients $f_{i,\ell}$ et $f_{\ell,j}$ sont en-dehors de la diagonale donc valent soit zéro, soit F. Donc la somme ne peut être nulle que si tous ses termes $f_{i,\ell} f_{\ell,j}$ sont nuls pour tout $0 \leq i, j \leq k-1$.

Pour tout $0 \leq \ell \leq k-1$, on doit prouver que la ligne ℓ et la colonne ℓ ne peuvent pas simultanément avoir un coefficient F. Supposons que la colonne ℓ contienne un F, disons $f_{i,\ell}$ avec $i \neq \ell$. Comme tous les $f_{i,\ell} f_{\ell,j}$ doivent être nuls, cela ne peut se produire que si $f_{\ell,j} = 0$ pour $j \neq \ell$. Donc la ligne ℓ ne contient pas de F.

Par transposition, on obtient le même résultat en inversant lignes et colonnes. □

On a ainsi complétement caractérisé les schémas de Feistel généralisés quasi-involutifs. Ce sont ceux pour lesquels aucun bloc n'apparaît à la fois en entrée et en sortie des fonctions de Feistel.

À l'aide de cette définition 8.5, on regarde les GFN quasi-involutifs à $k = 8$ blocs qui existent à équivalence près. On s'intéresse à trois paramètres :

- leur délai de diffusion d,
- le nombre Λ de fonctions de Feistel par tour (i.e. le nombre de F dans la matrice du GFN)
- le coût total pour diffuser c, qui est le nombre de fonctions de Feistel à évaluer pour atteindre la pleine diffusion. Par définition, on a $c = d \times \Lambda$.

Pour les valeurs de $d \leq 12$, le tableau 8.1 donne le plus petit s tel qu'il existe un GFN avec Λ fonctions de Feistel qui diffuse en d tours. Pour chaque valeur, il donne aussi le nombre de GFN qui atteignent une telle diffusion (ligne $\#\mathcal{M}$) en précisant le nombre de couches non-linéaires différentes possibles (ligne $\#\mathcal{F}$).

On remarque que le coût minimum est $c = 24$ et qu'il est en particuler atteint par les GFN de type 2 améliorés de [SM10] ($\Lambda = 4$ et $d = 6$) qui sont donc optimaux pour la diffusion au sein des GFN de la définition 8.5.

8.1.3.2 Cas général

Dans la section précédente, on a défini et caractérisé une classe de schémas de Feistel généralisés dont les blocs sont partitionnés entre émetteurs et récepteurs. Cependant, cette définition ne capture pas tous les schémas de Feistel généralisés existants puisque le type 3, cf. section 7.1.2.5, n'en fait pas partie. Parmi les façons possibles de placer les fonctions de

d	$1,2$	3	4	5	6	7	8	9	10	11	12
Λ	∞	16	7	6	4	4	4	3	3	3	2
c	∞	48	28	30	24	28	32	27	30	33	24
$\#\mathcal{F}$	0	1	1	8	3	13	13	1	6	6	1
$\#\mathcal{M}$	0	5	3	26	9	101	652	18	100	56	5

Tableau 8.1 – Nombre minimum Λ de fonctions de Feistel par tour requises pour diffuser en d tours et coût total $c = d \times \Lambda$ correspondant. Pour chaque cas, le nombre de matrices $\mathcal{F}$ (ligne $\#\mathcal{F}$) différentes possibles et le nombre total de GFN (ligne $\#\mathcal{M}$) sont aussi donnés à équivalence près.

Feistel, on va dans cette section caractériser celles qui donnent un schéma inversible dont l'inverse ne nécessite pas d'inverser ces fonctions de Feistel.

Pour résoudre ce problème, on définit une représentation graphique pour la couche non-linéaire du schéma comme on l'a déjà fait pour un tour de schéma tout entier. Par exemple, la figure 8.2 contient le graphe associé à la couche non-linéaire d'un Feistel de type-3 à $k = 8$ branches.

Définition 8.6. *Pour un GFN avec k blocs, le* graphe associé à la couche non-linéaire *est le graphe orienté dont l'ensemble des sommets est $\{0, \cdots, k-1\}$ et tel qu'il y a un arc du sommet i au sommet j s'il y a une fonction de Feistel dont le bloc x_i est une entrée et dont la sortie est xorée (au moins) au bloc x_j. En d'autres termes, il s'agit du graphe de matrice d'adjacence $\mathcal{F} - \mathcal{I}$.*

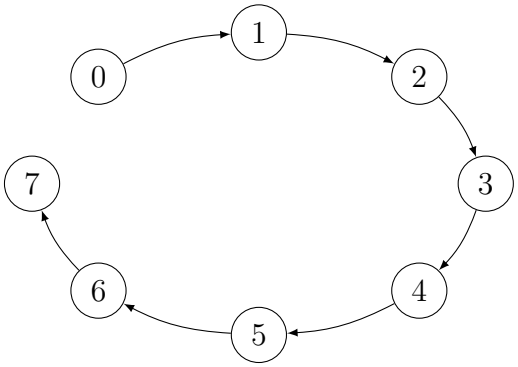


FIGURE 8.2 – Graphe associé à la couche non-linéaire d'un Feistel de type-3 à $k = 8$ branches (cf. section 7.1.2.5).

Rappelons que d'après la définition d'un schéma de Feistel généralisé si un des blocs x_j apparaît à la fois à l'entrée d'une fonction de Feistel et à la sortie d'une autre, on considère que la sortie de cette dernière est xorée *après* que le bloc x_j a servi d'entrée à la première fonction. En conséquence lors du déchiffrement, pour chaque arc (i, j) dans le graphe associé à la couche non-linéaire, la valeur du bloc x_i doit être connue pour pouvoir calculer celle du bloc x_j. S'il n'y a pas de fonction dont la sortie se fait sur x_i ou en termes de graphe s'il n'y a pas d'arc terminant sur le sommet i, la valeur du bloc x_i est directement connue. En revanche, s'il y a une fonction dont la sortie se fait sur x_i ou en termes de graphe s'il y a

un arc terminant sur le sommet i, cette fonction doit être évaluée d'abord avant de pouvoir évaluer celle entre x_i et x_j. En somme, le graphe associé à la couche non-linéaire d'un GFN représente l'ordre dans lequel doivent être calculés les blocs x_i en fonction des y_j lors du déchiffrement. On voit alors que cela n'est possible que lorsque ce graphe ne contient pas de cycle. Cela nous fournit donc un critère pour qu'une matrice $\mathcal{M} = \mathcal{P}\mathcal{F}$ corresponde à un schéma de Feistel généralisé : il faut que le graphe de matrice d'ajacence $\mathcal{F} - \mathcal{I}$ soit acyclique.

Une propriété des graphes orientés acycliques est que la relation $\preccurlyeq$ définie sur les sommets par $i \preccurlyeq j$ s'il existe un chemin allant du sommet i au sommet j est une relation d'ordre partielle. Il est alors possible d'étendre cet ordre en un ordre total sur l'ensemble des sommets $\{0, \cdots, k-1\}$ qui soit compatible avec $\preccurlyeq$. En d'autres termes, les sommets peuvent être ordonnés de telle manière que l'origine de chaque arc arrive plus tôt que sa destination. Un tel ordre s'obtient par exemple en effectuant un parcours en profondeur du graphe. Si les blocs d'un GFN sont réordonnés selon cet ordre alors la matrice de la couche non-linéaire $\mathcal{F}$ devient triangulaire inférieure. On est au final capable de définir ce qu'est une matrice de GFN de la manière suivante :

Définition 8.7. *Une matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$ est une matrice de GFN s'il est possible d'écrire $\mathcal{M} = \mathcal{P}\mathcal{F}$ où $\mathcal{P}$ est une matrice de permutation et $\mathcal{F}$ vérifie :*

1. *la diagonale principale est remplie de 1 ;*
2. *les coefficients en dehors de la diagonale sont soit 0 soit F ;*
3. *la matrice est triangulaire (inférieure) à permutation près,* i.e. *il existe une matrice de permutation π telle que $\pi\mathcal{F}\pi^{-1}$ soit triangulaire.*

Le théorème suivant est alors direct.

Théorème 8.6. *Soit une matrice de GFN $\mathcal{M}$ selon la définition 8.7 alors* $\det(\mathcal{M}) = \pm 1$.

On a ainsi totalement caractérisé les matrices qui correspondent à la couche non-linéaire d'un GFN : ce sont essentiellement les matrices triangulaires. Parmi ces matrices, on a en particulier isolé celles très utilisées en pratique qui sont les matrices quasi-involutives.

8.2 Profondeurs et Degrés de diffusion

Au chapitre 7, on a introduit la notion de délai de diffusion d'un schéma de Feistel généralisé comme le nombre minimum de tours au bout desquels tous les blocs en sortie dépendent de tous les blocs en entrée. Puis grâce à la représentation matricielle définie à la section 8.1, on a caractérisé ce délai de diffusion en termes matriciels. Dans cette section, on utilise cette représentation matricielle de manière plus poussée et on généralise la notion de délai de diffusion, en prenant en compte le nombre de fonctions de Feistel traversées.

8.2.1 Profondeurs de diffusion

Soit un schéma de Feistel généralisé à k branches représenté par sa matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$. On a vu à la section précédente que son délai de diffusion d^+ peut être défini comme le plus petit entier tel que $\mathcal{M}^{d^+}$ n'ait pas de coefficient nul. De

manière générale, si on considère la matrice $\mathcal{M}^r$ du Feistel après r tours, ses coefficients sont des polynômes en F. Le degré de ces polynômes a en fait une signification en terme de fonctions de Feistel traversées au cours du chiffrement.

Précisons dans un premier temps ce qu'on entend par fonction traversée.

Définition 8.8. *Soit un schéma de Feistel généralisé à k blocs, un nombre de tours $r \in \mathbb{N}$ et deux indices $i, j \in \{0, \cdots, k-1\}$. Notons $P^r_{i,j}$ l'ensemble des chemins (paths) de longueur r (dans le graphe associé au GFN) allant du bloc x_j vers le bloc y^r_i.*

Pour un chemin $p \in P^r_{i,j}$, on appelle nombre de fonctions traversées par le chemin p et on note $c(p)$, le nombre d'arêtes étiquetées avec un F que contient ce chemin.

On peut alors définir une notion de profondeur entre deux blocs comme étant le nombre maximal de fonctions qu'il est possible de traverser pour aller de l'un à l'autre.

Définition 8.9. *Avec les notations de la définition précédente, on appelle profondeur (depth) de diffusion au tour r du bloc x_j dans le bloc y^r_i et on note $\delta^r_{i,j}$, le nombre maximal de fonctions traversées pour aller de x_j à y^r_i, où le maximum est pris sur tous les chemins de $P^r_{i,j}$. En d'autres termes :*

$$\delta^r_{i,j} = \max \{c(p) \mid p \in P^r_{i,j}\}.$$

Faire augmenter la profondeur de diffusion entre tous les blocs rend le schéma de Feistel plus résistant à la cryptanalyse. En effet, chaque passage dans une fonction de Feistel ajoute sa petite dose de non-linéarité (i.e. confusion de Shannon). Ainsi des profondeurs de diffusion élevées rendent d'autant plus complexe la dépendance du chiffré $(y_0, \cdots, y_{k-1})$ en fonction du clair $(x_0, \cdots, x_{k-1})$.

Comme la profondeur de diffusion d'un bloc dans un autre est définie à l'aide de chemins dans le graphe associé au GFN, on s'attend à ce qu'il existe un lien entre cette notion de profondeur et la représentation matricielle du GFN. Ce lien est l'objet du théorème suivant.

Théorème 8.7. *Soit un schéma de Feistel généralisé à k blocs représenté par sa matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$. Pour $r \in \mathbb{N}$ et $i, j \in \{0, \cdots, k-1\}$, la profondeur de diffusion $\delta^r_{i,j}$ au tour r du bloc x_j dans le bloc y^r_i n'est autre que le degré du polynôme-coefficient se trouvant en ligne i colonne j de la matrice $\mathcal{M}^r$.*

Ainsi ce sont les degrés des coefficients de la matrice $\mathcal{M}^r$ qui sont importants. Afin de ne pas surcharger les formules, lorsqu'on donnera la valeur de $\mathcal{M}^r$, on ne donnera que les degrés de ses coefficients sous la forme suivante : F^δ en ligne i et colonne j signifie que la profondeur au tour r du bloc x_j dans le bloc y^r_i est δ. Par convention, $-\infty$ désignera le degré du polynôme nul mais on écrira 0 ou on mettra simplement un blanc ou un '.' (point) plutôt que $F^{-\infty}$, comme cela a été fait jusqu'à présent. De même, on écrira 1 au lieu de F^0. Par exemple, la figure 8.3 donne une matrice $\mathcal{M}$ d'un GFN et sa puissance quatrième.

8.2.2 Degrés de diffusion

À la section 8.1, on a caractérisé le délai de diffusion d^+ d'un schéma de Feistel généralisé comme étant le plus petit entier tel que $\mathcal{M}^{d^+}$ n'ait pas de coefficient nul. Avec l'idée de profondeur de diffusion que l'on vient de définir, il est possible de généraliser la notion de délai de diffusion en s'intéressant au tour où tous les coefficients de la matrice $\mathcal{M}^r$ sont au-dessus d'une certaine borne.

$$\mathcal{M} = \begin{pmatrix} . & . & F & 1 & . & . & . & . \\ 1 & . & . & . & . & . & . & . \\ . & . & . & . & . & . & F & 1 \\ . & . & . & . & 1 & . & . & . \\ . & . & . & . & F & 1 & . & . \\ . & . & . & . & . & . & 1 & . \\ F & 1 & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . & . \end{pmatrix} \qquad \mathcal{M}^4 = \begin{pmatrix} F^2 & . & F^4 & F^3 & F^2 & F & F^2 & F \\ F^3 & F^2 & F & . & F & 1 & . & . \\ F^2 & F & F^2 & F & F^2 & . & F^4 & F^3 \\ F & 1 & . & . & F^3 & F^2 & F & . \\ F^2 & F & F^2 & F & F^4 & F^3 & F^2 & . \\ . & . & F & 1 & F & . & F^3 & F^2 \\ F^4 & F^3 & F^2 & . & F^2 & F & F^2 & F \\ F & . & F^3 & F^2 & . & . & F & 1 \end{pmatrix}$$

FIGURE 8.3 – Exemple de notation de matrice $\mathcal{M}$ de GFN et sa puissance quatrième.

On définit dans un premier temps le degré minimal d'une matrice à coefficients polynomiaux.

Définition 8.10. *Soit une matrice A de taille $k \times k$ à coefficients dans $\mathbb{Z}[F]$. On appelle degré minimal de A et on note* $\min\deg(A)$*, le minimum des degrés des coefficients de A :*

$$\min\deg(A) = \min_{0 \leq i,j \leq k-1} \deg(A_{i,j})$$

où $\deg(A_{i,j})$ désigne le degré du coefficient en ligne i et colonne j de A.

On peut maintenant définir le ℓ-ième degré de diffusion d'un schéma de Feistel généralisé qui généralise la notion de délai de diffusion.

Définition 8.11. *Soit un schéma de Feistel généralisé à k blocs représenté par sa matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F\} \subset \mathbb{Z}[F]$. Pour un entier $\ell \in \mathbb{N}$, on appelle ℓ-ième degré de diffusion du schéma le plus petit entier d_ℓ^+, s'il existe, tel que :*

$$\min\deg(\mathcal{M}^{d_\ell^+}) \geq \ell.$$

Si un tel entier n'existe pas, on pose $d_\ell^+ = +\infty$.

On définit de même le ℓ-ième degré de diffusion inverse, noté d_ℓ^-, en considérant la matrice $\mathcal{M}^{-1}$ au lieu de $\mathcal{M}$ dans la définition précédente.

Comme dans le cas du délai de diffusion, on appelle ℓ-ième degré de diffusion dans les deux sens, l'entier $d_\ell = \max(d_\ell^+, d_\ell^-)$.

En termes moins matriciels, le ℓ-ième degré de diffusion d_ℓ^+ correspond au tour au bout duquel tous les blocs en sortie dépendent de tous les blocs en entrée via au moins ℓ fonctions de Feistel. Ainsi d_0^+ n'est autre que le délai de diffusion d^+ du schéma. Le premier degré de diffusion d_1^+, par exemple, correspond quant à lui au tour au bout duquel il n'y a plus d'influence purement linéaire des entrées sur les sorties.

Comme dans le cas du délai de diffusion, un bon schéma de Feistel généralisé essaiera d'avoir des degrés de diffusion les plus petits possibles afin de mélanger au mieux les blocs du schéma. Mais les degrés de diffusion permettent d'assurer la sécurité de l'algorithme plus finement que le délai de diffusion puisqu'ils tiennent compte du passage dans les fonctions de Feistel.

Chapitre 9

Schémas de Feistel généralisés étendus

Ce chapitre étend la représentation matricielle définie au chapitre précédent à une nouvelle classe de schémas de type Feistel appelée les schémas de Feistel généralisés étendus (Extended Generalized Feistel Network ou EGFN). Une famille particulière d'EGFN bien choisie est ensuite étudiée en profondeur. Ce travail [BMT13], ainsi que le chapitre précédant, a été présenté et publié à SAC en 2013 en collaboration avec Thierry Berger et Marine Minier.

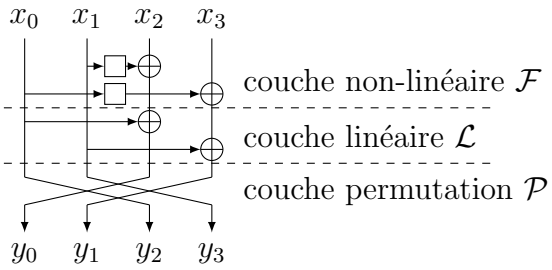
9.1 Définition et Représentation matricielle des EGFN

Considérons un schéma de Feistel généralisé représenté par sa matrice $\mathcal{M} = \mathcal{P}\mathcal{F}$ où $\mathcal{P}$ représente la couche permutation et $\mathcal{F}$ la couche non-linéaire. Si on veut améliorer sa diffusion, comme le montre le tableau 8.1 au chapitre précédent, on peut choisir une couche non-linéaire avec plus de fonctions de Feistel F. Mais cela signifie un schéma de Feistel généralisé plus coûteux. L'autre possibilité est de regarder la couche permutation $\mathcal{P}$. La définition 8.2 autorise déjà les permutations des blocs. Une généralisation possible est d'utiliser une application linéaire à la place et de considérer des GFN de la forme $\mathcal{M} = \mathcal{G}\mathcal{F}$ où $\mathcal{G}$ est une matrice $k \times k$ inversible. Il faut en revanche la choisir soigneusement si on veut conserver la propriété quasi-involutive du schéma. On propose alors de se restreindre aux applications $\mathcal{G}$ qui sont elle-mêmes des schémas de Feistel généralisés mais avec l'identité comme fonction de Feistel. En d'autres termes, on écrit $\mathcal{G} = \mathcal{P}\mathcal{L}$ où $\mathcal{P}$ est une matrice de permutation et $\mathcal{L}$ est une matrice similaire à la matrice $\mathcal{F}$ de la couche non-linéaire du GFN mais avec coefficients non-diagonaux et non-nuls I au lieu de F. On appelle cette matrice $\mathcal{L}$ la couche linéaire. Dans ce cas, le schéma de Feistel généralisé entier s'écrit $\mathcal{M} = \mathcal{P}\mathcal{L}\mathcal{F}$, voir par exemple la figure 9.1. Puisque les matrices $\mathcal{L}$ et $\mathcal{F}$ ont une structure commune, on les regroupe dans une même matrice $\mathcal{N} = \mathcal{L}\mathcal{F}$ et on écrit $\mathcal{M} = \mathcal{P}\mathcal{N}$. La matrice $\mathcal{N}$ représente la nouvelle partie du GFN contenant les fonctions de Feistel mais se compose désormais de deux paramètres formels : F pour les fonctions de Feistel non-linéaires pour garantir la sécurité cryptographique du schéma et I pour les fonctions de Feistel identités pour obtenir une diffusion rapide. On appelle ces nouveaux schémas des schémas de Feistel généralisés étendus (Extended Generalized Feistel networks en anglais, ou EGFN).

On se restreint aux EGFN quasi-involutifs.

Définition 9.1. *Une matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F, I\} \subset \mathbb{Z}[F, I]$ est la matrice d'un schéma de Feistel généralisé étendu quasi-involutif s'il existe une matrice de permutation $\mathcal{P}$ et une matrice $\mathcal{N}$ avec $\mathcal{M} = \mathcal{P}\mathcal{N}$ telle que :*

1. *la diagonale est remplie de un,*
2. *les coefficients non-nuls en dehors de la diagonale sont soit F, soit I,*
3. *pour tout $0 \leq i \leq k-1$, la ligne i et la colonne i ne peuvent pas toutes deux contenir un coefficient non-nul en dehors de la diagonale,*
4. *pour tout $0 \leq i \leq k-1$, si la ligne i contient un I alors elle contient aussi un F.*



$$\mathcal{M} = \begin{pmatrix} I & F & 1 & \\ F & I & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix} \quad \mathcal{P} = \begin{pmatrix} & & 1 & \\ & & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix}$$

$$\mathcal{L} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ I & & 1 & \\ & I & & 1 \end{pmatrix} \quad \mathcal{F} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & F & 1 \\ F & & & 1 \end{pmatrix}$$

FIGURE 9.1 – Exemple d'EGFN et matrices associées.

Comme dans le cas des GFN quasi-involutifs, la condition 3 revient à séparer les blocs en émetteurs et récepteurs. La condition 4 est utilisée dans la preuve de sécurité donnée à la section 9.3.1.

Comme la définition ci-dessus pour les EGFN est essentiellement la même que la définition 8.2 pour les GFN, on dispose du même théorème sur la quasi-involutivité.

Théorème 9.1. *Soit $\mathcal{M} = \mathcal{PF}$ un EGFN selon la définition 9.1, alors $\mathcal{F}$ est inversible et $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$ où $\mathcal{I}$ désigne la matrice identité.*

Démonstration. Idem que pour le théorème 8.4. □

À partir de la matrice $\mathcal{N}$ d'un EGFN, on définit les matrices de la couche non-linéaire $\mathcal{F}$ et de la couche linéaire $\mathcal{L}$ de la manière suivante.

Définition 9.2. *Soit $\mathcal{M} = \mathcal{PN}$ un EGFN comme défini par la définition 9.1. On définit la matrice de la couche non-linéaire $\mathcal{F} \in \mathbb{Z}[F]$ comme l'évaluation de $\mathcal{N}$ en $I = 0$ et, de manière similaire, la matrice de la couche linéaire $\mathcal{L} \in \mathbb{Z}[I]$ comme l'évaluation de $\mathcal{N}$ en $F = 0$.*

Le théorème suivant vérifie alors que cette définition correspond bien à ce que l'on souhaite, c'est-à-dire que $\mathcal{M} = \mathcal{PLF}$.

Théorème 9.2. *Soit $\mathcal{N}$, $\mathcal{L}$ et $\mathcal{F}$ trois matrices définies comme à la définition 9.2. Alors $\mathcal{N} = \mathcal{L} + \mathcal{F} - \mathcal{I}$ et $\mathcal{N} = \mathcal{L} \times \mathcal{F} = \mathcal{F} \times \mathcal{L}$.*

Démonstration. La première équation est une conséquence directe de la définition de $\mathcal{N}$, $\mathcal{L}$ et $\mathcal{F}$.

Pour la seconde, notons $a_{i,j}$ le coefficient en ligne i, colonne j de la matrice $\mathcal{LF}$ et montrons que $a_{i,i} = 1$ et que $a_{i,j} = \mathcal{L}_{i,j} + \mathcal{F}_{i,j}$ sinon (avec notations évidentes). On a :

$$a_{i,i} = \mathcal{L}_{i,i}\mathcal{F}_{i,i} + \sum_{\ell \neq i} \mathcal{L}_{i,\ell}\mathcal{F}_{\ell,i}.$$

Alors $a_{i,i} = \mathcal{L}_{i,i}\mathcal{F}_{i,i} = 1$ puisque la condition 3 de la définition 9.1 implique que tous les termes de la somme de droite sont nuls. Si $i \neq j$, on a :

$$a_{i,j} = \mathcal{L}_{i,i}\mathcal{F}_{i,j} + \mathcal{L}_{i,j}\mathcal{F}_{j,j} + \sum_{\substack{\ell \neq i \\ \ell \neq j}} \mathcal{L}_{i,\ell}\mathcal{F}_{\ell,j}.$$

Pour la même raison que précédemment, on a alors $a_{i,j} = \mathcal{F}_{i,j} + \mathcal{F}_{i,j}$. □

Finalement, la dernière propriété des GFN à généraliser aux EGFN est la relation d'équivalence discutée en section 8.1.2. La définition de l'équivalence de deux EGFN est la même que pour les GFN à la différence près que les EGFN possèdent des coefficients I.

En d'autres termes, une renumérotation des blocs qui morphe un EGFN en un autre échange les positions des F entre eux ainsi que la position des I entre eux, mais elle ne peut pas échanger un F avec un I. L'analogue du théorème 8.3 est alors immédiat.

Théorème 9.3. *Soit $\mathcal{M} = \mathcal{PLF}$ et $\mathcal{M}' = \mathcal{P}'\mathcal{L}'\mathcal{F}'$ deux EGFN selon la définition 9.1 et équivalents. Soit π une permutation telle que $\pi\mathcal{M}\pi^{-1} = \mathcal{M}'$. Alors $\pi\mathcal{P}\pi^{-1} = \mathcal{P}'$, $\pi\mathcal{L}\pi^{-1} = \mathcal{L}'$ et $\pi\mathcal{F}\pi^{-1} = \mathcal{F}'$.*

Démonstration. Idem que pour le théorème 8.3, en évaluant I ou F ou les deux en 0. □

9.2 Un Exemple intéressant

On donne dans cette section un exemple de famille d'EGFN avec un bon délai de diffusion et un coût raisonnable. Ces EGFN avec k blocs sont donnés par les schémas 9.2 et 9.3. Ses propriétés de diffusion sont l'objet du théorème 9.4. De plus la section 9.3 est dédiée à la sécurité de cette famille.

$$\mathcal{M} = \begin{pmatrix} & & & & F & 1 & & \\ & (0) & & F & I & & 1 & \\ & & \iddots & & I & & & \ddots \\ & F & & (0) & \vdots & & & 1 \\ F & I & I & \cdots & I & & & & 1 \\ 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & (0) & \\ & & & 1 & & & & \\ & & & & 1 & & & \end{pmatrix}$$

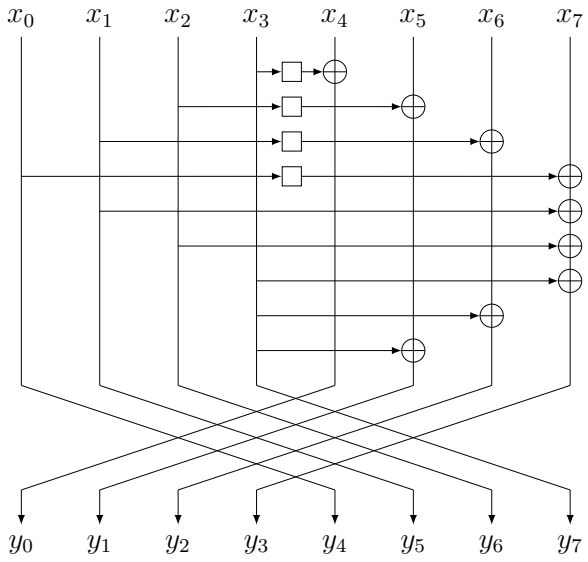


FIGURE 9.2 – À gauche, matrice $\mathcal{M}$ de l'EGFN avec $s = \frac{k}{2}$ fonctions de Feistel qui atteint la pleine diffusion en $d = 4$ tours et schéma correspondant à droite.

Comme le montre la figure 9.2, les k blocs du schéma sont divisés en deux groupes de $\frac{k}{2}$ blocs chacun : la moitié de gauche émet dans des fonctions tandis que la moitié de droite reçoit les sorties de ces fonctions. Plus précisément, les trois couches de cette famille d'EGFN sont les suivantes : la couche non-linéaire consiste à appliquer $\frac{k}{2}$ fonctions de Feistel sur la moitié des blocs qui émet et à xorer leurs sorties sur l'autre moitié des blocs de manière miroir par rapport à la verticale au milieu du schéma. La couche linéaire quant à elle consiste

à xorer certains blocs émetteurs avec certains récepteurs et elle permet d'atteindre la bonne diffusion telle que discutée dans le théorème 9.4. Plus précisément, tous les blocs émetteurs sont xorés sur le dernier bloc (i.e. x_{k-1}) et le dernier bloc émetteur (i.e. $x_{k/2-1}$) est xoré à tous les blocs récepteurs. Notons cependant que le bloc x_0 n'est pas xoré à x_{k-1} puisque x_0 influence déjà x_{k-1} dans la couche non-linéaire et qu'il est donc inutile du point de vue du délai de diffusion de le refaire dans la couche linéaire. Il en est d'ailleurs de même du bloc $x_{k/2-1}$ qui n'est pas xoré au bloc $x_{k/2}$. Finalement, la couche permutation échange les rôles des émetteurs et récepteurs via une rotation des blocs de $\frac{k}{2}$ positions.

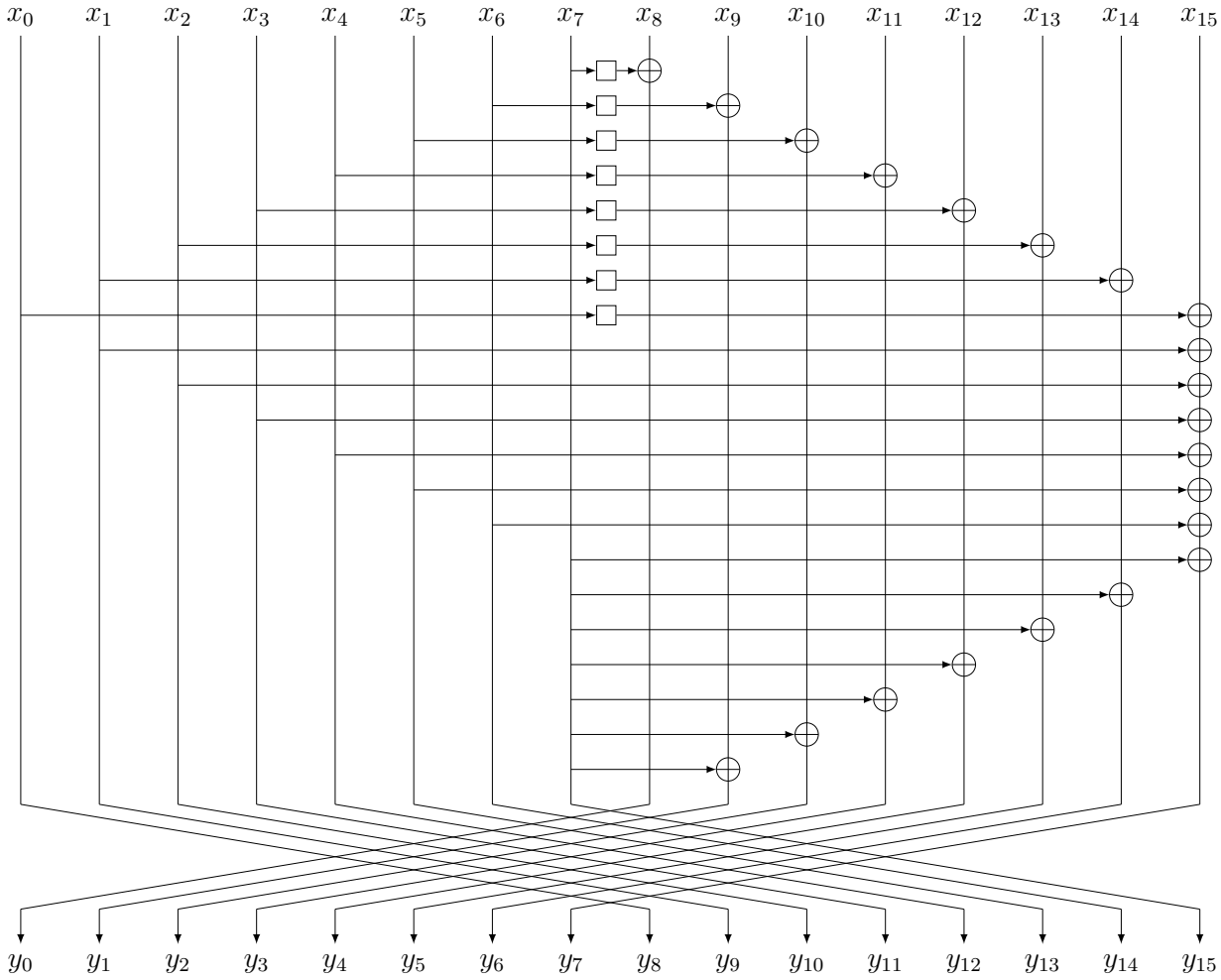


FIGURE 9.3 – EGFN avec $k = 16$ blocs et $s = 8$ fonctions de Feistel qui diffuse en $d = 4$ tours.

Cette famille existe pour tout nombre de blocs k pair. Son délai de diffusion fait l'objet du théorème suivant.

Théorème 9.4. *Pour un entier k pair, soit $\mathcal{M}$ l'EGFN défini ci-dessus et en figure 9.2 et soit d son délai de diffusion (dans les deux sens). Si $k = 2$ alors $d = 2$, et si $k \geq 4$ alors $d = 4$.*

Démonstration. Écrivons $\mathcal{M}$ par bloc :

$$\mathcal{M} = \begin{pmatrix} \mathcal{A} & \mathcal{I} \\ \mathcal{I} & 0 \end{pmatrix} \in \mathbb{Z}[F, I]$$

où $\mathcal{I}$ désigne la matrice identité de taille $\frac{k}{2}$ et le quart supérieur gauche de $\mathcal{M}$ est

$$\mathcal{A} = \begin{pmatrix} & & & & F \\ & (0) & & F & I \\ & & \iddots & & I \\ & F & & (0) & \vdots \\ F & I & I & \cdots & I \end{pmatrix}.$$

Remarquons dans un premier temps que A^2 ne possède pas de coefficient nul. En effet, un simple produit matriciel permet de voir que

$$\mathcal{A}^2 = \begin{pmatrix} F^2 & FI & \cdots & FI & FI \\ FI & F^2 + I^2 & & (I^2) & FI + I^2 \\ \vdots & & \ddots & & \vdots \\ FI & (I^2) & & F^2 + I^2 & FI + I^2 \\ FI & FI + I^2 & \cdots & FI + I^2 & F^2 + (\frac{k}{2} - 1)I^2 \end{pmatrix}.$$

On calcule ensuite le délai de diffusion direct d^+ qui est, par définition, la plus petite valeur positive telle que $\mathcal{M}^{d^+}$ n'ait pas de coefficient nul. On a

$$\mathcal{M}^2 = \begin{pmatrix} \mathcal{A}^2 + \mathcal{I} & \mathcal{A} \\ \mathcal{A} & \mathcal{I} \end{pmatrix}.$$

Ainsi si $k = 2$, alors la matrice $\mathcal{M}^2$ n'a pas de coefficients nuls, donc $d^+ = 2$ ce qui correspond au schéma de Feistel classique. En revanche si $k > 2$, il faut aller plus loin. En calculant $\mathcal{M}^3$ qui vaut

$$\mathcal{M}^3 = \begin{pmatrix} \mathcal{A}^3 + 2\mathcal{A} & \mathcal{A}^2 + \mathcal{I} \\ \mathcal{A}^2 + \mathcal{I} & \mathcal{A} \end{pmatrix},$$

on voit que $\mathcal{M}^3$ possède toujours des zéros puisque $\mathcal{A}$ en possède. On continue ainsi avec

$$\mathcal{M}^4 = \begin{pmatrix} \mathcal{A}^4 + 3\mathcal{A}^2 + \mathcal{I} & \mathcal{A}^3 + 2\mathcal{A} \\ \mathcal{A}^3 + 2\mathcal{A} & \mathcal{A}^2 + \mathcal{I} \end{pmatrix}.$$

On voit alors que $\mathcal{M}^4$ ne possède pas de coefficients nuls donc le délai de diffusion direct vaut $d^+ = 4$ lorsque $k \geq 4$.

Pour conclure, il suffit de remarquer que

$$\mathcal{M}^{-1} = \begin{pmatrix} 0 & \mathcal{I} \\ \mathcal{I} & -\mathcal{A} \end{pmatrix}$$

et que donc le raisonnment précédent sur $\mathcal{M}^{-1}$ va conduire à un délai de diffusion inverse

d^- égal au délai de diffusion direct d^+. On a donc au final

$$d = d^+ = d^- = \begin{cases} 2 & \text{si } k = 2, \\ 4 & \text{si } k \geq 4. \end{cases}$$

□

Ainsi grâce au théorème 9.4, nous disposons d'une famille d'EGFN avec $s = \frac{k}{2}$ fonctions de Feistel et un délai de diffusion de $d = 4$. On rappelle que le coût total c défini dans la section 8.1.3.1 est le nombre total d'appels à une fonction de Feistel nécessaires pour diffuser. Il vaut donc de manière générale $c = d \times s$. Dans le cas de notre famille, on a donc $c = 2k$.

En comparaison, les meilleurs schémas de Feistel généralisés connus auparavant sont les type 2 améliorés de Suzaki et Minematsu [SM10] qui font l'objet de la section 7.3.1 de l'état de l'art et dont le délai de diffusion est $d = 2\log_2 k$. Leur coût total est donc $c = k\log_2 k$. Lorsque $k > 4$, notre famille atteint la pleine diffusion à un coût moindre que le leur.

9.3 Évaluation de sécurité des schémas de Feistel généralisés étendus proposés

Comme fait par [SM10], on analyse à présent le schéma proposé dans la section précédente avec essentiellement $k = 8$ ou $k = 16$ blocs en termes d'indistinguabilité et en termes de résistance à certaines attaques classiques.

9.3.1 Preuve d'indistinguabilité

Comme on vient de définir une nouvelle structure de chiffrement par blocs, il est légitime de s'intéresser à l'avantage pseudo-random-permutation (prp-avantage) et à l'avantage strong-pseudo-random-permutation (sprp-avantage) d'un adversaire tel que fait dans de nombreux travaux [LR88, MV00, GM01] et présenté à la section 3.1. Pour cela, on rappelle les deux notations suivantes :

$$\mathrm{Adv}^{\mathsf{prp}}_{\mathsf{C}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CPA}} \left|\Pr[\mathsf{A}^{\mathsf{C}} = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n} = 1]\right| \tag{9.1}$$

$$\mathrm{Adv}^{\mathsf{sprp}}_{\mathsf{C}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CCA}} \left|\Pr[\mathsf{A}^{\mathsf{C},\mathsf{C}^{-1}} = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n,\mathsf{P}_n^{-1}} = 1]\right| \tag{9.2}$$

où C désigne la fonction de chiffrement d'un chiffrement par blocs n-bits composé de fonctions aléatoires uniformes (uniform random functions, URF) en tant de modules internes [LR88], C^{-1} est son inverse, P_n est une permutation aléatoire uniforme (uniform random permutation, URP) n-bits distribuée uniformément parmi toutes les permutations n-bits, et ${\mathsf{P}_n}^{-1}$ est son inverse. L'adversaire A essaye de distinguer C de P_n à l'aide de q requêtes à clairs choisis (chosen plaintext attack, CPA) et essaye de distinguer , toujours à l'aide de q requêtes, le couple $(\mathsf{C}, \mathsf{C}^{-1})$ du couple $(\mathsf{P}_n, {\mathsf{P}_n}^{-1})$ dans un modèle à chiffrés choisis (chosen ciphertext attack, CCA). On rappelle que ces notations signifient que la réponse finale de A est soit 0 s'il pense que les calculs sont faits à l'aide de P_n, soit 1 si A pense que les calculs sont faits à l'aide de C. Les maximums des équations (9.1,9.2) sont pris sur

tous les adversaires A avec q requêtes et une puissance de calcul non bornée. De nombreux résultats [LR88, MV00, GM01] évaluant la sécurité de variantes du schéma de Feistel dans ce modèle existent. Par exemple, Luby et Rackoff dans leur travail fondateur [LR88] (cf. section 3.1.3) ont prouvé la sécurité du schéma de Feistel classique $2n$-bits avec 3 tours dans le modèle prp et avec 4 tours dans le modèle sprp en considérant que le schéma de Feistel classique est composé de fonctions aléatoires uniformes n-bits vers n-bits. Les bornes qu'ils trouvent sont en $\mathcal{O}(q^2/2^n)$ dans les deux cas. Ces résultats initiaux ont été généralisés de nombreuses manières [ZMI89, Mau02].

Pour prouver les bornes sur notre schéma dans ces deux modèles, on suit la méthodologie introduite dans [SM10] basée sur les résultats de [MI08]. Pour cela, on introduit les notations suivantes : soit $\Phi_{kn,r}$ notre schéma à k blocs de n-bits chacun itéré sur r tours et soit d son délai de diffusion. On introduit dans un premier temps la définition suivante qui s'avèrera utile pour le lemme juste après.

Définition 9.3. *Soit H une permutation de $(\{0,1\}^n)^k$ dépendante d'une clé et soit $\boldsymbol{x} = (x_0, \cdots, x_{k-1}) \in (\{0,1\}^n)^k$ avec $\boldsymbol{x}_{[i]} = x_i$. On dit que H est une fonction ϵ-presque-universelle (ϵ-Almost-Universal, ϵ-AU) lorsque :*

$$\max_{\boldsymbol{x} \neq \boldsymbol{x'}} \Pr[H(\boldsymbol{x})_{[i]} = H(\boldsymbol{x'})_{[i]}, \text{ pour } i \in \{0, \cdots, k-1\}] \leq \epsilon.$$

Lemme 9.5. *Soit H et H' deux permutations de $(\{0,1\}^n)^k$ dépendantes d'une clé qui sont respectivement ϵ-AU et ϵ'-AU. Soit $\Phi_{kn,r}$ notre EGFN avec k blocs de n bits chacun sur r tours et avec délai de diffusion d où toutes les fonctions de Feistel n-bits sont des fonctions aléatoires uniformes indépendantes. On a alors :*

$$\mathrm{Adv}^{\mathsf{prp}}_{\Phi_{kn,2} \circ H}(q) \leq \left(\epsilon + \frac{k}{2^n}\right) \cdot \binom{q}{2}, \tag{9.3}$$

$$\mathrm{Adv}^{\mathsf{sprp}}_{H'^{-1} \circ \Phi_{kn,2} \circ H}(q) \leq \left(\epsilon + \epsilon' + \frac{k}{2^{n-1}}\right) \cdot \binom{q}{2}. \tag{9.4}$$

Démonstration. Intuitivement, pour l'équation (9.3), ce lemme utilise le fait qu'après l'application de H les entrées de $\Phi_{kn,2}$ sont suffisamment distinctes et sont des chaînes de bits aléatoires. Il y a alors de rares collisions possibles en sortie de $\Phi_{kn,2}$. Pour l'équation (9.4), les mêmes arguments sont utilisés dans les deux sens. La preuve de ce lemme n'est pas détaillée car elle est similaire à celle des théorèmes 3.1 et 3.2 de [NR99] ou est une extension directe du lemme 9 et du théorème 7 de [Mau02]. □

Théorème 9.6. *Étant donné $\Phi_{kn,r}$ notre EGFN avec k blocs de n bits chacun sur r tours et avec délai de diffusion d où toutes les fonctions de Feistel n-bits sont des fonctions aléatoires uniformes indépendantes, on a alors :*

$$\mathrm{Adv}^{\mathsf{prp}}_{\Phi_{kn,d+2}}(q) \leq \frac{kd}{2^n} q^2 \tag{9.5}$$

$$\mathrm{Adv}^{\mathsf{sprp}}_{\Phi_{kn,2d+2}}(q) \leq \frac{kd}{2^{n-1}} q^2 \tag{9.6}$$

Démonstration. Pour démontrer le théorème 9.6, on montre d'abord que $\Phi_{kn,d}$ est une fonction ϵ-AU et ensuite que ce que l'on note $\overline{\Phi_{kn,d}}$ et qui n'est autre que $\Phi^{-1}_{kn,d}$ sans la

permutation finale, est aussi une fonction ϵ-AU.

Montrons d'abord (comme fait dans [SM10]) que

$$\Pr[\Phi_{kn,d}(\mathbf{x})_{[i]} = \Phi_{kn,d}(\mathbf{x'})_{[i]}] \leq \frac{d}{2^n}, \text{ for all } i \in \{0, \cdots, k-1\} \tag{9.7}$$

On suppose que $(x_{k/2-1}, x_{k/2-2}, x_{k/2+1}) \neq (x'_{k/2-1}, x'_{k/2-2}, x'_{k/2+1})$, sans perte de généralité. On estime alors la probabilité que $\Phi_{kn,d}(\mathbf{x})_{[0]} = \Phi_{kn,d}(\mathbf{x'})_{[0]}$. Par définition du délai de diffusion d, il existe un chemin de longueur d dans le graphe du schéma commençant et finissant au sommet 0. Pour $h = 1, \cdots, d$, on peut définir la suite des états internes intermédiaires $Y_h = \Phi_{kn,h}(\mathbf{x})_{[s(h)]}$ suivant ce chemin. Il est alors facile de voir que

$$\Pr[Y_1 = Y'_1] = \Pr[F(x_{k/2-2}) \oplus x_{k/2-1} \oplus x_{k/2+1} = F(x'_{k/2-2}) \oplus x'_{k/2-1} \oplus x_{k/2+1}] \leq \frac{1}{2^n}$$

car la fonction de Feistel F est une fonction uniformément aléatoire. En utilisant le même raisonnement, ceci est aussi vrai pour les probabilités sur les autres blocs du schéma, y compris le bloc x_{k-1}, à cause de la présence d'une fonction de Feistel F. On regarde alors ce qui se passe au bout de d tours :

$$\Pr[Y_d = Y'_d] \leq \sum_{j=2}^{d} \Pr[Y_j = Y'_j | Y_{j-1} \neq Y'_{j-1}] + \Pr[Y_1 = Y'_1] \leq \frac{d}{2^n}$$

car toutes les fonctions de Feistel sont indépendantes, c'est-à-dire que $\Pr[Y_j = Y'_j | Y_{j-1} \neq Y'_{j-1}] \leq 1/2^n$. Ceci montre l'équation (9.7) et $\Phi_{kn,d}$ est ainsi une fonction $\frac{kd}{2^n}$-AU. L'équation (9.5) du théorème 9.6 est alors une conséquence directe de l'équation (9.3) du lemme 9.5.

Pour avoir la seconde équation du théorème, on utilise exactement le même raisonnement sur $\overline{\Phi_{kn,d}}$ pour montrer que $\Pr[Y_d = Y'_d] \leq d/2^n$ avec $Y_h = \overline{\Phi_{kn,h}}(\mathbf{x})_{[s(h)]}$ pour $h = 1, \cdots, d$. On en déduit alors que $\overline{\Phi_{kn,d}}$ est une fonction $\frac{kd}{2^n}$-AU. En combinant ceci avec le fait que $\Phi_{kn,d}$ est aussi une fonction $\frac{kd}{2^n}$-AU, on obtient l'équation (9.6) grâce à l'équation (9.4) du lemme 9.5. □

9.3.2 Résistance aux attaques classiques

On évalue à présent la sécurité de notre schéma face aux attaques dites structurelles que sont les attaques intégrales et différentielles impossibles. Ces attaques sont décrites de manière générale à la section 3.3. Enfin on dit deux mots sur les attaques différentielles et linéaires.

9.3.2.1 Attaque intégrale

Knudsen et Wagner [KW02] analysent la cryptanalyse intégrale comme un dual aux attaques différentielles particulièrement applicable aux chiffrements par blocs avec des composantes bijectives. On rappelle que dans une intégrale du premier ordre on considère une collection de m clairs qui diffèrent sur une composante particulière. Le but de cette attaque est alors de prédire la valeur des sommes (i.e. les intégrales) des chiffrés correspondants. Les mêmes auteurs généralisent aussi cette approche aux intégrales d'ordres supérieurs :

l'ensemble considéré devient un ensemble de $m\ell$ vecteurs qui diffèrent sur ℓ composantes et où la somme de cet ensemble est prédictible après un certain nombre de tours. La somme de cet ensemble est appelée une intégrale d'ordre ℓ. Dans [SW12], les auteurs améliorent les résultats antérieurs sur les schémas de Feistel en remarquant que les calculs des XOR-sommes des déchiffrements partiels peuvent être divisés en deux parties indépendantes via une approche meet-in-the-middle. On rappelle les notations suivantes sur un ensemble de 2^n mots de n bits :

- C (pour Constant) dans la i-ème entrée signifie que les valeurs des i-èmes mots dans la collection de textes sont égaux,
- P (pour Permutation) signifie que tous les mots dans la collection de textes sont différents,
- 0 signifie que la somme de tous les mots prise sur une certaine composante est nulle,
- ? signifie que la somme de tous les mots ne peut pas être prédite.

Les caractéristiques intégrales sont de la forme $(\alpha \to \beta)$ avec $\alpha \in \{C, P\}^k$ contenant au moins un P et $\beta \in \{C, P, 0, ?\}^k$ ne contenant pas que des ?. Pour trouver des caractéristiques intégrales, on applique la méthode et les propriétés décrites dans [BS01]. On regarde dans un premier temps les caractéristiques α contenant exactement un bloc A et des blocs C ailleurs. Par définition du délai de diffusion d, l'état après d tours ne contient plus de C. Si on suppose que l'état après d tours contient deux A sur les blocs les plus avantageux, disons i et j (par exemple les blocs aux indices $k/2-1$ et $k-1$), alors au tour d'après l'état du bloc $s = \mathcal{P}(j)$ devient un bloc $B = F(A) \oplus A$ ou bien $B = F(A) \oplus A \oplus A$ pour les transformations les plus simples, les autres transformations donnant immédiatement le même genre de résultat. Après un tour supplémentaire, le bloc $t = \mathcal{P}(s)$ est de la même forme puisqu'aucune fonction de Feistel n'a été rencontrée. Finalement, ajouter encore un tour transforme l'état en quelque chose de la forme $? = F(B) \oplus ?$ ou bien $? = F(B) \oplus B \oplus ?$ ou bien une expression plus compliquée pour y_1. Ainsi, une caractéristique intégrale contenant un A et $k-1$ C existe pour au plus $d+2$ tours. Si on essaie de l'étendre au début en une caractéristique d'ordre ℓ, on peut ajouter au plus d tours par définition de d. Ainsi le nombre maximal de tours que peut atteindre une intégrale d'ordre ℓ est $d+d+2 = 2d+2$ tours. On confirme expérimentalement cette borne en trouvant des caractéristiques intégrales du premier ordre pour au plus $d+2$ tours.

9.3.2.2 Différentielle impossible

Tandis que les attaques différentielles classiques s'intéressent à des différences qui se propagent dans le chiffrement avec la propabilité la plus grande possible, la cryptanalyse différentielle impossible exploite des différences avec une probabilité nulle sur l'un des tours intermédiaires du chiffrement dans le but d'éliminer les mauvaises clés candidates.

Plus formellement, les attaques différentielles impossibles sont représentées par une transition $\alpha \not\to \beta$ avec $\alpha, \beta \in (\{0,1\}^n)^k$ pour un chiffrement E avec k blocs de n bits chacun et tel que $\Pr[E(x) \oplus E(x \oplus \alpha) = \beta] = 0$, comme expliqué plus en détails à la section 3.3. Intuitivement, lorsqu'on veut former une différentielle impossible, on construit d'abord une différentielle sur r_1 tours de *chiffrement* entre une différence en entrée $\alpha^0 = (\alpha_0^0, \cdots, \alpha_{k-1}^0)$ et une différence après r_1 tours $\alpha^{r_1} = (\alpha_0^{r_1}, \cdots, \alpha_{k-1}^{r_1})$. Puis dans un second temps, on construit une différentielle sur r_2 tours de *déchiffrement* entre $\beta^0 = (\beta_0^0, \cdots, \beta_{k-1}^0)$ et $\beta^{r_2} = (\beta_0^{r_2}, \cdots, \beta_{k-1}^{r_2})$. On a alors une différentielle impossible $\alpha \not\to \beta$ sur $r_1 + r_2$ tours

si les différences α^{r_1} et β^{r_2} sont incompatibles au milieu du chiffrement.

Pour chercher ces différences, on utilise la $\mathcal{U}$-method de Kim *et al.* [KHL10] car elle s'applique aux chiffrements dont le message est divisé en plusieurs blocs. Les schémas de Feistel généralisés étendus sont donc une cible de choix pour cette méthode. Sur chacun des blocs, les différences $\alpha_i^{r_1}$ et $\beta_j^{r_2}$ peuvent être de cinq types :

- 0 lorsque la différence est nulle,
- γ lorsque la différence est fixée et non-nulle – les différences injectées en entrée de chiffrement sont de ce type,
- δ lorsque la valeur de la différence n'est pas fixée mais néanmoins non-nulle,
- $\gamma \oplus \delta$ lorsque la différence est le xor d'une différence de type γ avec une différence de type δ,
- t lorsque la différence n'est pas fixée (possiblement nulle).

Comme l'ont fait Suzaki et Minematsu [SM10], on peut utiliser cette méthode pour déterminer le nombre maximal de tours d'une différentielle impossible et l'exprimer en fonction du délai de diffusion d suivant différents cas :

- si α_i^d pour $i \in \{k/2, \cdots, k-1\}$ est de type γ, alors les seuls blocs de α^0 qui influencent α_i^d le font sans être jamais passés dans une fonction de Feistel F. Si de plus α_j^d pour $j \in \{0, \cdots, k/2-1\}$ est de type δ alors α_ℓ^{d+1} avec $\ell = \mathcal{P}(i)$ est de type $\delta \oplus \gamma$. Si finalement β_ℓ^d est de type γ, on est alors en mesure de construire une différentielle impossible sur $2d+1$ tours.
- Si tous les chemins de données sont passés par au moins une fonction de Feistel après d tours, alors ni α^d ni β^d ne contiennent de différences de type 0 ou γ. On ne peut dans ce cas monter des différences compatibles que sur au plus $d-1$ tours de chiffrement et d tours de déchiffrement (ou l'inverse). Le nombre maximal de tours de ces types de différentielles impossibles est donc $2d-1$ tours.

Finalement, une implémentation de la $\mathcal{U}$-method redonne les mêmes résultats : le nombre maximal de tours du schéma introduit à la section 9.2 pour lequel une différentielle impossible existe est $2d-2$, $2d-1$ ou $2d+1$.

9.3.2.3 Attaques différentielles et linéaires

Les cryptanalyses linéaires et différentielles sont parmi les attaques les plus connues contre les chiffrements par bloc. Elles ont été introduites respectivement par Matsui [Mat93] et par Biham et Shamir [BS90]. Depuis leurs découvertes, de nombreux travaux ont été faits pour établir un lien entre ces deux formes de cryptanalyse comme les travaux de Chabaud et Vaudenay [CV94] ou pour trouver de meilleurs manières de résister à de telles attaques comme ce qui a été fait pour le chiffrement AES [FIP01]. Le consensus le plus répandu à propos de ce dernier point consiste à compter le nombre minimal de S-box actives, c'est-à-dire de S-box traversées par les caractéristiques différentielles ou linéaires tout au long du chiffrement et à partir de cela estimer la probabilité différentielle ou linéaire maximale sur tout le chiffrement.

Si la probabilité différentielle ou linéaire d'une S-box est notée respectivement DP ou LP et si le nombre minimal de S-box actives est N alors la meilleure attaque respectivement différentielle ou linéaire contre l'algorithme a une complexité d'environ respectivement $1/DP^N$ ou $1/LP^N$. En conséquence, un chiffrement est considéré résistant à la cryptanalyse respectivement différentielle ou linéaire dès que ces quantités sont plus grandes que le

dictionnaire tout entier, soit 2^{kn} dans notre cas, où k est le nombre de blocs dans notre EGFN et n la taille des ces blocs.

Nous avons alors évalué le nombre minimal de S-boxes actives pour notre EGFN dans sa version $k = 16$ blocs (cf. figure 9.3) jusqu'à 20 tours et l'avons comparé avec la construction générique de Suzaki et Minematsu [SM10] qui diffuse en $d = 2 \log_2 k$.

Tour	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Type 2	0	1	2	3	4	6	7	9	10	13	15	17	19	22	24	26	28	32	35	39
[SM10]	0	1	2	3	4	6	8	11	14	19	21	24	25	27	30	31	33	36	37	39
EGFN 9.3	0	1	2	3	4	8	8	10	12	12	14	16	16	18	20	20	22	24	24	26

Tableau 9.1 – Nombre minimal de S-box actives sur un certain nombre de tours pour notre construction (cf. figure 9.3), celle de [SM10] ainsi qu'un GFN de type 2 classique, pour $k = 16$ blocs.

Comme on peut le voir au tableau 9.1 et comme noté par ses auteurs, la construction de [SM10] offre la même résistance aux attaques différentielles et linéaires malgré son meilleur délai de diffusion par rapport au GFN de type 2 classique. En ce qui concerne notre construction, on voit que sa résistance est en réalité bien moins bonne que pour les deux autres cas. Ceci peut en fait s'expliquer si l'on regarde cet EGFN sous un autre angle. La figure 9.4 montre le même EGFN qu'à la figure 9.3 mais après avoir replacé les blocs dans un certain ordre. On voit alors clairement que l'EGFN est composés de $k/4$ GFN reliés entre eux par des fonctions linéaires uniquement. On peut alors construire facilement un chemin différentiel qui reste sur un seul de ces GFN et n'active ainsi que peu de S-box, comme l'on fait Zhang et Wu [ZW14].

On touche ici à la limite de ce qu'on est capable de dire sur un schéma de chiffrement à l'aide du délai de diffusion. Car même si un délai de diffusion fini garantit que la sécurité augmente bien avec le nombre de tours et qu'on peut donc toujours se prémunir des attaques différentielles et linéaires quitte à effectuer suffisamment de tours, un délai de diffusion plus petit ne signifie pas toujours une meilleure sécurité vis à vis de ces attaques, comme déjà noté dans [SM10]. Cependant la résistance aux attaques structurelles comme les attaques intégrales ou les différentielles impossibles dépend plus directement du délai de diffusion et celles-ci sont donc bien écartées plus facilement par un délai de diffusion plus petit.

9.3.3 Nouveaux Exemples

À partir du constat que notre EGFN offre une résistance aux attaques différentielles et linéaires plus faible que les autres schémas existants, on va alors chercher à améliorer sa résistance face à ces attaques, tout en conservant sa résistance aux attaques structurelles. Pour un nombre k de blocs donné, nous n'avons pour l'instant donné en exemple qu'un seul EGFN diffusant en $d = 4$ tours. Comme fait par Suzaki et Minematsu [SM10] sur les GFN de type 2, nous choisissons de permettre n'importe quelle permutation des k blocs qui inverse émetteurs et récepteurs lors de la couche permutation pour généraliser notre famille d'EGFN.

Parmi ces permutations, on détermine dans un premier temps lesquelles engendrent un schéma qui diffuse en $d = 4$ tours. Puis on évalue la résistance aux cryptanalyses différentielles et linéaires de chacune d'elles.

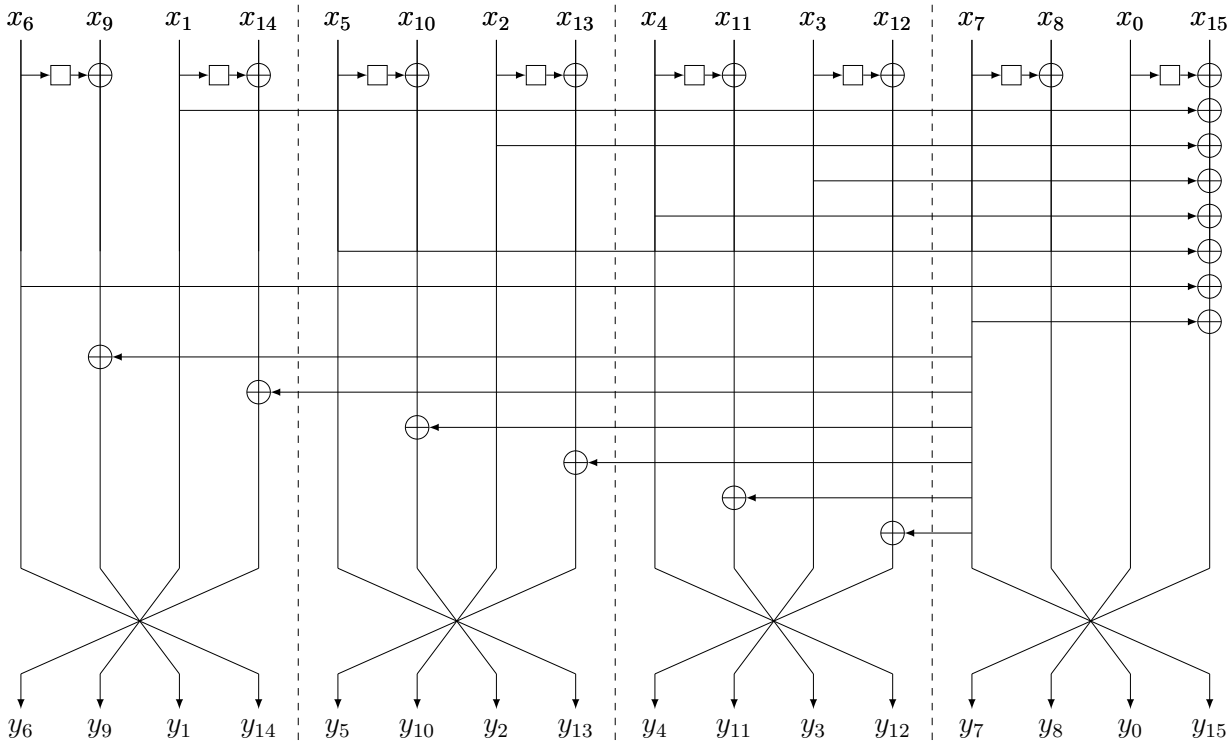


FIGURE 9.4 – Vision équivalente de l'EGFN donné à la figure 9.3.

Théorème 9.7. *Soit un EGFN comme défini à la section 9.2 avec $k \geq 4$ blocs mais avec une couche permutation $\mathcal{P}$ provenant d'une permutation $\pi \in S_k$ telle que $\pi(\{0, \cdots, k/2-1\}) = \{k/2, \cdots, k-1\}$ et $\pi(\{k/2, \cdots, k-1\}) = \{0, \cdots, k/2-1\}$. Soit d son délai de diffusion.*

Si $\pi(k/2-1) = k-1$ et $\pi(k-1) = k/2-1$ alors $d = 4$.

Démonstration. On adapte la démonstration du théorème 9.4 en prenant en compte la permutation. Soit la matrice par bloc $\mathcal{N} = \begin{pmatrix} \mathcal{I} & 0 \\ \mathcal{A} & \mathcal{I} \end{pmatrix} \in \mathbb{Z}[F, I]$ représentant les couches de fonctions (non-linéaires et linéaires) de l'EGFN où $\mathcal{I}$ désigne la matrice identité de taille $k/2$ et $\mathcal{A}$ est la matrice suivante :

$$\mathcal{A} = \begin{pmatrix} & & & & F \\ & (0) & & F & I \\ & & \cdots & & I \\ & F & & (0) & \vdots \\ F & I & I & \cdots & I \end{pmatrix}$$

Séparons la couche permutation $\mathcal{P}$ en deux parties puisque π inverse émetteurs et récepteurs :

$$\mathcal{P} = \begin{pmatrix} 0 & \mathcal{P}_1 \\ \mathcal{P}_2 & 0 \end{pmatrix}.$$

Alors la matrice $\mathcal{M}$ de l'EGFN est

$$\mathcal{M} = \begin{pmatrix} \mathcal{P}_1\mathcal{A} & \mathcal{P}_1 \\ \mathcal{P}_2 & 0 \end{pmatrix}.$$

Si $\mathcal{M}^3$ possède un coefficient nul et $\mathcal{M}^4$ non alors le délai de diffusion en chiffrement d^+ vaut 4. On a

$$\mathcal{M}^2 = \begin{pmatrix} (\mathcal{P}_1\mathcal{A})^2 + \mathcal{P}_1\mathcal{P}_2 & \mathcal{P}_1\mathcal{A}\mathcal{P}_1 \\ \mathcal{P}_2\mathcal{P}_1\mathcal{A} & \mathcal{P}_2\mathcal{P}_1 \end{pmatrix}$$

$$\mathcal{M}^3 = \begin{pmatrix} (\mathcal{P}_1\mathcal{A})^3 + \mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2 + \mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A} & (\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1 + \mathcal{P}_1\mathcal{P}_2\mathcal{P}_1 \\ \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^2 + \mathcal{P}_2\mathcal{P}_1\mathcal{P}_1\mathcal{P}_2 & \mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1 \end{pmatrix}$$

$$\mathcal{M}^4 = \begin{pmatrix} a & (\mathcal{P}_1\mathcal{A})^3 + \mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1 + \mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1 \\ \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^3 + \mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2 + (\mathcal{P}_2\mathcal{P}_1)^2\mathcal{A} & \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1 + (\mathcal{P}_2\mathcal{P}_1)^2 \end{pmatrix}$$

avec $a = (\mathcal{P}_1\mathcal{A})^4 + (\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1\mathcal{P}_2 + \mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A} + \mathcal{P}_2^2(\mathcal{P}_1\mathcal{A})^2 + \mathcal{P}_2^2\mathcal{P}_1\mathcal{P}_2$.

Comme la matrice $\mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1$ possède des zéros, $\mathcal{M}^3$ en possède aussi donc $d^+ \geq 4$. Si la matrice $(\mathcal{P}_1\mathcal{A})^2$ ne possède pas de zéro alors $\mathcal{M}^4$ non plus et dans ce cas $d^+ = 4$.

Or par hypothèse, la couche permutation $\mathcal{P}$ envoie le bloc $k-1$ sur le bloc $k/2-1$ donc la matrice $\mathcal{P}_1$ possède un '1' en dernière ligne et dernière colonne, c'est-à-dire :

$$\mathcal{P}_1 = \begin{pmatrix} & & & 0 \\ & \mathcal{P}_1' & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Et comme la dernière colonne et la dernière ligne de la matrice $\mathcal{A}$ n'ont pas de zéro, lorsqu'on calcule le produit $(\mathcal{P}_1\mathcal{A})^2$, on se retrouve avec des coefficients non-nuls partout.

Ainsi la condition $\pi(k-1) = k/2-1$ implique $d^+ = 4$. Pour conclure, on procède de même sur $\mathcal{M}^{-}1$, on s'intéresse cette fois-ci aux coefficients non-nuls de la matrice $(\mathcal{A}\mathcal{P}_2^{-1})^2$; la condition $\pi(k/2-1) = k-1$ implique alors $d^- = 4$. □

Grâce au théorème 9.7, on dispose d'un certain nombre d'EGFN diffusant en $d = 4$ tours. On se focalise sur les cas $k \in \{4, 8, 16\}$ blocs car ce sont les cas utilisés en pratique. On va alors évaluer la sécurité de ces schémas face aux attaques différentielles et linéaires en comptant le nombre minimal de S-box actives de 1 jusqu'à 20 tours. Caractérisons auparavant la relation d'équivalence sur la couche permutation découlant du fait qu'on a fixé les couches non-linéaires et linéaires.

Proposition 9.8. *Soit $\mathcal{M} = \mathcal{P}\mathcal{N}$ et $\mathcal{M}' = \mathcal{P}'\mathcal{N}$ deux EGFNs avec la matrice $\mathcal{N}$:*

$$\mathcal{N} = \begin{pmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & (0) & \\ & & & 1 & & & & & \\ & & & & 1 & & & & \\ & & & & F & 1 & & & \\ & (0) & & F & I & & 1 & & \\ & & \iddots & & I & & & \ddots & \\ & & & & \vdots & & & & \\ & F & & (0) & & & & 1 & \\ F & I & I & \cdots & I & & & & 1 \end{pmatrix}.$$

alors $\mathcal{P}$ et $\mathcal{P}'$ engendrent deux schémas équivalents si et seulement s'il existe $\pi \in S_k$ telle que :

1. $\mathcal{P} = \pi^{-1}\mathcal{P}'\pi$
2. $\pi(0) = 0$, $\pi(k/2-1) = k/2-1$, $\pi(k/2) = k/2$ *et* $\pi(k-1) = k-1$
3. $\forall i \in \{1, \cdots, k/2-2\}$, $\pi(k-1-i) = k-1-\pi(i)$.

Démonstration. Il s'agit d'une conséquence du théorème 9.3. Les permutations qui permettent de passer d'un schéma à l'autre globalement sont exactement celles qui transforment chaque couche du premier en la même couche du second. Comme dans notre cas, les couches non-linéaires et linéaires sont identiques, les seuls éléments de conjugaisons possibles sont ceux qui laissent invariant ces couches là. □

En d'autres termes, on ne change pas le schéma en intervertissant comme on le souhaite les blocs émetteurs intermédiaires $x_1, \cdots, x_{k/2-2}$ à condition d'intervertir les récepteurs correspondants $x_{k-2}, \cdots, x_{k/2+1}$ en conséquence. Le nombre n_k de schémas différents est donné au tableau 9.2 pour $k \in \{4, 8, 16\}$.

k	4	8	16
n_k	1	20	37108

Tableau 9.2 – Nombre n_k de nos EGFN différents pour $k \in \{4, 8, 16\}$ blocs.

On compte alors le nombre minimal de S-box actives sur r tours, pour $r \in \{1, \cdots, 20\}$. En prévision de la création du chiffrement léger LILLIPUT, on se focalise sur le cas $k = 16$ blocs. Le tableau 9.3 montre combien d'EGFN différents existent en fonction du nombre minimal de S-box actives au tour r.

On voit ainsi que, en fonction de la permutation que l'on choisit, la résistance aux attaques différentielles et linéaires varie fortement, presque du simple au double. Pourtant tous ces schémas ont un délai de diffusion $d = 4$. On remarque aussi que le schéma initialement créé à la section 9.2 est l'un des plus mauvais avec seulement 26 S-box actives après 20 tours, contre 42 pour le meilleur. Pour le schéma utilisé dans le chiffrement léger LILLIPUT décrit au prochain chapitre, on choisira un des 52 EGFN atteignant ce maximum. Enfin, il faut noter que les résultats dans le tableau 9.3 ont été obtenus sans réaliser d'instanciation des fonctions de Feistel. Il est donc possible que les chemins différentiels

S-box	Tour r																			
actives	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	37108	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	37108	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	37108	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	37108	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	5526	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	31582	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	6197	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	27212	3609	.	.	.	.	.	.	.	.	.	.	.	.	.
9	.	.	.	.	.	3572	8740	.	.	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	127	13359	4727	.	.	.	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	9792	8498	261	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	1604	8967	6239	3196	.	.	.	.	.	.	.	.	.	.
13	.	.	.	.	.	.	4	10542	5997	429	.	.	.	.	.	.	.	.	.	.
14	.	.	.	.	.	.	.	4372	7565	2947	3577	.	.	.	.	.	.	.	.	.
15	.	.	.	.	.	.	.	2	16504	5148	441	2	.	.	.	.	.	.	.	.
16	.	.	.	.	.	.	.	.	524	13195	2810	3665	3188	.	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	18	11481	4322	792	63	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	.	.	700	9034	2082	1428	3199	.	.	.	.	.	.
19	.	.	.	.	.	.	.	.	.	12	10113	2563	337	85	.	.	.	.	.	.
20	.	.	.	.	.	.	.	.	.	.	6692	6907	1724	1428	3519	3183	.	.	.	.
21	.	.	.	.	.	.	.	.	.	.	119	10826	2178	347	40	5	.	.	.	.
22	.	.	.	.	.	.	.	.	.	.	.	9373	5303	1550	1193	319	3196	.	.	.
23	.	.	.	.	.	.	.	.	.	.	.	898	9571	1959	346	442	54	.	.	.
24	.	.	.	.	.	.	.	.	.	.	.	.	10211	5447	2130	1232	1249	3379	3236	.
25	.	.	.	.	.	.	.	.	.	.	.	.	3076	7647	2045	353	131	66	.	.
26	.	.	.	.	.	.	.	.	.	.	.	.	29	9940	4273	1660	617	1097	283	3187
27	.	.	.	.	.	.	.	.	.	.	.	.	.	5373	7325	1770	276	103	7	2
28	.	.	.	.	.	.	.	.	.	.	.	.	.	133	11215	4228	1776	602	1047	337
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4765	8525	1870	391	149	418
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	257	9009	4067	1638	870	618
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5293	6060	1749	367	182
32	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1086	8549	3853	1523	860
33	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	6745	4875	1549	346
34	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2454	8299	3435	1557
35	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	64	6841	4370	1564
36	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3862	7600	3222
37	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	351	6544	3909
38	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	5171	6805
39	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	949	6382
40	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	8	6361
41	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1306
42	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	52

Tableau 9.3 – Pour $r \in \{1, \cdots, 20\}$, nombre d'EGFN différents en fonction du nombre minimal de S-box actives au tour r.

et linéaires que l'on a trouvé n'existent pas en pratique mais dans ce cas la résistance du schéma n'en est que meilleure.

Chapitre 10

Le chiffrement léger LILLIPUT

On présente dans ce chapitre, un chiffrement par bloc léger. Il est une application directe et pratique des travaux réalisés sur la représentation des GFN et des EGFN ainsi que de la diffusion. On donne dans un premier temps une description de l'algorithme. On justifie ensuite les choix de design de LILLIPUT, essentiellement par un argument de diffusion aussi bien au niveau du chiffrement qu'au niveau du cadencement de clés. Enfin, on s'intéresse aux résultats de l'implémentation de LILLIPUTen matériel. Ce travail, en collaboration avec Thierry Berger, Julien Francq et Marine Minier, a été soumis au journal IEEE Transactions on Computers comme version journal de l'article [BMT13] sur l'approche matricielle sur les schémas de Feistel généralisés (cf. chapitre 8) et les schémas de Feistel généralisés étendus (cf. chapitre 9).

10.1 Description de LILLIPUT

10.1.1 Processus de chiffrement

LILLIPUT est un schéma de chiffrement léger permettant de chiffrer des messages de 64 bits avec des clés de 80 bits. Le processus de chiffrement est résumé à la figure 10.1. Il utilise un schéma de Feistel généralisé étendu (EGFN) tel que défini au chapitre précédent avec un état interne de 64 bits et une fonction de tour agissant au niveau des nybbles. L'état interne, noté X, est vu comme 16 nybbles. Le chiffrement se compose de 30 tours, c'est-à-dire 30 répétitions d'un seul EGFN appelé `OneRoundEGFN` et représenté à la figure 10.2. Les 30 sous-clés de 32 bits RK^i sont générées à partir de la clé maître via le cadencement de clé.

La fonction de tour notée `OneRoundEGFN` sur la figure 10.1 se compose d'une couche d'applications non-linéaires dont le but est d'assurer la confusion, suivie d'une nouvelle couche constituée d'applications linéaires réalisée de façon Feistel et d'une permutation des blocs pour la diffusion. Comme expliqué au chapitre précédent, l'introduction d'une troisième couche permet de mélanger les blocs plus rapidement que ce qui était possible avec seulement la permutation des blocs tout en préservant l'auto-inversibilité du schéma. Dans le cas de LILLIPUT, ces trois couches sont respectivement nommées `NonLinearLayer`, `LinearLayer` et `PermutationLayer`, et constituent ensemble une itération de l'EGFN, comme montré sur la figure 10.2. Les trois couches agissent au niveau des nybbles sur l'état interne de l'EGFN X. Ces nybbles sont notés $X_{15}, \cdots, X_0$. Il est à noter que le dernier tour ne contient pas de `PermutationLayer` afin de préserver la symétrie du schéma.

De manière plus détaillée, `OneRoundEGFN` se compose de :

`NonLinearLayer :` il s'agit de la partie non-linéaire de l'EGFN. Elle est constituée de 8 mises à jour de l'état interne de celui-ci via la formule : $X_{8+j} \leftarrow X_{8+j} \oplus S(X_{7-j} \oplus RK^i_j)$, pour $j \in \{0, \cdots, 7\}$, où S est la S-box 4-bits donnée par le tableau 10.1 et RK^i_j est le j-ième nybble de RK^i.

`LinearLayer :` son but est de fournir une diffusion rapide entre les blocs et consiste à xorer

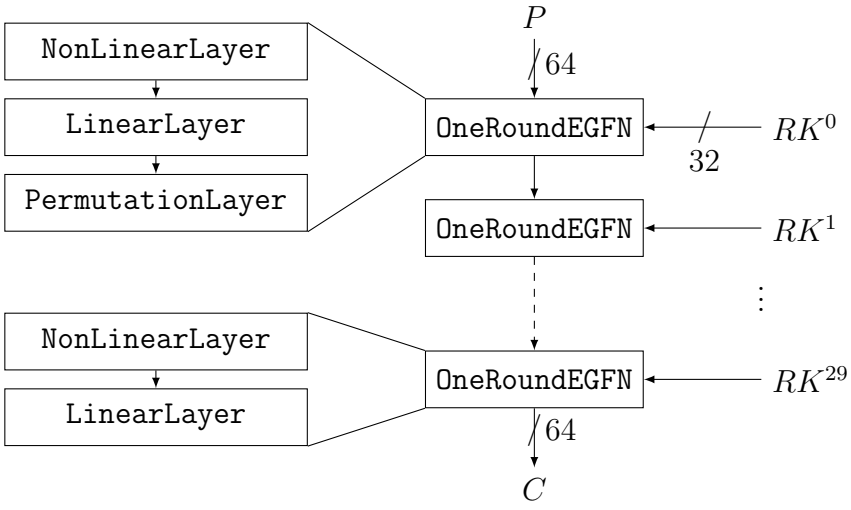


FIGURE 10.1 – Chiffrement LILLIPUT.

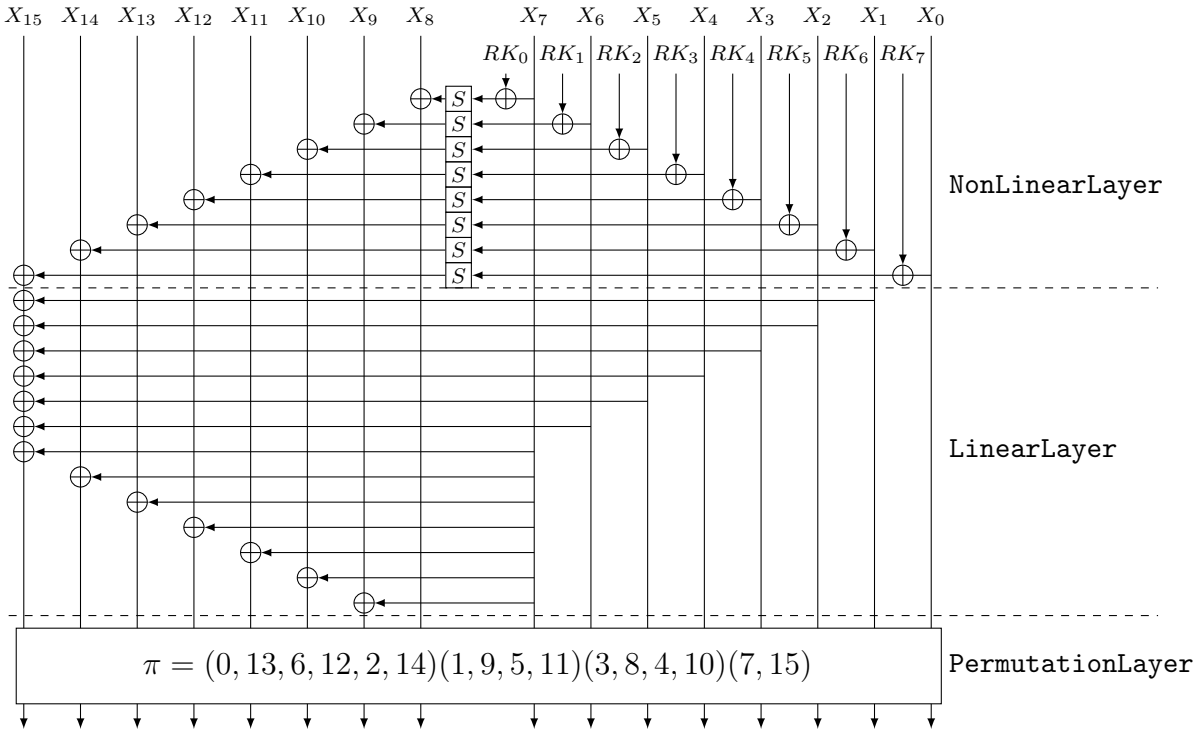


FIGURE 10.2 – OneRoundEGFN.

certains blocs avec d'autres. Plus précisément, comme montré sur la figure 10.2, les blocs X_1 à X_6 sont xorés au bloc X_{15} et le bloc X_7 est xoré aux blocs X_9 à $X15$.

`PermutationLayer :` elle consiste à appliquer la permutation π donnée par le tableau 10.2, c'est-à-dire que X_i devient $X_{\pi(i)}$ pour $i \in \{0, \cdots, 15\}$.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	4	8	7	1	9	3	2	E	0	B	6	F	A	5	D	C

Tableau 10.1 – S-box 4-bits utilisée dans LILLIPUT.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	13	9	14	8	10	11	12	15	4	5	3	1	2	6	0	7
$\pi^{-1}(i)$	14	11	12	10	8	9	13	15	3	1	4	5	6	0	2	7

Tableau 10.2 – Permutation des blocs π et son inverse.

10.1.2 Cadencement de clé

Le cadencement de clé représenté à la figure 10.3 produit les 30 sous-clés $RK^0, \cdots, RK^{29}$ à partir de la clé maître K et est concu pour permettre les calculs à la volée. Il utilise une machine à état fini linéaire (Linear Finite State Machine, LFSM). Son état interne est noté Y et est initialisé avec la clé maître K. Les sous-clés sont extraites de l'état interne Y à l'aide de `ExtractRoundKey` qui est une couche de S-box en parallèles. La sous-clé RK^0 est extraite de l'état initial de la LFSM (i.e. de la clé maître K). L'état interne Y de la LFSM est alors mis à jour via `RoundFnLFSM` puis la sous-clé suivante RK^1 est extraite ; et ainsi de suite jusqu'à RK^{29}.

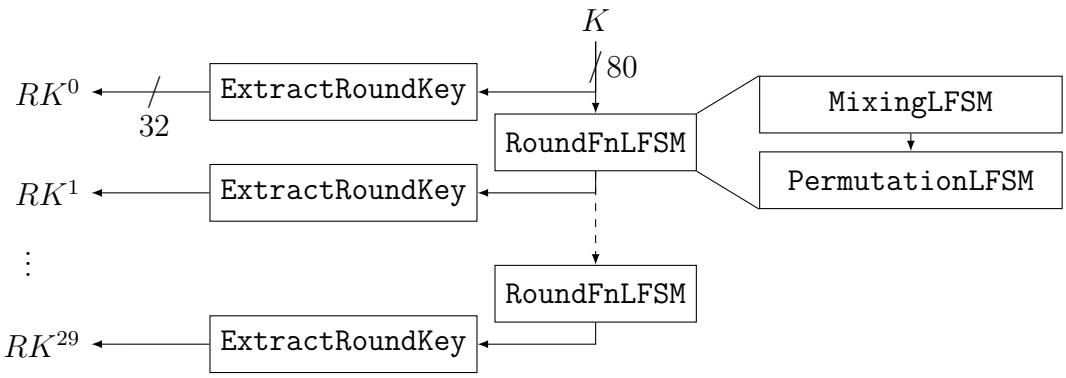


FIGURE 10.3 – Cadencement de clé de LILLIPUT.

L'état interne Y est constitué de 20 nybbles $Y_{19}, \cdots, Y_0$ répartis en 4 registres à décalage à rétroaction linéaire (Linear Feedback Shift Register, LFSR), $\mathcal{L}_0, \cdots, \mathcal{L}_3$, agissant sur 5 nybbles chacun : $\mathcal{L}_0$ agit sur $Y_0, \cdots, Y_4$, $\mathcal{L}_1$ sur $Y_5, \cdots, Y_9$ et ainsi de suite.

On utilise ici des LFSR s'inspirant des résultats de [BMP09] et [ABMP11] sur les LFSR. Comme décrit dans [BMP09], les 4 LFSR de LILLIPUT sont orientés nybble et vus comme des schémas de Feistel. Plus précisément, en utilisant la représentation matricielle introduite dans [BMP09], il est possible de construire des LFSR orientés mots dont les rétroactions se font au niveau des motsn et sont aussi choisies de manière à permettre une représentation Feistel généralisant ainsi les représentations classiques Fibonacci et Galois. De plus, les opérations utilisées pour calculer les rétroactions sont des opérations simples et orientées mots comme les shifts ou les rotations. Les deux principaux avantages de ces constructions sont d'une part une nette amélioration de la vitesse de diffusion et d'autre part, à cause de la construction Feistel, une inversion simple. Les 4 LFSR utilisés dans le cadencement de clé

de LILLIPUT sont des LFSR orientés mots agissant de manière Feistel sur 5 nybbles. Ainsi, la fonction `RoundFnLFSM` vue comme 4 de ces LFSR peut être divisée en deux transformations : `MixingLFSM` qui contient les rétroactions, suivie par `PermutationLFSM` qui est le décalage circulaire des nybbles, comme dépeint à la figure 10.4. Ceci est résumé par le figure 10.3. Une description précise de ces deux transformations est à présent donnée :

`MixingLFSM :` consiste, pour chacun des 4 LFSR $\mathcal{L}_0, \cdots, \mathcal{L}_3$, à xorer certains nybbles avec d'autres de manière Feistel :
- pour $\mathcal{L}_0$: $Y_0 \leftarrow Y_0 \oplus (Y_4 >>> 1)$ et $Y_1 \leftarrow Y_1 \oplus (Y_2 >> 3)$,
- pour $\mathcal{L}_1$: $Y_6 \leftarrow Y_6 \oplus (Y_7 << 3)$ et $Y_9 \leftarrow Y_9 \oplus (Y_8 <<< 1)$,
- pour $\mathcal{L}_2$: $Y_{11} \leftarrow Y_{11} \oplus (Y_{12} >>> 1)$ et $Y_{13} \leftarrow Y_{13} \oplus (Y_{12} >> 3)$,
- pour $\mathcal{L}_3$: $Y_{16} \leftarrow Y_{16} \oplus (Y_{15} << 3) \oplus (Y_{17} <<< 1)$.

`PermutationLFSM :` pour chacun des 4 LFSR $\mathcal{L}_0, \cdots, \mathcal{L}_3$, consiste en un décalage circulaire à gauche des ces 5 nybbles, c'est-à-dire $Y_i \leftarrow Y_{i-1 \bmod 5}$.

`ExtractRoundKey :` pour chaque tour de chiffrement, la sous-clé RK^i est extraite de l'état interne de la LFSM en utilisant une fonction d'extraction non-linéaire bitslicée. Dans un premier temps, certains nybbles de l'état interne sont extraits :
$Z_{(32)} \leftarrow Y_{18}||Y_{16}||Y_{13}||Y_{10}||Y_9||Y_6||Y_3||Y_1$.
Et si $Z_{31}, \cdots, Z_0$ désignent les bits de Z alors :
$RK^i_0 \leftarrow S(Z_0||Z_8||Z_{16}||Z_{24})$, $RK^i_1 \leftarrow S(Z_1||Z_9||Z_{17}||Z_{25})$,
$RK^i_2 \leftarrow S(Z_2||Z_{10}||Z_{18}||Z_{26})$, $RK^i_3 \leftarrow S(Z_3||Z_{11}||Z_{19}||Z_{27})$,
$RK^i_4 \leftarrow S(Z_4||Z_{12}||Z_{20}||Z_{28})$, $RK^i_5 \leftarrow S(Z_5||Z_{13}||Z_{21}||Z_{29})$,
$RK^i_6 \leftarrow S(Z_6||Z_{14}||Z_{22}||Z_{30})$, $RK^i_7 \leftarrow S(Z_7||Z_{15}||Z_{23}||Z_{31})$,
où S est la même S-box que dans le chemin de données. Finalement, le numéro du tour courant $i \in \{0, \cdots, 29\}$ est xoré aux 5 derniers bits de la sous-clé : $RK^i \leftarrow RK^i \oplus i_{(5)}||0_{(27)}$.

10.1.3 Processus de déchiffrement

Comme LILLIPUT est un schéma de Feistel, le déchiffrement est très similaire au chiffrement mais utilise la permutation des blocs inverse π^{-1} donnée par le tableau 10.2 ainsi que les sous-clés dans l'ordre inverse. Il est à noter que `NonLinearLayer` et `LinearLayer` commutent et peuvent donc être calculées dans n'importe quel ordre. Le processus complet de déchiffrement est dépeint à la figure 10.5. De plus, comme la LFSM utilisée dans le cadencement de clé est constituée de LFSR de type Feistel, le même argument permet de calculer les différentes sous-clés dans l'ordre inverse à partir de l'état interne Y^{29} obtenu en fin de chiffrement, comme le montre la figure 10.6.

10.2 Choix de l'EGFN

L'objectif principal lors du design de LILLIPUT était de maximiser la diffusion entre les blocs tout en gardant une implémentation avec des performances raisonnables. Des résultats récents [SM10, BMT13] suggèrent que cela peut être réalisé à l'aide de schémas de Feistel généralisés. Comme dans les chapitres précédents, on entend par diffusion le nombre minimum de tours requis pour que tous les blocs en sortie dépendent de tous les blocs en

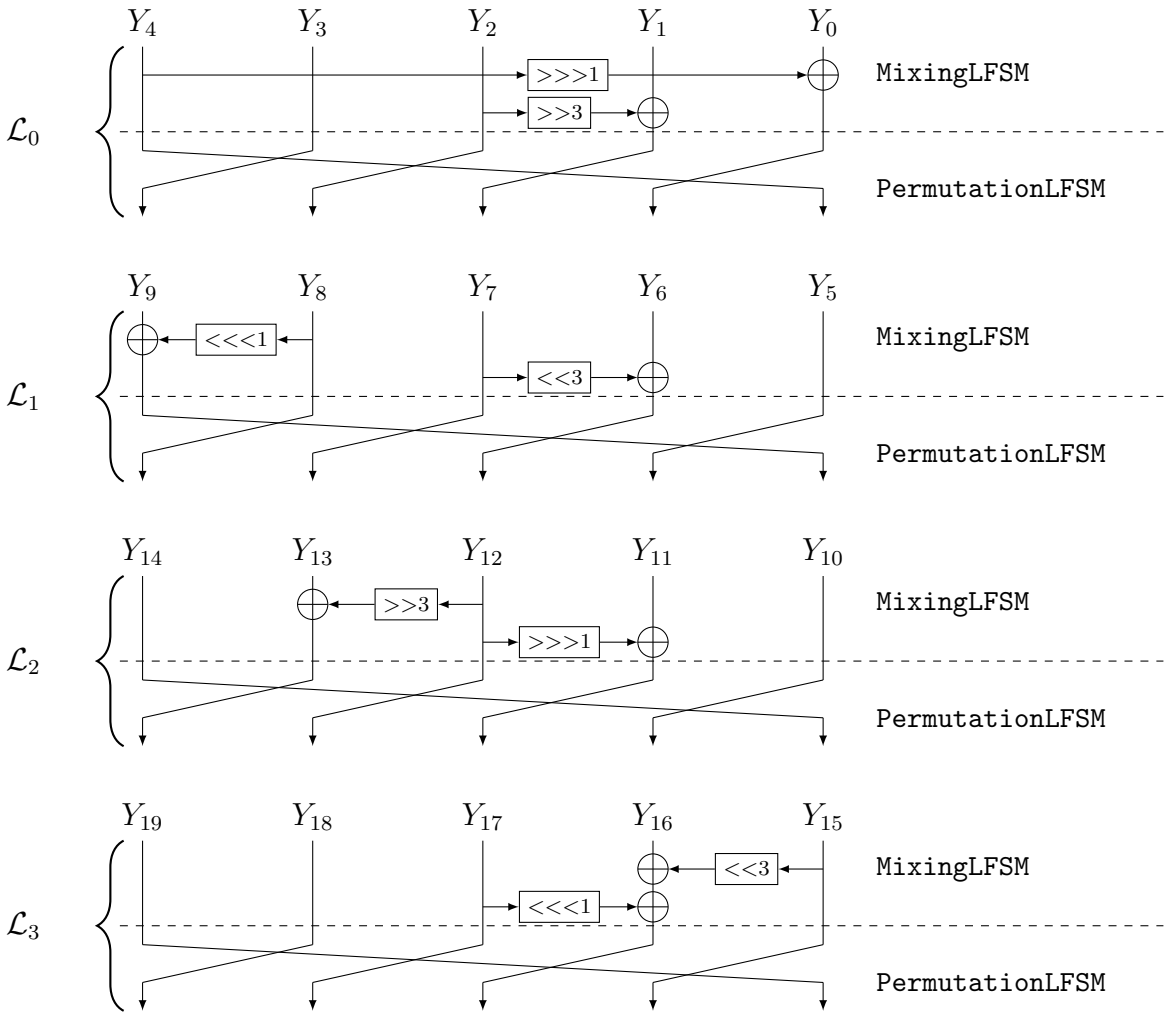


FIGURE 10.4 – LFSR $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$ et $\mathcal{L}_3$ du cadencement de clé de LILLIPUT.

entrée, que l'on nomme délai de diffusion et que l'on note d. Certaines attaques structurelles telles que les différentielles impossibles ou les attaques intégrales sont en effet fortement liées au délai de diffusion, comme montré à la section 9.3.1.

Pour minimiser le délai de diffusion, le choix s'est porté sur les schémas de Feistel généralisés étendus diffusant en $d = 4$ tours présentés au chapitre 9. On a choisi un état interne de 64 bits car cela est en accord avec l'objectif de cryptographie légère en limitant l'empreinte en mémoire. Cet état interne est ensuite divisé en 16 nybbles plutôt qu'en 8 octets afin que la taille des blocs soit égale celle de la S-box. Ainsi la couche non-linéaire se compose de 8 appels à une S-box 4 bits et de 8 additions de sous-clé.

Cependant un schéma de Feistel généralisé étendu à $k = 16$ avec délai de diffusion $d = 4$ est loin d'être unique. En reprenant pour le design de LILLIPUT les couches non-linéaires et linéaires de la famille présentée à la section 9.3.3, il existe 37108 couches permutations possibles avec $d = 4$. Pour chacune d'entre elles, on a calculé le nombre minimal de S-box actives en différentiel et en linéaire jusqu'à 20 tours (cf. tableau 9.3) et choisi la permutation qui maximise le nombre de S-box actives sur 18, 19 et 20 tours.

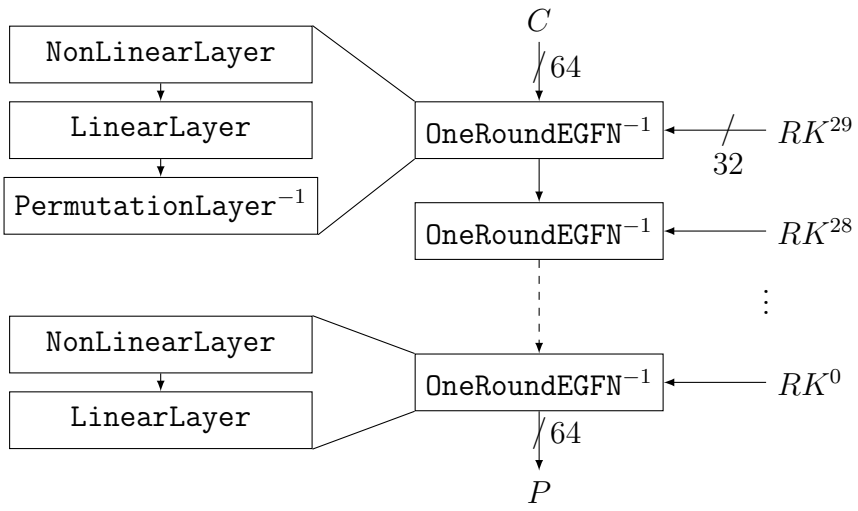


FIGURE 10.5 – Déchiffrement de LILLIPUT.

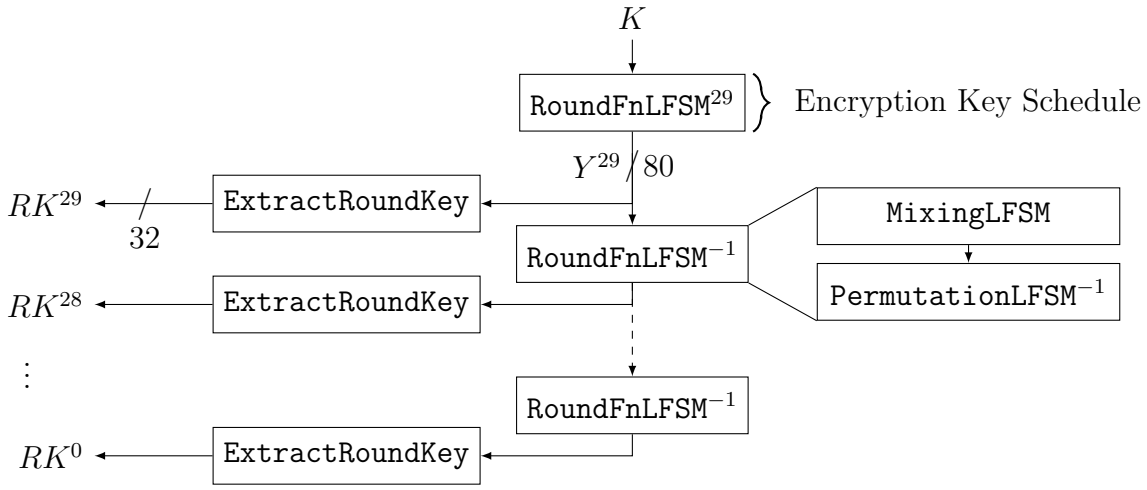


FIGURE 10.6 – Cadencement de clé pour le déchiffrement de LILLIPUT.

10.3 Choix de la S-box

LILLIPUT utilise une unique S-box comme seul composant non-linéaire. Son choix est donc d'une importance cruciale pour la sécurité et l'efficacité de LILLIPUT. Avec une future implémentation sérialisée matérielle optimisée en tête, on a décidé d'utiliser la même S-box 8 fois plutôt que d'avoir 8 S-box différentes. Cela permettra de n'implémenter qu'une seule S-box et de l'utiliser de manière intensive.

De nombreuses classifications [Can07, LP07, Saa11] des S-box 4-bits existent dans la littérature. Pour LILLIPUT, nous voulions que la S-box satisfasse les critères de sécurité suivants : les probabilités différentielles et biais linéaires maximums sont 2^{-2}, le degré algébrique est 3 et elle ne possède pas de point fixe. Nous avons sélectionné quatre telles S-box avec une expression algébrique simple. Pour choisir la S-box qui sera finalement implémentée, nous avons d'abord implémenté ces quatre S-box avec l'intention de choisir celle avec le plus faible coût en portes logiques. Il est apparu que ces quatres S-box nécessitent à peu près

la même aire, aux alentours de 23 GE (Gate Equivalent). Nous avons alors implémenté LILLIPUT avec chaque S-box pour voir lesquelles se combinent le mieux avec le reste du design. Il est finalement apparu que, parmi les quatre possibles, la S-box donnée par le tableau 10.1 induit le plus petit nombre de portes pour LILLIPUT et c'est ainsi elle que nous avons définitivement choisie.

10.4 Choix du Cadencement de clé

Comme cela est fait dans d'autres chiffrements légers tels que PRESENT ([BKL+07] et section 2.2.1.2), LBlock ([WZ11] et section 2.2.2.3), SIMON ([BSS+13] et section 2.2.2.6) ou TWINE ([SMMK12] et section 2.2.2.8), un registre de 80 bits contenant initialement la clé maître est mis à jour séquentiellement et les sous-clés sont extraites de ce registre. Cependant nous avons choisi de séparer la clé en quatre LFSR plus petits qui sont mis à jour en parallèle parce que des petits LFSR mélangent leur contenu plus rapidement et améliorent les performances de l'algorithme. L'inconvénient est que chaque LFSR pourrait être attaqué indépendamment si leurs contenus n'étaient pas recombinés lors de l'extraction de sous-clé, ce qui n'est pas le cas pour LILLIPUT. Ce design n'est pourtant pas entièrement nouveau puisqu'une chose similaire se passe dans le cadencement de clé du DES [Nat77].

10.4.1 Word-Ring-Feistel-LFSR

Pour le cadencement de clé de LILLIPUT, on utilise des LFSR inspirés des résultats de [BMP09] et [ABMP11] sur les LFSR. Dans [BMP09], les auteurs généralisent les LFSR au delà des représentations Fibonacci et Galois en permettant d'utiliser n'importe quelle cellule comme rétroaction sur n'importe quelle autre cellule. Ils appellent ces nouveaux LFSR "Ring-LFSR" à cause du décalage circulaire qui a lieu à chaque mise à jour.

Comme les LFSR de [ABMP11], les LFSR utilisés dans LILLIPUT possèdent aussi une structure orientée mot : au lieu de faire un décalage d'un bit à chaque itération et d'avoir des rétroactions binaires, ils sont décalés de tout un mot à la fois à chaque itération. Quant aux rétroactions, elles sont aussi orientées mot : un mot tout entier est xoré à un autre après éventuellement avoir été transformé par une opération logicielle simple tel que le shift ou la rotation. Ces LFSR sont appelés "Word-LFSR" par leurs auteurs [ABMP11].

Lorsqu'un LFSR est à la fois un Word-LFSR et un Ring-LFSR, on l'appelle un "Word-Ring-LFSR". Les Word-Ring-LFSR ont donc un délai de diffusion plus petit que les LFSR Fibonacci et Galois classiques. De plus, pour simplifier le cadencement de clé inverse, les LFSR utilisés dans LILLIPUT peuvent être représentés comme un schéma de Feistel généralisé (cf. la figure 10.4 pour des exemples) où les rétroactions jouent le rôle de fonction de Feistel. On appelle Word-Ring-Feistel-LFSR de tels LFSR.

10.4.2 Choix des LFSR

Les Word-Ring-Feistel-LFSR sont de bons candidats pour réaliser un cadencement de clé. En effet, le côté "Ring" permet de mélanger rapidement leur contenu, le côté "Word" permet une implémentation peu coûteuse et le côté "Feistel" permet d'implémenter le déchiffrement presque gratuitement en sus du chiffrement. On a donc recherché tous les Word-Ring-Feistel-LFSR agissant sur 5 nybbles vérifiant les conditions suivantes :

- les nybbles de rétroaction peuvent être décalés circulairement ou à perte avant d'être xorés,
- le polynôme de connection est primitif de degré 20 (pour produire des m-séquences),
- le nombre de portes XOR est minimal.

De ce fait, les LFSR choisis seront adaptés à la fois pour les implémentations matérielles et logicielles et peuvent être facilement inversés. Il se trouve qu'à renumérotation des nybbles près, il y a exactement seize tels LFSR, chacun avec un coût de 5 portes XOR. Cependant, en les regardant de plus près, on peut déduire ces 16 LFSR de seulement deux en appliquant certaines des 3 symétries/ propriétés suivantes :

Inverse pour la composition : 8 LFSR sont l'inverse pour la composition des 8 autres, c'est-à-dire inversent chiffrement et déchiffrement. Cela provient de la nature "Feistel" des LFSR étudiés.

Ordre des bits dans un nybble : 8 LFSR s'obtiennent à partir des 8 autres en inversant les rôles de la gauche et de la droite à l'intérieur des nybbles, par exemple $<< i$ devient $>> i$ et vice-versa.

Symétrie Galois/Fibonacci : 8 LFSR se déduisent des 8 autres via une transposition par rapport à la seconde diagonale. Cela généralise la dualité Galois/Fibonacci.

À partir d'un seul LFSR, ces trois symétries peuvent être appliquées indépendamment les unes des autres pour obtenir 8 LFSR différents satisfaisant toutes les conditions qui nous intéressent. Nos 16 LFSR peuvent donc être regroupés en deux groupes de 8 LFSR. Au sein de chaque groupe, chaque LFSR se déduit des autres en appliquant entre 0 et 3 des symétries ci-dessus.

Pour le cadencement de clé de Lilliput, nous devons choisir 4 LFSR parmi les 16 trouvés. Deux sont choisis dans chacun des deux groupes. De plus, les deux LFSR choisis dans chaque groupe sont choisis de manière à être opposés sur les trois symétries ci-dessus, *i.e.* on passe de l'un à l'autre en utilisant toutes ces symétries. Ce choix assure que des parties différentes de la clé sont traitées de manière différente par les LFSR, *i.e.* on casse les symétries dans le cadencement de clé.

En résumé, une structure faite de LFSR sous forme de schémas de Feistel a été choisie pour permettre les calculs à la volée aussi bien en chiffrement qu'en déchiffrement. De plus, les sous-clés peuvent être calculées dans l'ordre inverse à condition de fournir une clé de déchiffrement équivalente (cf. la figure 10.6).

10.4.3 Choix des points d'extraction

Pour l'opération `ExtractRoundKey`, les points d'extraction ont été choisis de manière à ce que trois sous-clés consécutives contiennent toute l'entropie de l'espace des clés (80 bits) et pour permettre une implémentation logicielle bitslicée.

10.5 Implémentation

Notre but lors de la création de Lilliput était de fournir un algorithme de chiffrement symétrique répondant aux exigences des environnements contraints ; en particulier dans le cas d'une authentification mutuelle à 3 passes entre le lecteur et le tag RFID telle que

définie dans la norme ISO 9798-2 [ISO08]. Dans un tel scenario, un protocole de challenge-réponse doit être implémenté où tant le chiffrement que le déchiffrement sont requis. De tels protocoles sont actuellement utilisés dans des produits industriels comme, par exemple, la famille DESFire de NXP. Comme souvent lorsqu'on parle de cryptographie légère sur tag RFID, la métrique principale à optimiser est la taille du circuit, exprimée en GE (cf. section 2.1.1).

On examine par la suite les avantages offerts par les choix de conception de LILLIPUT vis-à-vis de son implémentation. On donne ensuite une implémentation matérielle tour par tour (*round-based*) de LILLIPUT. Enfin, on compare ces résultats avec d'autres algorithmes de chiffrement légers possédant une taille de bloc et une taille de clé similaires à LILLIPUT.

10.5.1 Avantages de LILLIPUT

Le premier choix a avoir été fait fut celui de la structure généralle de LILLIPUT : SPN ou schéma de Feistel. Implémenter un schéma de Feistel apporte plusieurs avantages par rapport à un SPN en ce qui concerne la taille du circuit. D'abord, le chiffrement et le déchiffrement sont identiques à la permutation des blocs et à l'ordre des sous-clés près. Puisque l'on se place dans le cas où tant le chiffrement que le déchiffrement sont requis, le même circuit peut être utilisé pour les deux procédures. En comparaison, dans un SPN, le chiffrement et le déchiffrement doivent être implémentés séparément et ainsi très peu de ressources matérielles peuvent être partagées entre les deux. Un second avantage des schémas de Feistel par rapport aux SPN est que seulement la moitié de l'état interne de l'algorithme est modifiée à chaque tour. Par conséquent, on économise les ressources matérielles qui auraient été nécessaires pour modifier l'autre moitié.

Cependant, les schémas de Feistel possèdent en général deux inconvénients par rapport aux SPN. Le premier est que l'on a besoin de portes XOR pour recombiner avec l'autre moitié, la partie de l'état de l'algorithme qui a été modifiée par la fonction de Feistel. Dans le cas de LILLIPUT, il s'agit de 21×4 XOR, soit (sur la technologie ASIC que nous avons utilisée) $21 \times 4 \times 2.25 = 189$GE. Un second inconvénient est qu'en général les schémas de Feistel nécessitent plus de tours que les SPN pour obtenir un effet de diffusion comparable, ce qui introduit alors un surcoût en terme de débit et de consommation de courant. Il n'en est cependant pas ainsi pour LILLIPUT. En effet, une part importante de la conception de LILLIPUT a tournée autour du choix d'un (E)GFN optimal du point de vue de la diffusion, comme largement discuté au chapitre précédent. Au final, LILLIPUT ne comporte que 30 tours, ce qui est comparable à d'autres chiffrements comme le SPN PRESENT (31 tours) ou le GFN *Piccolo* (25 tours). Ainsi, nous n'avons qu'à prendre en compte le surcoût apporté par les 189GE de XOR ; on peut estimer que la contribution de ces 189GE sur la consommation électrique sera faible par rapport à celle du circuit complet.

Intéressons-nous à présent aux composantes élémentaires de LILLIPUT. En ce qui concerne la confusion de Shannon, LILLIPUT utilise des S-box 4-bits qui sont bien moins coûteuses que des S-box 8-bits ou 6-bits-vers-4-bits : la S-box de LILLIPUT coûte 23GE, comparé plusieurs douzaines pour les S-box 6-bits-vers-4-bits et plusieurs centaines pour les S-box 8-bits. De plus, elle a été choisie pour être un bon compromis entre la sécurité et le coût matériel. En ce qui concerne la diffusion, le choix d'un EGFN fait qu'elle ne coûte que les 189GE des XOR ainsi qu'une permutation des bits, ce qui est très efficace en matériel.

Les mémoires (flip-flops) utilisés pour stocker la clé et l'état interne sont les éléments les plus coûteux à implémenter en matériel et ceux qui consomment le plus d'énergie. Pour stocker les 64 bits d'état interne et les 80 bits de clé, nous avons besoin de $(64+80)\times 5.75 = 828$GE. Ce stockage est typiquement responsable d'au moins la moitié de la consommation de courant et de la surface du circuit.

Pour le cadencement de clé, nous avons choisi de le faire simple mais pas autant que ceux de KTANTAN [CDK09] et de PRINTCIPHER [KLPR10] pour le protéger contre les attaques à clés reliées [Bih93] et les attaques par glissement (*slide attacks*) [BW99]. Comme la même S-box est utilisée dans le chemin de donnée et dans le cadencement de clé, un partage de ressources entre ces deux parties est possible. Cependant, cela amène un surcoût dû aux multiplexeurs 2-vers-1 (2GE chacun) qu'il faut alors ajouter. De plus cela empêche de générer les sous-clés en parallèle du déroulement du chemin de données, ce qui augmente donc le nombre de cycles d'horloge nécessaires pour chiffrer. Ainsi nous avons décidé d'implémenter deux jeux complets de S-box : un pour le chemin de données et un pour le cadencement de clé.

Finalement, les différences entre le chiffrement et le déchiffrement sont minimes : seules les parties `PermutationLayer` et `PermutationLFSM` doivent être inversées. Ainsi le surcoût pour implémenter le déchiffrement en plus du chiffrement est négligeable.

10.5.2 Résultats d'implémentation

Nous donnons à présent les résulats d'une implémentation tour par tour de Lilliput. Elle traite les 64 bits de clair avec une clé de 80 bits en 30 cycles d'horloge. Les sous-clés sont calculées à la volée (*on-the-fly*) en parallèle du traitement de l'état interne de l'algorithme. Il n'y a pas de partage de ressources entre le chemin de données et le cadencement de clé. Les S-box sont implémentées sous forme de table (*lookup table*) et nous laissons le compilateur faire ses propres optimisations. Seul le chiffrement est implémenté.

Théoriquement, notre implémentation de Lilliput nécessite :

- 828GE pour stocker les sous-clés et l'état de l'algorithme,
- $16 \times 23 = 368$GE pour les S-box,
- $((21+8)\times 4 + 5\times 4 + 5)\times 2.25 = 317.25$GE pour tous les XOR,
- $(80+64)\times 2 = 288$GE pour les multiplexeurs 2-vers-1 qui choisissent entre la clé maître (resp. le clair) et la sous-clé (resp. l'état interne).

Au total, on peut estimer que notre implémentation tour par tour de Lilliput requiert au moins $828 + 368 + 317.25 + 288 = 1801.25$GE.

Nous avons implémenté en VHDL et synthétisé Lilliput à l'aide d'une bibliothèque *low-power high Vt 65 nm standart-cell*. Nous avons utilisé Synopsis Design Vision D-2010.03-SP5-2 pour la synthèse et la simulation de consommation ainsi que les valeurs de fonderie de 1.2V pour le voltage du cœur et de 25°C pour la température. En revanche, nous n'avons pas utilisé de *scan-flip-flops* pour les mémoires. Nous avons demandé à optimiser la surface du circuit.

Notre implémentation tour par tour, basse consommation, de Lilliput occupe 1832GE et possède une puissance simulée de 0.9 μW. Une comparaison avec d'autres chiffrement est données au tableau 10.3. Nous n'avons listé que des chiffrements par bloc avec une clé de 80 bits et des blocs de 64 bits, avec une implémentation matérielle respectant les contraintes des tags RFID HF, avec un débit raisonnable et pas d'attaque connue sur le cadencement de clé.

Lorsque cela est possible, nous donnons les résultats pour les circuits réalisant chiffrement et déchiffrement (comme TWINE). Les débits sont donnés avec une fréquence d'horloge de 100kHz.

	Latence (cycles)	**Débit (kbit/s)**	**Aire (GE)**	**Puissance (μW)**	**Logic Process**
PRESENT-80 [BKL+07]	31	200	1570	5	0.18μm
TWINE [SMMK12]	36	178	1799	N/A	90nm
LBlock [WZ11]	32	200	1320	N/A	0.18μm (theo.)
***Piccolo*-80** [SIH+11]	25	237	1274	N/A	0.13μm (theo.)
LILLIPUT	30	213	1832	0.9	LP 65nm

Tableau 10.3 – Comparaison de LILLIPUT avec d'autres chiffrements par bloc légers.

10.5.3 Comparaisons

En général, les comparaisons sont difficiles car les implémentations sont réalisées sur des technologies différentes et dans des conditions expérimentales différentes. On s'efforce néanmoins à présent de donner quelques éléments de comparaison entre LILLIPUT et les autres chiffrements par blocs donnés au tableau 10.3.

Tout d'abord en ce qui concerne le débit, à cause du nombre plus faible de tours, LILLIPUT est plus rapide que PRESENT-80, TWINE et LBlock mais plus lent que *Piccolo*-80.

En ce qui concerne la consommation électrique, les comparaisons sont très difficiles d'abord car nous utilisons une bibliothèsque ASIC basse consommation (*low-power*) contrairement aux autres implémentations, ensuite car les technologies CMOS utilisées dans les différents papiers ne sont pas les même et finalement car dans la plupart des cas aucun résultat de consommation de courant (ni moyenne, ni maximale) n'est donné par les auteurs des algorithmes. La seule choose que l'on peut dire est que notre implémentation consomme moins que celle de PRESENT-80 mais ce résultat semble principalement dû à l'utilisation d'un bibliothèque CMOS avancée.

Le reste de cette section se concentre désormais uniquement sur les comparaisons d'aire de circuit. En bref, les différences entre LILLIPUT et les autres chiffrements proviennent des points suivants : d'abord du surcoût du déchiffrement en sus du chiffrement, ensuite de notre désir de construire un cadencement de clé robuste et finalement des optimisations des circuits comme l'utilisation de scan-flip-flops au lieu d'une flip-flop et d'un multiplexeur. En détail, les comparaisons sont comme suit.

Comparé à PRESENT-80, on observe une différence de 262GE en notre défaveur. Mais ces résultats ne concernent que le chiffrement. Si l'on considère une implémentation avec chiffrement et déchiffrement, PRESENT nécessite l'implémentation de sa S-box et de son inverse ainsi que leur sélection par des multiplexeurs ce qui double la partie du circuit réservé aux S-box par rapport à une implémentation ne réalisant que le chiffrement. Si l'on considère que l'inverse de la S-box de PRESENT coûte autant que la S-box dans le sens direct, le surcoût du déchiffrement pour PRESENT est de $16 \times 28 = 448$GE. Dans le

cas de LILLIPUT, seuls les inverses de `PermutationLayer` et `PermutationLFSM` doivent être implémentés (uniquement du routage donc très peu coûteux) et on utilise alors des scan-flip-flops qui intègrent des multiplexeurs 2-vers-1. Ces derniers n'induisent qu'un surcoût de 0.25GE par bit de mémoire, soit au total $(64 + 80) \times 0.25 = 36$GE. En somme dans un scenario chiffrement/déchiffrement, LILLIPUT aura un nombre de portes plus faible que PRESENT avec $1570 + 448 = 2018$GE pour ce dernier contre $1832 + 36 = 1868$GE pour LILLIPUT.

Comparé à TWINE, les auteurs utilisent des scan-flip-flops qui leur permettent d'économiser 1GE par bit de mémoire. Dans notre bibliothèque, une flip-flop et un multiplexeur 2-vers-1 coûtent respectivement 5.75GE et 2GE tandis qu'une scan-flip-flop coûte 6GE, soit une économie de 1.75GE par bit de mémoire. Si nous avions utilisé des scan-flip-flops dans notre implémentation de LILLIPUT, on aurait pu économiser $(80+64)\times 1.75 = 252$GE au total. Ainsi LILLIPUT aurait un nombre de portes plus faible (1580GE) que TWINE (1799GE).

Comparé à LBlock, la valeur donnée (1320GE) est une estimation théorique et nous devons donc la comparer à notre propre estimation de 1801.25GE, soit une différence de 481.25GE en notre défaveur. Cette différence s'explique par la structure très simple du cadencement de clé de LBlock. À l'opposé, nous avons fait le choix d'un cadencement de clé plus complexe pour prévenir certaines propriétés indésirables comme celles montrées sur LBlock [BNS14].

À première vue, LILLIPUT ne joue pas dans le même catégorie que *Piccolo*-80 à cause de la nature très légère du cadencement de clé de ce dernier : il ne nécessite 32 multiplexeurs 3-vers-1 pour choisir la sous-clé appropriée. À nouveau, nous voulons pour LILLIPUT un cadencement de clé robuste afin de mieux résister aux attaques exploitant les propriétés du cadencement de clé [IS12]. De plus les auteurs de *Piccolo* utilisent des scan-flip-flops. Ainsi nous devons comparer 1580GE (LILLIPUT avec scan-flip-flops) avec 1274GE (*Piccolo* avec scan-flip-flops). Un autre facteur pour expliquer cette différence entre les deux implémentations est que les résultats de *Piccolo* ne contiennt pas les registres pour stocker la clé. Si ces registres sont requis alors environ 360GE supplémentaires sont nécessaires pour *Piccolo*-80[1]. L'aire de *Piccolo* devient alors dans ce cas $1274 + 360 = 1634$GE. Finalement, les auteurs de *Piccolo* infèrent des portes AND-NOR pour optimiser le nombre de portes XOR/XNOR, ce qui leur permet d'économiser 0.25GE par XOR, soit $((21+8)\times 4+5\times 4+5)\times 0.25 = 141\times 0.25 = 35.25$GE au total pour LILLIPUT. Au final, dans des conditions similaires, l'implémentation de LILLIPUT revient à $1580-35.25 = 1544.75$GE contre 1634GE pour *Piccolo*-80.

En résumé, LILLIPUT est plus petit que la plupart de ses concurrents lorsque chiffrement et déchiffrement sont requis, comme dans un scenario d'authentification mutuelle. Dans le cas d'une authentification simple où seul le chiffrement est requis, LILLIPUT n'est battu que par PRESENT-80 et LBlock (mais ce dernier possède certaines propriétés indésirables dans son cadencement de clé).

En conclusion de ce chapitre, on a introduit et analysé le chiffrement par bloc léger LILLIPUT, utilisant les EGFN introduits au chapitre précédent afin d'augmenter la diffusion du schéma. L'implémentation de LILLIPUT en matériel se compare raisonnablement bien

1. *"our results do not contain registers for storing keys. If such registers are needed, around 360 [...] extra GEs are required for Piccolo-80 [...]"*, cf. Section 5.1 de [SIH+11]

par rapport aux concurrents dans le cas où chiffrement et déchiffrement sont nécessaires. Le surcoût induit par l'ajout de `LinearLayer` au schéma de Feistel et le cadencement de clé robuste offre un bon compromis entre sécurité et nombre de portes.

Les recherches futures sur LILLIPUT concernent la réalisation d'une implémentation sérielle où, par exemple, seulement 8 bits de l'état interne sont calculés à chaque cycle d'horloge. En effet, les architectures sérielles sont connues pour réduire la consommation de courant et la dissipation d'énergie du circuit. En ce qui concerne les implémentations logicielles, nous voulons aussi évaluer les performances de LILLIPUT sur des microcontroleurs 4 bits, 8 bits ou 16 bits utilisés dans les cartes à puces ou les réseaux de capteurs sans-fil, comme par exemple respectivement les processeurs ATAM893-D, ATMega128L et MSP430. Il serait aussi intéressant de voir comment LILLIPUT pourrait être efficacement implémenté sur des plateformes haut-de-gamme, par exemple en utilisant des techniques bit-slice. Finalement, le plus intéressant sera certainement de voir ce que la communauté des cryptanalystes va inventer afin d'essayer de casser LILLIPUT.

Chapitre 11

Injections de fautes dans les schémas de Feistel généralisés

Ce travail [BTLT14] a été réalisé en collaboration avec Hélène Le Bouder, doctorante à l'École des Mines de Saint-Étienne, dirigée par Assia Tria (CEA Tech) et encadrée par Yanis Linge (STMicroelectronics). Il a été présenté et publié à FDTC en 2014. Hélène Le Bouder travaille sur "un formalisme unifiant les attaques physiques sur circuits cryptographiques". Plus précisément, le formalisme est pour les algorithmes à clé secrète. L'un des buts de ce formalisme est de trouver toutes les attaques existantes. Dans ce chapitre, c'est de manière systématique que nous souhaitons mettre en évidence des attaques de type "injection de faute".

L'objectif de ce travail est le suivant :

"Soit un algorithme de chiffrement basé sur un schéma de Feistel, quel est le registre idéal pour injecter une faute pour attaquer la clé ?"

On considère que les fautes injectées sont de type monobit sur un registre (bloc). Le fait d'injecter des fautes dans un circuit l'expose au risque d'être détérioré. Ainsi, le registre idéal pour injecter une faute sera celui pour lequel le nombre de fautes nécessaires pour retrouver la clé est minimal.

11.1 Attaque différentielle en faute

Les attaques différentielles en faute, notées DFA (Differential Fault Analysis) sont une technique de cryptanalyse qui exploitent les résultats erronés des injections de fautes. L'analyse de ces résultats se fait au niveau des données. Lorsqu'une donnée x manipulée par l'algorithme devient erronée elle est dite "fautée" et est notée avec une étoile en exposant x^*. On note $\Delta x = x \oplus x^*$, la différence entre la valeur correcte et la valeur fautée.

Si on considère un schéma classique à $\mathbf{r}$ tours, on note y_0^r et y_1^r les deux blocs du schéma au tour r, avec $r \in \{0, \cdots, \mathbf{r}\}$, avec la convention que l'entrée de la fonction au tour $r \geq 1$ est y_0^{r-1}. La sous-clé correspondante est notée K^r.

L'objectif d'une DFA sur un schéma de Feistel est de retrouver la sous-clé utilisée dans la fonction de Feistel du dernier tour, comme à la figure 11.1. Pour cela, une faute est injectée à un certain endroit dans l'algorithme de manière à impacter l'entrée de la fonction ciblée. En d'autres termes, on doit avoir $\Delta y_0^{\mathbf{r}-1} = y_0^{\mathbf{r}-1} \oplus y_0^{\mathbf{r}-1*} \neq 0$. On a alors la relation suivante :

$$\Delta y_0^{\mathbf{r}} = F(y_0^{\mathbf{r}-1}, K^{\mathbf{r}}) \oplus F(y_0^{\mathbf{r}-1*}, K^{\mathbf{r}}) \oplus \Delta y_1^{\mathbf{r}-1} \tag{11.1}$$

Parmi toutes les hypothèses possibles sur la sous-clé $K^{\mathbf{r}}$, seules certaines vont vérifier l'équation (11.1). Une DFA va donc réitérer le processus précédent un certain nombre de fois (avec une nouvelle faute à chaque fois) jusqu'à ce qu'il n'y ait plus qu'une sous-clé pouvant vérifier simultanément toutes les équations.

Néanmoins, il existe le cas où la sous-clé est xorée à la fin de la fonction de Feistel. Dans

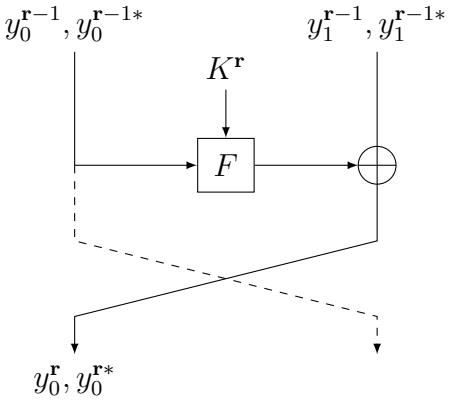


FIGURE 11.1 – Schéma d'attaque de type DFA dans un schéma de Feistel.

ce cas précis on a malheureusement :

$$\begin{aligned}\Delta y_0^{\mathbf{r}} &= F(y_0^{\mathbf{r}-1}, K^{\mathbf{r}}) \oplus F(y_0^{\mathbf{r}-1*}, K^{\mathbf{r}}) \oplus \Delta y_1^{\mathbf{r}-1} \\ &= \left(F(y_0^{\mathbf{r}-1}) \oplus K^{\mathbf{r}}\right) \oplus \left(F(y_0^{\mathbf{r}-1*}) \oplus K^{\mathbf{r}}\right) \oplus \Delta y_1^{\mathbf{r}-1} \\ &= F(y_0^{\mathbf{r}-1}) \oplus F(y_0^{\mathbf{r}-1*}) \oplus \Delta y_1^{\mathbf{r}-1}\end{aligned}$$

Il n'y a plus de dépendance avec la clé, il faut trouver un autre chemin d'attaque. C'est par exemple le cas de Piccolo [SIH+11] (cf. section 2.2.2.5) ou de SIMON [BSS+13] (cf. section 2.2.2.6). Ce cas particulier n'est pas pris en compte dans notre étude.

11.2 Approche de l'injection de fautes dans les schémas de Feistel généralisés

Cette étude porte sur la recherche du bloc y_e^r idéal pour injecter une faute monobit. L'emplacement est idéal pour injecter une faute lorsque le nombre de fautes nécessaires pour retrouver la clé est minimal. On souhaite avoir une bonne diffusion de la faute de manière à attaquer autant de bits de sous-clé que possible pour chaque injection de faute. D'où, a priori, la recherche du registre idéal dépend de deux choses : de la structure générale du schéma de Feistel généralisé mais aussi de la fonction de tour elle-même.

11.2.1 Raisonnement au niveau du schéma

Le but de cette section est d'étudier comment se propage la faute entre le moment où elle est injectée et les fonctions du dernier tour que l'on souhaite attaquer. Dans la suite, on suppose que l'on attaque un schéma de Feistel généralisé à k blocs à $\mathbf{r}$ tours, utilisant Λ fonctions de Feistel par tour. Chaque sous-clé K^r, avec $r \in \{1, \cdots, \mathbf{r}\}$, est divisée en Λ blocs K_λ^r, avec $\lambda \in \{0, \cdots, \Lambda - 1\}$, le bloc de sous-clé K_λ^r étant utilisé dans la λ-ième fonction de Feistel, notée F_λ. Par exemple, le GFN de type 2 à k branches (cf. section 7.1.2.4) possède $\Lambda = k/2$ fonctions.

Rappelons que le délai de diffusion, noté d, du schéma de Feistel généralisé tel que défini à la section 7.2 est le nombre minimal de tours requis pour que tous les blocs en sortie y_j^d

dépendent de tous les blocs en entrée y_i^0 $(i, j \in \{0, \cdots, k-1\})$. Ainsi par définition, il n'est pas intéressant d'injecter la faute au delà du délai de diffusion. La faute n'y est pas propagée à plus de blocs mais l'analyse en est complexifiée. Notre étude s'intéresse donc aux blocs y_e^r avec $e \in \{0, \cdots, k-1\}$, $r \in \{\mathbf{r}-(d+1), \cdots, \mathbf{r}-1\}$ où e indique l'indice du bloc où la faute monobit est injectée.

Pour qu'un bloc de sous-clé $K_\lambda^{\mathbf{r}}$ puisse être attaqué à l'aide d'injections de fautes, il faut que le bloc $y_i^{\mathbf{r}-1}$ en entrée de la fonction de tour $\mathcal{F}$ soit fauté. Il faut donc que le bloc sur lequel la faute a été injectée influence le bloc d'entrée de la fonction de tour.

Soit y_e^r, $e \in \{0, \cdots, k-1\}$, le bloc dans lequel est injectée une faute. Soit V_F le e-ième vecteur de base de taille k, *i.e.* toutes les coordonnées de V sont nulles sauf celle en position e.

En utilisant la représentation matricielle du schéma de Feistel généralisé introduite au chapitre 8, notée $\mathcal{M}$, on peut définir le vecteur V_F à l'aide de l'équation (11.2).

$$V_F = \mathcal{M}^{\mathbf{r}-(r+1)} \cdot V \tag{11.2}$$

Grâce aux propriétés de cette matrice en termes de caractérisation de profondeur de diffusion, ce vecteur permet de décrire "l'état" de la faute à l'entrée du dernier tour. En d'autres termes, on peut savoir pour chaque bloc en entrée du dernier tour, si l'erreur injectée dans y_e^r a influencé ce bloc et par combien de fonctions de tour elle est passée. Si à l'indice $i \in \{0, \cdots, k-1\}$ correspondant au bloc $y_i^{\mathbf{r}-1}$ on observe :

- 0 : alors le bloc n'est pas fauté ;
- 1 : le bloc contient la faute monobit ;
- F^x avec $x \in \mathbb{N}$: le bloc est fauté et la faute est passée x fois par une fonction de tour.

Ainsi ce vecteur permet entre autres de définir, pour chaque faute effectuée, le nombre n_λ de blocs de sous-clé attaqués.

11.2.2 Raisonnement au niveau de la fonction de Feistel

La fonction de Feistel F est construite à partir de trois types de fonctions :

- Le xor avec une sous-clé.
- Les S-box qui ne sont pas linéaires et assurent la confusion au sens de Shannon.
- Les fonctions de "mélange", c'est-à-dire les fonctions linéaires qui assurent la diffusion au sein d'un bloc.

Dans notre étude, on se restreint aux fonctions de Feistel ayant la structure suivante :

1. une application linéaire, en général l'identité (sauf par exemple dans le cas du DES) ;
2. un xor avec un bloc sous-clé $K_\lambda^{\mathbf{r}}$;
3. une couche de S-box en parallèle ;
4. une application linéaire de diffusion.

En d'autres termes, la fonction se compose d'une addition de clé suivie des S-box en parallèle, le tout se trouvant encadré par des applications linéaires.

On rappelle que la sous-clé $K^{\mathbf{r}}$ est divisée en Λ blocs $K_\lambda^{\mathbf{r}}$ correspondant aux Λ fonctions de Feistel de chaque tour. On divise alors chaque bloc en plusieurs morceaux selon les S-box ; si on appelle L le nombre de S-box en parallèle dans la fonction de Feistel, chaque bloc de

sous-clé $K_\lambda^{\mathbf{r}}$ est découpé en L morceaux $K_{\lambda,\ell}^{\mathbf{r}}$, avec $\ell \in \{0, \cdots, L-1\}$, correspondant chacun à l'entrée d'une S-box.

L'étude de la fonction de Feistel doit permettre d'identifier combien de morceaux de clé sont attaqués à chaque injection de faute. Ce nombre est noté $n_{\lambda,\ell}$.

L'étude de la fonction de Feistel doit également permettre de retrouver les valeurs possibles des différences $\Delta y_j^{\mathbf{r}-1}$ possibles sur le bloc auquel la fonction xore sa sortie. En effet $\Delta y_j^{\mathbf{r}-1}$ pouvant être fauté, il est important de connaître les différentes valeurs possibles de $\Delta y_j^{\mathbf{r}-1}$ afin d'utiliser l'équation (11.1).

L'étude qui suit est en fait une étude de la diffusion des bits dans la fonction de tour.

11.2.2.1 Nombre de morceaux de sous-clé attaqués

La première question est donc de savoir combien de morceaux de clé peuvent être attaqués, c'est-à-dire le nombre d'entrées de S-box fautées. En regardant la fonction de Feistel, il est possible de savoir quels bits en sortie peuvent être fautés avec une injection de faute monobit. Les bits en entrées des S-box sont linéairement dépendants des bits d'entrée de la fonction de Feistel, il est ainsi possible de déduire quelles S-box seront fautées à la fonction de Feistel suivante. À partir de la diffusion des bits dans la fonction de tour, on peut déterminer le nombre de morceaux de sous-clé attaqués $n_{\lambda,\ell}$ pour une injection de faute sachant le nombre de fonctions de tour traversées par la faute, *i.e.* en utilisant la valeur de V_F. Il faut d'ailleurs ne pas oublier de regarder aussi le cas sans passage dans la fonction de Feistel, c'est-à-dire le début de la fonction de Feistel. Par exemple dans le cas du DES, un unique bit fauté en entrée de fonction de Feistel peut impacter jusqu'à 2 S-box dans cette fonction à cause de la fonction d'expansion E (cf. section 2.2.2.2) qui duplique un bit sur deux.

11.2.2.2 Recherche de $\Delta y_j^{\mathbf{r}-1}$

Si la faute n'est pas propagée dans $y_j^{\mathbf{r}-1}$, la différence $\Delta y_j^{\mathbf{r}-1}$ est nulle. Sinon elle est inconnue, ce qui pose souci dans l'équation (11.1). La j-ème coordonnée $V_F(j)$ du vecteur V_F indique si $y_j^{\mathbf{r}-1}$ a été fauté ou non, mais aussi combien de fois la faute est passée par une fonction de Feistel. L'étude de la diffusion des bits dans une fonction de Feistel et la connaissance du nombre de passage dans celle-ci permettent d'évaluer le nombre de bits potentiellement fautés dans $y_j^{\mathbf{r}-1}$. Par exemple, si la faute n'est passée dans aucune fonction de Feistel, la différence $\Delta y_j^{\mathbf{r}-1}$ est monobit ; il y a autant de différences possibles que de bits dans un bloc. Dans le cas du DES, cela donne 32 différences $\Delta y_j^{\mathbf{r}-1}$ possibles. Cependant cette approche peut être améliorée. Pour cela, on recherche quel bit a été fauté dans le bloc y_e^r, à partir de ce que l'on observe en sortie de l'algorithme. On introduit donc le vecteur W_F décrivant "l'état" de la faute à la fin du chiffrement :

$$W_F = \mathcal{M} \cdot V_F.$$

Le degré du coefficient non nul de plus petit degré de W_F, noté $\min \deg_{\geq 0}(W_F)$, correspond au bloc qui contient la différence non nulle la plus simple (la moins diffusée par les fonctions de Feistel). Différents cas sont alors possibles :

- Si $\min \deg_{\geq 0}(W_F) = 0$, on observe une faute monobit sur la différence $\Delta y^{\mathbf{r}}_{\min \deg_{\geq 0}(W_F)}$; la faute a juste été recopiée et ainsi la faute sur y_e^{r*} est connue.
- Pour une faute monobit en entrée d'une fonction, l'énumération exhaustive de toutes les différences possibles à la sortie de la fonction de tour est en général possible en pratique à cause de la structure de la fonction puisque seule une ou deux S-box seront fautées. Ainsi si $\min \deg_{\geq 0}(W_F) = 1$, connaître $\Delta y^{\mathbf{r}}_{\min \deg_{\geq 0}(W_F)}$ permet de restreindre l'ensemble des bits fautés possibles sur y_e^{r*}.
- Si $\min \deg_{\geq 0}(W_F) \geq 2$, la même approche est encore envisageable. Cependant puisque pour chaque fonction traversée, il faut calculer toutes les différences intermédiaires, la complexité d'un tel algorithme croît exponentiellement en la taille du bloc et en le nombre de fonctions traversées $\min \deg_{\geq 0}(W_F)$ et est donc rapidement infaisable. De plus, comme les fonctions de Feistel sont construites de manière à maximiser la diffusion au sein d'un bloc, il est probable que cela n'aide que peu à réduire le nombre de fautes monobit possibles.

Si $\Delta y^{\mathbf{r}}_{\min \deg_{\geq 0}(W_F)}$ permet de restreindre l'ensemble des fautes candidates, chacun de ces candidats est propagé pour obtenir un ensemble réduit de différences $\Delta y_j^{\mathbf{r}-1}$ possibles. L'union de toutes ces différences (pour tous les candidats) indique le nombre de différences $\Delta y_j^{\mathbf{r}-1}$ possibles.

11.2.3 Raisonnement au niveau des S-box

Comme on considère que, dans une fonction de Feistel, il n'y a que des fonctions linéaires précédant ou suivant le xor de clé et les S-box, on peut considérer l'équation (11.1) indépendamment sur chaque S-box en parallèle, *i.e.* on peut faire du diviser pour régner et ainsi réduire la complexité de l'attaque. On se ramène ainsi à l'équation (11.3) où x, x^* et Δy sont connus.

$$S(x \oplus \hat{\kappa}) \oplus S(x^* \oplus \hat{\kappa}) = \Delta y \tag{11.3}$$

On s'inspire alors des méthodes de la cryptanalyse différentielle classique. En reprenant l'équation (11.3), on pose $e = x \oplus x^*$, on a alors l'équation suivante :

$$S(x \oplus \hat{\kappa}) \oplus S(x \oplus e \oplus \hat{\kappa}) = \Delta y \tag{11.4}$$

D'une expérience à l'autre, les valeurs de x, e et Δy varient. On cherche le nombre minimal n d'expériences (injections de fautes) tel qu'étant donnés $x_0, \cdots, x_{n-1}$, $e_0, \cdots, e_{n-1}$ et $\Delta y_0, \cdots, \Delta y_{n-1}$, il n'y ait qu'une seule clé qui vérifie les n équations (11.4).

La S-box S est une fonction booléenne de $\mathbb{F}_2^u$ dans $\mathbb{F}_2^v$. Soit $a \in \mathbb{F}_2^u$ et $b \in \mathbb{F}_2^v$, l'ensemble des entrées x de la S-box telles que si on applique la différence a en entrée, on observe la différence b en sortie, est noté $\mathcal{S}_{a,b}$. En d'autres termes :

$$\mathcal{S}_{a,b} = \{z \in \mathbb{F}_2^u \mid S(z) \oplus S(z \oplus a) = b\} \tag{11.5}$$

Par symétrie de l'équation (11.5), le cardinal $\#\mathcal{S}_{a,b}$ est pair. Cet ensemble $\mathcal{S}_{a,b}$ est défini $\forall a \in \mathbb{F}_2^u \backslash \{0\}$ et $\forall b \in \mathbb{F}_2^v$. Le logarithme en base 2 de la taille maximum des ensembles $\mathcal{S}_{a,b}$

est noté δ.

$$\begin{aligned} \delta &= \log_2 \max_{a\neq 0,\, b} \#\mathcal{S}_{a,b} \\ 2^{\delta} &= \max_{a\neq 0,\, b} \#\mathcal{S}_{a,b} \end{aligned}$$

Puisque les $\mathcal{S}_{a,b}$ ne sont pas tous vides et tous de cardinal pair, cela implique que $\delta \geq 1$. Plus généralement, on a $\delta \geq u - v$. En effet, soit une différence en entrée $a \neq 0$ quelconque fixée, on a $\sum_{b\in\mathbb{F}_2^v} \#\mathcal{S}_{a,b} = 2^u$. Donc il existe un b tel que $\#\mathcal{S}_{a,b} \geq 2^u/2^v$ et donc $\delta \geq u - v$. D'autre part, si z est tiré uniformément dans $\mathbb{F}_2^u$ alors

$$\Pr\left[z \in \mathcal{S}_{a,b}\right] = \frac{\#\mathcal{S}_{a,b}}{2^u} \leq 2^{\delta-u}$$

Ainsi, κ est solution de l'équation (11.4) si et seulement si $x \oplus \kappa \in \mathcal{S}_{e,\Delta y}$. Ce qui implique que le nombre de solutions de l'équation (11.4) est $\#\mathcal{S}_{e,\Delta y}$. L'ensemble des z tel que $z \oplus x \in \mathcal{S}_{e_i,\Delta y_i}$est noté $x \oplus \mathcal{S}_{e,\Delta y}$. Alors κ est solution de l'équation (11.4) si et seulement si $\kappa \in x \oplus \mathcal{S}_{e,\Delta y}$.

Pour n expériences, on s'intéresse à la probabilité suivante :

$$\Pr\left[k \in \bigcap_{i=0}^{n-1} (x_i \oplus \mathcal{S}_{e_i,y_i})\right].$$

Les fautes e_i sont indépendantes les unes des autres, les différences Δy_i et les entrées x_i indépendantes et identiquement distribuées. On a donc :

$$\Pr\left[\kappa \in \bigcap_{i=0}^{n-1} (x_i \oplus \mathcal{S}_{e_i,\Delta y_i})\right] = \prod_{i=0}^{n-1} \Pr\left[\kappa \in (x_i \oplus \mathcal{S}_{e_i,\Delta y_i})\right] \leq 2^{n(\delta-u)}$$

Par ailleurs, on a que la probabilité cherchée est au moins égale à 2^{-u} car il y a une solution au problème. Ainsi le nombre n de fautes recherché est tel que $2^{n(\delta-u)} \leq 2^{-u}$ d'où :

$$n \geq \left\lceil \frac{u}{u-\delta} \right\rceil \tag{11.6}$$

En pratique, il existe principalement deux cas possibles :

– La S-box est bijective ($u \to u$), optimale pour la différentielle, plus précisément avec δ aussi petit que possible. Typiquement, $u = 4$ ou $u = 8$. On sait que $\delta \geq 1$. Cependant, à l'heure actuelle, les seuls exemples connus atteignant cette borne sont pour u impair ou pour $u = 6$. En revanche, beaucoup d'exemples de S-box avec $\delta = 2$ sont connus. Le cas $u = 4$ est a été bien étudié, voir les classifications de Leander et Poschmann [LP07] et de Saarinen [Saa11]. Il est aujourd'hui facile de trouver une S-box 4×4 avec un δ minimal. L'autre cas usuel de S-box bijective est le cas $u = 8$. Si leur classification reste un problème ouvert à l'heure actuelle, de nombreux exemples de S-box avec $\delta = 2$ existent, la plus connue étant celle de l'AES. Dans les deux cas, on obtient alors que le nombre de fautes pour retrouver $\hat{\kappa}$ est $n = 2$.

– La S-box n'est pas bijective ($u \to v$). C'est le cas du DES avec des S-boxes ($6 \to 4$). En général, on doit avoir $\delta \geq 6 - 4 = 2$, et donc $n = 2$. Cependant à cause de leur structure, les S-box du DES n'atteignent pas cette borne, mais vérifient en fait $\delta = 4$, ce qui donne $n = 3$.

De manière générale, avec les paramètres u et v utilisés en pratique, il est toujours possible de calculer δ en temps raisonnable.

On regarde maintenant ce qui se passe quand il y a une incertitude sur la valeur Δ_y. C'est-à-dire lorsque le bloc en sortie de fonction est lui-aussi fauté. Il s'agit du cas de l'équation suivante, avec J le nombre de Δy possibles :

$$S(x \oplus \hat{\kappa}) \oplus S(x \oplus e \oplus \hat{\kappa}) = \Delta y_1 \text{ ou } \cdots \text{ ou } \Delta y_J \tag{11.7}$$

Puisque, à une différence d'entrée a fixée, les ensembles $\mathcal{S}_{a,b}$ sont disjoints deux à deux, on a :

$$\Pr\left[z \in \bigcup_{t=1}^{J} \mathcal{S}_{a,b_t}\right] = \sum_{t=1}^{J} \Pr\left[z \in \mathcal{S}_{a,b_t}\right]$$

Comme dans le cas où il y a seulement un Δy, κ est solution de l'équation (11.7) si et seulement si :

$$\kappa \in x \oplus \bigcup_{t=1}^{J} \mathcal{S}_{e,\Delta y_t}$$

L'union des hypothèses de sous-clés vérifiant des équations de type (11.7), $\bigcup_t (x \oplus \mathcal{S}_{e,\Delta y_t})$ n'est pertinente que si elle est plus restreinte que l'ensemble de toutes les hypothèses.

Pour n expériences :

$$\Pr\left[\kappa \in \bigcap_{i=0}^{n-1} \left(x_i \oplus \bigcup_{t=1}^{J} \mathcal{S}_{e_i,\Delta y_{i,t}}\right)\right] = \prod_{i=1}^{n-1} \Pr\left[\kappa \in x_i \oplus \bigcup_{t=1}^{J} \mathcal{S}_{e_i,\Delta y_{i,t}}\right] \tag{11.8}$$

Par conséquent, le nombre minimum de fautes n_J pour récupérer la clé est le plus petit n dans l'équation (11.8) tel que le membre de gauche soit inférieur à 2^{-u}.

Plus précisément, avec J le nombre moyen de Δy possibles :

$$n_J = \left\lceil \frac{u}{u - \delta - \log_2 J} \right\rceil.$$

11.3 Algorithme et Résultats

11.3.1 Algorithme

Cette partie décrit la méthode pour justifier le meilleur emplacement où injecter des fautes monobit dans les algorithmes de chiffrement utilisant des schémas de Feistel. L'algorithme 1 décrit notre procédure ; il prend en entrée la fonction de tour F et la matrice $\mathcal{M}$ représentant le schéma de Feistel. Il retourne pour tous les blocs concernés, c'est-à-dire les y_e^r avec $e \in \{0, \cdots, k-1\}$ et $r \in \{\mathbf{r} - (d+1), \mathbf{r} - 1\}$, le nombre de morceaux de blocs de clés et le nombre de différences à prendre en compte.

Pour chaque bloc étudié comme candidat à l'injection de faute, l'algorithme commence

par définir le nombre n_λ de blocs de clé attaqués en utilisant le schéma de Feistel. Puis pour chacun des blocs $K_\lambda^{\mathbf{r}}$, l'algorithme calcule le nombre de morceaux attaqués. Ensuite pour chaque bloc de clé attaquable, si $V_F(j) \neq 0$, c'est-à-dire si le bloc xoré en sortie de fonction de Feistel $y_j^{\mathbf{r}-1}$ est fauté, une première estimation du nombre de différentielles possibles est donnée. Puis, l'algorithme réutilise le schéma de Feistel pour identifier le bloc le moins influencé (influencé quand même) par la faute pour réduire encore le nombre de différences possibles. Si le nombre de différences ne permet plus d'éliminer de clé, $K_\lambda^{\mathbf{r}}$ n'est plus considéré comme un morceau de clé attaqué. Pour finir, le nombre de fautes nécessaire pour retrouver les morceaux de clés en fonction du nombre n_J des Δy (nombres moyens) est calculé.

Cet algorithme donne une première idée d'où injecter la faute, il ne dit pas quel bloc est le meilleur. C'est une étude pour se faire une première idée. Rien n'empêche de trouver une attaque encore plus performante qui tire partie de certaines spécificités de l'algorithme, cette amélioration vient alors peaufiner nos résultats. Par ailleurs, une étude du même genre avec des fautes non monobit est à envisager en perspective.

11.3.2 Résultats

L'algorithme 1 a été codé en magma et a permis de tester notre approche sur différents exemples.

11.3.2.1 DES

Le DES [Nat77] (cf. section 2.2.2.2) a été choisi car c'est le plus connu des schémas de Feistel classiques. Dans le cadre de notre étude :

- Le nombre de tours est $\mathbf{r} = 16$.
- Le nombre de blocs est $k = 2$.
- Le délai de diffusion est $d = 2$.
- La matrice de diffusion $\mathcal{M}$ est la suivante :

$$\mathcal{M} = \begin{pmatrix} . & 1 \\ 1 & F \end{pmatrix}$$

- Le nombre de blocs de clé à attaquer est : $\Lambda = 1$. On a $y_i^{\mathbf{r}-1} = R_{15}$ et $\Delta y_j^{\mathbf{r}-1} = \Delta L_{15}$.
- Le nombre de morceaux de bloc de clé (le nombre de s-box) est : $L = 8$.
- Le nombre de fautes pour retrouver un morceau de clé avec des équations de type (11.3) est : $n = 3$.

L'étude de la diffusion de la fonction de Feistel est la suivante :

- Du fait de l'expansion E, un bit modifié en entrée de fonction de Feistel peut impacter une ou deux S-box.
- Une des propriétés des S-box du DES exprime le fait que modifier un bit en entrée implique d'en modifier au moins deux en sortie
- La permutation P est telle que les bits en sortie des S-box d'un tour sont répartis sur 4 S-box différentes à l'entrée du tour suivant.

Ainsi le passage de la faute dans aucune fonction de Feistel peut quand même permettre d'attaquer deux S-box. Si la faute passe par une seule fonction de Feistel, il y a entre 2 et 8

Algorithm 1 Trouve le bloc dans lequel injecter des fautes monobit pour que l'attaque soit optimisée, *i.e.* injecter le moins de fautes possible.

Require: La fonction de Feistel F, la matrice du schéma $\mathcal{M}$
Ensure: Tous les blocs y_e^r avec le nombre de blocs de sous-clé attaqué n_λ, les nombres de morceaux de sous-clé attaqués $n_{\lambda,\ell}$, le nombre de différences en sortie possibles J et le nombre de fautes nécessaire n_J, associés
Calculer d le délai de diffusion à l'aide de $\mathcal{M}$.
Déduire le nombre de S-box L de F.
for 0 à d nombre de passage dans un F **do**
  Calculer le $n_{\lambda,\ell}$ associé.
  Calculer le n_J associé
end for
for $\forall y_e^r\ e \in \{0, \cdots, k-1\}, r \in \{\mathbf{r}-(d+1), \cdots, \mathbf{r}-1\}$ **do**
  Créer V le e-ième vecteur de base de taille k
  $V_F \leftarrow \mathcal{M}^{\mathbf{r}-(r+1)} \cdot V$
  Déduire n_λ
  for tous les blocs $K_\lambda^{\mathbf{r}}$ attaquables **do**
    Identifier le bloc d'entrée de $y_i^{\mathbf{r}-1}$ et bloc xoré $B_j^{\mathbf{r}-1}$
    Déduire $n_{\lambda,\ell}$ de $V_F(i)$
    if $V_{\mathcal{F}}(j) \neq 0$ **then**
      if $\deg V_F(j) \geq 2$ **then**
        Retirer le morceau de la liste $K_\lambda^{\mathbf{r}}$
      else
        Déduire le nombre des $\Delta y_j^{\mathbf{r}-1}$ possibles
        if $\min \deg_{\geq 0}(W) \leq 1$ **then**
          Réduire le nombre des $\Delta y_j^{\mathbf{r}-1}$ possibles
        end if
        Déduire les J
        if $J \geq \#\{k\}$ **then**
          Retirer le morceau de la liste $K_\lambda^{\mathbf{r}}$
        end if
      end if
    end if
    Calculer n_J
  end for
end for

bits fautés en sortie, ce qui induit $32 \times (256 - 1 - 8)$ différences possibles.

Inversement, le passage dans une fonction de tour est assez caractéristique pour que l'observation des bits fautés en sortie permette de diminuer le nombre des 32 différences en entrée à seulement deux.

Les résultats de notre analyse sont dans le tableau 11.1 où $n_{\lambda,\ell}$ est le nombre de morceaux de blocs de sous-clé attaqués et J est le nombre d'hypothèses sur ΔL_{15}.

Notre étude montre qu'il est plus judicieux d'injecter des fautes dans R_{14} pour le DES. Pourtant, Biham et Shamir ont montré dans [BS97] que pour le DES, l'attaque est plus

bloc y_r^e	V_F	W_F	$n_{\lambda,\ell}$	J
R_{15}	$(0,1)$	$(F,1)$	$1 \leq n_{\lambda,\ell} \leq 2$	1
R_{14}	$(1,F)$	(F^2,F)	$2 \leq n_{\lambda,\ell} \leq 8$	2
R_{13}	(F,F^2)	(F^3,F^2)	$2 \leq n_{\lambda,\ell} \leq 8$	$32 * 247 = 1504$

Tableau 11.1 – Résultats sur le DES

efficace en injectant la faute dans R_{13}. Pour justifier ce résultat, ils ont étudié la fonction de tour en profondeur, tandis que notre approche est générique. Notre approche confirme que, si nous sommes en mesure de réduire le nombre d'hypothèses sur ΔL_{15}, davantage de morceaux de clés seront attaqués en injectant les fautes sur R_{13} plutôt que sur R_{14}. De plus l'étude de Rivain dans [Riv09] montre qu'en injectant les fautes au tour 12, 11, 10 ou 9 (au delà du délai de diffusion), l'attaque nécessite d'avantage de fautes.

11.3.2.2 MIBS

MIBS [ISSK09] (cf. section 2.2.2.4) est choisi en exemple car comme le DES il n'est découpé qu'en deux blocs, ainsi c'est surtout sur la fonction de Feistel que se joue notre étude. De plus contrairement au DES, ses S-box sont bijectives, ce qui est plus classique. Dans le cadre de notre étude :

- Le nombre de tours est : $\mathbf{r} = 32$.
- Le nombre de blocs est : $k = 2$.
- Le délai de diffusion est : $d = 2$.
- La matrice de diffusion $\mathcal{M}$ est la suivante :

$$\mathcal{M} = \begin{pmatrix} F & 1 \\ 1 & . \end{pmatrix}$$

- Le nombre de blocs de clé à attaquer :$\Lambda = 1$ On a $y_i^{\mathbf{r}-1} = L_{31}$ et $\Delta B_{\mathbf{r}-1}^j = \Delta_{R_{31}}$.
- Le nombre de morceaux de bloc de clé (le nombre de S-box) est : $L = 8$.
- Le nombre de fautes pour retrouver un morceau de clé avec des équations de type (11.3) est : $n = 2$.

La diffusion par la fonction de tour préserve le découpage en nybbles défini par les S-box de MIBS. Ainsi, l'étude de la diffusion de la fonction de Feistel est la suivante :

- Fauter un bit en entrée implique fauter une S-box et donc jusqu'à 4 bits en sortie de fonction.
- Si la faute est passée dans une fonction de Feistel, il y a alors 5 ou 6 morceaux de clé attaqués.
- Si on observe un résultat fauté (faute passée une fois dans la fonction de Feistel) on peut en déduire quelle S-box était fautée et ainsi il y a 4 bits susceptibles de contenir la faute en entrée.
- Si la faute est passée dans plus d'une fonction de Feistel, tous les morceaux sont attaqués.

Les résultats de notre analyse sont dans le Tableau 11.2 où $n_{\lambda,\ell}$ est le nombre de morceaux de blocs de sous-clé attaqué et J est le nombre d'hypothèses sur ΔR_{31}. Comme pour le DES,

le nombre n_λ de blocs de sous-clé attaqués est $n_\lambda = 1$, car il s'agit d'un schéma de Feistel classique.

Blocs y_e^r	V_F	W_F	$n_{\lambda,\ell}$	J
L_{31}	$(1, 0)$	$(1, F)$	1	1
L_{30}	$(F, 1)$	(F, F^2)	$5 \leq n_{\lambda,\ell} \leq 6$	4
L_{29}	(F^2, F)	(F^2, F^3)	8	112

Tableau 11.2 – Résultats sur MIBS

Les résultats indiquent qu'il est préférable d'attaquer en L_{30}, mais l'attaquant ne cible pas tous les morceaux à la fois. En L_{29}, au contraire, tous les morceaux sont attaqués, cependant le nombre de différences possibles sur ΔR_{31} est plus grand. Une étude plus approfondie en L_{29} permettrait de diminuer le nombre de différences, mais elle n'est pas triviale.

11.3.2.3 TWINE

TWINE [SMMK12] (cf. section 2.2.2.8) est choisi car il est l'exact inverse du DES et de MIBS. La fonction de Feistel ne change rien au découpage, toute l'étude se fait sur le schéma de Feistel lui-même.

Dans le cadre de notre étude :

- Le nombre de tours est : $\mathbf{r} = 36$.
- Le nombre de blocs est : $k = 16$.
- Le délai de diffusion est : $d = 8$
- La matrice de diffusion $\mathcal{M}$ est la suivante :

$$\mathcal{M} = \begin{pmatrix}
F & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & F & 1 & . & . & . & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & F & 1 & . & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & F & 1 & . & . & . & . & . & . \\
. & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & F & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & F & 1 & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
. & . & . & . & F & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & F & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & .
\end{pmatrix}$$

- Le nombre de blocs de clé à attaquer est : $\Lambda = 8$.
- Le nombre de morceaux de bloc de clé (le nombre de S-box) $L = 1$. Ainsi l'étude de la fonction de tour est très limitée. Soit le bloc est fauté soit il ne l'est pas. De plus, si on observe la sortie d'un bloc fauté par une faute passée une seule fois dans la fonction de tour, cela laisse les 4 bits en entrée possibles.

– Le nombre de fautes pour retrouver un morceau de clé avec des équations de type (11.3) est : $n = 2$.

La fonction de Feistel n'est composée que d'un xor de clé et d'une seule S-box ($n_\lambda = 1$). Ainsi, si la faute de $y_{j*}^{\mathbf{r}-1}$ n'est passée que dans une seule fonction, il y a $J = 14$ différences possibles, voire 7 si la faute est connue. Par conséquent, il existe trois cas favorables :

– $y_j^{\mathbf{r}-1}$ n'est pas fauté : $J = 1$.
– $y_j^{\mathbf{r}-1}$ contient exactement la valeur de la faute : $J = 1$ si la valeur de la faute est connue, sinon $J = 4$.
– $y_j^{\mathbf{r}-1}$ est passé dans exactement une fonction de tour : $J = 7$ si la valeur de la faute est connue, sinon $J = 14$.

Le meilleur registre pour une injection de faute sur TWINE est alors celui où la faute atteint la plupart des fonctions de Feistel, avec la condition que les différences respectives soient dans l'un dans des trois cas ci-dessus.
Il apparaît que les meilleurs blocs sont les blocs y_i^{31} en l'entrée des fonctions de Feistel. De cette façon, il est possible d'attaquer simultanément $n_\lambda = 5$ blocs de clé sur le dernier tour. Dans cette configuration, quatre $\Delta y_j^{\mathbf{r}-1}$ sont connus ($J = 1$) et pour le dernier cas : $J = 7$. Injecter les fautes plus tôt dans l'algorithme (tour 30 et avant) ne permet d'attaquer qu'au plus 4 blocs de clé, ce qui n'est pas aussi intéressant que le cas précédent. En effet, plus de blocs sont fautés, cependant le nombre de différences possibles n'est plus réduit correctement. Injecter les fautes plus tard dans l'algorithme (tour 32 et après) ne permet d'attaquer que 3 blocs de clé à la fois.

11.3.2.4 CLEFIA

CLEFIA [SSA+07] (cf. section 2.2.2.1) est choisi comme exemple de "juste milieu" entre le schéma de Feistel et les GFN à nombreux blocs et petite fonction de Feistel. L'étude utilise à la fois la diffusion de la fonction de Feistel et la diffusion au niveau du schéma de Feistel. Il est l'exemple type où notre étude doit habilement mélanger ces deux composantes. Dans le cadre de notre étude :

– Le nombre de tours est $\mathbf{r} = 18$ ou 22 ou 26, cela dépend de la taille de la clé, pour notre étude $\mathbf{r} = 18$.
– Le nombre de blocs est $k = 4$.
– Le délai de diffusion est $d = 4$.
– La matrice de diffusion $\mathcal{M}$ est la suivante :

$$\mathcal{M} = \begin{pmatrix} F & 1 & . & . \\ . & . & 1 & . \\ . & . & F & 1 \\ 1 & . & . & . \end{pmatrix}$$

– Le nombre de blocs de clé à attaquer est : $\Lambda = 2$
– Le nombre de morceaux de bloc de clé (le nombre de S-box) est : $L = 4$.
– Le nombre de fautes pour retrouver un morceau de clé avec des équations de type (11.3) est : $n = 2$.

L'étude de la diffusion dans la fonction de Feistel est la suivante :

– Chaque bit d'entrée de la fonction impacte exactement une seule S-box. Les S-box sont bijectives, ainsi modifier un bit en entrée signifie modifier 1 à 8 bits en sortie.
– Par les propriétés de diffusion de matrices MC_0 et MC_1 de CLEFIA, si la faute a traversé une fonction de Feistel, alors toutes les S-boxes sont impactées.
– En observant une sortie de fonction, et supposant qu'un seul bit d'entrée est fauté, il est possible de déduire quelle S-box est fautée et ainsi la faute d'entrée est réduite à 8 possibilités.

Les résultats de notre analyse sont dans le Tableau 11.3

Blocs y_e^r	V_F	W_F	n_λ	$n_{\lambda,\ell}$	J
y_0^{17}	$(1,.,.,.)$	$(1,F,.,.)$	1	$(1,0)$	$(1,-)$
y_0^{16}	$(F,.,.,1)$	$(F,F^2,.,1)$	1	$(4,0)$	$(1,-)$
y_0^{15}	$(F^2,.,1,F)$	$(F^2,F^3,1,F)$	2	$(4,1)$	$(1,\ \leq 127)$
y_0^{14}	$(F^3,1,F,F^2)$	(F^3,F^4,F,F^2)	2	$(4,4)$	$(4,\text{grand})$
y_0^{13}	(F^4,F,F^2,F^3)	(F^4,F^5,F^2,F^3)	2	$(4,4)$	$(946,\text{grand})$

Tableau 11.3 – Résultats sur CLEFIA

Les résultats indiquent que l'injection dans y_0^{17} n'est pas optimale. En effet si l'injection est faite un tour plus tôt dans y_0^{16}, cela permet d'attaquer plus de blocs de clés simultanément. Ce constat a déjà été exposé dans l'attaque menée par Chen *et al.* [CWF07] où 18 fautes sont nécessaires pour réaliser l'attaque.

Si l'on veut attaquer deux blocs de clés en même temps, l'un aura une différence en sortie non triviale. Le cas y_0^{15} est encore gérable, mais une seule S-box de la deuxième fonction est attaquée, c'est le choix de Takahashi et Fukunaga dans [TF08]. L'étude approfondie des fonctions de CLEFIA leur permet de réduire significativement le nombre de fautes nécessaires pour réaliser leur attaque. Ce résultat est en adéquation avec notre analyse et permet de conclure qu'il est préférable d'injecter la faute au tour 15.

D'autre part, on peut peut-être mener une attaque en ciblant y_0^{14}, cela permet d'attaquer tous les 8 blocs à la fois, mais une meilleure étude de la fonction de tour est à prévoir.

Conclusions & Perspectives

Au cours de cette thèse, j'ai pu me pencher sur divers aspects de la cryptographie symétrique légère. En particulier, deux grand axes de recherche ont été developpés. Le premier concerne les registres à décalage à rétroaction avec retenue. Deux nouvelles propriétés cryptanalytiques sont présentées. La première concerne la présence de différentielles dont la probabilité peut être élevée même après un grand nombre d'itérations. La seconde est une adaptation des propriétés linéaires de Tian et Qi [TQ09]. Cependant aucune de ces deux propriétés ne remet en cause la sécurité des FCSR. Les FCSR sont aussi utilisés pour créer une fonction de hachage légère, GLUON, basée sur le modèle de l'éponge [BDPA08]

Le second axe de recherche concerne les schémas de Feistel généralisés (GFN). On développe une vision matricielle unifiant tous les schémas de Feistel généralisés connus. Cette représentation est en lien avec la notion de diffusion et permet une classification des schémas de Feistel généralisés en fonction du nombre de fonctions internes qu'ils possèdent et de leur délai de diffusion (nombre de tours pour que tous les blocs en sortie dépendent de tous les blocs en entrée). La notion de délai de diffusion est aussi raffinée pour prendre en compte le nombre de fonctions traversées. Puis, cette représentation est utilisée pour étendre la notion de schéma de Feistel généralisé en proposant les schémas de Feistel généralisés étendus (EGFN) en séparant les fonctions internes selon leur rôle : confusion ou diffusion. Ces schémas permettent de diminuer le délai de diffusion sans pour autant trop augmenter le coût de l'implémentation. Une famille particulière d'EGFN avec un bon délai de diffusion fait l'objet d'une étude poussée. On y évalue leur indistinguabilité ainsi que leur résistance aux attaques les plus classiques. On constate que certaines attaques dites structurelles comme les attaques intégrales ou les différentielles impossibles dépendent directement du délai de diffusion. Cependant d'autres attaques comme les attaques différentielles et linéaires ne sont – dans le meilleur des cas – pas affectées par un délai de diffusion faible. Dans le pire des cas, on constate une dégradation significative de la résistance de cette famille d'EGFN face à ces attaques. Dans les deux cas (meilleur et pire), le nombre minimal de S-box actives que l'on a trouvé est très proche de ceux qu'avaient trouvé Suzaki et Minematsu [SM10] lorsqu'ils avaient fourni un premier pas vers la généralisation des schémas de Feistel généralisés de type-2. Parmi la famille d'EGFN étudiée, nous avons ensuite choisi l'un des meilleurs schémas vis-à-vis des attaques différentielles et linéaires pour proposer un nouvel algorithme de chiffrement léger par bloc appelé LILLIPUT. Cet algorithme se base aussi sur des schémas de Feistel généralisés pour son cadencement de clé afin de pouvoir facilement réaliser le déchiffrement. Enfin, un dernier travail en rapport avec les schémas de Feistel généralisés a été effectué portant sur le lien entre la diffusion et les attaques différentielles en fautes sur les schémas de Feistel généralisés. Le but est alors de trouver, de manière générique, où se situe le meilleur endroit pour injecter une faute sur un schéma de Feistel généralisé.

Au terme de ces années de doctorat et malgré les réponses apportées, de nombreuses questions restent encore ouvertes. En ce qui concerne les FCSR, l'attaque de Perrin et Khovratovich [PK15, PK14] sur GLUON-64 met en évidence que les F-FCSR ne se comportent pas comme une fonction aléatoire. Dans le cadre de la construction éponge, cet effet pourrait être mitigé si le taux de l'éponge était plus élevé. Cependant cela n'explique pas pourquoi

les versions GLUON-80 et GLUON-112 n'exhibent pas ce comportement. En ce qui concerne les schémas de Feistel généralisés et les schémas de Feistel généralisés étendus, le lien entre la représentation matricielle d'un tel schéma et sa résistance aux attaques différentielles et linéaires est loin d'être clair : le nombre de S-box actives peut varier presque du simple au double pour une même famille simplement en changeant la permutation des blocs et en gardant un même délai de diffusion. De manière plus générale, les questions que je me pose à l'issue d'une réflexion plus globale sur la cryptographie symétrique sont multiples. Une première concernerait le cadencement de clé, qui n'a que très brièvement été évoqué dans ce manuscrit et pour cause. En effet, on a aujourd'hui une idée assez claire de ce que peut contenir le chemin de donnée d'un chiffrement par bloc, il y a en revanche encore beaucoup de flou sur ce qu'est un "bon" cadencement de clé. Une seconde question porterait sur les interactions qui existent entre le monde de la cryptanalyse classique et celui de la cryptanalyse physique. En effet, à travers mon étude des injections de fautes dans les schémas de Feistel généralisés, je me suis rendu compte que ce sont exactement les éléments qui rendaient le chiffrement résistant à la cryptanalyse classique qui le rendaient vulnérable aux injections de fautes. On pourrait alors se demander s'il ne serait pas possible, dans le cas d'un algorithme léger, de relacher un peu les exigences de la cryptanalyse classique afin de contrer plus efficacement les attaques par injections de fautes, comme c'est déjà le cas pour les implémentations à seuil dans le cas de certaines attaques par canaux auxilliaires.

Bibliographie

[AB05a] François Arnault et Thierry P. Berger : Design and Properties of a New Pseudorandom Generator Based on a Filtered FCSR Automaton. *IEEE Trans. Computers*, 54(11):1374–1383, 2005. 52

[AB05b] François Arnault et Thierry P. Berger : F-FCSR : Design of a New Class of Stream Ciphers. *In Fast Software Encryption, FSE 2005*, volume 3557 de *LNCS*, pages 83–97. Springer, 2005. 52

[ABK] Ross Anderson, Eli Biham et Lars Knudsen : Serpent : A Proposal for the Advanced Encryption Standard. 11

[ABL+09] François Arnault, Thierry P. Berger, Cédric Lauradoux, Marine Minier et Benjamin Pousse : A New Approach for FCSRs. *In SAC 2009*, volume 5867 de *LNCS*, pages 433–448. Springer, 2009. 48, 51, 54, 56, 57, 69

[ABM08] François Arnault, Thierry P. Berger et Marine Minier : Some Results on FCSR Automata With Applications to the Security of FCSR-Based Pseudorandom Generators. *IEEE Transactions on Information Theory*, 54(2): 836–840, 2008. 53

[ABMP11] F. Arnault, T. P. Berger, M. Minier et B. Pousse : Revisiting LFSRs for Cryptographic Applications. *IEEE Trans. on Info. Theory*, 57(12):8095–8113, 2011. 45, 121, 125

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier et Adi Shamir : Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. *In FSE 2009*, volume 5665 de *LNCS*, pages 1–22. Springer, 2009. 77

[AG99] C. Adams et J. Gilchrist : The CAST-256 Encryption Algorithm. Network Working Group, RFC 2612, june 1999. http ://tools.ietf.org/html/rfc2612. 87

[AHMNP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier et María Naya-Plasencia : Quark : A Lightweight Hash. *In CHES 2010*, volume 6225 de *LNCS*, pages 1–15. Springer, 2010. 69, 71

[AIK+00] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima et T. Tokita : Camellia : A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. *In Selected Areas in Cryptography - SAC 2000*, volume 2012 de *LNCS*, pages 39–56. Springer, 2000. 12, 15, 22

[BBS99] E. Biham, A. Biryukov et A. Shamir : Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *In Advances in Cryptology - EUROCRYPT '99*, volume 1592 de *LNCS*, pages 12–23. Springer, 1999. 32

[BCD+99] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas Jr, L. O'Connor, M. Peyravian, D. Stafford et N. Zunic : MARS - a candidate cipher for AES. *NIST AES Proposal*, 1999. 86

[BDM+12] Thierry P. Berger, Joffrey D'Hayer, Kevin Marquet, Marine Minier et Gaël Thomas : The GLUON Family : A Lightweight Hash Function Family

Based on FCSRs. *In Progress in Cryptology - AFRICACRYPT 2012*, volume 7374 de *LNCS*, pages 306–323. Springer, 2012. 1, 69

[BDPA08] Guido Bertoni, Joan Daemen, Michael Peeters et Gilles Van Assche : On the Indifferentiability of the Sponge Construction. *In EUROCRYPT 2008*, volume 4965 de *LNCS*, pages 181–197, 2008. 13, 79, 147

[BDPA11] G. Bertoni, J. Daemen, M. Peeters et G. Van Assche : The Keccak reference, January 2011. http://keccak.noekeon.org/. 13

[Bih93] E. Biham : New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). *In Advances in Cryptology - EUROCRYPT '93*, volume 765 de *LNCS*, pages 398–409. Springer, 1993. 128

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin et C. Vikkelsoe : Present : An ultra-lightweight block cipher. *In Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 de *LNCS*, pages 450–466. Springer, 2007. 18, 125, 129

[BMP87] E. Brickell, J. Moore et M. Purtill : Structure in the S-Boxes of the DES. *In Advances in Cryptology-CRYPTO'86*, pages 3–8. Springer, 1987. 21

[BMP09] T. P. Berger, M. Minier et B. Pousse : Software Oriented Stream Ciphers Based upon FCSRs in Diversified Mode. *In Progress in Cryptology - INDOCRYPT 2009*, volume 5922 de *LNCS*, pages 119–135. Springer, 2009. 51, 69, 71, 121, 125

[BMT13] T.P. Berger, M. Minier et G. Thomas : Extended Generalized Feistel Networks using Matrix Representation. *In Selected Areas in Cryptography - SAC 2013*, volume 8282 de *LNCS*. Springer, 2013. 2, 93, 103, 119, 122

[BNS14] Christina Boura, María Naya-Plasencia et Valentin Suder : Scrutinizing and Improving Impossible Differential Attacks : Applications to CLEFIA, Camellia, LBlock and Simon. *In Advances in Cryptology - ASIACRYPT 2014*, volume 8873 de *LNCS*, pages 179–199. Springer, 2014. 130

[BS90] E. Biham et A. Shamir : Differential Cryptanalysis of DES-like Cryptosystems. *In Advances in Cryptology - CRYPTO '90*, volume 537 de *LNCS*, pages 2–21. Springer, 1990. 31, 112

[BS91] E. Biham et A. Shamir : Differential Cryptanalysis of DES-like Cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991. 2

[BS97] E. Biham et A. Shamir : Differential Fault Analysis of Secret Key Cryptosystems. *In Advances in Cryptology - CRYPTO '97*, volume 1294 de *LNCS*, pages 513–525. Springer, 1997. 141

[BS01] A. Biryukov et A. Shamir : Structural Cryptanalysis of SASAS. *In Advances in Cryptology - EUROCRYPT '01*, volume 2045 de *LNCS*, pages 394–405. Springer, 2001. 111

[BS13] Andrey Bogdanov et Kyoji Shibutani : Generalized Feistel networks revisited. *Des. Codes Cryptography*, 66(1-3):75–97, 2013. 85

[BSS+13] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks et L. Wingers : The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. 24, 25, 125, 134

[BTLT14] Hélène Le Bouder, Gaël Thomas, Yanis Linge et Assia Tria : On Fault Injections in Generalized Feistel Networks. *In 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC*, pages 83–93. IEEE, 2014. 2, 133

[BW99] A. Biryukov et D. Wagner : Slide Attacks. *In Fast Software Encryption - FSE '99*, volume 1636 de *LNCS*, pages 245–259. Springer, 1999. 128

[Can07] C. De Cannière : *Analysis and Design of Symmetric Encryption Algorithms*. Thèse de doctorat, Katholieke Universiteit Leuven, 2007. 124

[CDK09] Christophe De Cannière, Orr Dunkelman et Miroslav Knezevic : KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. *In Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 de *LNCS*, pages 272–288. Springer, 2009. 128

[CV94] F. Chabaud et S. Vaudenay : Links Between Differential and Linear Cryptoanalysis. *In Advances in Cryptology - EUROCRYPT '94*, volume 950 de *LNCS*, pages 356–365. Springer, 1994. 112

[CWF07] H. Chen, W. Wu et D. Feng : Differential fault analysis on CLEFIA. *In Information and communications security*, pages 284–295. Springer, 2007. 145

[Dam89] Ivan Damgård : A Design Principle for Hash Functions. *In Advances in Cryptology - CRYPTO '89*, volume 435 de *LNCS*, pages 416–427. Springer, 1989. 13

[DH76] Whitfield Diffie et Martin E. Hellman : New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 6

[DKR97] J. Daemen, L. R. Knudsen et V. Rijmen : The block cipher square. *In Fast Software Encryption - FSE'97*, volume 1267 de *LNCS*, pages 149–165. Springer, 1997. 33

[DS09] Itai Dinur et Adi Shamir : Cube Attacks on Tweakable Black Box Polynomials. *In Advances in Cryptology - EUROCRYPT 2009*, volume 5479 de *LNCS*, pages 278–299. Springer, 2009. 74

[FIP01] FIPS 197 : Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T. 11, 15, 33, 112

[FO89] Philippe Flajolet et Andrew M. Odlyzko : Random Mapping Statistics. *In Advances in Cryptology - EUROCRYPT '89*, volume 434 de *LNCS*, pages 329–354. Springer, 1989. 79

[Gal63] Robert G. Gallager : *Low-density parity-check codes* . M.I.T. Press, Cambridge MA, 1963. 61

[Gam85] Taher El Gamal : A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. 6

[GC90] Henri Gilbert et Guy Chassé : A Statistical Attack of the FEAL-8 Cryptosystem. *In Advances in Cryptology - CRYPTO '90*, volume 537 de *LNCS*, pages 22–33. Springer, 1990. 31

[GGH97] Oded GOLDREICH, Shafi GOLDWASSER et Shai HALEVI : Public-Key Cryptosystems from Lattice Reduction Problems. *In Advances in Cryptology - CRYPTO '97*, volume 1294 de *LNCS*, pages 112–131. Springer, 1997. 6

[GK12] Mark GORESKY et Andrew KLAPPER : *Algebraic shift register sequences*. Cambridge University Press, 2012. 39

[GM01] H. GILBERT et M. MINIER : New Results on the Pseudorandomness of Some Blockcipher Constructions. *In Fast Software Encryption - FSE 2001*, volume 2355 de *LNCS*, pages 248–266. Springer, 2001. 108, 109

[GPP11] Jian GUO, Thomas PEYRIN et Axel POSCHMANN : The PHOTON Family of Lightweight Hash Functions. *In CRYPTO 2011*, volume 6841 de *LNCS*, pages 222–239. Springer, 2011. 69, 71

[GPPR11] J. GUO, T. PEYRIN, A. POSCHMANN et M. J. B. ROBSHAW : The LED Block Cipher. *In Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 de *LNCS*, pages 326–341. Springer, 2011. 17, 18

[HJ08] Martin HELL et Thomas JOHANSSON : Breaking the F-FCSR-H Stream Cipher in Real Time. *In Advances in Cryptology - ASIACRYPT 2008*, volume 5350 de *LNCS*, pages 557–569. Springer, 2008. 52, 54

[HKY08] S. HIROSE, H. KUWAKADO et H. YOSHIDA : SHA-3 Proposal : Lesamnta, october 2008. http://www.hitachi.com/rd/yrl/crypto/lesamnta/index.html. 87

[HR10] V. T. HOANG et P. ROGAWAY : On Generalized Feistel Networks. *In Advances in Cryptology - CRYPTO 2010*, volume 6223 de *LNCS*, pages 613–630. Springer, 2010. 85, 86, 87, 88

[HSH+06] D. HONG, J. SUNG, S. HONG, J. LIM, S. LEE, B. KOO, C. LEE, D. CHANG, J. LEE, K. JEONG, H. KIM, J. KIM et S. CHEE : HIGHT : A New Block Cipher Suitable for Low-Resource Device. *In Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 de *LNCS*, pages 46–59. Springer, 2006. 87

[IS12] T. ISOBE et K. SHIBUTANI : Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. *In Proc. Information Security and Privacy - ACISP 2012*, volume 7372 de *LNCS*, pages 71–86. Springer, 2012. 130

[ISO08] ISO/IEC : *ISO/IEC 9798-2 : Information technology – Security techniques – Entity authentication – Part 2 : Mechanisms using symmetric encipherment algorithms*. ISO/IEC, 2008. 127

[ISO11] ISO/IEC : *ISO/IEC 29192 : Information technology – Security techniques – Lightweight cryptography*. ISO/IEC, 2011. 18, 19

[ISSK09] Maryam IZADI, Babak SADEGHIYAN, Seyed Saeed SADEGHIAN et Hossein Arabnezhad KHANOOKI : MIBS : A New Lightweight Block Cipher. *In* Juan A. GARAY, Atsuko MIYAJI et Akira OTSUKA, éditeurs : *CANS*, volume 5888 de *LNCS*, pages 334–348. Springer, 2009. 21, 142

[KG93] Andrew KLAPPER et Mark GORESKY : 2-Adic Shift Registers. *In Fast Software Encryption, FSE'93*, volume 809 de *LNCS*, pages 174–178. Springer, 1993. 43

[KHL10] J. Kim, S. Hong et J. Lim : Impossible differential cryptanalysis using matrix method. *Discrete Mathematics*, 310(5):988–1002, 2010. 112

[KHS+03] J. Kim, S. Hong, J. Sung, C. Lee et S. Lee : Impossible Differential Cryptanalysis for Block Cipher Structures. *In Progress in Cryptology - INDOCRYPT 2003*, volume 2904 de *LNCS*, pages 82–96. Springer, 2003. ix, 32, 33

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann et Matthew J. B. Robshaw : PRINTcipher : A Block Cipher for IC-Printing. *In Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 de *LNCS*, pages 16–32. Springer, 2010. 128

[Knu98] L. Knudsen : DEAL - A 128-bit Block Cipher. *In NIST AES Proposal*, 1998. 32

[KW02] L. R. Knudsen et D. Wagner : Integral Cryptanalysis. *In Fast Software Encryption - FSE 2002*, volume 2365 de *LNCS*, pages 112–127. Springer, 2002. 110

[LP07] Gregor Leander et Axel Poschmann : On the Classification of 4 Bit S-Boxes. *In* Claude Carlet et Berk Sunar, éditeurs : *WAIFI*, volume 4547 de *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007. 124, 138

[LPPS07] G. Leander, C. Paar, A. Poschmann et K. Schramm : New Lightweight DES Variants. *In Fast Software Encryption, FSE 2007*, volume 4593, pages 196–210. Springer, 2007. 20, 21

[LR88] M. Luby et C. Rackoff : How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988. 12, 29, 108, 109

[LWD12] Yanjun Li, Wenling Wu et Le Dong : Integral distinguishers of JH and Grøstl-512. *Journal of Electronics (China)*, 29(1-2):94–102, 2012. 159

[LWLG09] Y. Luo, Z. Wu, X. Lai et G. Gong : A Unified Method for Finding Impossible Differentials of Block Cipher Structures. *IACR Cryptology ePrint Archive*, 2009:627, 2009. 33

[Mat93] M. Matsui : Linear Cryptoanalysis Method for DES Cipher. *In Advances in Cryptology - EUROCRYPT '93*, volume 765 de *LNCS*, pages 386–397. Springer, 1993. 31, 56, 61, 112

[Mat94] M. Matsui : The First Experimental Cryptanalysis of the Data Encryption Standard. *In* Yvo Desmedt, éditeur : *Advances in Cryptology - CRYPTO '94*, volume 839 de *LNCS*, pages 1–11. Springer, 1994. 31

[Mau02] U. M. Maurer : Indistinguishability of Random Systems. *In Advances in Cryptology - EUROCRYPT 2002*, volume 2332 de *LNCS*, pages 110–132. Springer, 2002. 29, 109

[McE78] Robert J McEliece : A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978. 6

[Mer89] Ralph C. Merkle : One Way Hash Functions and DES. *In Advances in Cryptology - CRYPTO '89*, volume 435 de *LNCS*, pages 428–446. Springer, 1989. 13

[MI08] A. Mitsuda et T. Iwata : Tweakable Pseudorandom Permutation from Generalized Feistel Structure. *In Provable Security, Second International Conference - ProvSec 2008*, volume 5324 de *LNCS*, pages 22–37. Springer, 2008. 109

[MT13] Marine Minier et Gaël Thomas : An Integral Distinguisher on Grøstl-512 v3. *In Progress in Cryptology - INDOCRYPT 2013*, volume 8250 de *LNCS*, pages 50–59. Springer, 2013. 159

[MV00] S. Moriai et S. Vaudenay : On the Pseudorandomness of Top-Level Schemes of Block Ciphers. *In Advances in Cryptology - ASIACRYPT 2000*, volume 1976 de *LNCS*, pages 289–302. Springer, 2000. 85, 86, 87, 88, 108, 109

[Nat77] National Bureau of Standards, U. S. Department of Commerce. *Data Encryption Standard*, 1977. 12, 20, 125, 140

[NR99] M. Naor et O. Reingold : On the Construction of Pseudorandom Permutations : Luby-Rackoff Revisited. *J. Cryptology*, 12(1):29–66, 1999. 85, 86, 109

[Nyb96] K. Nyberg : Generalized Feistel Networks. *In Advances in Cryptology - ASIACRYPT '96*, volume 1163 de *LNCS*, pages 91–104. Springer, 1996. 88

[PK14] Léo Perrin et Dmitry Khovratovich : Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64. *IACR Cryptology ePrint Archive*, 2014:223, 2014. x, xi, 69, 78, 79, 80, 81, 147

[PK15] Léo Perrin et Dmitry Khovratovich : Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64. *In Fast Software Encryption - FSE 2014*, volume to appear de *LNCS*, pages 3–18. Springer, 2015. x, xi, 69, 78, 79, 80, 81, 147

[Riv98] R. L. Rivest : A Description of the RC2(r) Encryption Algorithm. Network Working Group, RFC 2268, march 1998. http://tools.ietf.org/html/rfc2268. 86

[Riv09] M. Rivain : Differential Fault Analysis on DES Middle Rounds. *In Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 de *LNCS*, pages 457–469. Springer, 2009. 142

[RRSY98] R. L. Rivest, M. J. B. Robshaw, R. Sidney et Y. L. Yin : The RC6 Block Cipher, august 1998. http://people.csail.mit.edu/rivest/pubs/RRSY98.pdf. 87

[RSA78] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman : A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 6

[Saa11] Markku-Juhani O. Saarinen : Cryptographic Analysis of All 4 x 4 - Bit S-Boxes. Cryptology ePrint Archive, Report 2011/218, 2011. 124, 138

[Sha49] C. E. Shannon : Communication Theory of Secrecy Systems. *The Bell System Technical Journal*, 28(4):656–715, oct 1949. 8, 9

[SHS12] SHS : Secure Hash Standard. *In FIPS PUB 180-4, Federal Information Processing Standards Publication*, 2012. 86

[SIH+11] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita et T. Shirai : Piccolo : An ultra-lightweight blockcipher. *In Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 de *LNCS*, pages 342–357. Springer, 2011. ix, 23, 87, 129, 130, 134

[SM10] T. Suzaki et K. Minematsu : Improving the Generalized Feistel. *In Fast Software Encryption - FSE 2010*, volume 6147 de *LNCS*, pages 19–39. Springer, 2010. xi, 1, 26, 85, 87, 88, 89, 90, 91, 95, 97, 108, 109, 110, 112, 113, 122, 147

[SMMK12] T. Suzaki, K. Minematsu, S. Morioka et E. Kobayashi : TWINE : A Lightweight Block Cipher for Multiple Platforms. *In Selected Areas in Cryptography - SAC 2012*, volume 7707 de *LNCS*, pages 339–354. Springer, 2012. 21, 25, 26, 87, 125, 129, 143

[SPGQ06] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld et Jean-Jacques Quisquater : SEA : A Scalable Encryption Algorithm for Small Embedded Applications. *In CARDIS*, volume 3928 de *LNCS*, pages 222–236. Springer, 2006. 12

[SSA+07] T. Shirai, K. Shibutani, T. Akishita, S. Moriai et T. Iwata : The 128-Bit Blockcipher CLEFIA (Extended Abstract). *In Fast Software Encryption - FSE 2007*, volume 4593 de *LNCS*, pages 181–195. Springer, 2007. 19, 87, 144

[SW12] Y. Sasaki et L. Wang : Meet-in-the-Middle Technique for Integral Attacks against Feistel Ciphers. *In Selected Areas in Cryptography - SAC 2012*, volume 7707 de *LNCS*, pages 234–251. Springer, 2012. 111

[TF08] J. Takahashi et T. Fukunaga : Improved differential fault analysis on CLEFIA. *In Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*, pages 25–34. IEEE, 2008. 145

[TG91] Anne Tardy-Corfdir et Henri Gilbert : A Known Plaintext Attack of FEAL-4 and FEAL-6. *In Advances in Cryptology - CRYPTO '91*, volume 576 de *LNCS*, pages 172–181. Springer, 1991. 31

[TQ09] Tian Tian et Wen-Feng Qi : Linearity properties of binary FCSR sequences. *Des. Codes Cryptography*, 52(3):249–262, 2009. 1, 54, 55, 56, 67, 68, 147

[WSJ15] Hui Wang, Paul Stankovski et Thomas Johansson : A generalized birthday approach for efficiently finding linear relations in ℓ-sequences. *Des. Codes Cryptography*, 74(1):41–57, 2015. 54, 55, 56

[WZ11] W. Wu et L. Zhang : LBlock : A Lightweight Block Cipher. *In Applied Cryptography and Network Security - ACNS 2011*, volume 6715 de *LNCS*, pages 327–344, 2011. 21, 125, 129

[YI13] S. Yanagihara et T. Iwata : Improving the Permutation Layer of Type 1, Type 3, Source-Heavy, and Target-Heavy Generalized Feistel Structures. *IEICE Trans.*, 96-A(1):2–14, 2013. xi, 90, 91, 92, 95

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang et Ingrid Verbauwhede : RECTANGLE : A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms. *IACR Cryptology ePrint Archive*, 2014:84, 2014. 17, 19

[Zhe97] Y. Zheng : The SPEED Cipher. *In Financial Cryptography - FC 1997*, volume 1318 de *LNCS*, pages 71–90. Springer, 1997. 86

[ZMI89] Y. Zheng, T. Matsumoto et H. Imai : On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. *In Advances in Cryptology - CRYPTO '89*, volume 435 de *LNCS*, pages 461–480. Springer, 1989. 85, 87, 88, 109

[ZSW+12] W. Zhang, B. Su, W. Wu, D. Feng et C. Wu : Extending Higher-Order Integral : An Efficient Unified Algorithm of Constructing Integral Distinguishers for Block Ciphers. *In Applied Cryptography and Network Security - ACNS 2012*, volume 7341 de *LNCS*, pages 117–134. Springer, 2012. 34

[ZW14] Lei Zhang et Wenling Wu : Differential analysis of the Extended Generalized Feistel Networks. *Inf. Process. Lett.*, 114(12):723–727, 2014. 113

Appendices

Distingueur intégral sur Grøstl-512 v3

Cette annexe présente un travail réalisé durant cette thèse et publié à Indocrypt 2013 [MT13] sur une application des attaques intégrales (cf. section 3.3.2).

Il s'agit d'un distingueur intégral amélioré utilisant 2^{913} évaluations contre une version à 11 tours de la fonction de compression du candidat SHA-3 malheureux Grøstl-512 avec les paramètres du tour 3. Le résultat d'origine présenté dans [LWD12] est amélioré par l'utilisation de différentes propriétés intégrales.

An Integral Distinguisher on Grøstl-512 v3*

Marine Minier[1] and Gaël Thomas[2]

[1] Université de Lyon, INRIA
INSA-Lyon, CITI, F-69621, Villeurbanne, France
marine.minier@insa-lyon.fr
[2] XLIM (UMR CNRS 7252), Université de Limoges
123 avenue Albert Thomas, 87060 Limoges Cedex - France
gael.thomas@unilim.fr

Abstract. This paper presents an improved integral distinguisher using 2^{913} computations against an 11-round version of the compression function of the SHA-3 candidate Grøstl-512 with the round 3 parameters. The original result presented in [18] was enhanced through the use of different integral properties.
Keywords: Hash functions; cryptanalysis; integral distinguishers; SHA-3 competition

1 Introduction

The entire cryptographic community has been waiting until last October for the outcome of the SHA-3 competition[3]. Among the finalists, Grøstl [13] is a surviving proposal designed by P. Gauravaram et al. It is based on AES transformations and outputs 256 or 512 bits of hash according to the Grøstl-256/512 version.

During the second round, Grøstl has attracted a significant amount of cryptanalysis. For example, T. Peyrin presented rebound distinguishers against full version of the compression function Grøstl-256 [20]. This is one of the main reasons that forced the Grøstl designers to modify the parameters of Grøstl for the SHA-3 round 3. In the remainder of this paper, we refer to this last version as Grøstl v3 whereas the previous version is called Grøstl v2. Results against the compression function of Grøstl-256 v3 include a semi-free-start collision against 6 rounds that uses 2^{180} computations [22], a pseudo preimage against 8 rounds that uses $2^{507.32}$ computations [21], a rebound distinguisher against 10 rounds that uses 2^{392} computations [14], and an integral distinguisher against 11 rounds that requires 2^{953} computations [18]. In this paper, we improve this last distinguisher leading to an integral distinguisher on 11 rounds of the compression function of Grøstl-512 v3 with a complexity of 2^{913} computations.

This paper is organized as follows: Section 2 introduces related work, notations of the paper and the description of Grøstl-512; Section 3 describes the

* This work was partially supported by the French National Agency of Research: ANR-11-INS-011.
[3] see for example: http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

integral distinguisher against Grøstl-512 v3 reduced to 11 rounds and finally Section 4 concludes this paper.

2 Related Work and Notations

2.1 Integral Attacks

Integral cryptanalysis was first introduced against the Square block cipher in the original paper [6] in the unknown key setting to retrieve information on some key bytes. Then, it was applied to AES in the original submission paper [7, 8]. The original integral property on AES was extended by one round by Ferguson et al. in [10].

After those first attacks, many ciphers especially the ones that use a SPN structure have been studied with regard to this kind of distinguishers. Among all the integral cryptanalyses proposed in the literature, we could cite the attacks against SAFER [2], CRYPTON [9] and more recently on PRESENT [5]. The different Rijndael versions (Rijndael-192 and Rijndael-256) have also been attacked using integral properties [15, 11]. Other contributions also analyze the general framework of Integral cryptanalysis and especially focus on the conditions that a block cipher must fulfill to be attacked using this method [17, 3]. In [17], L. Knudsen and D. Wagner analyze integral cryptanalysis as a dual to differential attacks particularly applicable to block ciphers with bijective components. A first-order integral cryptanalysis considers a particular collection of m words in the plaintexts and ciphertexts that differ on a particular component. The aim of this attack is thus to predict the values in the sums (i.e. the integral) of the chosen words after a certain number of rounds of encryption. The same authors also generalize this approach to higher-order integrals: the original set to consider becomes a set of m^d vectors which differ in d components and where the sum of this set is predictable after a certain number of rounds. The sum of this set is called a d-th order integral.

More recently, in [16] Integral cryptanalysis has been proposed in the new model called known key settings where the key is known to the attacker. In the same settings, compression functions of hash functions could also be analyzed and some distinguishers have been proposed against SHA-3 candidates also using integral properties. Consider for instance integral distinguishers on the compression functions of Hamsi-256 [1, 19] and Keccak [4].

2.2 Notations

In the remainder of this paper, we use the consistent notations introduced in [17] and extend them for expressing word-oriented integral attacks. For a d-th order integral, we have:

- The symbol 'C' (for "Constant") in the i-th entry, means that the values of all the i-th words in the collection of texts are equal.
- The symbol 'P' (for "Permutation") means that all words in the collection of texts are different.

- The symbol '?' means that the sum of words cannot be predicted.
- The symbol 'P^d' corresponds to the components that participate in a d-th order integral, i.e. if a word can take m different values then $\mathcal{P}^d$ means that in the integral, the particular word takes all values exactly m^{d-1} times.
- The symbol '0' means that the sum of all values is zero.

2.3 Description of the Grøstl-512 Hash Function

Grøstl [12] is a SHA-3 candidate designed by Guaravaram *et al.*, notably Grøstl-256 outputs hash values of lengths 224 and 256 bits whereas Grøstl-512 outputs hash values of lengths 384 or 512 bits. We focus on Grøstl-512. It is an iterated hash function with a compression function built from two distinct permutations P and Q. A t-block message (after padding) $(M_1, \cdots, M_t)$ is hashed by computing successive chaining values H_i using the compression function $f(H_{i-1}, M_i)$ and then applying the output transformation $g(H_t)$ as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, M_i) = H_{i-1} \oplus P(H_{i-1} \oplus M_i) \oplus Q(M_i) \text{ for } 1 \leq i \leq t \\ h &= g(H_t) = trunc(H_t \oplus P(H_t)) \end{aligned}$$

where $trunc(\cdot)$ denotes the function that truncate its input by returning only the last 384 (or 512) bits.

The two permutations P and Q are constructed using the wide trail strategy, their design is very similar to AES with a fixed key input. Both permutations of the compression function of Grøstl-512 act on a 1024-bit state represented as a 8×16 matrix of bytes and have 14 rounds. The round transformations of Grøstl-512 are the following ones:

- AddRoundConstant (AC) adds a round-dependent constant to the state of P and Q.
- SubBytes (SB) is the non-linear layer that applies the AES Sbox to each byte of the state.
- ShiftBytes (ShB) rotates the bytes of row j in the following way: 0 for $j = 1$, 1 for $j = 2, \cdots 6$ for $j = 7$ and 11 for $j = 8$ for the P permutation and the shifted values are 1, 3, 5, 11, 0, 2, 4, 6 for the Q permutation.
- MixBytes (MB) is the linear diffusion layer where each column of the state is multiplied by a constant matrix B.

Note that the differences between Grøstl-512 v2 and the new version of Grøstl-512, Grøstl-512 v3 are localized in the two transformations AddRoundConstant and ShiftBytes.

3 Description of the 11-round Distinguisher of the Grøstl-512 v3 Compression Function

3.1 The Divide-and-Conquer Method to Find Integral Properties

In [18], the authors propose a divide-and-conquer method to efficiently find integral properties on several rounds of an AES-like cipher or hash function. Their

method works as follows: first, find some integral properties that sum to 0 before the last MixColumns (or MixBytes) operation. Then, combine several integral properties that sum together to 0 on a complete column and apply the MixColumns (or MixBytes) operation to fulfill the integral property. One thus obtain finally an integral property on a complete number of rounds.

3.2 Integral Properties for P and Q in the Forward Direction

We apply this method to the case of Grøstl-512 v3 and we find, for P and Q, the integral properties for 3.5 rounds shown in Fig. 1. In fact, we find the following integral properties with two active bytes for P:

- when the two active bytes are in position (0,0) and (1,1), then after 3.5 rounds, the bytes on three shifted columns have their sum equal to 0. This property also holds for two active bytes at positions (3,0) and (4,1); (5,0) and (6,1); (0,1) and (5,6); (7,1) and (0,6); (7,0) and (1,6). All those properties lead to three zero-sum shifted columns.

and the following integral properties with two active bytes for Q:

- when the two active bytes are in position (0,0) and (5,1), then after 3.5 rounds, the bytes on three shifted columns have their sum equal to 0. This property also holds for two active bytes at positions (1,0) and (6,1); (2,0) and (7,1); (4,0) and (0,1); (3,0) and (0,6); (3,1) and (4,6); (4,1) and (2,6). All those properties lead to three zero-sum shifted columns.

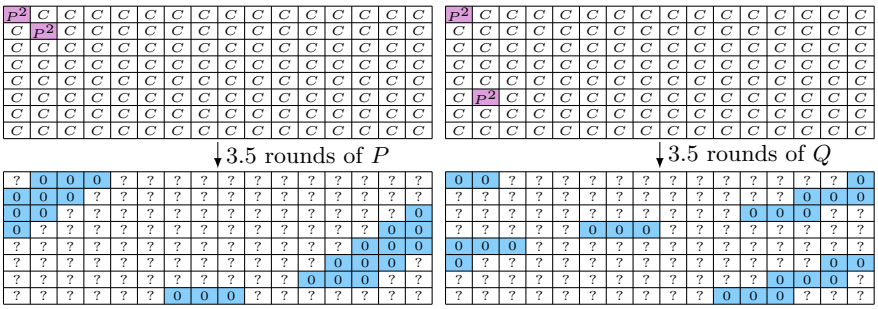


Fig. 1. The 3.5-round P integral property with 2 active bytes on the left and the 3.5-round Q integral property with 2 active bytes on the right.

Thus, we can combine those two bytes 3.5-round integral properties to mount integral properties on 4 rounds with respectively 12 active bytes for P and 14 active bytes for Q using the divide-and-conquer method of [18]. The deduced integral properties for P and Q are shown in Fig. 2. We are hence able to

distinguish P and Q from random permutations using respectively 2^{96} and 2^{112} chosen texts that sum to 0 at byte level on three particular columns (i.e. on 192 bits) after four applications of the round function of P (respectively Q). This distinguisher has a complexity equal to 2^{96} cipher operations for P and 2^{112} cipher operations for Q.

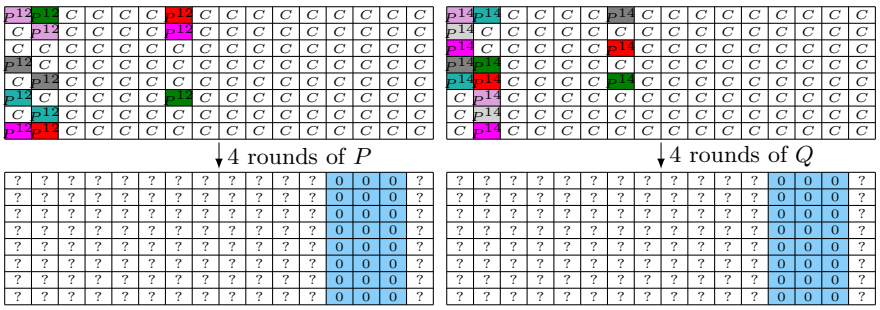


Fig. 2. The 4-round P integral property with 12 active bytes on the left and the 4-round Q integral property with 14 active bytes on the right.

Following the work of [15], we extended by two rounds at the beginning those 4-round integral properties using first a 24-th order integral property and second a 104-th order integral property as shown on Fig 3. We were able to distinguish P and Q from random permutations using 2^{832} chosen texts that sum to 0 at byte level on three particular columns (i.e. on 192 bits) after six applications of the round function of P (respectively Q). This distinguisher has a complexity equal to 2^{832} cipher operations.

3.3 Integral Properties for P and Q in the Backward Direction

Let us now analyze which integral properties exist for the backward direction. We use the backward integral property already described in [19], a 2nd order integral property on 3 backward rounds presented in Fig. 4.

This property leads to a distinguisher on 3 backward rounds where the sums taken at byte level over all the inputs on the three shifted columns marked in blue in Fig. 4 are equal to 0. It requires 2^{16} chosen texts to work and has a complexity equal to 2^{16} cipher operations. This property could be extended by first one round and second two backward rounds at the beginning using a 80th order integral property as shown on Fig. 5. This leads to an integral distinguisher that uses 2^{640} chosen texts with a complexity equal to 2^{640} cipher operations to test if the sums taken at byte level over $3 \times 8 \times 8 = 192$ bits are equal to 0 or not.

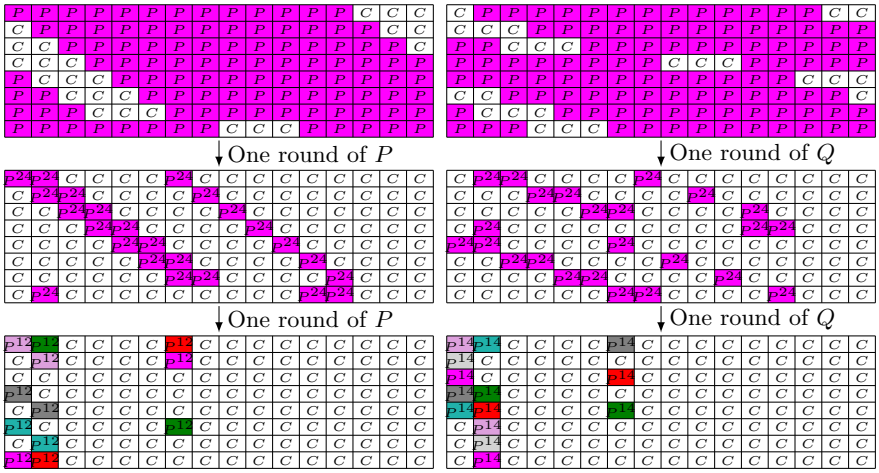


Fig. 3. The two added rounds of the integral property with 104 active bytes, for P on the left and Q on the right.

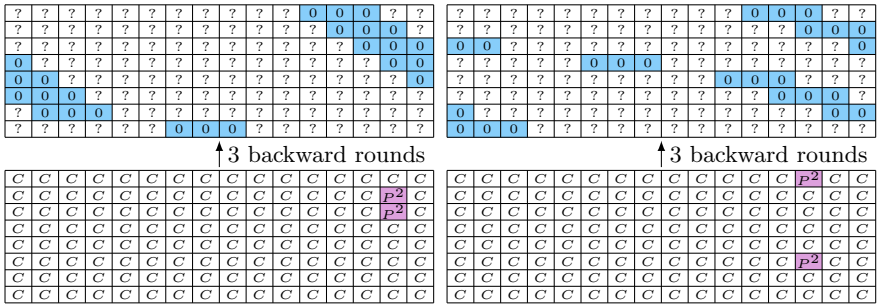


Fig. 4. The 2nd order Integral property on 3 backward rounds of P (left) and Q (right).

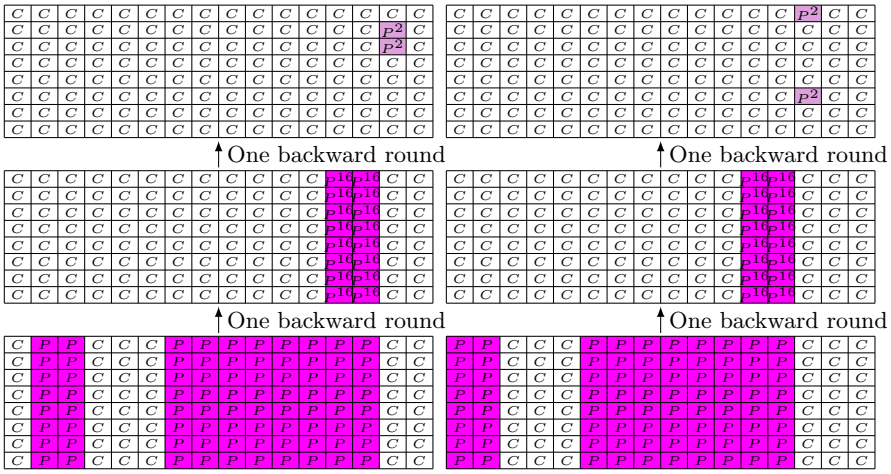


Fig. 5. The two added rounds of the P integral property with 80 active bytes on the left and the two added rounds of the Q integral property with 80 active bytes on the right.

3.4 Distinguisher on 11 Rounds of the Compression Function of Grøstl-512 v3

We combined those two properties (in the backward and in the forward directions) starting from both the middle of P and the middle of Q to build a structural property on the compression function of Grøstl-512 when 11 rounds are considered (see Fig. 6). For the permutation P, start from the middle with 2^{912} middletexts with 114 active bytes (the other are taken equal to a constant) then, go backward on five rounds to obtain inputs that sum to 0 on 3 shifted columns and go forward on 6 rounds to obtain outputs that sum to 0 on 3 columns. Do the same for the permutation Q. Using Q, get the 2^{912} corresponding M_t messages. Using those messages and the inputs of P, compute the corresponding 2^{912} H_{t-1} values. Those 2^{912} values also verify that their sums taken over all 2^{912} values on 6 bytes are equal to 0 (due to the linearity of the XOR operation and considering the intersection of all 0-sum bytes). Considering the knowledge of H_{t-1}, of the outputs of P and of the outputs of Q, the corresponding H_t values are such that the sums taken over all the 2^{912} values on the intersection of the 6 common bytes (for the backward direction) and of the 3 columns (for the forward direction) are equal to 0. In other words, the sum taken over all the 2^{912} outputs of the compression function is null at 4 byte positions whereas the corresponding inputs H_{t-1} and M_t have 0-sum on 6 bytes (see Fig. 7).

Thus, we have exhibited a structural property of the Grøstl-512 compression function when P and Q are limited to 11 rounds. The computational cost of this property is about 2^{913} cipher operations with modest memory requirements to

find some 0-sums at particular positions (4 bytes at the output of the compression function and 6 bytes at the input). This new structural property improves the one described in [18] that reaches 11 rounds also with a complexity equal to 2^{953} cipher operations.

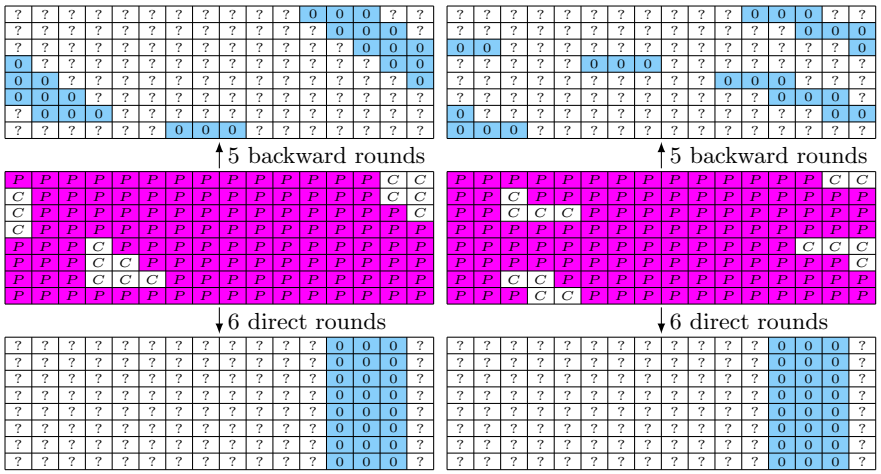


Fig. 6. Complete property on 11 rounds of P (on the left) and of Q (on the right) starting from the middle with a 114th order integral property.

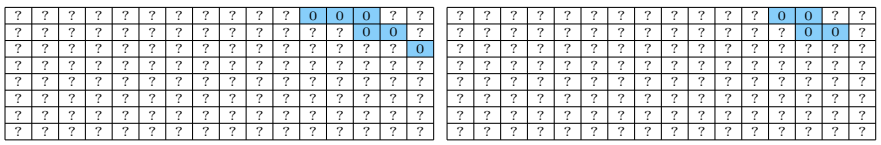


Fig. 7. Position of the 6 zero-sum bytes for H_{t-1} and M_t (left), and the 4 zero-sum bytes for H_t (right).

4 Conclusion

In this paper, we have improved the integral properties exhibited on the compression function of Grøstl-512 v3 presented in [18]. Table 1 sums up the main distinghuishers against the compression function of Grøstl-512 v3.

Table 1. Summary of distinguishers against the compression function of Grøstl-512 v3.

Nb rounds	Type of Attack	Time	Memory	Source
6	Semi-free-start Collision	2^{180}	2^{64}	[22]
8	Pseudo Preimage	$2^{507.32}$	2^{507}	[21]
10	Rebound Distinguisher	2^{392}	2^{64}	[14]
11	Integral Distinguisher	2^{953}	small	[18]
11	Integral Distinguisher	2^{913}	small	this paper

References

1. Jean-Philippe Aumasson, Emilia Käsper, Lars R. Knudsen, Krystian Matusiewicz, Rune Steinsmo Ødegård, Thomas Peyrin, and Martin Schläffer. Distinguishers for the compression function and output transformation of hamsi-256. In *Information Security and Privacy - ACISP 2010*, volume 6168 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2010.
2. Alex Biryukov, Christophe De Cannière, and Gustaf Dellkrantz. Cryptanalysis of SAFER++. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 195–211. Springer, 2003.
3. Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.
4. Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and *Luffa*. In *Fast Software Encryption - FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.
5. Baudoin Collard and François-Xavier Standaert. A statistical saturation attack against the block cipher PRESENT. In *Topics in Cryptology - CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2009.
6. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption - FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.
7. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. In *The First Advanced Encryption Standard Candidate Conference*. N.I.S.T., 1998.
8. Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.
9. Carl D'Halluin, Gert Bijnens, Vincent Rijmen, and Bart Preneel. Attack on Six Rounds of Crypton. In Lars R. Knudsen, editor, *Fast Software Encryption - FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 1999.
10. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In Bruce Schneier, editor, *Fast Software Encryption - FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2001.
11. Samuel Galice and Marine Minier. Improving integral attacks against Rijndael-256 up to 9 rounds. In *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.
12. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST (Round 1/2), 2008.

13. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl addendum. Submission to NIST (Round 2), 2009.
14. Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved rebound attack on the finalist Grøstl. In *Fast Software Encryption - FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2012.
15. Jorge Nakahara Jr., Daniel Santana de Freitas, and Raphael C.-W. Phan. New multiset attacks on Rijndael with large blocks. In *Progress in Cryptology - Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 2005.
16. Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.
17. Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.
18. Yanjun Li, Wenling Wu, and Le Dong. Integral distinguishers of JH and Grøstl-512. *Journal of Electronics (China)*, 29:94–102, 2012.
19. Marine Minier, Raphael C.-W. Phan, and Benjamin Pousse. Integral distinguishers of some SHA-3 candidates. In *Cryptology and Network Security - CANS 2010*, volume 6467 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 2010.
20. Thomas Peyrin. Improved differential attacks for ECHO and Grøstl. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 370–392. Springer, 2010.
21. Shuang Wu, Dengguo Feng, Wenling Wu, Jian Guo, Le Dong, and Jian Zou. (Pseudo) Preimage attack on round-reduced grøstl hash function and others. volume 7549 of *Lecture Notes in Computer Science*, pages 127–145. Springer, 2012.
22. Martin Schläffer. Updated differential analysis of Grøstl, Grøstl website (January 2011).

Design et Analyse de sécurité pour les constructions en cryptographie symétrique

Résumé : Les travaux réalisés au cours de cette thèse se situent au carrefour de la cryptographie symétrique et du monde des environnements contraints. Le but de cette cryptographie, dite cryptographie à bas coût, est de fournir et d'évaluer des algorithmes symétriques pouvant être implémentés sur des systèmes très limités en ressources. Les contributions de cette thèse portent d'une part sur l'évaluation de la sécurité des *registres à décalage à rétroaction avec retenue* (FCSR) face à de nouvelles attaques et d'autre part sur une vision unifiée des différents *schémas de Feistel généralisés* (GFN) qui permet de mieux cerner leurs propriétés cryptographiques. Ces études ont donné lieu à deux nouveaux algorithmes à bas coût ; d'une part GLUON une fonction de hachage à base de FCSR et d'autre part le chiffrement LILLIPUT basé sur une famille étendant plus avant la notion de schéma de Feistel généralisé. Enfin, une méthode générique permettant de réaliser des attaques différentielles en fautes sur des schémas de Feistel généralisés est esquissée.

Mots clés : cryptographie symétrique, cryptographie à bas coût, FCSR, schéma de Feistel généralisé, GFN.

Design and Security Analysis for constructions in symmetric cryptography

Abstract : The work done during this Ph.D. lies at the crossroads of symmetric cryptography and constraints environments. The goal of such cryptography, called lightweight cryptography, is to propose and evaluate symmetric algorithms that can be implemented on very ressource limited devices. The contributions of this thesis are first on the security evaluations of *feedback with carry shift registers* (FCSRs) to some new attacks and second on a unified vision of *generalized Feistel networks* (GFNs) that allows to better understand their cryptographic properties. These studies gave rise to two new lightweight algorithms : first GLUON a hash function based upon FCSRs and second the cipher LILLIPUT based upon a family further extending the notion of generalized Feistel network. Finally, a generic method for carrying out a differential fault attack on GFNs is outlined.

Keywords : symmetric cryptography, lightweight cryptography, feedback with carry shift register, FCSR, generalized Feistel network, GFN.

XLIM - UMR CNRS n° 7252
123, avenue Albert Thomas - 87049 LIMOGES