SGDSN/ANSSI — Université de Versailles Saint-Quentin

Laboratoire de Cryptographie — Laboratoire PRiSM

ÉCOLE DOCTORALE
STV

T H È S E

présentée pour obtenir le grade de

Docteur de l'Université de Versailles

Spécialité : Informatique

soutenue publiquement par

Jean-René Reinhard

Étude de Primitives Cryptographiques Symétriques : Chiffrements par Flot et Fonctions de Hachage

Le 14 décembre 2011

Jury :

Rapporteurs :	Anne Canteaut	-	INRIA
	Pierre-Alain Fouque	-	ENS Paris
Directeur de thèse :	Antoine Joux	-	DGA et UVSQ
Examinateurs :	Henri Gilbert	-	ANSSI
	Louis Goubin	-	UVSQ
	Pascal Paillier	-	CryptoExperts
	Serge Vaudenay	-	EPFL
	Marion Videau	-	LORIA

Travaux effectués au Laboratoire de Cryptologie de l'ANSSI
Agence Nationale de la Sécurité des Systèmes d'Information
51, boulevard de la Tour-Maubourg 75700 PARIS 07 SP

SGDSN/ANSSI — Université de Versailles Saint-Quentin

Laboratoire de Cryptographie — Laboratoire PRiSM

ÉCOLE DOCTORALE
STV

T H È S E

présentée pour obtenir le grade de

Docteur de l'Université de Versailles

Spécialité : Informatique

soutenue publiquement par

Jean-René Reinhard

Étude de Primitives Cryptographiques Symétriques : Chiffrements par Flot et Fonctions de Hachage

Le 14 décembre 2011

Jury :

Rapporteurs :	Anne Canteaut	-	INRIA
	Pierre-Alain Fouque	-	ENS Paris
Directeur de thèse :	Antoine Joux	-	DGA et UVSQ
Examinateurs :	Henri Gilbert	-	ANSSI
	Louis Goubin	-	UVSQ
	Pascal Paillier	-	CryptoExperts
	Serge Vaudenay	-	EPFL
	Marion Videau	-	LORIA

Travaux effectués au Laboratoire de Cryptologie de l'ANSSI
Agence Nationale de la Sécurité des Systèmes d'Information
51, boulevard de la Tour-Maubourg 75700 PARIS 07 SP

Remerciements

Au moment de remercier tout ceux sans qui ces travaux n'auraient pu aboutir, les mots me manquent pour exprimer toute ma gratitude et mon admiration.

En premier lieu, je tiens à remercier Antoine Joux de m'avoir fait l'honneur d'accepter de m'encadrer pendant cette thèse. Son talent et son expertise sont bien connus de toute la communauté des cryptologues. C'est l'enthousiasme avec lequel il mène ses travaux qui m'a le plus marqué.

Je remercie Anne Canteaut et Pierre-Alain Fouque de m'avoir fait l'honneur d'accepter la lourde tâche de rapporteur. Je remercie également Henri Gilbert, Louis Goubin, Pascal Paillier, Serge Vaudenay et Marion Videau pour avoir accepté de prendre part à mon jury de thèse.

La qualité de ce mémoire doit beaucoup au travail de ses relecteurs, qui ont traqué sans relâche, jusqu'à la dernière minute, les nombreuses coquilles et approximations que j'y avais glissées par mégarde. Mes remerciements vont donc à Joana Treger-Marim, Henri Gilbert, Thomas Fuhr, ainsi qu'à Anne Canteaut qui a joint à son rapport les fruits d'une relecture extensive.

Ces travaux ont été réalisés au sein du laboratoire de cryptologie de l'ANSSI. Ce laboratoire et de manière plus large la sous-direction ACE constituent un environnement très épanouissant et motivant : épanouissant en raison des compétences et des qualités humaines des personnes qui y travaillent, motivant car on s'y trouve confronté à un éventail très complet de problématiques. Je remercie plus particulièrement mes collègues passés et présents du laboratoire de cryptographie, Guillaume Poupard, Michel Mitton, Éliane Jaulmes, Gwenaëlle Martinet, Frédéric Muller, Sébastien Kunz-Jacques, Mathieu Baudet, Emmanuel Bresson, Thomas Fuhr, Marion Videau, Benoît Chevallier-Mames, Joana Treger-Marim, Aurélie Bauer, Yannick Seurin, Henri Gilbert et Jean-Pierre Flori, mes collègues du laboratoire composant, Karim Khalfallah, Jean-Claude Bourrée, Victor Lomné, Thomas Roche et Adrian Thillard. Je remercie également tout ceux qui à ACE s'intéressent de près ou de loin à la cryptologie, à ses applications et à son déploiement, et avec lesquels se nouent des échanges très fructueux, notamment Pierre-Michel Ricordel, Olivier Levillain, Chaouki Kasmi, Arnaud Ebalard, Guillaume Valadon et Éric Jaeger.

L'excellence de cet environnement de travail privilégié doit beaucoup à Florent Chabaud et Loïc Duflot, qui ont toujours considéré la recherche comme une activité essentielle des laboratoires de l'ANSSI, ainsi qu'à la direction qui donne aux agents la possibilité de se former par la recherche.

Une partie des travaux de cette thèse a été réalisée dans le cadre des projets collaboratifs SAPHIR et Saphir2. La soumission et la défense de la fonction de hachage Shabal furent des expériences enrichissantes, que j'ai eu l'honneur de partager avec Jean-François Misarsky, Thomas Fuhr, Benoît Chevallier-Mames, Emmanuel Bresson, Marion Videau, Pascal Paillier, Aline Gouget, Christophe Clavier, Thomas Pornin, Céline Thuillet, Thomas Icart, María Naya Plasencia et Anne Canteaut.

Enfin, je tiens à remercier ma famille pour son soutien inconditionnel et Marjorie, avec qui je partage tant de choses depuis tant d'années, pour ses encouragements et sa patience pendant la rédaction de ce mémoire de thèse.

Abstract : The use of symmetric cryptographic primitives is widely spread in many concrete applications requiring confidentiality and integrity. This work has been undertaken in the context of two international competitions, eSTREAM and SHA-3, which encouraged and enticed the design and cryptanalysis of many algorithms of two cryptographic algorithms families, respectively stream ciphers and hash functions.

In a first part, we study stream ciphers from a cryptanalysis point of view. We first present attack principles that apply to many stream ciphers. Through two examples, we expose at greater length algebraic and differential attacks. We perform an overview of algebraic attacks applied to the filtered LFSR and present practical time chosen IV attacks against the VEST stream cipher family, one of the candidate of the eSTREAM competition selected for phase 2.

In a second part, we study the conception of cryptographic hash functions. We participated to the submission of one candidate to the SHA-3 competition : Shabal. We focus in this document on the security of its domain extender in the indifferentiability framework, when the compression function is idealized. We also present a framework which allows to take into account non ideal properties of compression functions and expand the proof of Shabal domain extender in this model.

Keywords : stream cipher cryptanalysis, filtered LFSR, algebraic attacks, VEST, hash function domain extender, indifferentiability, Shabal.

Résumé : L'utilisation de primitives cryptographiques symétriques reste incontournable dans tout système mettant en oeuvre des mécanismes cryptographiques. Les travaux de cette thèse s'inscrivent dans le contexte de deux compétitions internationales, eSTREAM et SHA-3, qui ont stimulé le développement et la cryptanalyse de deux familles de primitives cryptographiques symétriques, respectivement les algorithmes de chiffrement par flot et les fonctions de hachage.

Dans une première partie, nous traitons d'algorithmes de chiffrement par flot du point de vue du cryptanalyste. Nous présentons des principes d'attaque s'appliquant de manière générale à de nombreux algorithmes de chiffrement par flot. À travers deux exemples, nous développons plus particulièrement la présentation des attaques algébriques et des attaques différentielles. Nous réalisons un état de l'art des attaques algébriques appliquées au registre linéaire filtré et présentons des attaques différentielles à IV choisis de complexité pratiquable contre la famille d'algorithmes VEST, soumise à la compétition eSTREAM et acceptée en phase 2.

Dans une deuxième partie, nous étudions la construction de fonctions de hachage cryptographiques. Nous avons travaillé à la conception d'un candidat à la compétition SHA-3 : Shabal. On se concentre dans ce mémoire sur la sécurité de son algorithme d'extension de domaine dans le modèle de l'indifférentiabilité d'un oracle aléatoire, en idéalisant la fonction de compression. On présente également une modélisation permettant de prendre en compte des propriétés non-idéales de la fonction de compression utilisée et on étend la preuve de l'extension de domaine de Shabal dans ce cadre.

Mots clés : Cryptanalyse d'algorithme de chiffrement par flot, registre linéaire filtré, attaques algébriques, VEST, extension de domaine de fonction de hachage, indifférentiabilité, Shabal.

Table des matières

Table des figures

Liste des tableaux

Introduction

La *cryptologie*, ou science du secret, trouve ses origines dans l'antiquité. L'étude et l'emploi de mécanismes cryptographiques ont longtemps été réservés à des usages militaires ou diplomatiques. L'histoire est parsemée d'exemples de schémas cryptographiques dont la mise en défaut a eu des répercussions importantes. Paradoxalement, c'est l'avènement d'une recherche académique en cryptographie, à la fin des années 1970 avec la publication d'un standard de chiffrement, le DES, et la découverte de la cryptographie à clé publique, qui a permis d'améliorer considérablement la qualité des techniques cryptographiques et a permis à ce domaine d'atteindre une certaine maturité. De plus, le développement des communications numériques, internet, GSM, etc, a conduit à une généralisation de l'emploi de ces techniques. Aujourd'hui, la sécurité de nombreuses applications, comme le commerce électronique, reposent sur la cryptologie moderne.

La cryptologie permet de *protéger l'information*. Elle permet d'atteindre des objectifs de sécurité comme :

- la confidentialité : une information est rendue inintelligible sauf pour les utilisateurs légitimes ;
- l'intégrité : une information ne peut être modifiée sans détection par les utilisateurs légitimes ;
- l'authentification : une preuve sur l'identité de l'émetteur d'un message est apportée.

Les mécanismes mis en œuvre pour remplir ces objectifs de sécurité reposent sur un secret, dont la connaissance permet de réaliser une opération sensible, lecture d'un message protégé en confidentialité, génération d'un message protégé en intégrité, etc. L'un des principes fondateurs de la cryptologie moderne est la séparation entre la description des mécanismes cryptographiques et le secret utilisé. En d'autres termes, les mécanismes cryptographiques emploient un paramètre secret appelé généralement clé, et peuvent être publiés sans compromettre leur sécurité tant que les clés utilitées restent secrètes.

On distingue parfois la conception de mécanismes cryptographiques, la *cryptographie*, et l'analyse et l'attaque de ces mécanismes, la *cryptanalyse*. Ces deux activités sont fortement liés : un cryptographe doit s'assurer que l'algorithme qu'il conçoit résiste (au moins) aux grandes techniques d'attaque connues ; le cryptanalyste peut proposer des contre-mesures permettant de résister à des attaques qu'il a identifié. Une bonne partie des travaux de recherche en cryptologie, notamment ceux portant sur les primitives, briques de base sur lesquelles sont construits les mécanismes cryptographiques, peut être vue comme une suite d'« aller-retours » entre cryptographes et cryptanalystes. Ces quinze dernières années ont été marquées notamment par trois compétitions internationales, publiques, concentrant l'attention de la communauté des cryptologues sur trois types de primitives :

- **NIST AES.** Cette compétition, conduite par le NIST entre 1997 et 2001 a conduit à la normalisation d'un algorithme de chiffrement par bloc, en remplacement du standard DES devenu obsolète.
- **ECRYPT eSTREAM.** Ce projet européen, qui s'est déroulé entre 2004 et 2008, a eu pour objet l'étude des algorithmes de chiffrement par flot. De nombreuses cryptanalyses d'algorithmes de chiffrement par flot déployés et l'émergence de l'AES comme standard de chiffrement avait conduit à questionner le principe même des algorithmes de chiffrement par flot. Le projet eSTREAM a eu pour objectif de faire émerger des principes de conception modernes pour cette famille de primitives cryptographiques.
- **NIST SHA-3.** Cette compétition lancée en 2008 par le NIST a pour objectif la normal-

isation d'un nouveau standard de hachage, qui pourra être amené à remplacer la famille SHA-2 si un défaut devait lui être découvert.

Ces trois évènements se sont déroulés selon des modalités proches. Dans un premier temps un appel à candidatures est émis et des chercheurs, académiques et industriels, soumettent des propositions d'algorithmes. Dans un deuxième temps ces candidats sont étudiés : la communauté académique cherche à évaluer leur sécurité, soit en la mettant en défaut par une attaque, soit en la garantissant par une preuve. Cette période d'analyse est découpée en phases entre lesquelles le nombre de candidats est réduit, afin de focaliser les efforts de la communauté sur les algorithmes les plus prometteurs. Enfin une sélection finale est réalisée.

Les travaux de cette thèse s'inscrivent dans le contexte des compétitions eSTREAM et NIST SHA-3. Dans une première partie nous présentons des techniques d'attaque contre des algorithmes de chiffrement par flot. On expose notamment la cryptanalyse différentielles à IV choisis de la famille d'algorithmes de chiffrement par flot VEST, candidat retenu pour la deuxième phase de la compétition eSTREAM. Dans une deuxième partie, nous présentons les preuves développées afin d'argumenter la sécurité de la construction de la fonction Shabal candidate à la compétition NIST SHA-3 à laquelle nous avons contribuée. Cette fonction a été sélectionnée pour participer à la deuxième phase de la compétition, mais n'a pas été retenue pour la phase finale. Les preuves que nous exposons en deuxième partie étudient la sécurité du mode sur lequel Shabal est construit, d'abord en considérant un composant interne de l'algorithme comme idéal, puis en intégrant l'existance de « distingueurs », identifiant des écarts entre les comportements du composant interne et d'un composant idéal.

Première partie

Chiffrement par Flot

Chapitre 1

Préliminaires

Sommaire

1.1 Définitions

1.1.1 Chiffrement symétrique

La cryptographie symétrique est caractérisée par le partage d'une même clé par tous les utilisateurs légitimes du cryptosystème. Pour remplir des services de confidentialité, on utilise deux algorithmes inverses l'un de l'autre appelés algorithmes de chiffrement et de déchiffrement. Ces algorithmes, dont les spécifications peuvent être rendues publiques, sont fonctions d'un paramètre K appelé clé. La sécurité du schéma cryptographique est assurée par le maintien du secret de la clé. L'algorithme de chiffrement $\mathcal{E}$ agit sur un message en clair, ou plaintext P, et produit un message chiffré, ou ciphertext C. Utilisé avec la même clé, l'algorithme de déchiffrement $\mathcal{D}$ réalise l'opération inverse. On a donc pour tout P, K :

$$\mathcal{D}(\mathcal{E}(P, K), K) = P.$$

On distingue deux familles d'algorithmes de chiffrement symétrique : les algorithmes de chiffrement par bloc et les algorithmes de chiffrement par flot.

1.1.2 Chiffrement par bloc

Un algorithme de chiffrement par bloc est un algorithme de chiffrement traitant des messages de taille fixe, appelés blocs. La taille n des blocs est usuellement de l'ordre de quelques centaines de bits. L'autre grandeur caractéristique de ces algorithmes est la taille k de la clé. Ainsi, pour un algorithme de chiffrement par bloc, on a $\mathcal{E} : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$.

La publication du Data Encryption Standard [108], ou DES, en 1977 par le NBS, organisme de standardisation américain, à destination de l'administration américaine, a contribué au développement de la cryptologie comme domaine de recherche académique. À la fin des années 90, le NIST, successeur du NBS a lancé une compétition destinée à définir un successeur au DES. Cette compétition a abouti à la sélection de l'algorithme Rijndael et à son inclusion dans le *Advanced Encryption Standard* [103]. L'acronyme AES désigne la version apparaissant dans ce standard.

Afin de pouvoir chiffrer des messages de taille arbitraire et obtenir des propriétés de sécurité additionnelles, comme par exemple le non-déterminisme du chiffrement, l'algorithme de chiffrement par bloc est mis en œuvre par un mode opératoire. Lors de la publication du DES et de l'AES des modes opératoires ont été standardisés [109]. L'activité de recherche académique a également conduit à la définition de modes ayant de meilleurs propriétés de sécurité [104] ou adaptés à de nouveaux contextes, comme le chiffrement de disque [105].

1.1.3 Chiffrement par flot synchrone

Les algorithmes de chiffrement par flot sont construits sur le principe de l'algorithme de chiffrement de Vernam. Le message est vu comme une suite de *symboles*. Chaque symbole formant le message à chiffrer est chiffré indépendamment par application d'un masque aléatoire. Pour un ensemble de symboles constitué des 26 lettres de l'alphabet, ceci consiste à réaliser un décalage dans l'alphabet. En modélisant chaque lettre par sa postion dans l'alphabet, entre 0 et 25, et en notant P_i, resp. C_i et M_i, le i-ème symbole du clair, resp. du chiffré et du masque, on a

$$C_i = P_i + M_i \mod 26.$$

Pour un ensemble de symboles constitué de valeurs booléennes, *i.e.* appartenant $\{0,1\}^\ell$, $\ell \geq 1$, l'application du masque s'obtient par ou exclusif

$$C_i = P_i \oplus M_i.$$

La suite de masques aléatoires utilisée doit :
- être de même longueur que le message à chiffrer ;
- être tirée uniformément ;
- ne pas être réutilisée pour chiffrer un autre message.

Shannon [128] montre que sous ces conditions le chiffrement est inconditionnellement sûr : les chiffrés n'apportent aucune information sur le message clair. La première condition est une condition nécessaire.

Si cet algorithme possède une sécurité parfaite, il est très difficile à mettre en œuvre en pratique, car les correspondants doivent partager préalablement des clés dont la longueur est supérieure à celle des messages et chaque clé ne peut être utilisée qu'une fois et une seule. Il est aisé de constater que la réutilisation d'une clé a des conséquences néfastes sur la confidentialité des données échangées. Si C est C' sont les chiffrés des messages P et P' sous la même clé

K, alors la différence des chiffrés est égale à la différence des clairs, ce qui constitue une fuite d'information conséquente :

$$C - C' = (P + K) - (P' + K) = P - P'.$$

Afin de faciliter la mise en œuvre de cette méthode de chiffrement, on fait appel à un algorithme appelé générateur pseudo-aléatoire, ou GPA. Il s'agit d'un algorithme prenant en entrée une valeur de taille fixe et générant une suite de masques de taille arbitraire. L'algorithme de chiffrement par flot correspondant consiste à fournir en entrée du générateur pseudo-aléatoire une clé et à appliquer sur le clair la suite de masques obtenue en sortie du générateur. Dans la quasi-totalité des cas, les symboles générés par un GPA consistent en un ensemble de bits (bit, octet, mot de 32 ou 64 bits). On se place par la suite dans le cas où la sortie du GPA est une suite de bits.

Les algorithmes de chiffrement par flot rendent le principe du chiffrement de Vernam utilisable dans la mesure où la taille de la clé à partager est à présent fixe et comme nous le verrons un peu plus loin relativement courte. Si on règle ainsi le problème de la taille de la clé, on perd cependant la sécurité inconditionnelle. De plus une clé reste à usage unique, car le GPA est déterministe. Afin de lever cette restriction, les algorithmes de chiffrement par flot récents acceptent une entrée additionnelle appelée *vecteur d'initialisation*, dénoté IV. Un IV, de longueur m bits, permet de diversifier la sortie du GPA pour une clé donnée. On peut utiliser une même clé pour chiffrer plusieurs messages, à condition de ne pas employer deux fois la même paire (clé, IV).

Afin de générer à partir de la clé une suite chiffrante de longueur supérieure à la taille de la clé, la plupart des GPA sont construits comme une machine ayant une mémoire de taille t bits, appelée *état interne*, sur laquelle agissent trois fonctions :

- une fonction d'évolution qui mélange les bits de l'état. Cette fonction est usuellement inversible pour s'assurer que l'évolution du GPA ne dégrade pas la variabilité de l'état interne ;
- une fonction d'extraction qui calcule à partir des valeurs des bits de l'état interne un symbole de la suite chiffrante de sortie (bit, octet, mot, ...).
- une fonction d'initialisation qui à partir d'une clé et éventuellement d'un IV produit une valeur d'état interne.

Deux limitations demeurent dans les algorithmes de chiffrement par flot de la manière décrite ci-dessus :

- Afin que le déchiffrement se déroule correctement, il est essentiel que le récepteur reste synchronisé avec le chiffré. La transmission sur un canal de transmission susceptible d'effacer ou d'intercaler des bits dans le chiffré transmis peut introduire des décalages rendant le message reçu indéchiffrable. La synchronisation devant être garantie par l'environnement, les algorithmes de chiffrement par flot dont la définition suit la structure décrite ci-dessus sont désignés sous le terme d'*algorithmes synchrones*. Des tentatives pour construire des algorithmes de chiffrement par flot essayant de lever cette limitation ont été réalisées [50, 125, 51]. L'idée est de faire dépendre le chiffrement, et donc le déchiffrement, d'une fenêtre des derniers bits de chiffré, garantissant ainsi la reprise d'un déchiffrement correct dès qu'une fenêtre suffisamment longue de chiffré est reçue. De tels algorithmes sont dits *auto-synchronisants*. Ils ne seront pas abordés dans la suite de ce mémoire.
- Seule la confidentialité est traitée par l'algorithme, et le chiffré est facilement manipulable. Les algorithmes de chiffrement par flot sont par nature très malléables : l'introduction d'une différence en une position donnée du chiffré conduira à un message clair différant en la même position. Afin d'ajouter un service d'intégrité, certains algorithmes prévoient le

calcul d'un motif d'intégrité, obtenu en introduisant au moyen d'une procédure spécifique le message clair ou chiffré dans l'état interne [139, 115].

1.1.4 Sécurité des algorithmes de chiffrement par flot

Les algorithmes de chiffrement par flot sont construits sur le principe du chiffrement de Vernam, en remplaçant la clé aléatoire par une suite chiffrante issue d'un GPA. Informellement, on s'attend donc à ce qu'un adversaire ne puisse pas remettre en cause la confidentialité des messages si la sortie du GPA lui apparaît comme aléatoire. A fortiori, l'attaquant ne doit pas pouvoir reconstituer tout ou partie de l'état interne du GPA, ni la clé utilisée pour initialiser le GPA. Pour formaliser cette intuition, on définit un « jeu de sécurité » dans lequel on attribue à l'attaquant des **moyens** et un **objectif**. Le jeu de sécurité est formalisé de la manière suivante : on fournit à l'attaquant une boîte noire contenant soit une source aléatoire idéale, soit un générateur pseudo aléatoire. L'attaquant réalise des requêtes à la boîte noire en fonction des moyens qui lui sont alloués. À la fin du jeu, il doit remplir son objectif.

Moyens de l'attaquant. Dans toutes les analyses, on considère que l'attaquant dispose des spécifications de l'algorithme attaqué. Les moyens de l'attaquant modélisent l'accès de l'attaquant à l'algorithme mis à la clé. Dans le cas des GPA synchrones, attaques à clairs connus, clairs choisis et chiffrés choisis coïncident : elles permettent toutes de remonter à la suite chiffrante générée par le GPA. On désigne ce moyen de l'attaquant sous le terme de *suite chiffrante connue*. Pour des algorithmes faisant usage d'IV, on donne généralement à l'attaquant accès aux IV utilisés. On parle d'attaque à *IV connus*. On considère également le cas où l'attaquant peut contrôler les IV. On parle alors d'attaques à *IV choisis*. Ces moyens semblent donner à l'adversaire un grand accès à l'algorithme. Cependant, comme l'illustrent les attaques pratiques contre les algorithmes A5/1 et A5/2 dans le cadre du système GSM [10, 106] et contre l'algorithme RC4 dans le cadre de sa mise en œuvre dans le système WEP [135], l'accès à la suite chiffrante peut être obtenu au travers du chiffrement de donnée de formatage fixe. Pour évaluer la sécurité intrinsèque des algorithmes cryptographiques, il est préférable de se placer dans le contexte le plus avantageux pour l'attaquant.

Pour les algorithmes incluant des mécanismes d'intégrité ou pour des algorithmes de chiffrement par flot auto-synchronisants, on peut de plus considérer des *attaques à clairs/chiffrés connus ou choisis*.

Objectifs de l'attaquant. L'objectif le plus difficile à atteindre pour l'attaquant est la *reconstitution de la clé de chiffrement*. On peut aussi demander à l'attaquant de *reconstituer tout ou partie de l'état interne*. L'objectif correspondant à l'intuition formulée ci-dessus est plus aisé à remplir pour l'attaquant. Il s'agit pour lui de distinguer les sorties du GPA de données aléatoires. On parle d'*attaques par distingueur*. L'attaquant remplit l'objectif s'il est en mesure de distinguer les sorties de l'algorithme de chiffrement par flot de suites parfaitement aléatoires avec une probabilité éloignée de manière notable de la valeur $\frac{1}{2}$, qui correspond à sa probabilité de succès quand il répond de manière aléatoire. Outre révéler une caractéristique non-aléatoire des sorties du GPA, ce type d'attaques est intéressant dans la mesure où il peut souvent être traduit en une attaque permettant de recouvrer une partie de l'état interne du GPA, comme nous le verrons au chapitre 3.

En cryptographie symétrique, il n'existe pas, en règle générale, de preuve de la sécurité d'un

algorithme. La sécurité d'un algorithme est la conjonction de deux points :

- Le dimensionnement de l'algorithme est tel que les meilleures attaques génériques connues sur l'algorithme ont des complexités inatteignables en pratique ;
- Il n'existe pas d'attaque plus rapide que ces attaques.

Si le premier point peut facilement être analysé, le deuxième point n'est jamais complètement garanti. La confiance dans un algorithme symétrique est liée à la durée consacrée à son étude et à la qualité des arguments de sécurité produits au cours de cette étude.

1.2 Historique

On donne ici un panorama rapide de l'histoire des algorithmes de chiffrement par flot en décrivant quelques principes de conception.

1.2.1 Constructions à base de registres linéaires

On commence par décrire les constructions basées sur des registres à décalage à rétroaction linéaire, plus communément désignés sous l'acronyme anglais LFSR, pour *linear feedback shift register*. Ces primitives peuvent être vues comme des GPA élémentaires et sont utilisées comme briques de base de nombreux algorithmes de chiffrement par flot du fait de leurs bonnes propriétés [99, Section 6.2.1] : distribution statistique des bits de sorties, garantie sur la période de la suite des bits de sorties, etc.

Malgré ces bonnes propriétés statistiques, un LFSR ne peut être utilisé comme algorithme de chiffrement par flot. En effet, de par la linéarité de la fonction de mise à jour, il est aisé de reconstituer le polynôme de rebouclage du LFSR. Par exemple, l'algorithme de Berlekamp-Massey [93], dont une description est donnée en section 2.2.4, permet de reconstruire le polynôme de rebouclage du LFSR le plus court générant une suite donnée. Étant donnée la taille L du LFSR, il suffit de collecter $2L$ bits de sortie afin de pouvoir déterminer de manière unique toutes les caractéristiques du LFSR.

On emploie donc les LFSR dans des constructions introduisant de la non-linéarité dans la mise à jour de l'état interne. Cette non-linéarité est introduite soit par l'utilisation d'une fonction non-linéaire, soit en rendant irrégulière l'avance des LFSR ou l'application de la fonction d'extraction. On rencontre dans la littérature les constructions classiques données ci-dessous. Ces principes de constructions peuvent être utilisés en parallèle dans un même algorithme.

Construction par combinaison. Plusieurs LFSR évoluent en parallèle dans l'état de l'algorithme. À chaque étape, une fonction est appliquée sur les sorties des LFSR pour former la sortie de l'algorithme, cf Figure 1.1. Parmi les algorithmes de chiffrement par flot construits sur

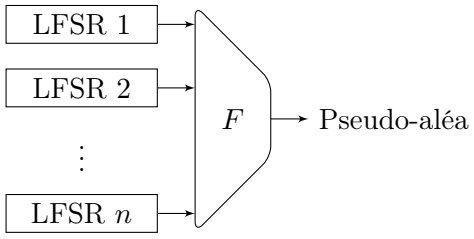


FIGURE 1.1 – Algorithme de chiffrement par flot construit selon le principe de combinaison

ce principe citons le générateur de Geffe [99, Example 6.50], basé sur 3 LFSR et pour lequel la fonction de calcul de la sortie est la fonction choix, et Achterbahn [66] où les LFSR sont remplacés par des registres à décalage à rebouclage non-linéaire.

Construction par filtrage. Un LFSR constitue l'intégralité de l'état du GPA, mais la fonction d'extraction classique est remplacée par une fonction non-linéaire prenant ces entrées dans l'état du LFSR, cf Figure 1.2.

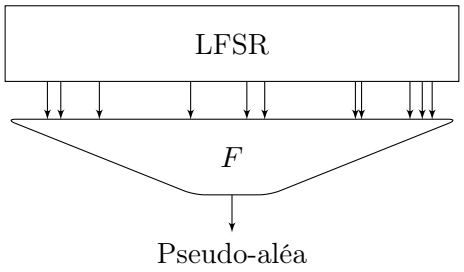


FIGURE 1.2 – Algorithme de chiffrement par flot construit selon le principe de filtrage

Le principe de construction du LFSR filtré se rencontre dans de nombreux algorithmes (Hitag-2, Crypto-1 [67], Sfinks [33], WG [70], ...).

Construction par avance irrégulière. La fonction de mise à jour d'un LFSR classique consiste à appliquer de manière déterministe une opération de décalage et de rebouclage. Afin d'introduire de la non-linéarité, certains algorithmes font dépendre le comportement de l'avance du LFSR d'une entrée auxiliaire, cf Figure 1.3.

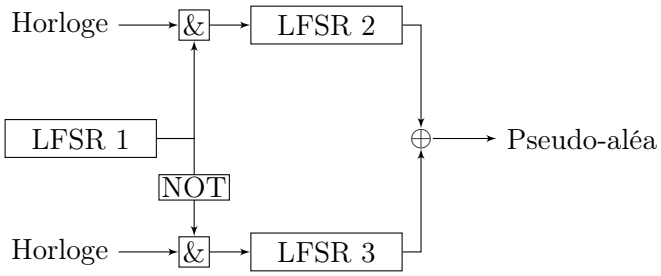


FIGURE 1.3 – Algorithme de chiffrement par flot construit selon le principe d'avance irrégulière

Suivant les cas, l'avance peut être tout simplement inhibée (A5/1, A5/2...) ou le polynôme de rebouclage peut varier (MICKEY [8], K2 [84],...)

Construction par décimation. Les constructions basées sur le principe de décimation sont analogues aux constructions par avance irrégulière, mais c'est à présent sur la fonction d'extraction que porte l'irrégularité de traitement. L'extraction d'éléments de la suite chiffrante n'est pas réalisée à chaque étape, mais contrôlée par une partie de l'état du GPA, cf Figure 1.4.

De nombreux algorithmes font usage de ce principe (Shrinking Generator [41], Self-Shrinking Generator [98], LILI [132], Decim [14], ...).

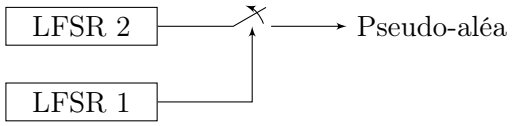


FIGURE 1.4 – Algorithme de chiffrement par flot construit selon le principe de décimation

1.2.2 Algorithme de chiffrement à flot « prouvé sûr »

Au milieu des années 1980, Blum, Blum et Shub proposent un générateur pseudo-aléatoire dont la sécurité peut être réduite au problème de la factorisation [26]. Bien que la réduction au problème mathématique sous-jacent reste asymptotique et ne permette pas de dimensionner l'algorithme pour tirer parti de la réduction (problème qui sera levé plus tard [130]), et malgré un débit médiocre de l'algorithme comparativement aux algorithmes de chiffrement par flot usuels, BBS traduit une préoccupation grandissante d'obtenir des algorithmes symétriques disposant d'une preuve de sécurité. L'algorithme QUAD [16] proposé en 2006 reprend également cette idée avec une réduction au problème de la résolution de systèmes multivariés quadratiques et de meilleures performances.

1.2.3 Chiffrements par flot industriels

Dans la première moitié des années 1990, on a assisté au déploiement de nombre de technologies de communication sans fil (GSM, Wifi, Bluetooth,...). Ces technologies, faisant usage d'ondes électromagnétiques pour transmettre des informations, sont bien plus vulnérables aux scénarios d'écoute passive que les communications utilisant un médium matériel (ethernet, ...). Afin de compenser cette vulnérabilité accrue, les standards décrivant ces technologies prévoient des mécanismes pour protéger les communications en confidentialité. La première génération d'algorithmes de confidentialité mise en œuvre est constituée presque exclusivement d'algorithmes de chiffrement par flot, parfois sous-dimensionnés :

- les algorithmes A5/1 et A5/2 protègent les communications GSM entre terminaux mobiles et stations de base ;
- la sécurité du WEP, *Wired Equivalent Privacy*, qui protège initialement les communications 802.11, *i.e.* du WiFi, repose sur l'algorithme RC4, complété pour pouvoir recevoir un IV ;
- la norme Bluetooth repose sur l'algorithme E0.

Ces algorithmes présentent des défauts qui ont conduit à leur cryptanalyse après leur publication.

- A5/1 et A5/2 sont deux algorithmes sous-dimensionnés (taille de clé de 64 bits, taille d'état interne de 64 bits pour A5/1, 81 bits pour A5/2. La structure de A5/2 permet de plus de réaliser une attaque algébrique très efficace [10]. Le précalcul d'une attaque générique de type compromis temps-mémoire, cf section 1.3.1.2, a récemment été réalisé pour A5/1 [106], et les tables issues de ce précalcul ont été rendues publiques. L'utilisation de ces tables permet de retrouver la clé utilisée par l'algorithme A5/1 pour chiffrer une communication GSM.
- la procédure de mise à la clé de RC4 est minimaliste. Par conséquent, les premiers octets de suite chiffrante font fuire beaucoup d'information sur les octets de clés. Une série d'attaques [64, 85, 135, 127] a conduit au développement d'un nouveau standard de protection, le WPA, remplaçant progressivement le WEP.

1.2.4 Constructions fondées sur des primitives cryptographiques sous-jacentes

Un autre principe de construction d'algorithmes de chiffrement par flot est l'utilisation d'une primitive cryptographique sous-jacente dans un mode adapté. On rencontre notamment les « modes de chiffrement de type flot », qui permettent de générer une suite chiffrante à l'aide d'un algorithme de chiffrement par bloc. Ainsi, le mode OFB itère un algorithme de chiffrement par bloc utilisant une clé sur un bloc de message initialisé avec un IV. Le mode compteur chiffre les valeurs successives d'un compteur initialisé avec un IV. Dans les deux cas, la concaténation des blocs de chiffré successifs forme la suite chiffrante.

Le candidat Salsa [17] à la compétition eSTREAM, cf section 1.2.5, peut également être vu comme une construction de ce type : les valeurs successives d'un compteur, initialisé avec la clé et l'IV, sont hachées pour obtenir une suite chiffrante.

1.2.5 Chiffrement par flot moderne – compétition eSTREAM

Au début des années 2000, de nombreux résultats de cryptanalyse contre les algorithmes de chiffrement par flot industriels et quelques propositions académiques remettent fortement en question le principe même des algorithmes de chiffrement par flot. Ceci conduit le projet européen ECRYPT à lancer un projet d'étude des algorithmes de chiffrement par flot, le projet eSTREAM [60]. Ce projet s'est déroulé sous forme d'une compétition. Suite à un appel à candidatures, 34 algorithmes de chiffrement par flot ont été soumis.

La conception de ces algorithmes vise deux profils, où l'on estime que les algorithmes de chiffrement par flot peuvent offrir de meilleures performances que les algorithmes de chiffrement par bloc. Les algorithmes concourant dans le profil matériel ont pour objectif une implémentation matérielle (FPGA, ASIC, ...) très compacte. Les algorithmes concourant dans le profil logiciel recherchent des débits en ligne, *i.e.* après établissement de l'état initial, très élevés.

L'objectif du projet eSTREAM était moins la définition d'un standard que l'avancement de l'état de l'art sur les algorithmes de chiffrement par flot. En conséquence, les candidats ont eu la possibilité, au cours de la compétition, d'évoluer et de réparer d'éventuelles faiblesses détectées suite à leur analyse, afin de permettre la poursuite de l'étude de principes de conception innovants. La liste des concurrents a été progressivement réduite afin de favoriser la concentration des efforts de cryptanalyse sur les candidats les plus prometteurs. La fin du projet a vu la sélection de huit algorithmes [7], quatre pour chaque profil. Les algorithmes retenus sont cependant encore considérés comme immatures et l'objectif est plus de fournir un ensemble d'algorithmes et de principes de conception à étudier plus avant. Si la cryptanalyse pratique de l'un de ces algorithmes peu après la fin de la compétition [73] confirme ce sentiment, il reste indéniable que la compétition eSTREAM a permis de faire progresser l'état de l'art de la cryptographie des algorithmes de chiffrement par flot et de restaurer quelque peu la confiance dans ce type d'algorithme comme l'atteste l'inclusion de SNOW 3G dans la suite de confidentialité du 3GPP.

1.3 Principes de cryptanalyse des algorithmes de chiffrement par flot

On présente ici dans les grandes lignes les principes de cryptanalyse les plus communs des algorithmes de chiffrement par flot.

1.3.1 Attaques génériques

Du fait de sa définition et de sa construction, un algorithme de chiffrement par flot peut être attaqué de manière générique. C'est le dimensionnement de l'algorithme qui permet de se prémunir contre ces attaques, *i.e.* il existe des attaques contre l'algorithme de chiffrement par flot mais leur complexité est telle qu'elles ne peuvent aboutir de manière pratique. On distingue les attaques directes des compromis temps-mémoire-donnée. La complexité des attaques génériques dépend de trois paramètres, dont ces attaques fixent donc la taille minimale :

- la taille de la clé ;
- la taille de l'état interne ;
- la taille de l'IV.

1.3.1.1 Attaque directe

Recherche exhaustive sur la clé. Il s'agit là d'une attaque élémentaire qui s'applique à tout algorithme cryptographique utilisant des tailles de clés fixes. L'attaque consiste à tester toutes les clés possibles. Elle est réalisable lorsque :

- l'attaquant dispose des spécifications de l'algorithme cryptographique ;
- l'attaquant dispose d'un *test d'arrêt*, lui permettant de détecter la clé utilisée avec une probabilité de fausse alarme faible. Pour la recherche exhaustive appliquée aux GPA, l'attaquant doit disposer d'une quantité de suite chiffrante correspondant à la taille de la clé.

Le nombre de clés d'un algorithme cryptographique doit être grand devant le nombre d'opérations réalisable en pratique avec les moyens de calculs actuels et envisageables à moyen terme. Des calculs requérant de l'ordre de 2^{64} opérations ont été réalisés par le passé par des projets de calcul distribué [56]. On estime qu'un calcul nécessitant 2^{128} opérations n'est pas réalisable à moyen terme. Une taille de clé de 128 bits permet donc de se prémunir à moyen terme contre la recherche exhaustive.

Il faut noter que dans de nombreux contextes cette attaque reste la meilleure attaque en pratique contre un algorithme, même quand il existe de meilleures attaques théoriques. Ceci est dû à trois points :

- Cette attaque ne requiert que très peu de données. Il est beaucoup plus simple d'obtenir les informations nécessaires à une recherche exhaustive (clair connu), que de réaliser des attaques ayant une forte complexité « en ligne », c'est à dire ayant besoin d'un nombre important d'interactions avec l'algorithme mis à la clé, par exemple sous la forme de requêtes de chiffrement ou de déchiffrement.
- Cette attaque se parallélise naturellement, ce qui permet de profiter au maximum de l'augmentation de la puissance de calcul disponible. Ce n'est pas le cas d'algorithmes requérant l'accès à une mémoire de taille importante.
- De nombreux cryptosystèmes pratiques sont sous-dimensionnés ou utilisent des clés de faible entropie.

Recherche exhaustive sur l'état interne. Dans la majorité des GPA, la clé utilisée intervient uniquement lors de l'initialisation de l'algorithme et permet de calculer un état initial. Cet état est ensuite soumis aux fonctions d'évolution et d'extraction. La connaissance de cet état permet donc de reconstituer la suite chiffrante correspondant à la clé. Dans les cas où la fonction d'avance est inversible, la connaissance d'un état à un instant quelconque permet de remonter à cet état initial. L'état du GPA peut donc également faire l'objet d'une recherche exhaustive et

l'information obtenue permet de dériver les bits suivants et précédents de la suite chiffrante. On remarque donc que la taille de l'état interne doit donc être au moins égale à la taille de la clé.

Collision sur l'IV. La propriété requise des IV du point de vue de la confidentialité des données protégées par un algorithme de chiffrement par flot est la non-répétition d'un IV pour une clé donnée. Lorsque ces IV sont tirés de manière aléatoire, ils doivent être dimensionnés de telle sorte qu'une collision sur ces IV n'apparaisse qu'avec une probabilité négligeable.

1.3.1.2 Compromis temps-mémoire-donnée

Les techniques décrites ci-dessous trouvent leur origine dans l'analyse de fonctions de chiffrement par bloc. L'objectif est de diminuer la complexité temporelle de l'inversion d'une fonction f difficile à inverser, par exemple la fonction qui à une clé associe le chiffré d'un message fixe, à travers l'utilisation d'un précalcul dont le résultat est stocké en mémoire [74, 107]. On rappelle ci-dessous le principe de ces compromis temps-mémoire ainsi que leurs caractéristiques.

On considère une fonction $f : E \to F$, avec $|E| \leq |F|$. On considère également des fonctions de F dans E appelées fonctions de réduction. On note N le cardinal de E. L'objectif est d'inverser la fonction f. Pour ce faire, on réalise un précalcul de complexité P et on stocke en mémoire le résultat de ce précalcul. On note M la quantité de mémoire requise. Puis pour une instance donnée du problème, on cherche une solution en réalisant un calcul de complexité en temps T.

L'application de compromis temps-mémoire aux algorithmes de chiffrement par flot a deux spécificités :

- on dispose de deux cibles, la clé et l'état interne. Bien que par construction ces deux données soient équivalentes, le dimensionnement de la clé et de l'état interne peut rendre préférable d'attaquer l'une des deux valeurs.
- pour une clé, ou pour une paire (clé, IV) donnée, on peut disposer d'une quantité importante de suite chiffrante, quantité que l'on notera D. Quand la fonction d'avance est inversible, recouvrer l'état interne du GPA à un instant donné, par exemple l'instant initial, est équivalent à le recouvrer à n'importe quel instant. On peut alors utiliser la suite chiffrante disponible pour résoudre une inversion parmi D, mettant ainsi à profit la quantité de données disponible pour réaliser des compromis temps-mémoire-donnée.

Dans la suite, on s'attache à donner les ordres de grandeur et relations liant les différents paramètres du problème en supposant P, M, T et D grands devant $\log_2(N)$. On omet les termes négligeables et les termes logarithmiques.

Compromis de Babbage-Golić. Le compromis de Babbage-Golić [6, 69] est un compromis temps-mémoire-donnée attaquant l'état interne d'un algorithme de chiffrement par flot. Plus précisément, on considère n la taille en bits de l'état interne, l'ensemble $E = \{0,1\}^n$ des $N = 2^n$ valeurs d'états internes du GPA, la fonction d'avance du GPA $\varphi : E \to E$ et la fonction d'extraction du GPA $\psi : E \to \{0,1\}$. On construit alors la fonction Ψ^l qui associe à un état interne la fenêtre de l bits générée par le GPA initialisé par cet état interne :

$$\begin{array}{rcl} \Psi^l : E & \to & E, \\ x & \to & (\psi \circ \varphi^0(x), \psi \circ \varphi^1(x), \ldots, \psi \circ \varphi^{l-1}(x)). \end{array}$$

La fonction $\Psi = \Psi^n$ est une fonction difficile à inverser. L'idée du compromis de Babbage-Golić est de recouvrer un état interne correspondant à une fenêtre de n bits parmi les D fenêtres disponibles.

Précalcul. Le précalcul consiste à calculer et stocker en mémoire un certain nombre de paires formées d'un élément de E et de son image par la fonction Ψ. Plus précisément, on tire m éléments $x \in E$ distincts et on stocke en mémoire les paires $(x, \Psi(x))$. À la fin du précalcul on trie ces couples selon la valeur de $\Psi(x)$. On a $P = M = m$.

Résolution. On suppose disposer d'une suite chiffrante de longueur $D = \frac{N}{m}$. On peut construire de l'ordre de D fenêtres de n bits de suite chiffrante. Pour chacune de ces fenêtres, on teste si la table donne un antécédent. Si c'est le cas, on retourne cet antécédent. La complexité de cette phase est $T = D = \frac{N}{m}$. Heuristiquement, on considère $MT = N$ paires de la forme (fenêtre de suite chiffrante, image de Ψ dans la table) chacune ayant probabilité $1/N$ de répéter la même valeur. On s'attend donc à ce que la table donne un antécédent sur l'ensemble des fenêtres considérées.

Remarque 1. Quitte à abandonner la phase de précalcul, on peut commencer par collecter et stocker en mémoire m bits de suite chiffrante, puis tester aléatoirement $\frac{N}{m}$ valeurs d'état interne en calculant leur image par Ψ et en recherchant le résultat parmi les fenêtres de suite chiffrante collectée. On a alors $D = M = m$ et $T = \frac{N}{m}$. Cet autre point du compromis vérifie la même relation $MT = N$. On peut adopter le compromis qui minimise la quantité de données utilisée $D = \min(M, T)$.

Remarque 2. La valeur retournée par l'algorithme ne correspond pas forcément à la valeur de l'état interne. En effet rien ne garantit l'injectivité de Ψ. Cependant, la probabilité de trouver le bon résultat est non négligeable et la probabilité d'erreur liée à ce phénomène peut être dimininuée en augmentant légèrement la taille des fenêtres considérées ($f = \Psi^{n+\varepsilon}$).

Remarque 3. Un point classique de ce compromis est la valeur $T = M = 2^{\frac{n}{2}}$. Étant donnée la quantité de mémoire et/ou de données nécessaire pour réaliser ce compromis, il ne pose généralement pas de problème en pratique. Cependant il indique qu'il est sain de dimensionner l'état interne du GPA pour que sa taille soit au moins égale à deux fois le niveau de sécurité visé.

Compromis de Hellman, tables arc-en-ciel et compromis de Biryukov-Shamir. Le compromis de Hellman [74] peut-être utilisé pour attaquer aussi bien la clé que l'état interne d'un GPA. Il repose sur un précalcul différent. Le compromis d'Oeschlin [107], plus connu sous le nom de table arc-en-ciel, est apparenté au compromis de Hellman. Le compromis de Biryukov-Shamir [25] est une variante du compromis de Hellman spécifique au GPA qui porte spécifiquement sur l'état interne et reprend l'idée du compromis de Babbage-Golić de résoudre une inversion parmi D instances du problème.

Principe du précalcul. Le précalcul dans ces compromis se fonde sur la construction de chaînes de valeurs par itération de la fonction f. Considérons une valeur $x \in E$, un entier t et une fonction de réduction τ. On peut construire une chaîne de $t+1$ valeurs de E commençant par x en itérant la fonction $\tau \circ f$:

$$x = x_0 \overset{\tau\circ f}{\to} x_1 \overset{\tau\circ f}{\to} x_2 \ldots \overset{\tau\circ f}{\to} x_t = y.$$

L'idée du précalcul est de construire un certain nombre de chaînes de ce type et, pour chacune de ces chaînes, de stocker la valeur initiale et la valeur finale. Ces chaînes généralisent les paires utilisées par le compromis de Babbage-Golić. On remarque qu'elles peuvent être utilisées pour inverser toute valeur $z \in \{x_i, 1 \leq i \leq t\}$: on peut détecter que z fait partie de la chaîne en

vérifiant qu'il existe $0 \leq j < t$ tel que $(\tau \circ f)^j(z) = y$. Un antécédent de z est alors donné par $(\tau \circ f)^{t-j-1}(x)$.

Construction d'une table de précalcul, collisions. Idéalement, le précalcul doit construire des chaînes de valeurs de telle sorte que toutes les valeurs de E apparaissent dans les chaînes.[1] Une première idée naïve consiste à construire une table de m chaînes de longueur t de telle sorte que $mt = N$. On s'attend à rencontrer N valeurs mais cette intuition est erronée en raison de l'apparition de collisions entre les valeurs rencontrées. Ces collisions ont un impact notable sur le nombre de valeurs distinctes apparaissant au cours du précalcul car deux chaînes distinctes x^1, x^2 ayant deux valeurs en collision $x_i^1 = x_j^2$ partagent une même fin de chaîne $x_{i+1}^1 = x_{j+1}^2$, etc. La couverture de l'espace E correspondant à chaque ligne, égale à t en l'absence de collision, diminue notablement quand des collisions commencent à apparaître. Afin de dépasser ce problème, on peut utiliser les deux stratégies suivantes.

Compromis de Hellman. Dans le compromis de Hellman, l'approche adoptée consiste à limiter la taille des tables de précalcul construites. On fixe m pour arrêter la construction d'une table à partir du moment où des collisions commencent à apparaître. Lorsqu'on ajoute une chaîne à une table comptant m chaînes sans collision, la probabilité d'apparition d'une collision est de l'ordre de $t \times mt/N$. On fixe donc la taille limite m par la relation $mt^2 = N$. L'espace des valeurs de E couvert par les chaînes de cette table est $mt = N/t$, qui constitue une partie relativement faible de l'espace des valeurs d'entrée. Si on se contente de cette table, la probabilité de succès de l'attaque sera de $1/t$. Hellman résout le problème en construisant différentes tables en faisant varier la fonction de réduction τ. Ceci fait disparaitre le problème de fusion de chaînes puisqu'une collision entre deux chaînes de tables différentes n'entraîne plus automatiquement la fusion des fins des chaînes. Comme chaque table couvre une fraction $1/t$ de l'espace des valeurs d'entrée de f, il faut construire t tables. Lorsqu'on recherche un antécédent il faut reproduire la procédure de recherche pour chacune des t tables. On peut alors estimer le coût du compromis temps-mémoire. On a $P = t \times mt = N$, $M = t \times m$, $T = t \times t$. Le compromis peut s'écrire $N^2 = TM^2$.

Compromis de Oechslin. Dans le compromis de Oechslin on construit toujours une seule table, mais on fait varier τ en chaque position. Ceci a pour conséquence de réduire l'impact d'une collision au cours de la construction des chaînes. Pour qu'une collision entraîne une fusion de lignes, elle doit avoir lieu en la même position. On peut alors fixer m et t tel que $P = mt = N$. On a également $M = m$. La procédure de recherche d'un antécédent doit être adaptée. En effet, le coût de détection d'un élément en position i est $t - i$ car la détection d'un élément en position $i - 1$ dans une chaîne ne profite plus du test réalisé pour détecter un élément en position i. Par conséquent $T = \sum_{i=1}^{t} i \sim t^2$. On retrouve la même relation de compromis $N^2 = T \times M^2$. Un choix classique pour réaliser ce compromis est $T = M = 2^{\frac{2n}{3}}$. Le précalcul dans les deux cas a un coût de l'ordre d'une recherche exhaustive. Ces deux compromis peuvent être vus comme des méthodes permettant de factoriser l'effort fait par une recherche exhaustive pour accélérer les recherches suivantes.

Compromis de Biryukov-Shamir. Ce compromis est une adaptation du compromis de Hellman reprenant l'idée du compromis de Babbage-Golić, *i.e.* recouvrer un état interne parmi

1. En pratique, on se contente de couvrir une fraction suffisante des valeurs de E, en acceptant une certaine probabilité d'échec à l'inversion de f par le compromis.

les D parcourus pendant la génération des données disponibles. Disposant de D cibles, il n'est plus nécessaire de couvrir l'intégralité de E au cours du précalcul. Si le précalcul couvre une fraction $1/D$ de E, l'attaque a une bonne probabilité de réussite. En adaptant le compromis de Hellman, on a toujours la relation $N = mt^2$, mais le nombre de tables nécessaire pour couvrir N/D valeurs est à présent t/D. La mémoire nécessaire est adaptée en conséquence : $M = mt/D$. La complexité du précalcul s'en trouve également réduite : $P = N/D$. Pour la partie recherche de solution il faut tenir compte de ce qu'on travaille avec moins de tables, mais que le calcul doit être réalisé pour chacune des D fenêtres de suites chiffrantes, soit $T = t^2D/D = t^2$. L'équation du compromis s'écrit à présent $N^2 = TM^2D^2$. Un point classique du compromis est $D = M = 2^{\frac{n}{3}}, T = 2^{\frac{2n}{3}}$. On remarque que, contrairement au compromis de Hellman, le précalcul est ici plus rapide que la recherche exhaustive, $P = 2^{\frac{2n}{3}}$, ce qui fournit une attaque globale plus rapide que la recherche exhaustive. La taille de l'état interne doit donc être supérieure au niveau de sécurité visé.

Problèmes liés à l'implémentation pratique d'un compromis temps-mémoire Lors de l'implémentation pratique de compromis temps-mémoire-donnée, les différents paramètres ne jouent pas le même rôle. Les paramètres les plus importants et restrictifs en pratique sont la mémoire et la quantité de données disponibles.

Impact du paramètre M. Si on considère les implémentations de grande mémoire disposant d'un accès aléatoire, nécessaire pour réaliser la recherche d'un élément dans une table, on constate que plus la taille de la mémoire est grande, plus le temps d'accès est important. On donne en table 1.1 des ordres de grandeur de taille de mémoire et de temps d'accès pour des types de mémoire très répandus. Dans le cas du compromis de Hellman, la complexité en temps est estimée en nombre de calculs de la fonction f. Cependant, on fait là l'hypothèse implicite que le temps de l'évaluation de la fonction f, noté T_f, est le terme prédominant dans la complexité temporelle. En supposant que la mémoire est une table de hachage dont la clé est la valeur de fin de chaîne, que le test d'appartenance d'une valeur à l'ensemble des valeurs de fin de chaîne requiert C accès mémoire et en notant $T_M = g(M)$ le temps d'un accès mémoire (qui est fonction de la taille de la mémoire, de la technologie utilisée, etc) on peut écrire $T_{\text{horloge}} = T(T_f + CT_M)$. Ceci modifie donc le compromis. Dans le cas général, cette description reste encore imparfaite, car elle ne prend pas en compte la possibilité de paralléliser certains aspects du calcul, ce qui pénalise également les algorithmes utilisant de grandes mémoires à accès aléatoire [18]. Tous ces éléments concourent à choisir en pratique des points du compromis avec M relativement petit et à introduire des optimisations permettant de réduire M. La technique des points distingués, mentionnée par Rivest dans [53], modifie le compromis de Hellman en rendant t variable. La construction d'une chaîne commence en un point et s'arrête en un point dont la représentation en mémoire est compacte. Le taux de compression choisi donne la moyenne des longueurs de chaînes. Un exemple récent d'implémentation pratique de compromis temps-mémoire est le calcul de table pour l'algorithme de chiffrement du GSM A5/1 [106].

Impact du paramètre D. Le paramètre D joue également un rôle important dans l'implémentation d'une attaque dans un scénario pratique. En effet, si dans les modèles théoriques utilisés pour réaliser l'analyse de sécurité de générateurs pseudo-aléatoires les bits de suite chiffrante sont mis à disposition de l'attaquant, l'obtention de ces données dans une attaque pratique est plus difficile et plus contraint. Il n'est pas possible d'énoncer d'ordre de grandeur générique puisque tout dépend du contexte d'application. Par exemple, dans le contexte du WEP, algorithme de chiffrement des communications WiFi, l'attaquant obtient de l'ordre d'une dizaine d'octets de suite chiffrante par paquet en examinant le chiffré des en-têtes des paquets protégés

Type	Technologie	Taille (bits)	Temps d'accès (s)
Cache L1	SRAM	2^{20}	10^{-8}
Mémoire principale	DRAM	2^{30}	10^{-7}
Disque dur	Stockage Magnétique	2^{37}	10^{-5}
Bande magnétique	Stockage Magnétique	2^{42}	10

TABLE 1.1 – Ordre de grandeur des temps d'accès et capacité des technologies courantes de mémoire

(clair connu) [135]. Dans le cas du GSM, un canal de synchronisation émettant des messages en partie prédictibles est chiffré par la clé de protection des données, ce qui fournit une bonne source de clairs connus [10]. Lorsque D est limité par le contexte d'emploi, l'avantage du compromis de Biryukov-Shamir sur le compromis de Hellman est faible.

1.3.2 Attaques par corrélation

On présente dans la fin de ce chapitre des classes d'attaques exploitant la structure interne des GPA étudiés.

Les attaques par corrélation s'appliquent aux algorithmes de chiffrement par flot construits sur des primitives sous-jacentes linéaires, comme des LFSR. Elles tirent partie de cette linéarité pour essayer de linéariser complètement le comportement de l'algorithme. L'attaquant tente de remplacer les composants non-linéaires de l'algorithme par des composants linéaires qui approximent leurs comportement. Il compare ensuite les sorties de l'algorithme avec celles d'une simulation linéarisée en faisant une hypothèse sur une partie de son état interne. La concordance entre ces sorties valide l'hypothèse sur l'état interne.

On détaille ici à titre d'exemple l'attaque par corrélation [131] contre le générateur de Geffe [99, Example 6.50]. Le générateur de Geffe est un algorithme de chiffrement par flot basé sur la combinaison de trois LFSR définis sur $\mathbb{F}_2$. La fonction de combinaison utilisée est la fonction choix, *i.e.*

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 \oplus x_1)x_3.$$

Cette fonction présente des approximations linéaires ayant un biais relativement élevé, c'est à dire dont la probabilité est relativement éloignée de $\frac{1}{2}$:

$$\Pr_x [f(x) = x_2] = \Pr_x [f(x) = x_3] = \frac{3}{4}.$$

À une probabilité p, on associe le biais ε défini par $p = \frac{1}{2}(1+\varepsilon)$. Le biais du générateur de Geffe est $\frac{1}{2}$. On notera $a = b\ [\varepsilon]$ une égalité entre deux valeurs vérifiée avec probabilité $\frac{1}{2}(1+\varepsilon)$.

Ce biais important et le fait que l'approximation linéaire de f ne fasse intervenir qu'une seule variable permet de mener une attaque par corrélation. En effet, on peut approximer le GPA par le LFSR 2 dont la sortie est bruitée avec probabilité $\frac{1}{4}$, cf Fig 1.5.

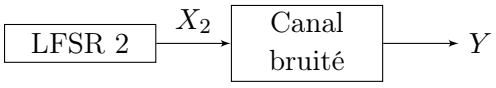


FIGURE 1.5 – Modélisation du GPA par un LFSR bruité

On note n_i la taille du LFSR 2. En supposant que l'attaquant dispose de D bits $(z_i)_{i=0}^{D-1}$ de sortie du GPA, il peut exécuter l'algorithme 1.

Algorithme 1 Algorithme de recouvrement de l'état initial du LFSR

Entrée : la définition d'un LFSR (taille, polynôme de rebouclage)
D bits de suite chiffrante $(z_i)_{i=0}^{D-1}$
Sortie : l'état initial du LFSR
pour tout S initial non nul du LFSR 2 **faire**
Générer la suite des sorties du LFSR $(x_i)_{i=0}^{D-1}$
Calculer la concordance entre la suite chiffrante et la sortie du LFSR 2, c'est à dire $\ell = \#\{0 \leq i < D | x_i = z_i\}/M$.
si $\ell > \ell_{limit}$ **alors**
retourner S
fin si
fin pour

Dans le cas où l'algorithme considère une valeur initiale erronée du LFSR, les suites (x_i) et (z_i) sont indépendantes et ℓ suit la distribution de la moyenne de D bits suivant une loi de Bernoulli de paramètre $\frac{1}{2}$. Dans le cas où l'algorithme considère la bonne valeur initiale du LFSR, ℓ suit une loi similaire, mais le paramètre des lois de Bernoulli est $\frac{1}{2}(1+\varepsilon)$. Lorsque D est grand, on peut approximer ces lois par des gaussiennes de moyenne $\frac{1}{2}$ (resp. $\frac{1}{2}(1+\varepsilon)$) et d'écart type $\sqrt{\frac{1}{4D}}$ (resp., $\sqrt{\frac{(1-\varepsilon^2)}{4D}}$). On peut s'attendre à ce que la valeur de ℓ soit proche de $\frac{1}{2}(1+\varepsilon)$ lorsque la bonne hypothèse est effectuée, et proche de $\frac{1}{2}$ lorsqu'elle est erronée. Lorsque M augmente, l'écart type diminue de telle sorte que les valeurs de ℓ sont de plus en plus concentrées autour de leur valeur moyenne. Le critère de sélection choisi va conduire à réaliser deux types d'erreurs :

- **fausses alarmes :** une fausse alarme se produit lorsqu'une hypothèse erronnée est considérée comme la valeur initiale du LFSR.
- **non détection :** une non-détection se produit lorsque l'hypothèse correcte n'est pas reconnue comme la valeur initiale du LFSR.

Dans le cas des attaques par corrélation, ces deux causes d'erreur ne jouent pas le même rôle. En effet, au cours de l'exploration des valeurs initiales possibles du LFSR, la probabilité a priori de considérer une valeur erronée est très grande devant la probabilité de considérer la bonne hypothèse. Par conséquent si on veut mener l'attaque, il est nécessaire de choisir la limite ℓ_{limit} et le nombre d'échantillons D tels que

- l'espérance du nombre de fausses alarmes sur l'ensemble des itérations soit proche de 0 ;
- la probabilité de non-détection ne soit pas trop importante.

Afin d'avoir un taux de fausses alarmes et un taux de non-détection plus petits que $\frac{1}{2}$, on fixe la contrainte

$$\frac{1}{2} \leq \ell_{limit} \leq \frac{1}{2}(1+\varepsilon).$$

Pour une hypothèse sur l'état initial, on peut exprimer la probabilité de fausse alarme p_{fa} et la probabilité de non-détection p_{nd} en fonction de M et ε au moyen de la fonction d'erreur complémentaire :

$$\mathsf{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^{+\infty} e^{-t^2} dt,$$

$$\begin{aligned} p_{fa} &= \tfrac{1}{2}\mathsf{erfc}(2\sqrt{2}(\ell_{limit} - \tfrac{1}{2})\sqrt{D}) \\ p_{nd} &= \tfrac{1}{2}\mathsf{erfc}(\tfrac{2\sqrt{2}}{\sqrt{1-\varepsilon^2}}(\tfrac{1}{2}(1+\varepsilon) - \ell_{limit})\sqrt{D}). \end{aligned}$$

Pour une probabilité de fausse alarme et de non détection maximales, on déduit la quantité d'échantillons D nécessaire en fonction de ε. Par exemple, pour un générateur de Geffe avec un LFSR de taille 45 bits, on a $\varepsilon = \frac{1}{2}$ et on souhaite $p_{fa} \leq 2^{-45}$ et $p_{nd} \leq 0,1$. On en déduit

$$\begin{cases} 2\sqrt{2}(\ell_{limit} - \frac{1}{2})\sqrt{M} & \geq & 5.35, \\ \frac{4\sqrt{2}}{\sqrt{1-\varepsilon^2}}(\frac{1}{2}(1+\varepsilon) - \ell_{limit})\sqrt{M} & \geq & 0.91, \end{cases}$$

d'où, en éliminant ℓ_{limit},

$$\begin{array}{rcl} M & \geq & 4(\frac{5.35}{2\sqrt{2}} + \frac{0.91(1-\varepsilon^2)}{2\sqrt{2}}))^2 \frac{1}{\varepsilon^2}, \\ M & \geq & 73. \end{array}$$

En général ε est petit et le terme en $\frac{1}{\varepsilon^2}$ domine la borne inférieure. On retient que la quantité de données D nécessaires pour réaliser cette attaque est de l'ordre de $\frac{1}{\varepsilon^2}$. Par ailleurs la complexité en temps de cette attaque est de l'ordre de $\frac{1}{\varepsilon^2}2^n$ et elle permet de recouvrer l'état du LFSR. Dans le cas du générateur de Geffe, une attaque du même type peut être menée pour recouvrer les états initiaux des LFSR 2 et 3. Une fois les LFSR 2 et 3 connus, on peut recouvrer par recherche exhaustive l'état initial du LFSR 1. Finalement, l'attaque par corrélation du générateur de Geffe permet de recouvrer son état interne avec complexité $2^{n_1} + N2^{n_2} + N2^{n_3}$ ce qui constitue un gain notable sur la recherche exhaustive, de complexité $2^{n_1+n_2+n_3}$ lorsque l'état est scindé de manière équilibrée entre les 3 LFSR.

Pour pouvoir mener des attaques par corrélation, il faut pouvoir trouver des relations linéaires biaisées entre bits de sortie et une partie de l'état interne dont le biais ne soit pas trop important. Ce type d'attaque n'est pas applicable directement quand il n'est pas possible de scinder l'état interne du GPA facilement, par exemple dans le cas d'un LFSR filtré, même en présence d'une fonction d'extraction ayant une approximation linéaire fortement biaisée. Pour pouvoir attaquer dans ce cas, on a recours à une généralisation des attaques par corrélation, les attaques par corrélation rapides [96, 97].

De la manière la plus générale possible, l'existence d'approximations linéaires biaisées reliant k bits de l'état interne et les bits de sortie permet de construire une instance d'un problème de décodage. On note x_i^t la valeur à l'instant t du bit i de la partie de l'état interne évoluant de manière linéaire. On note L la taille de cette partie de l'état. L'évolution de l'état interne étant linéaire, on peut exprimer facilement ses valeurs en fonction des valeurs initiales des bits de l'état interne. On note φ la fonction d'évolution linéaire de l'état interne.

Par ailleurs on note z_t le bit de suite chiffrante produit à l'instant t. La possibilité de réaliser une attaque par corrélation repose sur l'existence d'approximations de combinaisons linéaires des bits de l'état interne par des bits de sortie.

À titre d'exemple,

- dans le cas du LFSR filtré, toute approximation linéaire de la fonction booléenne utilisée permet de relier la sortie à l'instant t à l'état à l'instant t et donc à l'état initial avec un biais égal à celui de l'approximation linéaire de la fonction.

$$\varphi^t(x^0) \cdot a = z_t \ [\varepsilon].$$

- dans le cas de GRAINv0 [15], en combinant deux bits de sortie on obtient 16 approximations linéaires de même biais

$$\varphi^t(x^0) \cdot a_j = z_t \oplus z_{t+80} \ [2^{-8.67}], 1 \leq j \leq 16.$$

Supposons que l'attaquant dispose de D bits de suite chiffrante $z = (z^0, \ldots, z^{D-1})$ et qu'il peut en dériver M approximations de combinaisons linéaires de bits de l'état initial $x = x^0$ de biais ε

$$\sum_{i=0}^{L-1} x_i a_{i,j} = f^i(z)\ [\varepsilon], 0 \leq j < M.$$

L'objectif de l'attaquant est de retrouver $x = (x_0, \ldots, x_{L-1})$ à partir de ces approximations. Considérons le code linéaire $\mathcal{C}$ de paramètre (M, L), de matrice génératrice $A = (a_{i,j})_{0 \leq i < L, 0 \leq j < M}$. Le problème que l'attaquant doit résoudre est le décodage du mot $f = (f^0(z), \ldots, f^{M-1}(z))$. L'approximation peut-être interprétée comme la transmission du mot $x^0 A$ à travers un canal binaire symétrique de probabilité d'erreur $\frac{1}{2}(1-\varepsilon)$. Intuitivement, la quantité d'information portée par une approximation est égale à la capacité du canal de transmission qui peut être approximée par $\frac{\varepsilon^2}{2 \log 2}$ lorsque ε est petit. Pour reconstituer l'intégralité de l'état, il faut donc collecter assez d'approximations pour que la quantité d'information soit supérieure à la taille de l'état à recouvrer. En suppposant les approximations indépendantes, on obtient

$$L \leq \frac{M\varepsilon^2}{2 \log 2},$$

ce qui donne une borne inférieure sur le nombre d'approximations à collecter,

$$M \geq \frac{2L \log 2}{\varepsilon^2}.$$

Cependant cela suppose l'existence d'algorithmes de décodage efficace. Dans le cas des attaques par corrélation simple, l'algorithme de décodage utilisé est le décodage à maximum de vraisemblance, dont la complexité est en 2^L. Lorsque cette complexité est trop élevée, on a recours à une étape intermédiaire entre la collecte d'approximations et la résolution du problème : la construction de *tests de parité*. L'idée est de combiner les approximations collectées de manière à obtenir un code plus facile à décoder [78, 39]. La forme usuellement retenue est l'expression d'un bit de l'état à recouvrer en fonction de bits de la suite chiffrante et d'un sous-ensemble $\mathcal{B}$ de taille $B \leq L$ de l'état interne :

$$x_j = f^j(z) \oplus \sum_{i \in \mathcal{B}} x_i a_{i,j}\ [\varepsilon'].$$

L'application d'un décodage à maximum de vraisemblance est plus aisée sur le code constitué des équations de parité car seule la partie $\mathcal{B}$ de l'état interne est visée dans un premier temps par le décodage et le recouvrement du reste de l'état est facilité une fois la partie $\mathcal{B}$ de l'état obtenue. On remarque également que la combinaison d'approximations linéaires a un coût, puisqu'on obtient des approximations linéaires de biais plus faible. La perte de qualité des approximations linéaires peut être quantifiée par le lemme suivant, connu sous le nom de *piling-up lemma* et qui peut être énoncé de la manière suivante dans le cas booléen ($\mathbb{K} = \mathbb{F}_2$) :

Lemme 1 *Soit $X_i, 1 \leq i \leq h$, h variables aléatoires indépendantes distribuées selon des lois de Bernoulli de biais ε_i. Alors $\bigoplus_{i=1}^{h} X_i$ est une loi de Bernoulli de biais $\prod_{i=1}^{h} \varepsilon_i$.*

Ainsi la combinaison de h approximations de biais ε produit une approximation de biais ε^h. L'idée sous-jacente aux algorithmes de construction de tests de parité est de construire des approximations linéaires faisant intervenir les mêmes bits de l'état interne, sauf pour les bits du sous-ensemble de taille B.

L'algorithme décrit en [78] réalise un compromis temps-mémoire pour trouver des collisions sur les bits de l'état interne ne faisant pas partie du sous-ensemble de taille B entre deux combinaisons d'approximations, chaque collision donnant un test de parité. Une optimisation algorithmique permet de diminuer la quantité de mémoire nécessaire à la réalisation de ce compromis temps-mémoire [39]. Cette phase de construction de tests de parité ne dépend pas des sorties de l'algorithme de chiffrement par flot et peut être précalculée.

Le même article donne également une méthode pour évaluer efficacement les tests de parité au moyen de la transormée de Walsh rapide.

1.3.3 Attaques algébriques

Une grande partie des algorithmes cryptographiques symétriques peuvent complètement être décrits par un système d'équations algébriques multivariées dans $\mathbb{F}_2$, dont les variables sont des bits de clé ou d'états internes à l'algorithme et les équations sont constituées à partir des spécifications de l'algorithme et les données publiques (couple clair-chiffré, suites chiffrantes,...). Une voie d'attaque à envisager est la résolution directe d'un tel système par un adversaire. Cette approche a fait l'objet d'une grande attention [48, 40] dans le domaine de la cryptanalyse des algorithmes de chiffrement par bloc suite à la sélection par le NIST de Rijndael comme standard de chiffrement [103]. Cet algorithme de chiffrement par bloc possède une structure algébrique caractéristique, qui permet de construire à partir d'une paire (clair, chiffré) un système quadratique dont la résolution permet le recouvrement de la clé. Les attaques algébriques sont particulièrement attirantes du fait du peu de données qu'elles nécessitent en théorie : un système construit à partir de quelques paires (clair, chiffré) dispose en effet a priori d'une solution unique correspondant à la clé de l'algorithme. Cependant, en dehors de résultats sur des algorithmes réduits, cette approche n'a jusqu'à présent pas permis de casser d'algorithmes de chiffrement par bloc modernes, en raison de la taille des systèmes algébriques produits et de la grande complexité de la résolution d'un système d'équations multivariées [44].

Dans le cadre du chiffrement par flot, des résultats plus intéressants ont pu être obtenus. Ils découlent :

- de la possibilité de construire des systèmes algébriques surdéterminés. En effet, chaque bit de suite chiffrante fournit une nouvelle équation. On peut donc construire des systèmes comportant beaucoup plus d'équations que d'inconnues. Si on peut collecter plus d'équations que le nombre de monômes en les inconnues de degré d, degré maximal des fonctions booléennes rencontrées, on peut résoudre le système par linéarisation, *i.e.* en considérant chaque monôme comme une variable indépendante ;
- de la structure des systèmes d'équations générées : on peut parfois retraiter le système d'équations obtenu pour obtenir un système d'équations de degré moindre. On parle d'attaques algébriques rapides [45, 72]. Ces attaques ont conduit à définir des critères de sécurité additionnels pour les composants des algorithmes de chiffrement par flot.

On décrira plus en détail au chapitre 2 l'application de ces attaques au cas du LFSR filtré.

1.3.4 Attaques différentielles

La notion d'attaque différentielle émerge de la cryptanalyse des algorithmes de chiffrement par bloc et a notamment permis d'obtenir les premiers résultats contre l'algorithme DES [21]. Il s'agit d'étudier le comportement « différentiel » de l'algorithme, c'est à dire comment pour des couples de clairs de différence donnée et bien choisie, cette différence initiale se propage au cours du calcul de la fonction de chiffrement.

L'application d'attaques différentielles contre les algorithmes de chiffrement par flot est plus récente et a été formalisée par Muller en 2004 [102]. En effet, pour les premiers algorithmes de chiffrement par flot, le seul paramètre du GPA est la clé, paramètre sur lequel l'attaquant n'exerce aucun contrôle dans les scénarios d'attaque traditionnels. L'inclusion à part entière des IV dans les procédures d'initialisation des GPAs et l'utilisation de mécanismes introduisant clair (chiffrement par flot authentifié) ou chiffré (algorithme de chiffrement par flot auto-synchronisant) dans l'état du GPA qui fournit à l'attaquant un contrôle nouveau sur l'état du GPA, contrôle qu'il est susceptible d'exploiter en tirant parti du comportement différentiel des fonctions d'avance et d'extraction. De nombreuses attaques de ce type ont été publiées au début de la compétition eSTREAM, contre des algorithmes n'ayant pas suffisamment pris en considération le contrôle éventuel de l'attaquant sur la mise à l'IV de l'algorithme [77, 71, 141, 80, 142, 76]. On donne dans le chapitre suivant un exemple d'attaque différentielle sur la mise à l'IV d'un algorithme de chiffrement par flot.

Chapitre 2

Structure Algébrique du Registre Linéaire Filtré

Sommaire

On présente dans ce chapitre la cryptanalyse algébrique du registre linéaire filtré. Cet algorithme classique de chiffrement par flot, d'une grande simplicité, est défini par un registre linéaire et une fonction booléenne. L'étude des propriétés de la fonction booléenne peut conduire à des *attaques algébriques* contre ce GPA. Afin d'améliorer ces attaques, on peut de plus prendre en compte la riche structure liée à l'emploi d'un registre linéaire sous-jacent. Ceci permet d'étendre les attaques algébriques en *attaques algébriques rapides*, mais aussi d'établir des correspondances entre LFSR filtrés et LFSR combinés.

On commence par rappeler la définition et les propriétés des LFSR et des suites pseudo-aléatoires qu'ils génèrent. Dans un deuxième temps, on introduit les fonctions booléennes. On présente alors les attaques algébriques sur le LFSR filtré. On étudie ensuite les propriétés des suites produites par le LFSR filtré. Ces propriétés révèlent une structure sous-jacente qui peut être exploitée pour réaliser des attaques algébriques rapides.

2.1 Préliminaires

On commence par rappeler quelques définitions et propriétés d'objets mathématiques utiles à la cryptanalyse des LFSR.

2.1.1 Corps finis

Comme nous le verrons dans les sections suivantes, il existe des liens très forts entre les LFSR et LFSR filtrés et la théorie des corps finis. On commence par effectuer quelques rappels de théorie des corps finis. On renvoie à [90] pour une description plus exhaustive et les preuves des résultats énoncés.

Théorème 1 (Théorème de Wedderburn) *Tout corps fini est commutatif.*

Définition 1 *Soit $\mathbb{K}$ un corps fini. On note $1_{\mathbb{K}}$ l'élément neutre de la loi multiplicative. Le plus petit entier non nul p tel que $p \cdot 1_{\mathbb{K}} = 0_{\mathbb{K}}$ est appelé caractéristique du corps $\mathbb{K}$.*

Proposition 1 *La caractéristique d'un corps fini est un nombre premier.*

Proposition 2 *Soit $\mathbb{K}$ un corps fini. Si on note p sa caractéristique, alors le cardinal de $\mathbb{K}$ est de la forme p^m avec $m \in \mathbb{N}^*$.*

Proposition 3 *Soit $\mathbb{K}$ un corps fini de cardinal $q = p^m$. Le groupe des éléments inversibles $\mathbb{K}^*$ est cyclique. On désigne par éléments primitifs les générateurs de $\mathbb{K}^*$. Soit α un élément primitif de $\mathbb{K}$. Alors l'ensemble des éléments primitifs de $\mathbb{K}$ est*

$$\{\alpha^k : k \wedge (p^m - 1) = 1\}.$$

Le cardinal de cet ensemble est donné par $\varphi(p^m - 1)$, où φ est la fonction d'Euler.

Proposition 4 *Soit $\mathbb{K}$ un corps fini de cardinal $q = p^m$. Tout élément non nul γ de $\mathbb{K}$ vérifie $\gamma^{q-1} = 1$. On en déduit*

$$X^{p^m} - X = \prod_{\gamma \in \mathbb{K}} X - \gamma.$$

Proposition 5 (Construction d'un corps fini de caractéristique p) *Soit P un polynôme irréductible unitaire de $\mathbb{Z}/p\mathbb{Z}[X]$ de degré m. L'égalité modulo P des polynômes de $\mathbb{Z}/p\mathbb{Z}[X]$ définit une relation d'équivalence dont le quotient $(\mathbb{Z}/p\mathbb{Z}[X])/P$ est un corps de cardinal p^m.*

Proposition 6

$$X^{p^m} - X = \prod_{\substack{Q \in \mathbb{Z}/p\mathbb{Z}[X] \\ \text{irréductible unitaire} \\ \text{de degré } r | m}} Q(X)$$

On en déduit pour tout d l'existence de polynômes unitaires irréductibles de $\mathbb{Z}/p\mathbb{Z}[X]$ de degré d.

On en déduit également une caractérisation des polynômes irréductibles.

Proposition 7 *Soit P un polynôme unitaire de $\mathbb{Z}/p\mathbb{Z}[X]$ de degré m. P est irréductible ssi*

- *P divise $X^{p^m} - X$;*
- *P est premier avec $X^{p^r} - X, r|m$.*

Proposition 8 *Soit p entier premier et m entier positif. Il existe un corps de cardinal $q = p^m$. De plus deux corps de cardinal p^m sont isomorphes. On note $\mathbb{F}_q$ le corps à q éléments.*

Exemple : $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$.

Proposition 9 *Les sous-corps de $\mathbb{F}_q$, $q = p^m$, sont les corps $\mathbb{F}_d$ avec $d|m$. En particulier $\mathbb{F}_p$ est un sous-corps de $\mathbb{F}_q$.*

Définition 2 *Soit $\mathbb{F}_q$, $q = p^m$, un corps fini. La fonction*

$$\begin{aligned} \mathrm{Fr} : \mathbb{F}_q &\to \mathbb{F}_q \\ \alpha &\to \alpha^p \end{aligned}$$

est un automorphisme de $\mathbb{F}_q$, appelé morphisme de Frobenius. Cet automorphisme laisse les éléments de $\mathbb{F}_p$ invariants. L'ordre de Fr *dans le groupe des automorphismes de $\mathbb{F}_q$ laissant $\mathbb{F}_p$ invariant est m.*

On remarque également que Fr est un automorphisme d'espace vectoriel de $\mathbb{F}_q$ vu comme un $\mathbb{F}_p$-espace vectoriel.

Définition 3 *Soit $\mathbb{F}_q$ un corps fini et $\mathbb{F}_{q'}$ un sous-corps de $\mathbb{F}_q$. On note m la dimension de $\mathbb{F}_q$ comme $\mathbb{F}_{q'}$ espace vectoriel. On a $q = {q'}^m$. On définit la fonction trace comme*

$$\begin{aligned} \mathrm{Tr}_{q'} : \mathbb{F}_q &\to \mathbb{F}_{q'}, \\ \alpha &\to \sum_{i=0}^{m-1} \alpha^{{q'}^i}. \end{aligned}$$

Cette application est une forme linéaire de $\mathbb{F}_q$ vu comme un $\mathbb{F}_{q'}$-espace vectoriel. Lorsqu'il n'y a pas d'ambiguïté sur le corps d'arrivée, on note Tr *au lieu de $\mathrm{Tr}_{q'}$.*

Soit $\alpha \in \mathbb{F}_q^*$ et n un entier tel que $\alpha^{{q'}^n} = \alpha$. Par extension de l'écriture de la fonction trace, on note

$$\mathrm{Tr}_{q'}^n(\alpha) = \sum_{i=0}^{n-1} \alpha^{{q'}^i}.$$

Cette valeur est un élément de $\mathbb{F}_{q'}$.

Définition 4 *Soit $\mathbb{F}_q$, $q = p^m$, un corps fini et $\alpha \in \mathbb{F}_q$. L'application*

$$\begin{aligned} \Phi_\alpha : \mathbb{F}_p[X] &\to \mathbb{F}_q \\ P = \sum_{i=0}^{d} p_i X^i &\to P(\alpha) = \sum_{i=0}^{d} p_i \alpha^i \end{aligned}$$

est un morphisme d'anneau. L'ensemble des polynômes annulateurs de α est $\Phi_\alpha^{-1}(0) = \{P \in \mathbb{F}_p[X] | P(\alpha) = 0\}$. C'est un idéal de $\mathbb{F}_p[X]$, engendré par un polynôme unitaire m_α irréductible sur $\mathbb{F}[X]$. On désigne m_α sous le terme de polynôme minimal de α.

Définition 5 *Soit $\mathbb{F}_q$ un corps et $\mathbb{F}_{q'}$ un sous corps de $\mathbb{F}_q, q = {q'}^m$. La relation sur $\mathbb{F}_q$*

$$\gamma_1 \mathcal{R} \gamma_2 \text{ ssi } \exists r \text{ tel que } \gamma_2 = \gamma_1^{{q'}^r}$$

est une relation d'équivalence. Les éléments de la classe d'équivalence $\mathcal{C}_\gamma$ d'un élément γ sont désignés sous le nom de conjugés de γ. La taille d'une classe d'équivalence n_γ divise m. Le polynôme

$$P_\gamma = \prod_{\gamma' \in \mathcal{C}_\gamma} (X - \gamma'),$$

est un polynôme unitaire de degré $|\mathcal{C}_\gamma|$ dans $\mathbb{F}_{q'}[X]$ et est le polynôme minimal des éléments de $\mathcal{C}_\gamma$ sur ce corps. On note $\mathfrak{R}$ un ensemble contenant un membre de chaque classe d'équivalence.

Démonstration : Cette définition établit le lien entre racines dans $\mathbb{F}_q$ d'un polynôme de $\mathbb{F}_{q'}[X]$. Ce lien étant exploité par la suite, on donne une démonstration des propriétés énoncées.
La réflexivité et la transitivité de $\mathcal{R}$ sont immédiates, la symétrie découle de la relation $x^{q'^m} = x$ pour $x \in \mathbb{F}_q$, ce qui prouve que $\mathcal{R}$ est une relation d'équivalence.
L'ensemble des conjugués de $\gamma \in \mathbb{F}_q$ est de la forme $\{\gamma, \gamma^{q'}, \dots \gamma^{q'^{n_\gamma - 1}}\}$, ces éléments étant deux à deux distincts. n_γ est le plus petit entier tel que la suite obtenue en élevant à la puissance q' l'élément précédent et ayant pour terme initial γ répète une valeur précédemment atteinte de $\mathbb{F}_q$. On a $n_\gamma \leq m$. On montre de plus que $\gamma^{q'^{n_\gamma}} = \gamma$. En effet, supposons $\gamma^{q'^{n_\gamma}} = \gamma^{q'^i}, i > 0$. Dans ce cas les éléments suivant de la suite sont $\{\gamma^{q'^i} : i \geq n_\gamma\} = \{\gamma^{q'^i} \dots \gamma^{q'^{n_\gamma - 1}}\}$ Cet ensemble ne contient pas γ, ce qui est en contradiction avec $\gamma^{q'^m} = \gamma$. On en déduit que n_γ divise m.
On a

$$\begin{aligned} P_\gamma(X)^{q'} &= \textstyle\sum a_i^{q'} X^{iq'} \\ &= \textstyle\prod_{\gamma' \in \mathcal{C}_\gamma} (X - \gamma')^{q'} \\ &= \textstyle\prod_{\gamma' \in \mathcal{C}_\gamma} (X^{q'} - \gamma'^{q'}) \\ &= \textstyle\prod_{\gamma'' \in \mathcal{C}_\gamma} (X^{q'} - \gamma''), \text{ avec } \gamma'' = \gamma'^{q'} \\ &= P_\gamma(X^{q'}) \\ &= \textstyle\sum a_i X^{iq'} \end{aligned}$$

Chaque coefficient de P_γ vérifie l'équation de corps de $\mathbb{F}_{q'}$, donc $P_\gamma \in \mathbb{F}_{q'}[X]$.
Finalement, considérons $\gamma' = \gamma^{q'^i}$ un conjugué de γ et notons μ_γ le polynome minimal de γ sur $\mathbb{F}_{q'}$. On a

$$\begin{aligned} \mu_\gamma(\gamma') &= \mu_\gamma(\gamma^{q'^i}), \\ &= \mu_\gamma(\gamma)^{q'^i}, \\ &= 0, \end{aligned}$$

d'où $P_\gamma | \mu_\gamma$. □

Cette relation d'équivalence permet d'expliquer la correspondance entre les décompositions de $X^{p^m} - X$ comme produit de polynômes irréductibles et comme produit des polynômes unitaires de degré 1 de $\mathbb{F}_q[X]$.

Lorsqu'on considère un élément primitif α de $\mathbb{F}_q$, cette relation a une traduction naturelle sur les logarithmes en base α des éléments de $\mathbb{F}_q^*$.

Proposition 10 *Soit $\mathbb{F}_q$, $q = p^m$ un corps fini ainsi que α un élément primitif de $\mathbb{F}_q$. Les trois propositions suivantes sont équivalentes :*

- *α^{i_1} et α^{i_2} sont conjugués ;*
- *$\exists j$ tel que $i_2 = p^j i_1 \mod p^m - 1$;*
- *l'écriture en base p de i_2 se déduit de celle de i_1 par rotation de ses chiffres.*

Par extension, on note $\mathcal{C}_i$ la classe d'équivalence de l'exposant i, n_i sa taille et $\mathfrak{R}$ un ensemble contenant un représentant de chaque classe d'équivalence. De plus on note $\mathfrak{R}_d$ (resp. $\mathfrak{R}_{\leq d}$) le sous-ensemble de $\mathfrak{R}$ contenant des entiers dont l'écriture en base p comporte un nombre de chiffres non-nuls égal à d (resp. $\leq d$).

Définition 6 *Un polynôme $P \in \mathbb{F}_p[X]$ unitaire de degré m est dit primitif si c'est le polynôme minimal d'un élément primitif de $\mathbb{F}_{p^m}$.*

Proposition 11 *Un polynôme unitaire de degré m $P \in \mathbb{F}_p[X]$ est primitif si et seulement si les conditions suivantes sont satisfaites*

- *P est irréductible ;*
- $\min(r$ *tel que* P *divise* $X^r - 1) = p^m - 1$.

2.1.2 Algèbre linéaire

Nous utiliserons également par la suite les résultats d'algèbre linéaire suivants.
On rapelle la définition de la matrice compagnon d'un polynôme.

Définition 7 *Soit $\mathbb{K}$ un corps et $A = \sum_{i=0}^{L-1} a_i X^i + X^L \in \mathbb{K}[X]$ de degré L unitaire. La matrice compagnon de A est la matrice $M_A \in \mathcal{M}_{L,L}(\mathbb{K})$ définie par*

$$M_A = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{L-1} \end{bmatrix}.$$

La polynôme minimal et le polynôme caractéristique de cette matrice sont égaux à A.

Proposition 12 *Soit E un espace vectoriel de dimension finie n et u un endomorphisme de E, dont le polynôme minimal μ_u est de degré m. Il existe $x \in E$ tel que $\big(x, u(x), u^2(x), \ldots, u^{m-1}(x)\big)$ est libre. Le sous-espace vectoriel engendré par ces vecteurs est stable par u et appelé clôture stable de x par u.*

Les deux résultats suivants établissent des décompositions d'endomorphisme d'espace vectoriel.

Théorème 2 (Décomposition de Frobenius [89][Théorème 14.2.1]) *Soit E un espace vectoriel de dimension finie, non nul, sur un corps $\mathbb{K}$ et u un endomorphisme de E. Alors E admet une décomposition en somme directe*

$$E = E_1 \oplus \cdots \oplus E_r,$$

où chaque E_i est la clôture stable d'un élément de E par u. De plus, en notant μ_u^i le polynôme annulateur de la restriction de u à E_i et μ_u le polynôme annulateur de u on a

$$\mu_u^r | \mu_u^{r-1} | \cdots | \mu_u^1 = \mu_u.$$

Théorème 3 (Décomposition selon les facteurs du polynôme minimal) *Soit E un espace vectoriel de dimension finie, non nul, sur un corps $\mathbb{K}$ et u un endomorphisme de E. Le polynôme minimal de u admet une décomposition en puissances de facteurs irréductibles distincts*

$$\mu_u = \prod_{i=1}^{r} P_i^{e_i} (e_i \geq 1).$$

E peut être écrit comme la somme directe des noyaux des $P_i^{e_i}(u)$

$$E = \ker(P_1^{e_1}) \oplus \cdots \oplus \ker(P_r^{e_r}(u)).$$

On note $Q_i = \mu_u / P_i^{e_i}$. Les polynômes Q_i sont premiers entre eux, et on peut écrire une relation de Bezout $\sum A_i Q_i = 1$. L'endomorphisme $(A_i Q_i)(u)$ est le projecteur sur l'espace $\ker(P_i^{e_i})$.

2.1.3 Suite récurrente linéaire

On procède à présent à quelques rappels sur les suites récurrentes linéaires.

Définition 8 *Soit $\mathbb{K}^{\mathbb{N}}$ l'ensemble des suites d'éléments de $\mathbb{K}$. On munit cet ensemble d'une loi de composition interne $+$ et d'une loi de composition externe $\cdot$ définies de la manière suivante :*

$$\begin{array}{rcl} + : \mathbb{K}^{\mathbb{N}} \times \mathbb{K}^{\mathbb{N}} & \to & \mathbb{K}^{\mathbb{N}} \\ a = (a_0, a_1, \ldots), b = (b_0, b_1, \ldots) & \to & a + b = (a_0 + b_0, a_1 + b_1, \ldots), \\ \cdot : \mathbb{K} \times \mathbb{K}^{\mathbb{N}} & \to & \mathbb{K}^{\mathbb{N}} \\ \lambda, a = (a_0, a_1, \ldots) & \to & \lambda \cdot a = (\lambda a_0, \lambda a_1, \ldots). \end{array}$$

$(\mathbb{K}, +, \cdot)$ est un espace vectoriel. On note $\mathbf{0}$ la suite nulle.

Définition 9 *On appelle fonction de décalage et on note $\mathcal{D}$ l'endomorphisme de $\mathbb{K}^{\mathbb{N}}$ défini par*

$$\begin{array}{rcl} \mathcal{D} : \mathbb{K}^{\mathbb{N}} & \to & \mathbb{K}^{\mathbb{N}} \\ (a_0, a_1, a_2, \ldots) & \to & (a_1, a_2, \ldots) \end{array}$$

Proposition 13 *L'application*

$$\begin{array}{rcl} \Psi : (\mathbb{K}[X], +, \times, \cdot) & \to & (End_{\mathbb{K}^{\mathbb{N}}}, +, \circ, \cdot) \\ P = \sum_{i=0}^{d} p_i X^i & \to & \left(s \to \sum_{i=0}^{d} p_i \cdot \mathcal{D}^i(s) \right) \end{array}$$

est un morphisme d'algèbre.

Définition 10 *Soit $s \in \mathbb{K}^{\mathbb{N}}$ une suite d'éléments de $\mathbb{K}$. On dit que s est une* suite récurrente linéaire d'ordre L *s'il existe $A \in \mathbb{K}[X]$ de degré L tel que*

$$\Psi(A)(s) = \mathbf{0},$$

i.e. il existe $(a_0, a_1, \ldots, a_L) \in \mathbb{K}^{L+1}, a_L \neq 0$ tel que

$$\forall t \in \mathbb{N}, \sum_{i=0}^{L} a_i s_{t+i} = 0.$$

On dit que A est un polynôme de récurrence *de s.*

Le morphisme d'algèbre Ψ permet de dériver facilement quelques propriétés utiles des suites linéaires récurrentes, dont les propositions suivantes.

Proposition 14 *Soit $s \in \mathbb{K}^{\mathbb{N}}$. L'ensemble des polynômes de récurrence de s forme un idéal de $\mathbb{K}[X]$. $\mathbb{K}[X]$ étant principal, il est engendré par un polynôme μ_s unitaire appelé polynôme minimal de s.*

Proposition 15 *Soit $s_1, s_2 \in \mathbb{K}^{\mathbb{N}}$ deux suites récurrentes linéaires, de polynômes minimaux μ_{s_1}, μ_{s_2}. Si μ_{s_1} et μ_{s_2} sont premiers entre eux, alors $\mu_{s_1+s_2} = \mu_{s_1}\mu_{s_2}$.*

Démonstration : Il est clair que $\mu_{s_1+s_2} | \mu_{s_1}\mu_{s_2}$. Soit P un polynôme de récurrence de $s_1 + s_2$, *i.e.* $\Psi(P)(s_1 + s_2) = 0$. On a $0 = \Psi(\mu_{s_1}P)(s_1 + s_2) = \Psi(\mu_{s_1}P)(s_1) + \Psi(\mu_{s_1}P)(s_2) = \Psi(\mu_{s_1}P)(s_2)$. On a donc $\mu_{s_2} | \mu_{s_1}P$, et comme μ_{s_1} et μ_{s_2} sont premiers entre eux, $\mu_{s_2} | P$. De même $\mu_{s_1} | P$, et comme μ_{s_1} et μ_{s_2} sont premiers entre eux, $\mu_{s_1}\mu_{s_2} | P$. □

Cette proposition s'étend naturellement au cas de n suites récurrentes linéaires dont les polynômes minimaux sont premiers entre eux deux à deux.

2.2 Registre à décalage à rétroaction linéaire

Définition 11 *Un LFSR de taille L sur un corps fini $\mathbb{K} = \mathbb{F}_{p^m}$ est constitué d'une mémoire de L éléments de $\mathbb{K}$. L'opération de mise à jour du LFSR est basée sur un décalage des valeurs présentes dans son état, complété par une opération linéaire, appelée* rebouclage. *L'opération d'extraction consiste à retourner la valeur disparaissant suite au décalage.*

On rencontre principalement deux rebouclages qui déterminent deux types de LFSR.

- **LFSR de type Fibonacci.** Le rebouclage consiste à calculer une combinaison linéaire des valeurs présentes dans l'état avant le décalage et à l'introduire dans l'espace laissé vacant par le décalage.

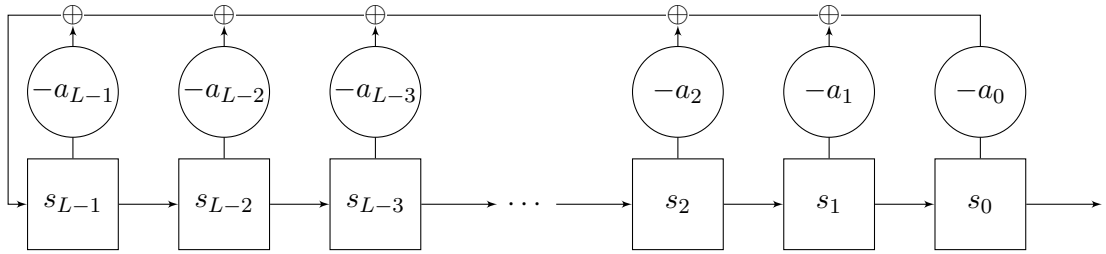


FIGURE 2.1 – LFSR de type Fibonacci

- **LFSR de type Galois.** Le rebouclage consiste à ajouter à l'état après décalage, en attribuant au préalable la valeur 0 à la case laissée inoccupée par le décalage, un multiple de la valeur extraite par le décalage.

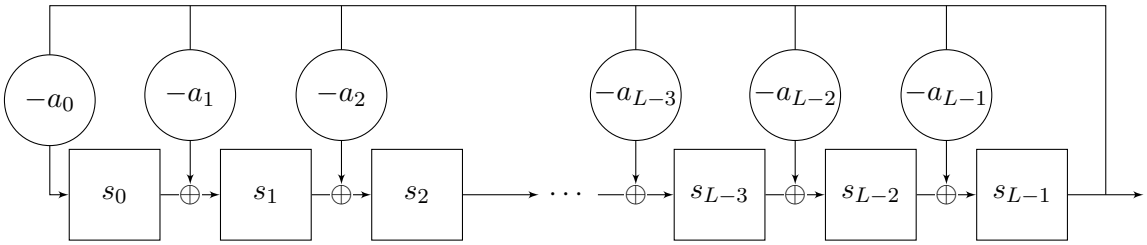


FIGURE 2.2 – LFSR de type Galois

2.2.1 Représentation matricielle

On peut représenter l'état d'un LFSR comme un élément de l'espace vectoriel $\mathbb{K}^L$. L'opération de mise à jour étant linéaire, elle définit un endomorphisme de cet espace vectoriel. On note

$$S^t = (s_0^t, s_1^t, \ldots, s_{L-1}^t)^T$$

le vecteur des valeurs de l'état du LFSR à l'instant t. Avec ce choix de base, on obtient les représentations matricielles suivantes :

$$S^{t+1} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{L-2} & -a_{L-1} \end{bmatrix} S^t$$

LFSR de type Fibonacci

$$S^{t+1} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ddots & 0 & -a_{L-1} \\ 0 & 0 & 0 & \cdots & 1 & -a_{L-1} \end{bmatrix} S^t$$

LFSR de type Galois

On constate qu'avec les conventions adoptées la matrice de mise à jour d'un LFSR de Galois est la matrice compagnon d'un polynôme P de $\mathbb{K}[X]$ et que la matrice de rebouclage d'un LFSR de Fibonacci faisant usage des mêmes coefficients de rebouclage dans l'ordre inverse est égale à la transposée de la matrice du LFSR de type Galois. On appelle ce polynôme le polynôme de rebouclage du LFSR.

Remarquons que le polynôme caractéristique de la matrice de rebouclage M du polynôme de Fibonacci est le polynôme P de degré L, et que si l'on note (e_i) la base canonique de $\mathbb{K}^L$, $(e_{n-1}, Me_{n-1}, \ldots, M^{L-1}e_{n-1})$ est un système libre, qui constitue une base de $\mathbb{K}^L$. Par conséquent le poynôme minimal de M est de degré au moins L et il est donc égal à P. En écrivant la matrice de rebouclage dans cette base, on obtient la matrice compagnon d'un polynôme, égal au polynôme P par égalité des polynômes caractéristiques de matrice semblable. À un changement de base près, les deux types de registres sont donc équivalents.

2.2.2 Lien avec les suites récurrentes linéaires

Une autre manière de constater l'équivalence de ces deux types de registres est de considérer les suites des bits produites par les LFSR. Dans les deux cas il s'agit de suites linéaires récurrentes d'ordre L, de coefficients $a_0, \ldots, a_{L-1}, 1$. Dans le cas du LFSR de type Fibonacci, il suffit de constater que l'élément en position i dans le LFSR à l'instant t est la valeur extraite à l'instant $t+i$, $s(t+i)$. L'équation de rebouclage se traduit immédiatement en l'équation de récurrence linéaire de la suite de sortie. Pour le cas du LFSR de Galois, on considère L avances consécutives du LFSR. À l'instant initial, noté t, la valeur $-a_0 s(t)$ est introduite à la première position du registre. Lors de la mise à jour suivante cette valeur est décalée en deuxième position et $-a_1 s(t+1)$ y est ajouté. Cette opération est répétée jusqu'à ce que $\sum_{i=0}^{L-1} -a_i s(t+i)$ se trouve en dernière position après l'étape $t+L-1$. C'est cette valeur qui est produite à l'instant $t+L$, d'où l'équation de récurrence.

Considérons l'ensemble des suites générées par un LFSR sur $\mathbb{F}_{p^m}$ de polynôme de rebouclage A. A est un polynôme de récurrence commun à toutes ces suites. Afin de s'assurer que toutes les suites ont des propriétés similaires, on choisit A polynôme irréductible. Dans ces conditions, le polynôme minimal de toute suite générée par le LFSR est soit 1, cas de la suite nulle, soit A. Si $\deg(A) > 1$, A divise $X^{p^m-1} - 1$ et la période maximale d'une suite récurrente générée par le LFSR est $p^m - 1$. Le choix de A primitif permet d'atteindre cette période maximale.

2.2.3 Représentation algébrique

Soit $A = \sum_{i=0}^{L} a_i X^i$ un polynôme irréductible de $\mathbb{F}_q[X]$ de degré L. Le nombre de suites sur $\mathbb{F}_q$ récurrentes d'ordre L ayant A comme polynôme de récurrence est q^L (il suffit de fixer les L premiers éléments de la suite pour déterminer tous les éléments suivants par application de la relation de récurrence). Considérons à présent le corps $\mathbb{F}_q[X]/A$ et notons α la classe d'équivalence de X. Considérons l'ensemble des suites

$$\{(Tr_q(\beta\alpha^t))_{t\in\mathbb{N}}, \beta \in \mathbb{F}_{q^L}\}.$$

Ces suites sont à valeurs dans $\mathbb{F}_q$, par définition de la trace. Il s'agit de suites récurrentes ayant A pour polynôme de rebouclage :

$$\sum_{i=0}^{L} a_i Tr_q(\beta\alpha^{t+i}) = Tr_q(\beta\alpha^t \sum_{i=0}^{L} a_i\alpha^i) = 0.$$

Enfin les suites sont deux à deux distinctes. En effet, la trace étant linéaire, il suffit de montrer qu'une suite nulle correspond à $\beta = 0$. On introduit γ un élément primitif du corps quotient et k tel que $\alpha = \gamma^k$. On a

$$\begin{aligned} \forall t,\ \textstyle\sum_i \beta^{q^i} \alpha^{tq^i} &= 0, \\ \forall t,\ \textstyle\sum_i \beta^{q^i} (\gamma^t)^{kq^i} &= 0, \end{aligned}$$

les exposants kq^i étant pris modulo $q^L - 1$. Le polynôme $\sum_i \beta^{q^i} X^{kq^i}$ est un polynôme de $\mathbb{F}_{q^L}[X]$ de degré $< q^L - 1$ ayant $q^L - 1$ racines distinctes. Par conséquent il s'agit du polynôme nul et $\beta = 0$.

Par un argument de dénombrement, on en déduit le théorème suivant :

Théorème 4 *Les suites d'éléments de $\mathbb{F}_q$ récurrentes linéaires d'ordre L, de polynôme de récurrence A irréductible, sont de la forme $(Tr(\beta\alpha^t))$, où $\alpha, \beta \in \mathbb{F}_{q^L}$, α racine de A.*

Dans cette forme, la multiplication par α encode la fonction d'avance du LFSR et β la valeur initiale du LFSR.

2.2.4 Algorithme de Berlekamp-Massey

Pour conclure cette présentation des LFSR, on décrit l'algorithme de Berlekamp-Massey[93]. Cet algorithme permet de construire le polynôme de récurrence de plus petit degré d'une suite finie. Si la suite est produite par un LFSR, cet algorithme permet de retrouver son polynôme de rebouclage dès que la longueur de la suite est supérieure à deux fois le degré de ce polynôme.

Le degré du polynôme retourné est appelé complexité linéaire de la suite $(s_0, \ldots, s_{N-1})$.

La correction de cet algorithme se montre en considérant les invariants de boucle suivants, considérés après le calcul de e_f :

- d_f est le degré de f, d_g est le degré de g ;
- v_f est le nombre vérifié de 0 initiaux de la suite $\Psi(f)(a)$, $e_f = \Psi(f)(a)_{v_f}$;
- v_g est le nombre de 0 initiaux de la suite $\Psi(g)(s)$, e_g est la première valeur non nulle de cette suite, *i.e.* $\Psi(g)(s)_{v_g} = e_g$;
- $v_f + d_f = n$;
- $d_f + v_f > d_g + v_g$;
- $v_f + v_g = n - 1$.

La propriété suivante est également utile : pour $v \leq \deg(h), \Psi(X^v h)(a)$ est la suite obtenue en supprimant les v premiers éléments de la suite $\Psi(h)(a)$.

La complexité de l'algorithme de Berlekamp-Massey est de l'ordre de $\mathcal{O}(LN)$ pour une suite de longueur N et de complexité linéaire L. Lorsqu'il est appliqué à une suite générée par un LFSR de taille L, on l'applique sur une suite de longueur $2L$ et la complexité est $\mathcal{O}\left(L^2\right)$.

Le problème de la reconstruction du polynôme de rebouclage d'un LFSR peut encore s'exprimer au moyen de l'algorithme d'Euclide étendu. On écrit $S(x) = \sum_{i=0}^{2L-1} s_i X^i$ et $\tilde{A}(X) = X^L P(\frac{1}{X}) = \sum_{i=0}^{L} a_{L-i} X^i$. Les termes de degré supérieur à L et strictement inférieur à $2L$ du produit $C = \tilde{A}S$ sont nuls :

$$i \geq L, c_i = \sum_{j=0}^{L} a_{L-j} s_{i-j} = \sum_{j=0}^{L} a_j s_{j+i-L} = 0.$$

On a donc

$$\tilde{A}S + Q * X^{2L} = R, \deg(R) < L.$$

Algorithme 2 Algorithme de Berlekamp-Massey

Entrée : $s_0, s_1, \dots s_{N-1}$ une suite de N éléments de $\mathbb{F}_q$

Sortie : L un entier et A un polynôme de récurrence satisfait par la suite, de degré minimum.

si pour tout $0 \leq i < N$, $s_i = 0$ **alors**
  retourner 0, 1
fin si
$k = \min(i \text{ tel que } a_i \neq 0)$
$f(X) = X^{k+1} + 1, d_f = k + 1, v_f = 0$
$g(X) = 1, d_g = 0, v_g = k, e_g = s_k$
pour $n = k + 1 \to N - 1$ **faire**
  $e_f = \sum_{i=0}^{d_f} f_i s_{n-d_f+i}$
  si $e_f = 0$ **alors**
    $v_f = v_f + 1$
  sinon
    si $v_f < v_g$ **alors**
      $f(X) = f(X) - e_f(e_g)^{-1} x^{v_g - v_f} g(X)$
      $v_f = v_f + 1$
    sinon
      $T(X) = f(X)$
      $f(X) = X^{v_f - v_g} f(X) - e_f(e_g)^{-1} g(X)$
      $g(X) = T(X)$
      $d_g = d_f$
      $d_f = d_f + v_f - v_g$
      $v_f = v_g + 1$
      $v_g = v_f$
      $e_g = e_f$
    fin si
  fin si
fin pour
retourner d_f, f

En appliquant l'algorithme d'Euclide étendu à S et X^{2L}, interrompu dès qu'un reste est de degré $< L$ on peut reconstituer $\tilde{A}$ et donc A. La complexité de cet algorithme est là encore quadratique.

Dans la suite de la section on se restreindra aux corps de caractéristique 2.

2.3 Fonction booléennes

On présente dans cette section quelques résultats sur les fonctions booléennes, l'un des composants du LFSR filtré.

2.3.1 Définition

Définition 12 *On appelle fonction booléenne à $n \geq 1$ variables une fonction*

$$f : \{0,1\}^n \to \{0,1\}.$$

On note $\mathcal{B}_n$ l'ensemble des fonctions booléennes à n variables.

On utilisera le terme de *bit* pour désigner un élément de $\{0,1\}$. Dans la suite on identifiera souvent les éléments de $\{0,1\}^n$ aux entiers compris entre 0 et $2^n - 1$ à travers leur écriture en base 2. On notera également $\mathbf{x} = (x_0, \ldots, x_{n-1})$ les composantes d'un élément de $\{0,1\}^n$.

Une fonction booléenne est complètement définie par les valeurs qu'elle prend sur les 2^n éléments de son ensemble de départ.

Définition 13 *On appelle table de vérité de f et on note $\mathcal{T}(f)$ le vecteur $(f(0), \ldots, f(2^n-1))^T$ de $\{0,1\}^n$.*

Définition 14 *Soit f une fonction booléenne. On appelle* support *de f, et on note* $\mathrm{supp}(f)$ *l'ensemble des valeurs où f prend la valeur 1. Son complémentaire, où f prend la valeur* 0*, est appelé* noyau *de f, noté* $\ker(f)$*. Le* poids *de f, noté $wt(f)$ est le cardinal de son support. Il s'agit du poids de Hamming de $\mathcal{T}(f)$.*

Les fonctions booléennes ont d'autres représentations les définissant de manière complète. Des transformations permettent de passer de $\mathcal{T}(f)$ à ces autres représentations. On présente la forme algébrique normale, qui nous intéressera plus particulièrement.

2.3.2 Représentation algébrique

Soit $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ une liste de n indéterminées, $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}) \in \{0,1\}^n$ et $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1}) \in \mathbb{N}^n$. On note $\mathbf{X}^{\mathbf{e}}$ le monôme $\prod_{i=0}^{n-1} X_i^{e_i}$ et $\mathbf{x}^{\mathbf{e}}$ le bit $\prod_{i=0}^{n-1} x_i^{e_i}$.

Définition 15 *On dit que f est une fonction booléenne monomiale s'il existe un monôme $m = X^e$ pour $e \in \mathbb{N}^n$ tel que*

$$\begin{array}{rcl} f : \{0,1\}^n & \to & \{0,1\} \\ \mathbf{x} & \to & \mathbf{x}^e. \end{array}$$

Or $\forall x \in \mathbb{F}_2, x^2 - x = 0$. Deux monômes dont les exposants s'annulent aux mêmes positions s'évaluent donc de la même manière sur $\{0,1\}^n$, et on peut remplacer dans la définition $\mathbb{F}_2[\mathbf{X}]$ par $\mathbb{F}_2[\mathbf{X}]/\langle X_0^2 - X_0, X_1^2 - X_1, \ldots, X_{n-1}^2 - X_{n-1}\rangle$, l'anneau obtenu en quotientant $\mathbb{F}_2[\mathbf{X}]$ par l'idéal engendré par les $X_i^2 - X$. Les monômes dans ce quotient sont représentés par un monôme dans

les valeurs de l'exposant sont égales à 0 ou 1. Il y a 2^n tels exposants, définissant 2^n fonctions monomiales distinctes. On note $\mathbb{T}^n$ cet ensemble de monômes. De même que pour les vecteurs de n bits, on peut représenter les fonctions monomiales de n variables par un entier compris entre 0 et $2^n - 1$ en définissant un ordre total sur les monômes de $\mathbb{T}^n$. On considèrera les deux ordres suivants :

Définition 16 (Ordre lexicographique inverse) *Soit* $\mathbf{e}, \mathbf{e}' \in \mathbb{N}^n$. *L'ordre lexicographique inverse, noté* rl, *est défini par*

$$x^{\mathbf{e}} \overset{\mathsf{rl}}{\leq} x^{\mathbf{e}'} \Leftrightarrow \begin{cases} \mathbf{e} = \mathbf{e}' \text{ ou} \\ e_i < e'_i \text{ où } i = \max\left(0 \leq j < n | e_j \neq e'_j\right) \end{cases}$$

Restreint aux monômes de $\mathbb{T}^n$, cet ordre correspond à associer à chaque monôme l'entier dont l'écriture en base 2 est donnée par les exposants du monôme.

On rappelle que le degré d'un monôme $\mathbf{X}^{\mathbf{e}}$, noté $\deg(\mathbf{X}^{\mathbf{e}})$ est égal à la somme de ses exposants.

Définition 17 (Ordre du degré lexicographique inverse) *Soit* $\mathbf{e}, \mathbf{e}' \in \mathbb{N}^n$. *L'ordre du degré lexicographique inverse, noté* drl, *est défini par*

$$x^{\mathbf{e}} \overset{\mathsf{drl}}{\leq} x^{\mathbf{e}'} \Leftrightarrow \begin{cases} \mathbf{e} = \mathbf{e}' \text{ ou} \sum e_i < \sum e'_i \text{ ou} \\ \sum e_j < \sum e'_j \text{ et } e_i < e'_i \text{ où } i = \max\left(0 \leq j < n | e_j \neq e'_j\right) \end{cases}$$

La matrice des évaluations des fonctions monomiales joue un rôle particulier dans l'étude des propriétés des fonctions booléennes.

Définition 18 *On considère les valeurs de* $\{0,1\}^n$ *ordonnées* $\mathbf{z} = (z_0, \ldots z_{2^n-1})$,*et les monômes de* $\mathbb{T}^n$ *ordonnés selon l'ordre lexicographique inverse,* $\alpha = (\alpha_0, \ldots, \alpha_{2^n-1})$. *On appelle matrice monomiale à n variables la matrice de* $\mathcal{M}_{2^n,2^n}(\mathbb{F}_2)$ *définie par :*

$$\mathcal{V}_n = \begin{bmatrix} z_0^{\alpha_0} & z_0^{\alpha_1} & \cdots & z_0^{\alpha_{2^n-1}} \\ z_1^{\alpha_0} & z_1^{\alpha_1} & \cdots & z_1^{\alpha_{2^n-1}} \\ \vdots & \vdots & \vdots & \vdots \\ z_{2^n-1}^{\alpha_0} & z_{2^n-1}^{\alpha_1} & \cdots & z_{2^n-1}^{\alpha_{2^n-1}} \end{bmatrix}.$$

Le terme général de cette matrice est

$$\mathcal{V}_{n,i,j} = \delta_{\{(i \oplus (2^n-1)) \cdot j = 0\}}$$

où $\oplus$ *et* $\cdot$ *s'appliquent bit à bit aux entiers* i *et* j, *et* $\delta_{\{\mathsf{Ev}\}}$ *vaut* 1 *ssi* Ev *est vrai,* 0 *sinon.*

Cette matrice est une matrice de Vandermonde multivariée évaluant les fonctions définies par α en les points $\mathbf{z}$ [113].

Proposition 16 *Les matrices* $\mathcal{V}_n, n \in \mathbb{N}$, *possèdent une structure récursive par bloc On a*

$$\mathcal{V}_0 = [1],$$

$$\mathcal{V}_{n+1} = \begin{bmatrix} \mathcal{V}_n & 0 \\ \mathcal{V}_n & \mathcal{V}_n \end{bmatrix}.$$

De plus pour tout $n \in \mathbb{N}$, $\mathcal{V}_n$ *est inversible et égale à son inverse.*

On en déduit que les fonctions booléennes monomiales sont linéairement indépendantes et par un argument de dimension que toute fonction booléenne peut être écrite de manière unique comme somme de fonctions booléennes monomiales.

Définition 19 *Soit f une fonction booléenne à n variables. On appelle coefficients de f et on note $\mathcal{C}(f) = (f_0, f_1, \ldots, f_{2^n-1})$ l'unique vecteur de $\{0,1\}^{2^n}$ tel que*

$$\forall \mathbf{x} \in \{0,1\}^n, f(\mathbf{x}) = \sum_{i=0}^{2^n-1} f_i \mathbf{x}^i,$$

où i est identifié avec le i-ème monôme de $\mathbb{T}^n$ selon l'ordre lexicographique inverse. L'écriture de f sous forme de fonction booléenne polynomiale est appelé forme algébrique normale de f.

On appelle degré de f et on note $\deg(f)$ *le degré maximum d'un monôme apparaissant dans la forme algébrique normale de f*

$$\deg(f) = \max\left(\deg(\mathbf{x}^i) | f_i \neq 0\right).$$

La matrice monomiale à n variables permet de passer de la représentation de $\mathcal{T}(f)$ à la représentation $\mathcal{C}(f)$ et inversement. En effet

$$\mathcal{T}(f)^T = \mathcal{V}_n \mathcal{C}(f)^T, \mathcal{C}(f)^T = \mathcal{V}_n \mathcal{T}(f).$$

On peut tirer partie de la structure récursive de $\mathcal{V}_n$ pour réaliser ces produits matrice-vecteur en temps $\mathcal{O}(n2^n)$ grâce à l'algorithme 3.

Algorithme 3 Transformée de Moebius rapide

Entrée : $n \in \mathbb{N}, v = (v_0, v_1, \ldots, v_{2^n-1}) \in \{0,1\}^{2^n}$.
Sortie : $V_n.v$, calculé en place dans le vecteur v.

```
pour k = 0 → n − 1 faire
  pour i = 0 → 2^{n−k−1} − 1 faire
    pour j = 0 → k − 1 faire
      v_{i∗2^{k+1}+2^k+j} = v_{i∗2^{k+1}+j} + v_{i∗2^{k+1}+2^k+j}
    fin pour
  fin pour
fin pour
```

2.4 Cryptanalyse algébrique du LFSR filtré

On s'intéresse à présent au LFSR filtré et plus particulièrement à sa cryptanalyse algébrique.

Définition 20 *Un LFSR filtré sur $\mathbb{F}_2$ est un générateur pseudo-aléatoire défini par :*

- *un LFSR de taille L sur $\mathbb{F}_2$ et de polynôme de rebouclage irréductible $A(X)$ de degré L. On adopte la représentation de Fibonacci. On note $\mathbf{s}^t = (s_0^t, \ldots, s_{L-1}^t)$ les valeurs des cases du LFSR à l'instant t et ϕ la fonction d'avance, linéaire, du LFSR. Si $s(t)$ désigne la sortie du LFSR à l'instant t, on a $s_i^t = s(t+i)$;*
- *une fonction booléenne f à L variables.*[1] *On note d son degré ;*

1. En général, on utilise des fonctions booléennes utilisant moins de variables que la taille du LFSR et on précise les positions du LFSR alimentant les entrées de f. Ces fonctions s'expriment également dans le formalisme adopté.

L'initialisation consiste à écrire la clé dans l'état du LFSR. La fonction d'évolution de ce GPA est la fonction d'évolution de son LFSR. La fonction d'extraction à l'instant t est donnée par

$$z(t) = f(s_0^t, s_1^t, \ldots, s_{L-1}^t).$$

On notera de manière abrégée $z(t) = f(\mathbf{s}^t)$.

Remarque : On se restreint par la suite au cas où le polynôme de rebouclage $A(X)$ est primitif. Nous verrons plus bas que le cas d'un LFSR filtré utilisant un polynôme de rebouclage simplement irréductible correspond à une version dégénérée du LFSR filtré avec polynôme primitif.

La cryptanalyse du LFSR a fait l'objet de nombreux travaux. On trouve principalement deux classes d'attaques : les attaques statistiques exploitant des biais de la fonction booléenne pour essayer de reconstruire l'état interne, cf section 1.3.2, et les attaques algébriques qui vont être détaillées ci-dessous.

2.4.1 Attaques algébriques

Le principe des attaques algébriques du LFSR filtré a été développé à partir de 2003 suite à la publication de l'article de Courtois et Meier [47]. Le principal résultat de cet article est la possibilité d'améliorer l'attaque algébrique naïve du LFSR filtré en faisant usage d'annulateurs de f. Il décrit quantité de scénarios d'attaque qui ont été unifiés dans des articles ultérieurs (voir par exemple [95]). Il motive l'étude de la construction d'annulateurs de petit degré de fonctions booléennes et définit un critère de résistance des fonctions booléennes aux attaques algébriques, l'*immunité algébrique*. De nombreux algorithmes ont été proposés pour construire des annulateurs de bas degré et réaliser le calcul de l'immunité algébrique [3, 2, 54, 55]. En 2003, une amélioration des attaques algébriques, appelée attaques algébriques rapides, est proposée [45]. Elle est basée sur un compromis temps-données. Elle a conduit à la cryptanalyse de Sfinks [46], un candidat à la compétition eSTREAM [60]. Le critère de résistance contre les attaques algébriques s'étend naturellement à un critère de résistance contre les attaques algébriques rapides. Hawkes et Rose [72] se concentrent sur la complexité des attaques algébriques rapides et optimisent deux parties délicates de ces attaques. Initialement, les attaques algébriques et attaques algébriques rapides exploitent essentiellement la représentation matricielle du LFSR filtré et reposent sur des techniques d'algèbre linéaire. En 2007, [124] réexplicite les attaques algébriques rapides sans recherche d'annulateur d'un point de vue algébrique. [123] étend l'analyse et fournit une expression algébrique explicite du LFSR filtré.

Attaques algébriques naïves. Les attaques algébriques reposent sur le constat que chaque bit de sortie du LFSR filtré fournit une équation polynomiale en les bits de l'état initial :

$$z(t) = f(\mathbf{s}^t) = f(\phi^t(\mathbf{s}^0)).$$

La fonction booléenne $f \circ \phi^t$ est un polynôme de degré d dont la forme algébrique normale peut être déduite de la forme algébrique normale de f et de celle de ϕ. On peut donc, en collectant N bits de suite chiffrante, constituer un système de N équations polynomiales en L inconnues. Dès que le système est surdéterminé, *i.e.* $N > L$, il possède avec une forte probabilité une seule solution, qui est l'état initial du LFSR.

La taille du système construit au cours des attaques algébriques est liée à la technique de résolution du système adoptée. Deux techniques de résolution ont principalement été étudiées :

- **La linéarisation :** on considère chaque monôme intervenant dans le système polynomial comme une variable indépendante, ce qui permet d'obtenir un système linéaire, résoluble à l'aide de techniques classiques d'algèbre linéaire. Pour un LFSR de taille L et une fonction booléenne de degré d, tous les monômes de degré inférieur ou égal à d en les L variables sont susceptibles d'apparaître. Le système linéarisé comporte donc $\mathcal{N}_L^d$ variables avec :

$$\mathcal{N}_L^d = \sum_{i=0}^{d} \left(\begin{array}{c} L \\ i \end{array} \right).$$

  Afin d'obtenir une solution unique, on doit collecter $\mathcal{N}_L^d$ équations linéairement indépendantes. La complexité de résolution de ce système correspond à l'inversion d'une matrice carrée de taille $\mathcal{N}_L^d$. L'algorithme de Gauss classique a une complexité cubique. L'algorithme de Strassen [134] offre de meilleures performances avec une complexité en $\mathcal{O}\left(n^{2.807}\right)$. L'algorithme de Coppersmith-Winograd améliore la complexité asymptotique [42], mais cette complexité cache des constantes importantes. On notera ω l'exposant intervenant dans la formule de complexité de la résolution du système.
- **Le calcul d'une base de Gröbner :** Il s'agit d'une méthode générale de résolution de systèmes polynomiaux. L'idée est de trouver un système générateur de l'idéal engendré par la famille de polynômes composant le système, ayant des propriétés spécifiques pour un ordre monomial donné. On renvoie à [49] pour une présentation générale des bases de Gröbner. L'application des bases de Gröbner à la cryptanalyse algébrique du LFSR filtré a été étudiée dans [62].

On constate ici un premier compromis temps-donnée : le choix de la méthode de résolution peut conduire à des systèmes linéaires, mais qui requièrent une grande quantité de suites chiffrantes, ou à de petits systèmes mais dont la complexité de résolution est élevée.

Par la suite on se placera dans le cas où la méthode de résolution choisie est la linéarisation, et on construira les systèmes à résoudre en conséquence. Lorsqu'on choisit la linéarisation, l'état du LFSR peut être étendu pour incorporer les monômes en les bits de l'état initial. On ordonne les monômes de $\mathbb{T}^L$ suivant l'ordre drl. Si la fonction booléenne est de degré d, seuls les monômes de degré inférieur ou égal d seront pertinents. On a donc $\mathbf{s}^d = (\mathbf{s}^{\alpha_0}, \mathbf{s}^{\alpha_1}, \ldots, \mathbf{s}^{\alpha_{\mathcal{N}_L^d - 1}})^T$. On complète également la matrice d'avance du polynôme pour rendre compte de l'évolution des monômes de degré inférieur ou égal à d. Un monôme à l'instant $t+1$ est

- égal à un monôme de même degré de l'état à l'instant t si ce dernier ne fait pas intervenir la variable X_{L-1} ;
- égal à la somme d'au plus L monômes de degré inférieur ou de même degré de l'état précédent si la variable X_{L-1} intervient dans le monôme précédent.

La matrice obtenue, appelée matrice d'avance monomiale, notée $\mathcal{R}_A^d$, est donc creuse.

L'algorithme 4 décrit l'attaque algébrique naïve.

La complexité de l'algorithme 5 correspond à $\mathcal{N}_L^d$ étapes durant lesquelles on extrait $wt(f)$ lignes de tmp et où on réalise une multiplication de tmp par $\mathcal{R}_A^d$. Elle est donc de $\mathcal{O}\left(\mathcal{N}_L^d(wt(f)\mathcal{N}_L^d + (L\mathcal{N}_{L-1}^{d-1}\mathcal{N}_L^d + \mathcal{N}_L^d(\mathcal{N}_L^d - \mathcal{N}_{L-1}^{d-1}))\right) = \mathcal{O}\left(L(\mathcal{N}_L^d)^3\right)$

On résume dans la table 2.1 la complexité de cet algorithme. Pour simplifier les notations, on note $F = \mathcal{N}_L^d$.

Remarque : La matrice $\mathcal{S}$ étant indépendante de la suite chiffrante $\mathbf{z}$, on peut précalculer son inverse pour un coût F^ω et résoudre chaque instance par une simple multiplication matrice vecteur pour un coût F^2.

Algorithme 4 Attaque algébrique

Entrée : un LFSR de taille L d'état $\mathbf{s}^t$ et de fonction d'avance ϕ, une fonction booléenne à f de degré d, $\mathbf{z} = (z(0), z(1), \ldots, z(N-1)), N$ bits de suite chiffrante produite par le LFSR filtré initialisé avec une valeur $\mathbf{s}^0$ inconnue

Sortie : $\mathbf{s}^0$

$\mathcal{S}_p = \mathsf{ConstruireSystèmePrécalcul}(f, \phi, L)$
$\mathcal{S}, \mathbf{b} = \mathsf{ConstruireSystème}(\mathcal{S}_p, \mathbf{z})$
$\mathbf{s}^* = \mathsf{RésoudreSystème}(\mathcal{S}, \mathbf{b})$
retourner $\mathbf{s}^*$

Algorithme 5 ConstruireSystèmePrécalcul (Attaque naïve)

Entrée : $\mathcal{C}_{\mathsf{drl}}(f)$ les coefficients de f ordonnés selon l'ordre drl,
$A(X) = \sum_{i=0}^{L-1} a_i X^i + X^L$ le polynôme de rebouclage du LFSR,
L la taille du LFSR

Sortie : $\mathcal{S}_p$ une matrice carrée de taille $\mathcal{N}_L^d$

$\mathcal{S}_p = 0_{\mathcal{N}_L^d}$
$\mathsf{tmp} = \mathsf{Id}_{\mathcal{N}_L^d}$
pour $i = 0 \to \mathcal{N}_L^d - 1$ **faire**
  $\mathcal{S}_p[i] \leftarrow \mathcal{C}_{\mathsf{drl}}(f)^T \cdot \mathsf{tmp}$
  $\mathsf{tmp} \leftarrow \mathcal{R}_A^d \cdot \mathsf{tmp}$
fin pour
retourner $\mathcal{S}_p$

Annulateurs et améliorations On voit que le degré de la fonction booléenne influence de manière exponentielle la complexité de l'attaque algébrique. [47] constate qu'il est possible pour un LFSR filtré de construire des systèmes faisant intervenir des fonctions booléennes de degré plus faible, ce qui améliore considérablement la complexité des attaques algébriques. Ceci passe par l'étude des annulateurs de la fonction f.
Remarque : Un résultat essentiellement équivalent a été établi indépendamment dans [62].

Définition 21 *Soit $f \in \mathcal{B}_n$ une fonction booléenne à n variables. On dit qu'une fonction booléenne $g \in \mathcal{B}_n$ est un annulateur de f si*

$$\forall x \in \{0,1\}^n, f(x)g(x) = 0.$$

L'ensemble des annulateurs de f est noté $\mathsf{Ann}(f)$. Cet ensemble est un idéal de l'ensemble des fonctions booléennes muni de l'addition et de la multiplication. L'ensemble des annulateurs de degré inférieur ou égal à e est noté $\mathsf{Ann}_{\leq e}(f)$.

Soit g soit un annulateur de f de degré e. On a

$$\begin{array}{lrcl} \forall t & (f \cdot g)(\phi^t(\mathbf{s}^0))) & = & 0, \\ \forall t & z_t g(\phi^t(\mathbf{s}^0)) & = & 0. \end{array}$$

À chaque instant où $z_t = 1$, on obtient une équation $g(\phi^t(\mathbf{s}^0)) = 0$. Si $e < d$, on peut donc réaliser une attaque algébrique de complexité moindre, aussi bien en temps qu'en données. En fait d'autres scénarios sont envisageables et peuvent être résumés par le scénario suivant :

Supposons qu'on dispose de g et h tels que $f \cdot g = h$. On remarque qu'alors

Algorithme 6 ConstruireSystème (Attaque naïve)
Entrée : $\mathcal{S}_p$ la matrice résultat du précalcul, $\mathbf{z}$ la suite des sorties du générateur
Sortie : $\mathcal{S}, \mathbf{b}$ un système linéaire dont la solution fournit l'état du LFSR filtré
retourner $\mathcal{S}, \mathbf{b}$

		T	M	D
Construction	Précalcul	LF^3	F^2	
	Application	F^2	F^2	F
Résolution		F^ω	F^2	

TABLE 2.1 – Résumé de la complexité de l'attaque algébrique naïve

- $h \in \mathsf{Ann}\,(f+1)$ car $(f+1).h = fh + h = f^2.g + h = fg + h = 0$.
- $g + h \in \mathsf{Ann}\,(f)$ car $f(g+h) = fg + fh = h + h = 0$.

Si $g + h$ ou h est de degré inférieur à d, on peut réaliser une attaque algébrique plus rapide que l'attaque naïve. Pour estimer la résistance du LFSR filtré contre les attaques algébriques, il faut donc étudier le problème de l'existence d'annulateurs de f et de $f + 1$ de bas degré.

Définition 22 *On appelle immunité algébrique d'une fonction booléenne f l'entier*

$$\mathcal{IA}(f) = \min\left(e \in \mathbb{N} | \mathsf{Ann}_{\leq e}\,(f) \bigcup \mathsf{Ann}_{\leq e}\,(f+1) \neq \{0\{\right).$$

Proposition 17 ([47][Théorème 6.0.1]) *Soit f une fonction booléenne à n variables. On a*

$$\mathcal{IA}(f) \leq \lceil \frac{n}{2} \rceil.$$

Démonstration : On considère la famille de fonctions booléennes à n variables constituées des monômes de degré inférieur ou égal à $\lfloor \frac{n}{2} \rfloor$ et des multiples de f par les monômes de degrés inférieur ou égal à $\lceil \frac{n}{2} \rceil$. Le nombre d'éléments de cette famille est

$$\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{i} + \sum_{i=0}^{\lceil \frac{n}{2} \rceil} \binom{n}{i} = 2^n + \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Étant donné que la dimension de l'espace vectoriel des fonctions booléennes est 2^n, cette famille est liée et une relation de dépendance fournit un annulateur de f ou de $f + 1$. □

Construction d'annulateurs de bas degré. Une série d'articles [3, 2, 54, 55] étudie le problème de la construction d'annulateurs de degré le plus bas possible d'une fonction booléenne. Ils procèdent de manière itérative en éliminant l'existence d'annulateurs de degré e avant d'étudier l'existence d'annulateurs de degré $e + 1$. Une approche directe consiste à écrire, pour chaque valeur de supp(f) une équation linéaire en les coefficients de l'annulateur représentant son annulation en cette valeur, et à résoudre le système obtenu par élimination gaussienne. [2] décrit un algorithme déterministe qui repose sur l'interpolation multivariée, précédemment développée dans [101, 113]. L'idée est de trouver un polynôme de bas degré s'annulant sur le support de f. Cet article propose des optimisations pour tirer parti de la forme de la matrice d'interpolation et de la caractéristique 2. [54, 55] développent des algorithmes probabilistes. [55] décrit un algorithme de complexité quasi-optimale, permettant de construire les annulateurs d'une classe de

fonctions booléennes. [54] reprend les idées de [2] mais utilise le caractère creux de la matrice d'interpolation pour mettre en oeuvre l'algorithme de Wiedemann [140], une technique d'inversion nécessitant une quantité moindre de mémoire. La procédure, probabiliste, doit être répétée pour un certain nombre de vecteurs choisis aléatoirement pour obtenir une bonne probabilité de succès.

On rappelle dans la table 2.2 les complexités de ces différents algorithmes. On note e l'immunité algébrique calculée, d le degré de la fonction en n variables [2] et de poids w. Pour simplifier les notations, on note $\mathcal{E} = \mathcal{N}_n^e$. On renvoie aux articles pour leur description détaillée.

Algorithme	T	M
Élimination gaussienne	$w\mathcal{E}^2$	$w\mathcal{E}^2$
[2]	$\mathcal{E}2^n$	$\mathcal{E}^2$
[55]	$\mathcal{E}$, en moyenne	$\mathcal{E}$
[54]	$n2^n\mathcal{E}$	$n2^n$

TABLE 2.2 – Complexité $\mathcal{C}_{\text{Ann}}$ du calcul d'annulateurs de degré minimum e d'une fonction booléenne à n variables

On adapte l'attaque algébrique naïve afin de tirer parti de l'existence d'annulateurs. Un précalcul permet de déterminer un annulateur de bas degré. La construction du système dépend à présent de la suite chiffrante observée car seuls les instants correspondant à une sortie égale à 1 (resp. 0) sont retenus lorsqu'on emploie un annulateur de f (resp. $f+1$). On peut cependant continuer à précalculer l'essentiel de la construction du système en construisant un système plus grand, dont on ne retiendra que les équations correspondant aux instants où la valeur de sortie correspond à l'annulateur utilisé. Les complexités des différentes phases de l'attaque sont résumées en Table 2.3. Elles sont exprimées en fonction de $E = \mathcal{N}_L^e$.

		T	M	D
Construction	Précalcul	$\mathcal{C}_{\text{Ann}} + LE^3$	E^2	
	Application	E^2	E^2	E
Résolution		E^ω	E^2	

TABLE 2.3 – Résumé de la complexité de l'attaque algébrique avec prise en compte des annulateurs

2.4.2 Structure linéaire du LFSR filtré.

Les attaques algébriques classiques requièrent une quantité de données de l'ordre de la taille du système à résoudre et construisent une équation du système à résoudre à partir de chaque bit de sortie utile. Ces attaques exploitent les propriétés de la fonction booléenne pour abaisser le degré des systèmes construits, mais n'exploitent pas les propriétés de récurrence de la suite sous-jacente produite par le LFSR. [83, 124, 123, 119] constatent que les propriétés de récurrence linéaire satisfaites par les suites générées par des LFSR se généralisent dans une certaine mesure aux suites produites par extraction à travers une fonction de filtrage booléenne.

2. Lorsqu'on calcule les annulateurs d'une fonction, on élimine pendant le calcul les variables qui n'apparaissent pas dans la forme algébrique normale de la fonction

Considérons en effet la suite produite par le LFSR. Il s'agit d'une suite linéaire récurrente d'ordre L, de polynôme de récurrence A. Si α désigne une racine de A dans $\mathbb{F}_{2^L}$, la suite (s_t) s'écrit sous la forme $Tr(\beta\alpha^t)$, avec $\beta \in \mathbb{F}_{2^L}$ qui encode l'état initial du LFSR. La représentation algébrique permet d'exprimer finement la structure des sorties produites par le LFSR filtré. Commençons par considérer la suite obtenue par filtrage du LFSR par un monôme $\prod_{i=0}^{e-1} X_{u_i}$. On a

$$\begin{aligned} z_t &= \prod_{i=0}^{e-1} s_{u_i}^t = \prod_{i=0}^{e-1} s_{t+u_i} \\ &= \prod_{i=0}^{e-1} Tr(\beta\alpha^{t+u_i}) = \prod_{i=0}^{e-1}\sum_{j_i=0}^{L-1} (\beta\alpha^{t+u_i})^{2^{j_i}} \\ &= \sum_{\mathbf{j}\in[\![0,L-1]\!]^e} \alpha^{\sum_{i=0}^{e-1} u_i 2^{j_i}} (\beta\alpha^t)^{\sum_{i=0}^{e-1} 2^{j_i}}. \end{aligned}$$

On note $k_{\mathbf{j}} = \sum_{i=0}^{e-1} 2^{j_i} \mod 2^L - 1$, $u_{\mathbf{j}} = \sum_{i=0}^{e-1} u_i 2^{j_i} \mod 2^L - 1$ et $\mathbf{j} + l = (j_0 + l, \dots j_{e-1} + l) \mod L$.

On a

$$z_t = \sum_{k=0}^{2^L-2} \Big(\sum_{\mathbf{j}:k_{\mathbf{j}}=k} \alpha^{u_{\mathbf{j}}} \Big)(\beta\alpha^t)^k$$

Par ailleurs on a

$$\begin{aligned} k_{\mathbf{j}+l} &= 2^l k_{\mathbf{j}} \\ u_{\mathbf{j}+l} &= 2^l u_{\mathbf{j}} \end{aligned}$$

On en déduit en regroupant les k conjugués

$$\begin{aligned} z_t &= \sum_{k\in\mathfrak{R}} Tr^{n_k}\Big(\big(\sum_{\mathbf{j}:k_{\mathbf{j}}=k} \alpha^{u_{\mathbf{j}}} \big)(\beta\alpha^t)^k \Big) \\ &= \sum_{k\in\mathfrak{R}_{\leq e}} Tr^{n_k}(C_k^{u_0,\dots,u_{e-1}} (\beta^k (\alpha^k)^t) \end{aligned}$$

La dernière équation restreint la somme aux exposants de poids inférieur ou égal à e. On remarque tout d'abord que ceci est bien défini car le poids de Hamming est constant dans une classe d'équivalence. La restriction est correcte car le poids de Hamming de $k_{\mathbf{j}}$ est inférieur ou égal à e.

Les suites produites par un LFSR filtré par une fonction booléenne quelconque partagent la même structure. Arrêtons-nous sur cette structure. On reconnaît une somme de suites récurrentes linéaires. Les polynômes de rétroaction de ces suites ont pour racine α^k, où le poids de Hamming de k est inférieur ou égal au degré d de la fonction booléenne. Ainsi, un LFSR filtré par une fonction booléenne de degré d peut être vu comme la combinaison linéaire de $t_d = |\mathfrak{R}_{\leq d}|$ LFSR de polynômes de rebouclage P_{α^k}, où α est une racine de A et k un entier $\leq d$. Ces LFSR sont initialisés par une valeur $C_k^f \beta^k$ liée à la fonction booléenne utilisée et à l'état initial du LFSR filtré. Cette correspondance permet de transformer un petit GPA non linéaire en un GPA linéaire équivalent de grande taille. En fait l'intégralité de la non linéarité est déplacée dans la fonction d'initialisation du GPA-linéaire.

Remarque : lorsque α n'est pas un élément primitif, les P_d ne sont plus nécessairement premiers entre eux. Par conséquent le nombre de LFSR intervenant dans la représentation linéaire du LFSR filtré est inférieur à sa valeur maximale.

On peut noter que cette structure peut être traduite en terme de représentation matricielle au travers de la matrice d'avance monomiale.

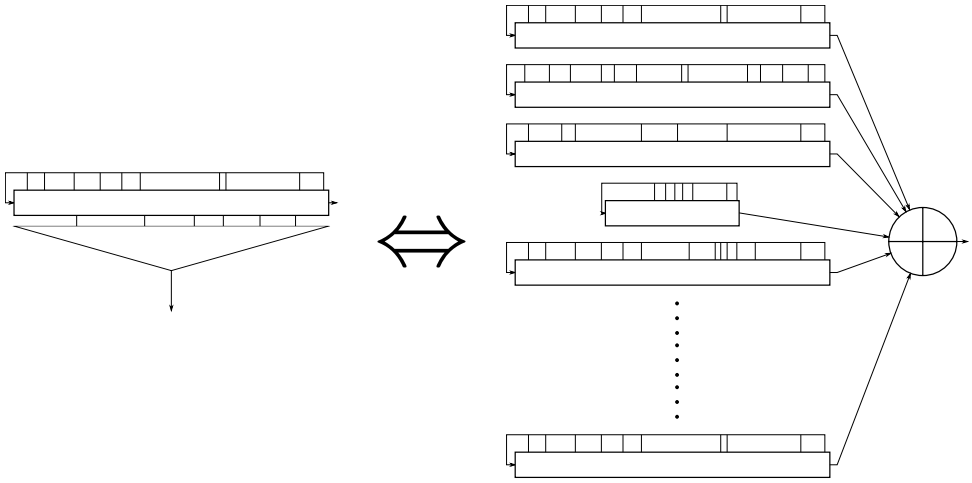


FIGURE 2.3 – Structure linéaire équivalente du LFSR filtré

Théorème 5 *Soit A un polynôme irréductible de $\mathbb{F}_q[X]$, d un entier et $\mathcal{R}_A^d$ la matrice d'avance monomiale de degré d du polynôme A. $\mathcal{R}_A^d$ est une matrice triangulaire inférieure par bloc*

$$\mathcal{R}_A^d = \begin{bmatrix} A_0 & 0 & \cdots & 0 \\ * & A_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & A_d \end{bmatrix},$$

avec $A_i \in \mathcal{M}_{\mathcal{N}_L^i,\mathcal{N}_L^i}(\mathbb{F}_q)$ de polynôme minimal et de polynôme caractéristique $P_i = \prod_{k\in\mathfrak{R}_i} X-\alpha^k$. $P_{\leq d}$ est le polynôme minimal et caractéristique de $\mathcal{R}_A^d$.

De plus, en notant $P_{e\cdot d} = \prod_{i=e}^{d} P_i$, on a $\forall 0 \leq e \leq d$,

$$P_{e\cdot d}(\mathcal{R}_A^d) = \begin{bmatrix} B_0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ * & B_1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & B_{e-1} & 0 & \cdots & 0 \\ * & * & \cdots & * & 0 & \cdots & 0 \end{bmatrix},$$

avec $B_i = P_{e\cdot d}(A_i)$.

Démonstration : La structure triangulaire par blocs et la taille des matrices carrées diagonales découlent de la définition de la matrice d'avance monomiale. De plus $\mathcal{R}_A^0 = [1]$ est une matrice de polynôme minimal et caractéristique égaux à $P_0 = X + 1$.
Supposons que les propriétés sur les matrices carrées de la diagonale soient vraies pour tout entier inférieur ou égal à $d-1$. On va montrer qu'elles sont également vraies pour d.
On commence par remarquer que A étant primitif, α est primitif et les polynômes P_i sont premiers entre eux.
Pour $k \in \mathfrak{R}_e$, on a

$$C_k^{0,\ldots,d-1} = \sum_{\mathbf{j}\in[\![0,L-1]\!]^d:\sum 2^{j_i}=k} \prod_{i=0}^{d-1} \alpha^{i2^{j_i}}.$$

Comme le poids de Hamming de k est d, la condition sur les e-uplets $\mathbf{j}$ retenus dans la somme est équivalente à exiger que $\mathbf{j}$ est une permutation des positions des bits à 1 dans l'écriture binaire

de k. En notant $k_0 < \ldots < k_{d-1}$ ces positions, on peut écrire :

$$C_k^{0,\ldots,d-1} = \sum_{\sigma \in \mathfrak{S}_d} \prod_{i=0}^{d-1} \alpha^{i 2^{k_{\sigma(i)}}}.$$

On reconnaît le déterminant de la matrice $\mathcal{M}_{d,d}(\mathbb{F}_{2^L})$ de terme général $(\alpha^{i 2^{k_j}})_{0 \leq i < e, 0 \leq j < e}$. Cette matrice est une matrice de Vandermonde. Les k_j étant distincts, ce déterminant $C_k^{0,\ldots,d-1}$ est non nul.
Considérons à présent la suite générée par le LFSR filtré par le monôme $X_0 \ldots X_{d-1}$. On a vu qu'elle peut être écrite comme la somme de suites récurrentes linéaires :

$$z_t = \sum_{k \in \mathfrak{R}_{\leq d}} Tr^{n_k}(C_k^{0,\ldots,u_{d-1}}(\beta^k(\alpha^k)^t)).$$

On vient également de voir que les coeffients C_k sont non nuls pour $HW(k) = d$. Les polynômes de récurrence des suites correspondant à $HW(k) < d$ étant premiers avec ceux correspondant à $HW(k) = d$, d'après la proposition 15, P_d divise le polynôme minimal de la suite z_t. Par ailleurs, la suite z_t peut être écrite sous forme matricielle :

$$z_t = f \cdot (\mathcal{R}_A^d)^t \cdot x,$$

où f est un vecteur nul en toutes ses composantes, sauf celle correspondant au monôme $X_0 \ldots X_{d-1}$ et x est le vecteur représentant les monômes de degré inférieur ou égal à d en les bits de l'état initial. Par conséquent, pour tout polynôme P,

$$\Psi(P)(z) = f \cdot P(\mathcal{R}_A^d)(\mathcal{R}_A^d)^t \cdot x.$$

En particulier $\Psi(\mu_{\mathcal{R}_A^d})(z) = 0$, donc le polynôme minimal de z divise $\mathcal{R}_A^d$, par transitivité P_d divise $\mu_{\mathcal{R}_A^d}$ et donc le polynôme caractéristique de $\mathcal{R}_A^d$. Du fait de la structure triangulaire inférieure de $\mathcal{R}_A^d$, son polynôme caractéristique est le produit du polynôme caractéristique de $\mathcal{R}_A^{d-1}$ et du polynôme caractéristique de A_d. P_d étant premier avec $P_{\leq d-1}$, P_d divise le polynôme caractéristique de A_d. De plus son degré est égal à la taille de la matrice A_d. Il est donc égal au polynôme caractéristique de A_d. Comme il est produit de polynômes irréductibles distincts, c'est également le polynôme minimal de A_d. Le polynôme caractéristique de $\mathcal{R}_A^d$ est égal à $P_{\leq d}$. Comme ce polynôme est produit de polynômes irréductibles distincts, c'est également le polynôme minimal de $\mathcal{R}_A^d$. Ceci conclut la preuve par récurrence de la première partie du théorème.
On montre à présent que $P_{e \cdot d}(\mathcal{R}_A^d)$ a la forme annoncée dans le théorème. On sépare les lignes et colonnes de $P_{e \cdot d}(\mathcal{R}_A^d)$ en deux blocs de longueurs respectives N_L^e et $N_L^d - N_L^e$. La matrice $\mathcal{R}_A^d$ s'écrit alors

$$\mathcal{R}_A^d = \begin{bmatrix} M_1 & 0 \\ * & M_2 \end{bmatrix}.$$

M_1 est une matrice triangulaire inférieure par bloc dont les blocs diagonaux sont les $A_i, 0 \leq i < e$. Par ailleurs le polynôme caractéristique de M_2 est $P_{e \cdot d}$, d'où $P_{e \cdot d}(M_2) = 0$. On en déduit la forme annoncée dans le théorème.

□

2.4.3 Attaques algébriques rapides.

On explique ici comment la structure linéaire des LFSR filtrés peut être utilisée pour accélérer les attaques algébriques, au prix d'une augmentation de la complexité en temps d'un précalcul et de la complexité en données. L'idée est de constater que la sortie d'un LFSR filtré de polynôme de rebouclage A, supposé primitif, avec une fonction de degré d suit une récurrence linéaire de polynôme de récurrence $P_{\leq d}$. En appliquant cette récurrence linéaire sur une telle suite, on l'annule. Le théorème 5 permet d'affiner encore ce résultat en établissant des polynômes de récurrence permettant d'annuler l'influence de monômes de degré élevé.

Revenons aux attaques algébriques. On a vu qu'une équation de la forme $f.g = h$ pouvait être exploitée pour réaliser une attaque algébrique si h ou $g+h$ est de degré faible. En exploitant les relations de récurrence présentées ci-dessus, une amélioration de l'attaque devient également possible sous la condition plus faible que g est de degré faible [45]. En effet

$$\forall t, z_t g(\mathbf{s}^t) = h(\mathbf{s}^t).$$

La fonction h est potentiellement de degré élevé. On suppose [3] $\deg(g) < \deg(h) < \deg(f)$. Afin de pouvoir exploiter ces équations, il faut en abaisser le degré.

Les monômes apparaissant à l'instant t dans l'expression de $h(\mathbf{s}^t)$ sont les monômes donnés par

$$\mathcal{C}_h \cdot (R_A^{\deg(h)})^t \cdot \mathbb{T}_{\deg(h)}.$$

En appliquant la récurrence linéaire donnée par un polynôme $P = \sum p_i X^i$ à cette suite d'expression formelle on obtient la suite

$$\mathcal{C}_h \cdot P(R_A^{\deg(h)})(R_A^{\deg(h)})^t \cdot \mathbb{T}_{\deg(h)}.$$

Le théorème 5 énonce que les colonnes de droite de $P_{e\cdot\cdot\deg(h)}(\mathcal{R}_A^{\deg(h)})$ correspondant aux monômes de degré $> e$ sont nulles. Ces monômes n'apparaissent donc pas dans l'expression formelle de $h(\mathbf{s}^t)$ et on peut les ignorer dans la construction de cette expression formelle. Cette construction consiste donc à calculer

$$\mathcal{C}_h \cdot P_{e\cdot\cdot\deg(h)}(R_A^{\deg(h)})^*(R_A^e)^t \cdot \mathbb{T}_e,$$

où l'astérisque dénote la suppression des colonnes nulles de la matrice. Lorsque $\deg(g) > 0$, l'application de la même récurrence linéaire sur l'expression de $z_t g(\mathbf{s}^t)$ n'élimine pas a priori les monômes de degré $\deg(g)$. On peut donc choisir $0 \leq e \leq \deg(g)$ pour obtenir un système d'équations de degré $\deg(g)$. Lorsque $\deg(g) = 0$, on choisit $e > 0$ pour former un système de degré e, ce qui correspond à l'attaque de Rønjom et Helleseth [124].

On note $E = \mathcal{N}_L^e, G = \mathcal{N}_L^{\deg(g)}$ et $H = \mathcal{N}_L^{\deg(h)}$.

Pour mener l'attaque il est nécessaire de collecter $\max(E, G)$ équations de ce type. Le coût de construction de la partie des équations correspondant à h décimée par le polynôme de récurrence $P_{e\cdot\cdot\deg(h)}$ est donc égal à la somme des coûts suivants :

- le coût de construction du polynôme $P_{e\cdot\cdot\deg(h)}$, soit $\mathcal{O}\left(H(\log H)^2\right)$, cf [72] ;
- le coût de construction de $P_{e\cdot\cdot\deg(h)}(R_A^{\deg(h)})^*$, soit $\mathcal{O}\left(LEH^2\right)$
- le coût de construction de $\max(E, G)$ expressions de $h(\mathbf{s}^t)$, soit $\mathcal{O}\left(LHE\max(E, G)\right)$

3. le cas $g = 1, h = f$ montre en effet qu'il n'est pas nécessaire de considérer h de degré supérieur au degré de f.

Il est donc de l'ordre de $\mathcal{O}\left(LEH^2\right)$. Il faut noter un cas particulier pour cette complexité en fonction du degré e choisi, le cas $e = 0$. Dans ce cas la décimation annule complètement l'expression et la complexité de la construction de l'expression est nulle.

L'expression de $z_t g(\mathbf{s}^t)$ est décimée par le même polynôme. Cependant, on ne peut plus précalculer cette expression car elle dépend des valeurs de suite chiffrante observées. Seule l'expression formelle des $g(\mathbf{s}^t), 0 \leq t < H$ peut-être précalculée, pour un coût en $\mathcal{O}\left(LG^2H\right)$. L'instanciation du système, *i.e.* la prise en compte des valeurs de suite chiffrante pour construire l'expression de $z_t g(\mathbf{s}^t)$ s'obtient alors par calcul rapide d'une convolution pour un coût de $\mathcal{O}\left(GH \log H\right)$, cf [72]. La quantité de données nécessaires pour obtenir $\max(E, G)$ équations après décimation par un polynôme de degré $H - E$ est $H - E + \max(E, G)$.

Finalement, la résolution du système est obtenue en $\mathcal{O}\left((\max(E, G))^\omega\right)$.

Afin de pouvoir mener une attaque algébrique rapide il faut commencer par trouver une relation de type $fg = h$, avec g de bas degré. Les algorithmes de calcul d'annulateurs peuvent être adaptés pour réaliser cette tâche. La table 2.4 donne la complexité du calcul de relations de type $fg = h$. Cette complexité est exprimée en fonction de $\mathcal{G} = \mathcal{N}_n^{\deg(g)}, \mathcal{H} = \mathcal{N}_n^{\deg(h)}$, où n est le nombre de variables de f.

Algorithme	T	M
Élimination gaussienne	$w\mathcal{G}\mathcal{H}$	$w\mathcal{G}\mathcal{H}$
[2]	$\mathcal{G}2^n$	$\mathcal{G}^2$
[55]	complexité non prouvée, conjecture $\mathcal{G}$, en moyenne	-
[54]	$n2^n\mathcal{G}$	$n2^n$

TABLE 2.4 – Complexité $\mathcal{C}_{\text{Ann}}$ du calcul d'annulateurs de degré minimum d'une fonction booléenne à n variables

La table 2.5 résume les complexités des différentes phases des attaques algébriques rapides. Ces complexités sont exprimées en fonction de $G = N_L^{\deg(g)}, H = \mathcal{N}_L^{\deg(h)}$.

		T	M	D
Construction	Précalcul	$\mathcal{C}_{\text{Ann}} + LEH^2 + LG^2H$	GH	
	Application	$GH \log H$	GH	$H - E + \max(E, G)$
Résolution		$\max(E, G)^\omega$	$\max(E, G)^2$	

TABLE 2.5 – Résumé de la complexité de l'attaque algébrique rapide

Chapitre 3

VEST

Sommaire

3.1 Introduction

VEST [114, 115] est un ensemble de quatre familles d'algorithmes de chiffrement par flot soumis à l'appel à candidature du projet eSTREAM [60] par S. O'Neil, B. Gittins et H. Landman. VEST-v, $v \in \{4, 8, 16, 32\}$, est une famille d'algorithmes de chiffrement par flot visant un niveau de sécurité de respectivement 80, 128, 160 et 256 bits, et produisant une sortie de v bits par cycle d'horloge. Les quatre familles partagent un même principe de conception. Seules les tailles des différents composants changent pour atteindre le niveau de sécurité visé. Un algorithme de sélection permet, étant donné un paramètre appelé clé de famille, d'obtenir un algorithme membre de la famille VEST.

Lors du passage en phase 2 du projet eSTREAM, les spécifications de VEST ont été mises à jour. Les modifications réalisées lors de cette mise à jour portent sur certains paramètres de l'algorithme et sur la définition d'un mode additionnel permettant d'obtenir un algorithme de MAC ou de chiffrement authentifié à partir de la famille VEST.

Dans cette section, nous nous intéressons à des faiblesses basiques de certains composants de VEST. Ces faiblesses peuvent être mises à profit pour engendrer des collisions internes dans

l'état de l'algorithme, semblables aux collisions internes dans les fonctions de hachage de la famille SHA [37]. Par conséquent, nous sommes en mesure de réaliser une attaque à IV choisis contre les algorithmes de chiffrement par flot de la famille VEST permettant de recouvrer 53 bits de l'état interne après mise à la clé. Cette information peut être utilisée pour diminuer la complexité d'une attaque par recherche exhaustive. Cette attaque à IV choisis sur l'algorithme de chiffrement par flot VEST peut également être utilisée comme attaque en forge existentielle contre VEST MAC.

Nous commençons par donner une description des composants de VEST et de leurs mode opératoire. Nous exhibons ensuite la faiblesse élémentaire mise à jour dans les composants *compteur* et *diffuseur linéaire du compteur*. Nous décrivons alors deux attaques à IV choisis permettant de recouvrer de manière efficace 53 bits de l'état interne de l'algorithme mis à la clé. Enfin, nous discutons de l'impact de ces attaques sur la complexité de la recherche exhaustive et donc sur la sécurité de l'algorithme et décrivons brièvement comment elles peuvent se traduire en des attaques de type forge existentielle sur VEST MAC.

3.2 Description de VEST

Les membres de la famille d'algorithmes de chiffrement par flot VEST sont constitués de quatre composants :

- un ensemble de 16 registres à décalage à rétroaction non linéaire, appelé *compteur* ;
- un *diffuseur linéaire du compteur* ;
- un *accumulateur* ;
- un *filtre de sortie* sans mémoire.

Les algorithmes de la famille VEST possèdent trois modes de fonctionnement interne :

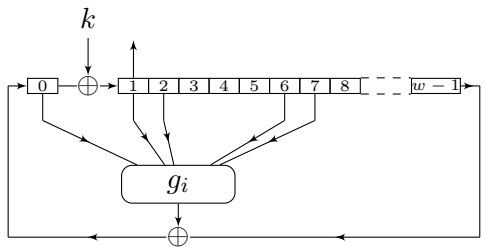
- le mode de *mise à la clé* permet d'introduire une clé dans l'état de l'algorithme. Tous les bits de l'état sont d'abord mis à 1, puis la clé est insérée, ce qui aboutit à un état dit *mis à la clé*. Dans ce mode, l'algorithme ne produit pas de bits de sortie ;
- le mode de *mise à l'IV* permet d'introduire un IV dans un état mis à la clé. Ce mode ne génère pas de bits de sortie ;
- le mode de génération de suite chiffrante produit une suite de taille arbitraire de bits pseudo-aléatoires.

Nous commençons par décrire de manière un peu plus détaillée les quatre composants de l'algorithme VEST, notamment leurs différentes fonctions de mise à jour. Nous nous appuyons ensuite sur ces premières descriptions pour expliciter les modes de mise à la clé et de mise à l'IV de l'algorithme VEST, en donnant notamment pour chaque composant la fonction de mise à jour mise en œuvre.

3.2.1 Compteur

Le compteur est un ensemble de 16 registres $c_i, (0 \leq i < 16)$. Il constitue la partie autonome de l'algorithme de chiffrement par flot dans le mode de génération de suite chiffrante. Chaque registre est un registre à décalage à rétroaction non linéaire, en anglais non linear feedback shift register, abrégé en NLFSR, de taille $w = 10$ ou 11 bits. Nous adoptons les notations de [115] et désignons par c_{ij}^r la valeur du bit j du registre i à l'instant r. La fonction de mise à jour des registres est donnée en Figure 3.1.

Chaque registre est mis à jour de manière indépendante. Les registres ont deux mode d'opération. En *mode compteur*, les registres sont mis à jour de manière autonome. En *mode initialisa-*



- **Mode Initialisation**
  - $c_{i\ 0}^{r+1} = g_i(c_{i\ 0}^r, c_{i\ 1}^r, c_{i\ 2}^r, c_{i\ 6}^r, c_{i\ 7}^r) \oplus c_{i\ w_i-1}^r$
  - $c_{i\ 1}^{r+1} = c_{i\ 0}^r \oplus k_i^r$
  - $c_{i\ j}^{r+1} = c_{i\ j-1}^r$
- **Mode Compteur**
  - $c_{i\ 0}^{r+1} = g_i(c_{i\ 0}^r, c_{i\ 1}^r, c_{i\ 2}^r, c_{i\ 6}^r, c_{i\ 7}^r) \oplus c_{i\ w_i-1}^r$
  - $c_{i\ j}^{r+1} = c_{i\ j-1}^r$

FIGURE 3.1 – Mise à jour d'un registre du compteur

tion, la fonction de mise à jour d'un registre est perturbée par un bit à chaque coup d'horloge. Les fonctions g_i sont non linéaires et choisies de telle sorte que les graphes des fonctions de mise à jour des registres en mode compteur soient composés de deux cycles de longueur de même ordre de grandeur, les longueurs de ces cycles étant premières deux à deux. Ainsi, un minimum sur la période de la fonction de mise à jour de l'ensemble des registres fonctionnant en mode compteur peut-être obtenue. 32 fonctions non linéaires satisfaisant les propriétés énoncées ci-dessus sont spécifiées dans [115], 16 pour des registres de longueur 11 et 16 pour des registres de longueur 10. La sélection d'un membre de la famille VEST attribue à chaque registre une fonction de rebouclage de longueur adéquate. À chaque coup d'horloge le bit en position 1 est extrait de chaque registre.

On remarque que la fonction de mise à jour d'un registre est inversible en mode compteur. Elle est également inversible en mode initialisation si le bit de perturbation est connu.

3.2.2 Diffuseur linéaire du compteur

Le diffuseur linéaire du compteur est une valeur de 10 bits utilisée pour perturber l'avance de l'accumulateur. À chaque cycle, le diffuseur linéaire du compteur est mis à jour de manière linéaire à l'aide des 16 bits de sortie du compteur. Chaque bit mis à jour est somme d'un bit du diffuseur linéaire du compteur et de cinq bits prevenant du compteur. En reprenant les notations de [115], on note d_j^r la valeur à l'instant r du bit j du diffuseur linéaire du compteur. On note

$$\begin{aligned} D^{(r)} &= (d_0^r, d_1^r, \ldots, d_9^r)^T \\ C^{(r)} &= (c_{0\ 1}^r, c_{1\ 1}^r, \ldots, c_{15\ 1}^r)^T \end{aligned}$$

La mise à jour du diffuseur linéaire peut alors être écrite sous forme matricielle

$$D^{(r+1)} = A \cdot D^{(r)} \oplus M \cdot C^{(r)} \oplus B,$$

avec

$$A = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1 \end{bmatrix} \quad M = \begin{bmatrix} 0&1&0&0&1&1&0&0&0&0&0&1&0&1&0&0 \\ 1&0&1&0&0&0&1&0&1&0&0&0&0&0&1&0 \\ 0&0&0&1&1&0&0&1&0&0&1&0&0&0&0&1 \\ 1&0&0&1&0&1&0&0&0&1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&1&0&0&0&0&0&1&0&0&1 \\ 1&0&0&0&0&0&0&0&1&0&1&0&0&0&1&1&0 \\ 0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&1 \\ 0&0&1&0&0&1&1&0&0&0&1&0&1&0&0&0 \\ 1&0&0&1&0&0&0&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&1&0&1&1&0&1 \end{bmatrix} \quad B = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

3.2.3 Accumulateur

Le reste de l'état d'un algorithme de chiffrement par flot VEST est constitué d'un accumulateur. À chaque cycle, l'état de l'accumulateur subit une phase de substitution et une phase

de permutation. Puis le diffuseur linéaire du compteur est ajouté par ou exclusif aux dix premiers bits de l'état de l'accumulateur. Pour une description complète de l'accumulateur, nous renvoyons à [115].

3.2.4 Filtre de sortie

À chaque cycle d'horloge, un membre de la famille VEST-v produit v bits de suite chiffrante. Chaque bit est calculé en prenant le ou exclusif de 6 bits de l'état de l'accumulateur. Le nombre de bits de sortie est au moins 18 fois plus petit que la taille de l'accumulateur. Le principe de conception de l'accumulateur et du filtre de sortie, véritable cœur de l'algorithme, suit la même stratégie que celle adoptée par l'algorithme de chiffrement par flot LEX, autre candidat soumis au projet eSTREAM [23, 24]. L'idée est d'extraire une petite partie d'un état mis à jour par un tour d'un algorithme de chiffrement par bloc, la clé de tour étant fournie par un composant évoluant de manière autonome.

3.2.5 Mode de mise à la clé

La *mise à la clé* prend en entrée une clé K de F bits, où F est un multiple de 8 et l'introduit dans l'état du générateur pseudo-aléatoire. Au début de la mise à la clé, tous les bits de l'état sont initialisés à 1. À la fin de la mise à la clé, la valeur de l'état interne est appelée état mis à la clé. Pendant la mise à la clé, aucun bit de sortie n'est produit par l'algorithme. Le diffuseur linéaire de compteur et l'accumulateur évoluent comme décrit précédemment. Les registres du compteur passent par deux phases :

- Pendant la première phase, K est introduite dans les registres. On commence par concaténer 15 bits à 0 devant la clé et 1 bit à 1 suivi de 15 bits à 0 après la clé, obtenant ainsi une clé K'. Pendant $F + 16$ étapes, les registres fonctionnent en mode initialisation. À l'étape r, les bits $K'_r, \ldots, K'_{r+15}$ perturbent l'évolution des registres $0, \ldots, 15$ respectivement. Chaque bit de clé influence donc chaque registre, le bit de clé $0 \leq i < F$ étant introduit dans le registre j à l'étape $15 + i - j$.
- Pendant la deuxième phase, une constante de 8 bits est introduite dans les 8 premiers registres, puis l'état est mélangé. Pendant l'intoduction de la constante, les 8 premiers registres sont en mode initialisation, le registre i étant perturbé par le i-ème bit de la constante, et les 8 derniers registres sont en mode compteur. Puis l'état de l'algorithme passe par 31 étapes où l'évolution des registres se fait en mode compteur.

L'état mis à la clé peut être stocké et rechargé plus tard dans l'algorithme, pour accélérer l'initialisation de l'algorithme quand une même clé est employée avec différents IV. L'état mis à la clé décrit complètement l'influence de la clé sur la suite du fonctionnement de l'algorithme.

3.2.6 Mode de mise à l'IV

La mise à l'IV peut être réalisée après la mise à la clé. Elle prend pour entrée un IV de longueur W bits, où W est un multiple de 8. Comme pour la mise à la clé, l'algorithme ne génère aucune sortie pendant la mise à l'IV, et le diffuseur linéaire et l'accumulateur évoluent comme décrit plus haut. Les registres du compteur passent par deux phases :

- Pendant la première phase de la mise à l'IV, l'IV est introduit dans le compteur. Cette phase s'étend sur $W/8$ étapes. On remarque que contrairement aux bits de clé, les bits d'IV ne sont pas introduits dans tous les registres : ils perturbent uniquement les 8 premiers

registres et chaque bit d'IV n'est introduit que dans un registre. À chaque cycle, un octet d'IV est utilisé.
- La deuxième phase de la mise à l'IV est identique à la deuxième phase de la mise à la clé, mais utilise une constante différente.

3.3 Propriétés des composants de VEST

Dans cette section, nous identifions deux propriétés des composants de VEST. Elles concernent le comportement différentiel des NLFSR en mode initialisation et du diffuseur de compteur.

3.3.1 Caractéristiques différentielles des registres

Pendant la première phase de la mise à l'IV, la fonction d'évolution d'un registre $i, 0 \leq i < 8$ peut être vue comme une fonction

$$f_i : \{0,1\}^{w_i} \times \{0,1\}^n \to \{0,1\}^{w_i} \times \{0,1\}^n,$$

qui modifie un état de longueur w_i à partir d'un IV d'entrée de longueur n. La sortie de la fonction est un état mis à jour et une sortie de longueur n. En étudiant les propriétés différentielles de cette fonction relativement à son deuxième argument, nous mettons en évidence un comportement non idéal.

Les masques différentiels pertinents dans le contexte de la mise à l'IV sont de la forme

$$0 \times \Delta \to 0 \times \alpha.$$

En partant d'une même valeur d'état (par exemple celle de l'état mis à la clé), on introduit une paire d'IV de différence Δ et on recherche une collision sur l'état du registre après la première phase de la mise à l'IV et une différence fixe sur les sorties α. Pour une paire d'IV de différence Δ, on dit qu'un état *collisionne* si en partant de cet état et en introduisant les deux IV de la paire on obtient une collision sur l'état de sortie et la différence attendue α sur les bits de sortie. On rappelle en Figure 3.2 le lien entre motif différentiel et états en collision.

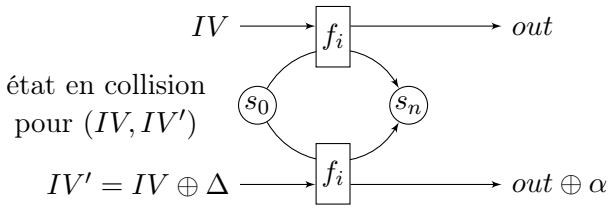


FIGURE 3.2 – Motif différentiel et états en collision

Comportement différentiel sur un cycle. Afin de comprendre la source du comportement non idéal, considérons l'effet d'un bit de différence dans l'état d'un registre pendant la mise à l'IV. Nous distinguons trois cas :
- la différence est en position $w - 1$: la fonction de mise à jour déplace cette différence en position 0,
- la différence est en position $j \notin \{0, 1, 2, 6, 7\}$: la mise à jour décale la différence en position $j + 1$,

- la différence est en position $j \in \{0, 1, 2, 6, 7\}$: la mise à jour décale la différence en position $j + 1$ et est susceptible de créer une différence en position 0, en fonction de la fonction non linéaire de rebouclage utilisée et des valeurs des bits utilisés pour calculer le bit de rebouclage.

Nous devons encore prendre en compte le bit d'IV introduit à cette étape. Après le calcul du bit de rebouclage et le décalage du registre, mais avant de produire le bit de sortie à cette étape, un bit d'IV est ajouté par ou-exclusif en position 1. Dans notre contexte d'analyse différentielle, ce bit peut être utilisé aussi bien pour introduire une nouvelle différence en position 1 que pour corriger une différence présente en position 0 à la fin de l'étape précédente.

À la fin de chaque étape, le bit de sortie est le bit en position 1 de l'état du registre.

Collisions locales. Nous adoptons une stratégie semblable à celle des collisions locales utilisées dans [37] pour attaquer SHA-0. L'idée clé est d'introduire un unique bit de différence dans l'état du registre et de contrôler sa propagation jusqu'à sa disparition. Nous commençons par considérer la partie linéaire de la fonction d'évolution avant d'intégrer la partie non linéaire.

À l'étape 0, on introduit une différence à travers des bits d'IV différents. Ainsi, après l'étape 0, il y a une différence en position 1 du registre. La partie linéaire de la fonction d'évolution du registre consiste simplement à faire subir une rotation aux bits du registre. Après l'étape 1, la différence est en position 2, après l'étape 2 en position 3 ... et après l'étape $w - 1$ la différence est en position 0. À l'étape w, elle revient en position 1, où elle peut être supprimée en utilisant une différence dans la paire d'IV. Ainsi, lorsque le rebouclage non linéaire n'intervient pas, $w+1$ étapes sont nécessaires pour introduire et supprimer une différence. La sortie du registre étant extraite de la position 1, on constate que la différence de sortie α est constituée d'un bit à 1 suivi de w bits à 0. Dans la suite, nous montrons comment construire des collisions suivant un motif similaire sur $w + 1$ étapes avec différence α sur la sortie.

On tient maintenant compte de la fonction non linéaire (NLF). La collision décrite ci-dessus ne se produit plus systématiquement. En effet, lorsque la fonction non linéaire est active, i.e. quand il y a un bit de différence sur au moins une de ses entrées, sa sortie peut porter une différence. Heuristiquement, cela se produit avec probabilité $\frac{1}{2}$. Ainsi, après une étape où la NLF est active, il peut y avoir une différence additionnelle en position 0. Afin de prévenir la propagation d'une différence, on utilise l'insertion de l'IV en position 1 lors de l'étape suivante. En combinant tous les bits de différence sur l'IV utilisés au cours des $w + 1$ étapes, on obtient un motif de correction. Remarquons qu'avec des masques de différence sur $w + 1$ étapes, il n'est pas possible de corriger une différence produite à l'étape w.

En raison de la propagation de la différence initiale à travers toutes les positions du registre, le nombre d'étapes actives est d'au moins 5, les étapes 1, 2, 6, 7 et w. Heuristiquement, après chaque étape active, la probabilité d'apparition d'une différence additionnelle en position 0 est $\frac{1}{2}$. Lorsque c'est le cas l'étape suivante est également active. On peut donc s'attendre à ce que des motifs de correction conduisent à des collisions avec probabilité 2^{-5}. Ainsi, pour certaines paires d'IV ayant une différence Δ correspondant à un motif de correction, on peut s'attendre à 2^{w-5} états qui collisionnent.

De manière pratique $w = 10$ ou $w = 11$. On peut donc calculer toutes les caractéristiques différentielles de longueur $w + 1$ pour un compteur. En fait, pour un IV donné et un état de départ, il y a au plus une différence d'IV qui crée une collision après $w + 1$ étapes et présente la différence de sortie correcte α. Nous avons calculé ces valeurs et stocké pour chaque paire d'IV, les états initiaux pour lesquels une collision se produit. Nous avons réalisé cette recherche exhaustive pour chacune des fonctions non linéaires. Ce calcul prend quelques minutes sur un

Intel Celeron 1.4 Ghz.

Les résultats de ce calcul confirment notre analyse heuristique : pour quelques bonnes paires d'IV ayant une différence appropriée, le nombre d'états qui collisionnent est de l'ordre de 2^{w-5}, parfois supérieur à cette valeur d'un facteur plus grand que 2. Nous donnons en Table 3.1, pour chaque fonction non linéaire $0 \leq i < 32$ proposée dans [115, Appendice F], le nombre maximum d'états en collision N_i pour la meilleure paire d'IV. Pour les 16 premières (resp. dernières) fonctions, $w = 11$ (resp. 10) et le nombre d'états en collision attendu est de 64 (resp. 32).

TABLE 3.1 – Taille N_i du plus grand ensemble d'états en collision pour la fonction i de mise à jour de registre

i	N_i	i	N_i	i	N_i	i	N_i	i	N_i	i	N_i	i	N_i	i	N_i
0	127	4	106	8	122	12	102	16	70	20	44	24	59	28	52
1	107	5	107	9	95	13	96	17	67	21	60	25	76	29	64
2	117	6	96	10	90	14	104	18	74	22	62	26	65	30	54
3	128	7	150	11	156	15	136	19	52	23	77	27	54	31	77

3.3.2 Collision dans le diffuseur de compteur

En Section 3.2.2, une description matricielle du diffuseur linéaire de compteur est donnée. La matrice M ayant plus de colonnes que de lignes, elle a un noyau non trivial, engendré par les vecteurs :

$$(\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0})^{\mathbf{T}},$$
$$(1,1,1,1,0,1,1,0,1,1,1,0,0,0,0,0)^T,$$
$$(0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0)^T,$$
$$(0,1,0,1,1,0,1,0,1,0,0,0,1,0,0,0)^T,$$
$$(1,1,0,1,1,0,0,0,0,0,0,0,0,0,1,0)^T,$$
$$(0,1,0,1,0,0,0,0,0,1,0,0,0,1,0,1)^T$$

Ceci contredit la propriété d'absence de collision annoncée dans [115, Section 5.4]. Si les sorties du compteur à l'étape r diffèrent d'une combinaison linéaire de ces vecteurs, la contribution du compteur à la mise à jour des bits de diffusion $\{d_j^r\}_j$ sera la même. Le vecteur du noyau mis en valeur ne fait intervenir que des bits provenant des 8 premiers registres qui peuvent être, dans une certaine mesure, contrôlés par l'attaquant pendant la mise à l'IV. Ce vecteur est particulièrement utile à notre attaque.

3.4 Reconstitution partielle de l'état mis à la clé

Dans cette section, nous exploitons les faiblesses décrites dans la section précédente pour recouvrer la valeur d'une partie donnée de l'état mis à la clé. Toutes les attaques décrites ici sont des attaques différentielles à IV choisis. La première attaque ne considère pas de contrainte sur la longueur de l'IV. La deuxième attaque s'efforce d'utiliser un IV aussi court que possible et peut être mise en œuvre quand la longueur de l'IV est contrainte à une valeur standard, plus précisément 128 bits. En fait, cette attaque est réalisable dès lors qu'on dispose d'IV de longueur supérieure à 96 bits.

3.4.1 Attaque avec des IV longs

Nous décrivons dans cette sous-section une première attaque qui suppose que la longueur des IV est plus grande que $23 \times 8 = 184$ bits. Nous notons que [115] ne précise pas de limite à la longueur des IV.

Le but de cette attaque est d'exploiter les faiblesses décrites dans la section précédente pour construire une paire d'IV en collision, c'est à dire une paire d'IV tels que l'état de l'algorithme est le même après la mise à la clé utilisant une clé inconnue K et la mise à l'IV utilisant chaque IV de la paire. Nous avons étudié dans la section précédente le comportement différentiel des NLFSR en mode initialisation et du diffuseur de compteur. Notre attaque utilise les propriétés identifiées pour introduire une différence dans le compteur et pour l'éliminer, tout en évitant l'introduction d'une différence dans l'accumulateur.

En effet, pendant la première phase de la mise à l'IV, tous les NLFSR ne sont pas mis à jour de la même manière. Les NLFSR 0 à 7 fonctionnent en mode initialisation et sont perturbés par les bits d'IV, chaque bit d'IV étant utilisé exactement une fois, tandis que les NLFSR 8 à 15 évoluent en mode compteur. Par conséquent, un attaquant peut uniquement influencer les 8 premiers NLFSR. Afin de ne pas introduire de différence dans l'accumulateur, aucune différence ne doit être introduite dans le diffuseur de compteur bien que des différences soient présentes dans le compteur. La mise à jour du diffuseur de compteur dépendant linéairement de la sortie des NLFSR au travers de la matrice M, ceci est possible à condition que la sortie du compteur soit dans le noyau de M. De plus, comme l'attaquant ne peut introduire de différence que dans les 8 premiers NLFSR, la différence des sorties du compteur ne peut être non nulle que pour ces registres. Un vecteur du noyau de M mis en valeur en section 3.3.2 satisfait cette contrainte supplémentaire. Ainsi, si les sorties du compteur diffèrent exactement pour les NLFSR 0, 4, 5, 6, et 7, le diffuseur de compteur est mis à jour comme s'il n'y avait pas de différence. Par conséquent, s'il n'y avait pas de différence dans le diffuseur de compteur et dans l'accumulateur, et si la différence en sortie du compteur est telle que décrite ci-dessus, aucune différence n'est introduite dans le diffuseur de compteur et dans l'accumulateur.

De plus, nous avons décrit en section 3.3.1 les motifs différentiels des NLFRs. La différence de sortie de ces motifs est constituée d'un 1 suivi d'un nombre w de 0. En combinant de telles différentielles, une pour chacun des cinq registres 0, 4, 5, 6, 7, en les faisant commencer à la même étape, et en n'introduisant aucune différence dans ces registres après la fin du motif différentiel, et aucune différence dans les autres registres, on obtient une caractéristique différentielle sur l'intégralité du compteur telle que :

- il y ait une collision sur l'intégralité du compteur si l'état initial est dans un ensemble approprié ;
- la sortie du compteur n'introduise pas de différence dans le diffuseur de compteur et par conséquent dans l'accumulateur. En effet la différence en sortie du compteur est soit nulle soit égale à l'élément du noyau de M mis en valeur en section 3.3.2.

Afin d'expliciter l'ensemble des bonnes valeurs, nous commençons par rappeler que pendant la première phase de la mise à l'IV, les NLFSR sont mis à jour de manière indépendante. De plus, on peut partitionner les bits de l'IV en fonction des registres qu'ils affectent. Ainsi, obtenir une collision sur l'intégralité du compteur est équivalent à obtenir une collision sur chacun des registres 0, 4, 5, 6 et 7. L'ensemble des états initiaux du compteur qui conduit à une collision est le produit cartésien $\mathcal{S}$ des ensembles $\mathcal{S}^r$ des états initiaux des registres qui conduisent à des collisions partielles pour les paires d'IV utilisées. Notons qu'en raison de la différence de sortie des collisions partielles considérées, ceci conduit à une collision sur l'intégralité de l'état de l'algorithme de chiffrement par flot à la fin de la première phase de la mise à l'IV. Comme

aucune différence n'est introduite dans la deuxième phase de la mise à l'IV, on obtient une collision sur tout l'état à la fin de la mise à l'IV.

L'idée de l'attaque de base est de choisir pour chaque registre la paire d'IV qui maximise le nombre d'états en collision. Ceci maximise également le nombre d'états initiaux conduisant à une collision sur l'intégralité du compteur. Si l'état initial est tiré aléatoirement, la probabilité qu'il conduise à une collision est le nombre d'états en collision divisé par le nombre total d'états. Pour le membre par défaut de la famille VEST, la valeur de cette probabilité est :

$$p \approx 2^{-21.24}.$$

On construit alors des paires d'IV de longueur 23 octets, telles que les 11 premiers octets soient identiques et tirés aléatoirement, et les 12 derniers octets soient une paire d'IV fixe, la composition des meilleures paires d'IV pour les registres 0, 4, 5, 6 et 7, et des valeurs fixes pour les registres 1, 2 et 3. On remarque qu'en moyenne, 11 étapes sont suffisantes pour randomiser complètement l'état du compteur. Par conséquent, p est la probabilité qu'une paire d'IV construite de la sorte produise une collision après la mise à l'IV, ce qui conduit à une collision sur l'ensemble de l'état de l'algorithme comme expliqué plus haut. Pour tester cette collision, on compare les 32 premiers bits du flux de clés après chaque mise à l'IV. Si une collision se produit dans la mise à l'IV, les bits pseudo-aléatoires sont identiques. Dans le cas contraire, la probabilité que deux IV conduisent à la même suite de 32 bits de pseudo-aléa est heuristiquement de 2^{-32}. Après de l'ordre de $1/p$ paires de mise à l'IV, on trouve avec forte probabilité une paire d'IV produisant les mêmes bits de pseudo-aléa, et avec une forte probabilité cela correspond à une collision de l'état du compteur.

Une fois une telle paire d'IV obtenue, il est aisé de recouvrer les 53 bits de l'état mis à la clé présents dans les registres 0, 4, 5, 6 et 7 ($w = 11$ pour les registres 0, 4 et 7, $w = 10$ pour les registres 5 et 6). On procède registre par registre. Ayant obtenu une collision après mise à l'IV, l'état du registre i après l'étape 11 de la mise à l'IV est un état en collision pour la paire d'IV extraite de la paire d'IV en collision en prenant la partie relative au registre r, restreinte aux étapes 12 à 23. On retrouve alors l'état du registre en vérifiant les différentes valeurs possibles, tirées de l'ensemble $\mathcal{S}^r$ des états en collision. En remontant les 11 premières étapes de la mise à l'IV, on obtient un candidat pour la valeur du registre après la mise à la clé. Afin de tester cette valeur, on construit une nouvelle paire d'IV à partir de la paire d'IV en collision en ne modifiant que la partie correspondant au registre r. Les 11 premiers bits sont choisis pour assurer que la nouvelle valeur du registre r après les 11 premières étapes appartienne à $\mathcal{S}^r$ si l'hypothèse est correcte. Les 12 bits suivants sont laissés non modifiés dans les deux IV. Si la mise à l'IV utilisant la nouvelle paire d'IV ne conduit pas à une collision de l'état interne, la valeur de l'hypothèse est éliminée des valeurs candidates. La probabilité qu'une valeur candidate erronée passe un test avec succès est de $\#\mathcal{S}^r/2^{w_r}$. En itérant cette procédure on retrouve rapidement les bonnes valeurs de registres. En la répétant sur chacun des registres intéressants pour quelques exemples, nous avons été en mesure de recouvrer les valeurs des registres dans l'état mis à la clé en utilisant moins de 500 mises à l'IV supplémentaires. Il est possible d'améliorer cette première approche en éliminant plusieurs valeurs simultanément en utilisant une seule mise à l'IV. Ceci réduit le nombre de mises à l'IV nécessaires à moins de 200.

Bien entendu, cette attaque peut aisément être adaptée aux contextes où la longueur des IV est supérieure à 23 octets. Elle peut également être utilisée pour des IV plus courts, tant que la longueur des IV est supérieure à 12 octets. Dans ce cas, elle ne permettra plus de recouvrer toutes les clés, car la randomisation des registres ne garantit plus de transformer un état mis à la clé quelconque en un élément de $\mathcal{S}$. Ainsi l'attaque dévient dépendante de la clé et permet

d'identifier une classe de clés faibles.

La partie de l'attaque de plus grande complexité est la découverte d'une paire d'IV réalisant une collision de l'état interne. Sa complexité est d'approximativement $2/p \approx 2^{22.24}$ mises à l'IV. Nous avons implémenté cette attaque. Sur un Pentium Xeon 2.8 GHz, en utilisant l'implémentation de VEST des auteurs, trouver une paire d'IV en collision ne nécessite que quelques minutes.

3.4.2 Attaque avec des IV courts

Dans la section précédente, nous avons décrit une attaque différentielle qui permet de recouvrer une partie de l'état mis à la clé étant donnée la possibilité de réaliser des mises à l'IV pour des IV de longueur suffisante. Nous avons également mentionné que le succès de l'attaque dépend de la valeur de la clé lorsque la taille de l'IV est réduite et que la taille minimale de l'IV pour laquelle l'attaque est possible est de 12 octets. Dans cette sous-section, nous développons une autre approche qui permet de recouvrer l'état mis à la clé des registres intéressants quelle que soit la clé en utilisant des IV de longueur minimales, i.e. 12 octets.

L'attaque présentée dans la sous-section précédente génère des états aléatoires à partir de l'état mis à la clé jusqu'à ce que l'un de ces états tombe dans un ensemble particulier $\mathcal{S}$. L'idée de l'attaque décrite ici est de générer une famille d'ensembles $\{\mathcal{S}_i\}_i$ associés à différentes paires d'IV qui couvre tout l'espace des états.

En effet, pour un registre particulier d'indice r, on peut associer à la paire d'IV C_i^r pour ce registre l'ensemble des états qui collisionnent pour cette paire d'IV en produisant une différence de sortie constituée d'un 1 suivi de 0. Nous avons vu en section 3.3.1 que pour certaines paires d'IV C^r, la taille de l'ensemble d'états en collision associé $\mathcal{S}^r$ peut être particulièrement importante. La taille w des registres étant de seulement 10 ou 11 bits, nous avons vu qu'il est possible de calculer pour chaque paire d'IV de longueur $w+1$ l'ensemble des états en collision pour cette paire. On obtient ainsi une famille d'ensembles. Nous avons vérifié que l'union de ces ensembles couvre l'intégralité de l'espace des valeurs de registres pour toutes les fonctions non linéaires proposées. Ainsi, il est possible de sélectionner $N^{(r)} < 2^{w_r}$ ensembles, $\mathcal{S}_1^r, \mathcal{S}_2^r, \ldots \mathcal{S}_{N^{(r)}}^r$, de manière à ce que l'union de ces ensembles soit $\{0,1\}^{w_r}$. En prenant le produit cartésien de 5 familles d'ensembles choisies pour les registres 0, 4, 5, 6 et 7, on obtient une famille d'ensembles couvrant toutes les valeurs possibles des 53 bits de l'état mis à la clé correspondant aux 5 registres intéressants. La taille de cette famille $\{\mathcal{S}_i\}_i$ est le produit de la taille des familles choisies pour chaque registre : $N = N^{(0)}N^{(4)}N^{(5)}N^{(6)}N^{(7)}$. Les éléments de la famille sont des ensembles d'états en collision après mise à l'IV utilisant une paire d'IV obtenue en combinant les paires d'IV utilisées pour chaque registre.

Supposons que nous disposions d'une telle famille $\{\mathcal{S}_i\}_i$ et de la famille de paires d'IV associée. En réalisant deux mises à l'IV pour chaque paire, on est assuré de trouver une paire d'IV pour laquelle on obtienne une collision après la mise à l'IV. La complexité de cette attaque en le nombre de mises à l'IV nécessaires est de $2N$ dans le pire cas. De plus, si on commence par tester les paires dont les ensembles d'états en collision sont les plus grands, le nombre moyen de mises à l'IV nécessaires diminue.

Afin d'obtenir la meilleure complexité, nous devons construire pour chaque registre une famille *couvrante* de paires d'IV $\{C_i^r\}_i$. Afin d'améliorer la complexité de l'attaque dans le cas moyen, on utilise un algorithme glouton pour générer ces familles. On construit d'abord pour chaque paire d'IV l'ensemble des états en collision. On les trie par ordre décroissant de taille et on sélectionne la première paire. On retire alors les éléments de l'ensemble des états en collision des ensembles des paires non sélectionnées et on les retrie par ordre décroissant de taille. On

itère cet étape jusqu'à obtenir une liste de paires d'IV telles que tout état est en collision pour une paire de la liste. Nous donnons à titre d'exemple en Table 3.2 la taille des listes obtenues pour les fonctions utilisées par les registres 0, 4, 5, 6 et 7 lorsque la clé de famille par défaut est utilisée pour sélectionner un algorithme de la famille VEST.

TABLE 3.2 – Taille de familles couvrantes de quelques fonctions non linéaires

indice de la fonction	taille d'une famille couvrante
0	59
1	93
19	77
20	86
2	96

Une fois obtenues les familles pour les cinq registres intéressants, nous devons les combiner pour obtenir une famille de paires d'IV pour l'intégralité du compteur. Afin de minimiser la complexité dans le cas moyen pour l'attaque en utilisant ces 5 familles, nous devons tester les paires par ordre décroissant de la taille de leur ensemble d'états en collision. Pour le membre par défaut de la famille VEST-v, le nombre de paires est approximativement $2^{31.70}$. Les algorithmes de tri génériques sont difficilement applicables en raison de la quantité de mémoire nécessaire. Cependant, nous n'avons pas vraiment besoin de stocker la liste d'IV dans l'ordre, tant qu'on est en mesure de l'énumerer rapidement dans l'ordre. De plus, la seule information requise pour réaliser la comparaison est la taille des ensembles d'états en collision pour chaque paire d'IV. Ainsi nous devons résoudre le problème suivant : Étant donné deux listes d'entiers $(a_i)_{1\leq i\leq N_a}, (b_j)_{1\leq j\leq N_b}$ triées par ordre décroissant, énumérer par ordre décroissant leurs produits $a_i b_j$.

Dans [126] et [28], un algorithme est proposé pour résoudre ce problème. Cet algorithme ne stocke par tous les produits. En supposant sans perte de généralité que $N_a > N_b$, sa complexité temporelle est de $\mathcal{O}(N_a N_b \log N_b)$ et sa complexité en mémoire est de $\mathcal{O}(N_b)$.

En implémentant cet algorithme et en utilisant les 5 familles obtenues, nous avons été en mesure d'énumerer les $2^{31.70}$ paires d'IV de la famille couvrante par ordre décroissant de leur ensemble d'états en collision en moins de 2 heures sur un processeur Celeron M à 1.4 GHz. En notant $(C_i)_{1\leq i\leq N}$ cette liste de paire, $(\mathcal{S}_i)_{1\leq i\leq N}$ la liste associée d'états en collision et $N_i = \#(S_i)_{1\leq i\leq N}$ la liste décroissante des cardinaux de ces ensembles, la complexité moyenne de la recherche de collision en nombre de paires de mises à l'IV est de :

$$\mathcal{C} = \sum_{i=1}^{N} i \cdot \frac{N_i}{2^{53}}.$$

On calcule cette somme pendant l'énumération des paires en ordre décroissant. Pour le membre par défaut de la famille VEST et les familles de paires d'IV choisies on obtient $\mathcal{C} \approx 2^{27.73}$.

Une fois une collision obtenue, on a une petite liste de valeurs candidates pour la partie de l'état mis à la clé correspondant aux cinq registres intéressants. En testant ces candidats séparément pour chaque registre, en modifiant l'IV spécifique au registre, on est en mesure de reconstituer 53 bits de l'état mis à la clé. Le nombre de mises à l'IV additionnelles pour ces tests est négligeable par rapport au nombre de mises à l'IV nécessaires pour trouver une paire d'IV en collision.

L'attaque décrite dans cette sous-section permet à un attaquant de trouver 53 bits de l'état mis à la clé en $2^{28.73}$ mises à l'IV en moyenne et $2^{32.70}$ dans le pire cas. C'est légèrement plus

que la complexité de l'attaque avec des IV longs. Cependant cette attaque utilise des IV de taille minimale. En effet, au moins 12 étapes sont nécessaires pour créer une collision dans les registres 0, 4 et 7. Avec des IV plus longs, de taille comprise entre 12 et 23 octets, on peut utiliser le début de l'IV pour randomiser partiellement l'état et ajouter une stratégie d'early-abort à l'attaque décrite dans cette section pour améliorer sa complexité.

3.5 Recouvrement de clé

Dans cette section, nous montrons comment la partie de l'état mis à la clé obtenue à la suite des attaques présentées dans les sections précédentes peut être utilisée pour accélérer la recherche exhaustive de la clé de l'algorithme de chiffrement par flot. Ceci permet d'évaluer l'impact sur la sécurité de l'algorithme des collisions mises à jour dans le mécanisme de mise à l'IV.

3.5.1 Inversion de la deuxième phase de la mise à la clé

On commence par défaire la deuxième phase de la mise à la clé. Comme tous les bits perturbant les registres intéressants (0, 4, 5, 6 et 7) sont connus, nous sommes en mesure de reconstituer les états de ces registres après l'introduction des bits de clé. On sait également que tous les bits de l'état sont égaux à 1 avant l'introduction des bits de clé. On peut donc réaliser une attaque par le milieu sur la clé, en dépit du fait que les bits de clé soient introduits dans tous les registres selon un mécanisme de fenêtre glissante. Le même type d'attaque a conduit à la disqualification du double-DES en tant que successeur du DES.

3.5.2 Attaque par le milieu

On peut remarquer que les bits 0 à $\ell - 1$ (resp. ℓ à $F - 1$) de la clé sont introduits dans le registre r entre les étapes $15 - r$ et $\ell - 1 + 15 - r$ (resp. $\ell + 15 - r$ et $F - 1 + 15 - r$). En faisant une hypothèse sur les ℓ premiers bits ou les $F - \ell$ derniers bits de la clé, on est en mesure de calculer la valeur du registre r avant l'étape $\ell + 15 - r$. On peut ainsi réaliser une attaque de type compromis temps-mémoire contre les algorithmes VEST. On construit la table $\mathcal{A}$ des valeurs, sur 53 bits, des 5 registres intéressants à l'étape $\ell + 15 - r$. Ceci requiert une mémoire de taille $53 \cdot 2^{\ell}$.

Puis, pour chacune des $2^{F-\ell}$ valeurs j possibles pour la fin de la clé on réalise les opérations suivantes :

- remonter aux valeurs des registres à partir des 53 bits de l'état mis à la clé et de l'hypothèse j : on note x la valeur de 53 bits ainsi obtenue ;
- rechercher i tel que $\mathcal{A}[i] = x$. La probabilité de trouver une telle valeur est de 2^{-53}.
- pour chaque valeur trouvée, vérifier la clé $i \| j$, où $\|$ désigne la concaténation.

Complexité. L'algorithme décrit ci-dessus permet d'explorer toutes les valeurs de clé pour lesquelles la mise à la clé initialise les 5 registres intéressants aux valeurs reconstituées par l'attaque à IV choisis. Il permet de retrouver la clé de l'algorithme de chiffrement par flot VEST avec une complexité de l'ordre de $2^{\max(F-53, F-\ell)}$ en temps et de 2^{ℓ} en mémoire. Le nombre moyen de clés testées est 2^{F-53}.

3.5.3 Recouvrement de clé par attaque à clés corrélées

Il est également possible de réaliser des attaques à clés corrélées très efficaces. Supposons que l'on donne à l'attaquant la possibilité d'accéder à l'algorithme VEST mis à la clé avec des clés K et K' ne différant qu'en position $F-\ell-1$. En réalisant une attaque à IV choisis, il est en mesure de récupérer pour chaque clé l'état mis à la clé des cinq registres intéressants. En faisant une hypothèse sur les ℓ derniers bits de la clé K, il peut remonter l'introduction de ces bits de clés à partir de l'état mis à la clé jusqu'à l'étape suivant l'introduction du bit de différence de clé. Pour la bonne hypothèse, les états obtenus en partant des états mis à la clé pour K et K' ne diffèrent qu'en position 1 des registres. Pour les mauvaises hypothèses, la probabilité que cette propriété soit satisfaite est de 2^{-53}. L'attaquant peut donc confirmer son hypothèse dès lors que $\ell < 53$. Pour que les états se comportent de manière aléatoire et que l'heuristique décrite ci-dessus soit applicable, ℓ doit être supérieur à la taille des registres.

Une fois les ℓ derniers bits de la clé déterminés, on itère le procédé sur la partie inconnue de la clé. Pour une clé de 128 bits, en choisissant $\ell = 16$, on est en mesure de retrouver la clé en utilisant 8 clés liées et en réalisant de l'ordre de 2^{26} mises à l'IV et 2^{19} tests d'hypothèses.

3.5.4 Discussion

Les attaques décrites ci-dessus montrent que les produits des attaques différentielles à IV choisis exposées en section 3.4 peuvent théoriquement être exploités pour retrouver la clé d'un algorithme VEST plus rapidement qu'en réalisant une recherche exhaustive. Ceci remet en cause la sécurité de VEST lorsqu'il est utilisé avec des clés de taille égale au paramètre de sécurité. En [115, Section 3.3], les auteurs de VEST recommandent l'utilisation de clés de taille égale à au moins deux fois le paramètre de sécurité. Dans ce cas, on peut considérer que VEST résiste aux attaques ne faisant pas intervenir de clés liées dans la mesure où la complexité de notre compromis temps-mémoire est supérieure au niveau de sécurité visé. Cependant, dans le cadre d'attaques à clés liées, la sécurité s'écroule. Dans tous les cas, il n'est pas souhaitable d'employer des algorithmes de chiffrement par flot pour lesquels il est possible de reconstituer une partie de l'état mis à la clé.

La spécification de VEST autorise l'usage de clés de longueur égale au paramètre de sécurité. Dans ce cas d'utilisation, elle recommande même l'utilisation d'IV longs. Dans sa version 2, VEST n'atteint pas le niveau de sécurité annoncé.

3.6 Forge existentielle pour le mode VEST Hash MAC

VEST peut être utilisé comme une fonction de hachage avec clé en utilisant la procédure décrite dans [115, Section 3.4]. Dans ce mode l'algorithme VEST est d'abord mis à la clé. Puis le message dont on veut calculer un motif d'intégrité est introduit dans l'état comme un IV en utilisant la procédure de mise à l'IV classique, en modifiant uniquement la constante utilisée pendant la deuxième phase de la mise à l'IV. Pour terminer, $2n$ bits sont produits par l'algorithme en mode génération de pseudo-aléa et constituent le MAC. Pour spécifier complètement cet algorithme de MAC, un schéma de padding doit être ajouté. Cependant, indépendamment de ce schéma de padding, les attaques à IV choisis se traduisent naturellement en attaques en forge existentielle à messages choisis.

En effet les IV dans les attaques précédentes sont ici remplacés par les messages. Ainsi, en demandant le MAC d'approximativement $2^{22.24}$ messages choisis, un attaquant peut reconstituer 53 bits de l'état mis à la clé. Étant donné la valeur obtenue, l'attaquant peut ensuite créer une

paire de messages en collision après introduction du message. En demandant un MAC pour l'un de ces messages, il obtient également un MAC pour l'autre message. Ceci constitue une attaque de type forge existentielle à message choisis.

Deuxième partie

Fonctions de Hachage

Chapitre 4

Préliminaires

Sommaire

On s'intéresse dans cette partie aux *fonctions de hachage cryptographiques* et plus particulièrement à leur construction à travers l'utilisation de modes opératoires appelés *extensions de domaine*. On introduit dans ce chapitre ces deux notions. Après quelques rappels sur les preuves de sécurité en cryptographie, on définit les fonctions de hachage et leur sécurité, on présente un bref historique de l'étude des fonctions de hachage et des principaux résultats du domaine. On aborde ensuite les extensions de domaine, notamment celle attribuée à Merkle et Damgård. On définit leur sécurité et on dresse la liste des extensions de domaine les plus répandues.

4.1 Preuves de sécurité en cryptographie

4.1.1 Principe

Une primitive, un protocole, un système cryptographique remplit un objectif de sécurité. L'évaluation de la sécurité d'un système cryptographique consiste à éliminer l'existence d'attaquants (efficaces) susceptibles de remettre en cause ces objectifs de sécurité. Pour les primitives cryptographiques (algorithme de chiffrement par bloc, générateur pseudo-aléatoire, fonction de hachage...) il est difficile d'obtenir une garantie générale car il est difficile d'identifier de manière

exhaustive tous les attaquants envisageables. Pour les protocoles cryptographiques, la situation est légèrement différente. En effet, ces protocoles sont des constructions mettant en œuvre des primitives cryptographiques. En formalisant les ressources de l'attaquant et/ou en idéalisant les primitives cryptographiques, il est possible de dériver des preuves de sécurité réduisant la sécurité du protocole cryptographique à un problème mathématique réputé difficile ou à la sécurité des primitives utilisées.

Les résultats de ces preuves par réduction sont parfois difficiles à interpréter. Trois éléments concourent à rendre subtile l'interprétation des preuves en cryptographie :

- La *pertinence de la modélisation* adoptée dans la preuve, notamment les moyens de l'attaquant. Une preuve reposant sur un modèle d'attaquant passif, qui se contente d'observer les messages échangés sans les modifier, n'apportera aucune garantie de sécurité contre les attaques actives, où l'attaquant a la possibilité d'insérer des messages de son choix dans les communications, de supprimer ou de modifier des messages. Pour qu'une preuve apporte des garanties de sécurité, elle doit prendre en compte des classes d'attaquants les plus puissants possibles, c'est à dire disposant de beaucoup de moyens. La prise en compte de ces moyens est détaillée en section 4.1.2.
- L'utilisation d'un *modèle idéalisé*. Lorsqu'une primitive cryptographique est remplacée par une représentation idéalisée, la preuve suppose qu'un attaquant n'accède à la primitive en question qu'à travers son interface et que la primitive se comporte de manière idéale. Les preuves dans des modèles idéalisés, comme les modèles de l'*oracle aléatoire* (ROM) ou du *chiffrement idéal* (ICM), ne prouvent pas directement la sécurité d'un cryptosystème complet, comme l'attestent de nombreux résultats d'ininstanciabilité [36, 68, 11] qui démontrent qu'une primitive concrète ne peut instancier sa représentation idéalisée en exhibant des protocoles cryptographiques sûrs dans les modèles idéalisés et non sûrs dès qu'une primitive concrète est utilisée. De nombreux auteurs, dont [13, 133], estiment que les preuves de sécurité dans un modèle idéalisé attestent de la correction structurelle des protocoles cryptographiques étudiés et qu'elles restent donc utiles en apportant notamment une garantie sur leur sécurité contre des attaquants ne cherchant pas à exploiter la structure interne de la primitive idéalisée.
- L'*efficacité de la réduction*. Les preuves par réduction consistent à prouver que l'existence d'un attaquant contre le protocole cryptographique entraîne l'existence d'un attaquant contre un problème bien étudié. Ces preuves sont en général constructives, c'est-à-dire qu'elles exhibent un moyen d'utiliser un attaquant contre le protocole pour construire un attaquant contre la primitive. Cependant, un point important dans ces preuves est l'estimation des moyens des attaquants en termes de puissance de calcul et de nombre de requêtes au protocole cryptographique. En effet, dans une preuve par réduction constructive, l'attaquant obtenu contre la primitive sous-jacente est susceptible d'exécuter de nombreuses fois l'attaquant contre le protocole, de réaliser des calculs de complexité élevée, etc. On dit que la réduction est *efficace* si les ressources utilisées par l'attaquant contre la primitive sous-jacente sont proches des ressources utilisées par l'attaquant contre le protocole, et non-efficace quand elles sont beaucoup plus élevées. Une preuve efficace est préférable du point de vue dimensionnement du cryptosystème. En effet, on déduit de l'absence d'attaquant contre la sécurité de la primitive ayant des ressources inférieures à c et de la preuve l'absence d'attaquant contre le protocole ayant des ressources inférieures à c'. Lorsque la preuve est efficace, $c \approx c'$ et le protocole est sûr dès que la primitive est dimensionnée de manière traditionnelle. Si la preuve n'est pas efficace, alors $c' \ll c$ et la preuve ne permet d'éliminer que des attaquants disposant de ressources faibles, plus

faibles que celles dont dispose un attaquant réaliste. Afin de disposer de bonnes garanties de sécurité en tirant parti de la preuve, on doit donc surdimensionner la primitive afin qu'elle résiste à des attaquants plus puissants, ce qui a un coût en terme de performances.

4.1.2 Adversaire, Objectifs, Moyens et Avantage

La sécurité des primitives et protocoles cryptographiques est formalisée au travers de la notion de jeux de sécurité. Dans le cadre de ces jeux, un *attaquant*, encore désigné sous le terme d'*adversaire*, modélisé par une machine de Turing probabiliste, cherche à réaliser un *objectif*, comme distinguer une primitive cryptographique d'une fonction idéale ou retrouver la clé d'une primitive cryptographique. Pour ce faire, l'attaquant dispose de *moyens*. Pour représenter ces moyens :

- on précise les oracles auxquels il a accès. Ces oracles, auxquels l'attaquant a un accès de type boîte noire, possèdent une interface lui permettant d'obtenir des réponses à ses requêtes. Ils peuvent être utilisés pour représenter par exemple des primitives cryptographiques mises à la clé avec une clé à laquelle l'attaquant n'a pas accès, ou encore, dans des modèles idéalisés une primitive publique accessible par l'ensemble des participants du système. Ils sont également formalisés comme des machines de Turing probabilistes.
- la quantification des ressources de l'attaquant, en termes de temps, de mémoire, de nombre de requêtes à ses oracles, etc.

Enfin, on estime la probabilité de succès de l'attaquant, calculée par rapport aux jetons aléatoires des oracles de l'attaquant et à ses propres jetons aléatoires. La sécurité d'un objet cryptographique est estimée par la valeur maximale de cette probabilité sur la classe des attaquants disposant de ressources bornées.

Un type d'adversaire, très répandu dans les preuves de sécurité, est traité de manière légèrement différente. Lorsqu'un adversaire produit une valeur binaire, on le désigne par le terme de *distingueur*. Un tel adversaire $\mathcal{D}$ cherche à identifier un objet cryptographique inconnu $\mathcal{O}$ auquel il a un accès en boîte noire parmi deux objets cryptographiques d'interface identique $\mathcal{O}_0$ et $\mathcal{O}_1$, par exemple une primitive cryptographique et sa version idéalisée. Il retourne l'indice de l'objet auquel il pense être confronté. Le jeu de sécurité consiste alors à choisir de manière uniforme l'un de ces objets $\mathcal{O} = \mathcal{O}_b$, exécuter le distingueur en lui donnant accès à cet objet, et constater si le distingueur reconnaît bien l'objet avec lequel il a interagit. Dans ce scénario, un attaquant naïf répondant de manière aléatoire a une probabilité $\frac{1}{2}$ de répondre correctement. La probilité de succès de l'attaquant est donc moins intéressante que le biais de cette probabilité. En règle générale, la quantité estimée est

$$\left|\Pr\left[\mathcal{D}^{\mathcal{O}_0} = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{O}_1} = 1\right]\right| = \left|2\Pr\left[\mathcal{D}^{\mathcal{O}_b} = b\right] - 1\right|.$$

On la désigne usuellement sous le terme d'avantage de l'attaquant $\mathcal{D}$ et pour une classe d'attaquants disposant de ressources bornées, la difficulté de distingueur les deux objets est estimée par la valeur maximale de l'avantage sur cette classe.

Cette formalisation, qui précise les moyens des attaquants, permet de suivre leur évolution au cours d'une preuve de sécurité. On parle de sécurité concrète.

On motive dans les deux sections suivantes deux cadres d'analyse utilisés usuellement dans les preuves de sécurité. On présente au préalable les deux oracles publics les plus fréquemment rencontrés.

Oracle aléatoire. Un oracle aléatoire est un oracle public, *i.e.* accessible à tous les participants au jeu de sécurité. Il modélise une fonction aléatoire. À chaque nouvelle requête, une nouvelle

valeur est tirée uniformément dans l'image de l'oracle aléatoire et indépendamment des valeurs précédemment tirées. Un oracle aléatoire peut être vu comme évaluant de manière paresseuse une fonction tirée uniformément dans l'ensemble des fonctions de même domaine et de même image que l'oracle aléatoire.

Fonction de chiffrement idéale. Une fonction de chiffrement idéale est également un oracle public. Elle possède deux interfaces, permettant de réaliser des requêtes de « chiffrement » et de « déchiffrement » sous une clé explicitée dans la requête. Elle modélise un algorithme de chiffrement par bloc tiré aléatoirement, *i.e.* pour chaque valeur du paramètre de clé, les fonctions de chiffrement et de déchiffrement sont inverses l'une de l'autre et obtenues par tirage aléatoire dans l'ensemble des permutations de l'image de la fonction de chiffrement idéale. Les tirages de ces permutations sont également indépendants de la valeur du paramètre de clé.

4.1.3 Indistinguabilité

Dans le cadre de l'indistinguabilité, on cherche à prouver que l'avantage de tout distingueur cherchant à distinguer un objet cryptographique d'une version idéalisée est faible. Par exemple, un générateur pseudo-aléatoire est comparé avec une source idéale d'aléa. On dit qu'un système $\mathcal{C}$ est au moins aussi sûr qu'un système $\mathcal{C}'$, si pour tout attaquant $\mathcal{A}$ contre le système $\mathcal{C}$ il existe un attaquant $\mathcal{A}'$ contre le système $\mathcal{C}'$ dont la probabilité de succès est proche de celle de $\mathcal{A}$. Les preuves en indistinguabilité ont une portée importante du fait du théorème de composition suivant.

Théorème 6 ([94]) *Soit $\mathcal{S}$ et $\mathcal{T}$ deux objets cryptographiques. $\mathcal{S}$ et $\mathcal{T}$ sont indistinguables si et seulement si pour tout cryptosystème $\mathcal{C}$ utilisant $\mathcal{T}$ comme primitive, le cryptosystème $\mathcal{C}(\mathcal{S})$ obtenu en remplaçant $\mathcal{T}$ par $\mathcal{S}$ est au moins aussi sûr que $\mathcal{C}(\mathcal{T})$.*

Ainsi, pour prouver la sécurité d'un schéma cryptographique $\mathcal{C}$ reposant sur un objet cryptographique $\mathcal{S}$ indistinguable d'une version idéalisée $\mathcal{T}$, il suffit de montrer la sécurité du schéma cryptographique lorsqu'il fait appel à l'objet cryptographique $\mathcal{T}$.

Afin de pouvoir appliquer ce théorème de composition, il faut cependant que l'attaquant ne dispose pas d'information supplémentaire sur les aléas utilisés par le schéma cryptographique. Le cadre de l'indistinguabilité n'est donc pas adapté à l'analyse de constructions cryptographiques faisant usage d'une primitive publique, comme un oracle aléatoire.

4.1.4 Indifférentiabilité

Le cadre de l'indifférentiabilité est une généralisation du cadre de l'indistinguabilité permettant de traiter les cas où l'attaquant a accès à des éléments internes de l'objet étudié. Cette information supplémentaire n'est pas nécessairement disponible dans les deux systèmes qu'on cherche à distinguer. Par exemple, considérons un objet cryptographique obtenu en appliquant une construction à une primitive idéale publique, et une modélisation idéalisée de cet objet. Lorsqu'on considère l'objet cryptographique de départ, la primitive utilisée par la construction est accessible à l'attaquant. Cependant dans la modélisation idéalisée de cette construction, elle n'a pas d'équivalent. Il n'y a pas d'équivalent à l'information additionnelle fournie par un accès à la primitive sous-jacente dans la modélisation idéalisée. La cadre doit donc être adapté Afin de généraliser le théorème de composition obtenu dans le cadre de l'indistinguabilité, l'idée est de montrer que l'accès aux éléments internes est *simulable* par l'attaquant et que cet accès ne lui

apporte donc pas d' information additionnelle permettant de distinguer les deux cryptosystèmes considérés. On reviendra sur cette notion de simulateur dans les chapitres 5 et 6, où l'on donnera des exemples de construction de simulateurs. On renvoie à [94] pour une preuve du théorème de composition dans le cadre de l'indifférentiabilité.

4.1.5 Techniques de preuve : Preuve par séquence de jeux

On termine cette présentation rapide des preuves de sécurité en cryptographie en abordant quelques techniques usuelles permettant de systématiser et de simplifier l'établissement et la validation de preuves cryptographiques. L'outil général est connu sous le nom de *preuve par jeu* et a été formalisé par Shoup [129], lequel s'est appuyé sur des travaux antérieurs pour formuler ce cadre.

4.1.5.1 Principe

Les preuves par séquence de jeux s'emploient à montrer que deux jeux de sécurité G et G' sont proches et permettent dans ces cas de borner la probabilité qu'ils se comportent de manières différentes. L'intérêt de ces preuves reposent sur deux idées :

- On ne prouve pas directement la proximité des jeux G et G', mais on procède de manière itérative en passant par une succession (finie) de jeux $G = G_0, G_1, \ldots, G_n = G'$. La proximité de G et G' peut être dérivée de la proximité pour tout i de G_i et G_{i+1}. Ceci simplifie les preuves de sécurité, en remplaçant une preuve monolithique complexe à analyser en une succession de preuves plus élémentaires.
- On travaille de manière formelle, en modifiant petit à petit, d'un jeu à l'autre, le code des algorithmes mis en œuvre. Cette manière de procéder donne une plus grande confiance dans l'exhaustivité de la preuve. De plus les transitions entre jeux peuvent êtres classées en trois types classiques, dont l'impact sur les probabilités de comportement des jeux successifs est bien étudié.

4.1.5.2 Type de transition

Dans sa formalisation, Shoup identifie trois types de transition. On les rappelle brièvement ici.

Transition par changement de distribution. Ce type de transition consiste à modifier la distribution de tirage d'une variable aléatoire entre deux jeux. Dans ce cas, les deux jeux peuvent servir à distinguer les deux distributions. La différence de probabilités entre les deux jeux est donc bornée par l'avantage maximum d'un distingueur entre les deux distributions. Lorsqu'on borne le nombre de tirages de cette valeur aléatoire par N au cours du jeu, l'avantage maximum peut être borné par N fois la distance statistique entre les deux distributions de tirage. Pour deux distributions de probabilité $\Pr_0$ et $\Pr_1$ sur un ensemble Ω la distance statistique Δ est donnée par

$$\Delta = \frac{1}{2} \sum_{x \in \Omega} \left| \Pr_0 [x] - \Pr_1 [x] \right| .$$

Transition par délimitation des différences. Ce type de transition est utilisé quand on arrive à montrer que deux jeux successifs se comportent de manière identique tant qu'un évènement Ev ne se produit pas. Le lemme de différence permet alors de borner la différence de probabilités entre les deux jeux.

Lemme 2 *Lemme de différence Soit* Ev_i, $1 \leq i \leq 4$, *quatre évènements tels que* $|\Pr[\mathsf{Ev}_1 \wedge \neg\mathsf{Ev}_2] - \Pr[\mathsf{Ev}_3 \wedge \neg\mathsf{Ev}_4]| \leq \alpha$. *Alors on a*

$$|\Pr[\mathsf{Ev}_1] - \Pr[\mathsf{Ev}_3]| \leq \alpha + \max(\Pr[\mathsf{Ev}_2], \Pr[\mathsf{Ev}_4]).$$

Démonstration : On a

$$\begin{aligned} \Pr[\mathsf{Ev}_1] - \Pr[\mathsf{Ev}_3] &= \Pr[\mathsf{Ev}_1 \wedge \mathsf{Ev}_2] + \Pr[\mathsf{Ev}_1 \wedge \neg\mathsf{Ev}_2] - \Pr[\mathsf{Ev}_3 \wedge \mathsf{Ev}_4] - \Pr[\mathsf{Ev}_3 \wedge \neg\mathsf{Ev}_4], \\ &= (\Pr[\mathsf{Ev}_1 \wedge \mathsf{Ev}_2] - \Pr[\mathsf{Ev}_3 \wedge \mathsf{Ev}_4]) + (\Pr[\mathsf{Ev}_1 \wedge \neg\mathsf{Ev}_2] - \Pr[\mathsf{Ev}_3 \wedge \neg\mathsf{Ev}_4]). \end{aligned}$$

Ainsi

$$-\Pr[\mathsf{Ev}_4] - \alpha \leq \Pr[\mathsf{Ev}_1] - \Pr[\mathsf{Ev}_3] \leq \Pr[\mathsf{Ev}_2] + \alpha,$$

ce qui conclut la preuve. □

Transition par reformulation. Ce type de transition est purement formel. Il consiste à modifier le code d'un jeu en celui d'un autre jeu sans modifier les probabilités des variables aléatoires intervenant au cours de l'exécution du jeu. Dans ce cas de figure, la différence de probabilité entre les deux jeux est nulle.

4.2 Fonctions de hachage cryptographiques

On présente dans cette section les fonctions de hachage cryptographiques, leurs usages et les notions de sécurité qu'elles doivent satisfaire.

4.2.1 Définition - Attaques génériques

4.2.1.1 Définitions - Sécurité

Nous définissons ici les *fonctions de hachage*. Nous énonçons leurs propriétés, et notamment les propriétés que doivent remplir les fonctions de hachage *cryptographiques*. Nous terminons par une discussion sur la manière d'évaluer la *sécurité* des fonctions de hachage.

Définition 23 *On appelle fonction de hachage une fonction*

$$h : \{0,1\}^* \to \{0,1\}^{\ell_h},$$

où $\{0,1\}^*$ *désigne l'ensemble des suites de bits de longueur finie arbitraire. On note* ℓ_h *la taille de la sortie de la fonction de hachage. On appelle haché d'un message* $M \in \{0,1\}^*$ *son image* $H = h(M)$ *par la fonction de hachage* h.

Une fonction de hachage est donc une fonction qui, à un message de taille quelconque, associe une valeur de longueur fixe. Les applications de telles fonctions sont nombreuses. Pour avoir une utilité pratique, une fonction de hachage doit vérifier une propriété supplémentaire : elle doit pouvoir être évaluée efficacement.

En cryptographie, on attend de plus des fonctions de hachage qu'elles se comportent comme des fonctions aléatoires. Pour des raisons pratiques, on ne peut pas mettre en œuvre des procédés aléatoires pour réaliser des fonctions de hachage. On utilise des procédés déterministes, « pseudo-aléatoires ». On attend d'une fonction de hachage qu'il soit aussi difficile de la mettre en défaut qu'une fonction aléatoire. Suivant le contexte d'utilisation, cette attente peut se traduire par les exigences définies ci-dessous :

Définition 24 *Une fonction de hachage est dite*

- ***Résistante en préimage*** *si étant donné un haché H, il est calculatoirement impossible de trouver un message M tel que $h(M) = H$,*
- ***Résistante en seconde préimage*** *si étant donné un message M, il est calculatoirement impossible de trouver un message $M' \neq M$ tel que $h(M) = h(M')$,*
- ***Résistante en collision*** *s'il est calculatoirement impossible de trouver deux messages $M \neq M'$ tels que $h(M) = h(M')$.*

Notons que l'existence de collisions est inévitable, du fait des tailles des ensembles de départ et d'arrivée d'une fonction de hachage. Il n'est pas pour autant facile de trouver des collisions. Le terme « calculatoirement impossible » sera explicité dans les paragraphes suivants. En première approximation, on peut dire que « calculatoirement impossible » signifie qu'il n'existe aucun algorithme connu permettant de trouver dans un temps réaliste une solution au problème étudié.

Ces propriétés permettent de répondre à différents besoins des protocoles cryptographiques. À titre d'exemple, une fonction de hachage résistante en collision peut être utilisée dans un schéma de signature pour obtenir une empreinte du message à signer. Une fonction de hachage résistante en préimage, en deuxième préimage et en collision permet de réaliser un engagement sur une valeur.

Proposition 18 *Si h est une fonction de hachage résistante en collision, elle est résistante en seconde préimage.*

La résistance en collision n'implique cependant pas la résistance en préimage (cf [99], note 9.20). En pratique, la propriété la plus facilement mise en défaut est la résistance en collision et la mise en défaut de la résistance en collision ne remet pas forcément en cause la résistance en (seconde) préimage. La sécurité apportée par une fonction de hachage peut dépendre de son contexte d'utilisation : certains contextes requièrent la résistance en collision, alors que pour d'autres la résistance en préimage est suffisante.

4.2.1.2 Attaques génériques

Nous présentons dans cette section des algorithmes génériques permettant de trouver collisions et (secondes) préimages pour des fonctions aléatoires. Ces algorithmes sont dits génériques car ils ne requièrent aucune connaissance du fonctionnement de la fonction de hachage attaquée. On considère une fonction aléatoire $f : \mathcal{X} \to \mathcal{Y}$. L'ensemble d'arrivée $\mathcal{Y}$ est supposé fini, de cardinal 2^{ℓ_h}.

Attaque en (seconde) préimage. Pour une fonction aléatoire, la probabilité que l'image d'un élément de $\mathcal{X}$ soit $y \in \mathcal{Y}$ est $\frac{1}{2^{\ell_h}}$. L'algorithme 7 requiert en moyenne $\mathcal{O}\left(2^{\ell_h}\right)$ calculs de f. Pour trouver une seconde préimage de f pour x, on prend $y = f(x)$.

Attaque en collision. La meilleure attaque générique en collision repose sur une propriété mathématique contre-intuitive désignée par le terme *paradoxe des anniversaires*. Cette propriété énonce que trouver une collision pour une fonction aléatoire requiert un nombre d'évaluations de la fonction bien inférieur à la taille de l'espace d'arrivée.

Proposition 19 *Soit $f : \mathcal{X} \to \mathcal{Y}$ une fonction aléatoire, où $\mathcal{Y}$ est un ensemble fini de cardinal t. Alors le nombre d'évaluations nécessaires pour trouver une collision est en $\mathcal{O}\left(\sqrt{t}\right)$.*

Algorithme 7 Recherche générique de préimage

Entrée : $y \in \{0,1\}^n$
Sortie : x tel que $f(x) = y$
  tant que *true* **faire**
    choisir aléatoirement $x \in \mathcal{X}$
    si $f(x) = y$ **alors**
      return x
    fin si
  fin tant que

L'algorithme 8 permet de rechercher une collision sur une fonction aléatoire. Il fait appel à une fonction recherche qui cherche parmi les valeurs de haché déjà obtenues la valeur que l'on vient de calculer, renvoyant l'index de la première valeur trouvée, ou -1 s'il n'existe pas de telle valeur. Le paradoxe des anniversaires prévoit que le nombre d'évaluations à effectuer sera de l'ordre de $\sqrt{2^{\ell_h}} = 2^{\frac{\ell_h}{2}}$.

Algorithme 8 Recherche générique de collision

Sortie : x, x' tel que $f(x) = f(x')$
  $j \leftarrow -1$
  tant que *true* **faire**
    choisir aléatoirement $x_i \in \mathcal{X}$
    $y_i = f(x_i)$
    $j \leftarrow \mathsf{recherche}(y_i)$
    si $j \neq -1$ et $x_i \neq x_j$ **alors**
      return (x_i, x_j)
    fin si
    stocker (x_i, y_i)
  fin tant que

On peut également utiliser l'algorithme de Floyd ([87, exercices 6 et 7, page 4]) qui permet de s'abstraire du coût en mémoire de l'algorithme précédent.

4.2.2 Évaluation de la sécurité d'une fonction de hachage

On considère qu'une fonction de hachage est sûre si :

- ll n'existe pas d'attaque connue remettant en cause sa résistance en collision ou en préimage qui soit plus rapide que les attaques génériques correspondantes. Il ne doit donc pas être possible de trouver une préimage en un nombre d'appels à la fonction de hachage inférieur à $\mathcal{O}\left(2^{\ell_h}\right)$, ni de trouver une collision en un nombre d'appels inférieur à $\mathcal{O}\left(2^{\frac{\ell_h}{2}}\right)$;
- La taille de la sortie est suffisamment grande pour que les complexités de ces attaques dépassent largement le nombre d'opérations réalisables.

L'existence de propriétés non-idéales des composants internes d'une fonction de hachage peut remettre en question la sécurité de cette fonction. On détaille ce point dans la section 6.1.

4.2.3 Formalisation de la sécurité d'une fonction de hachage

Lorsqu'on prouve la sécurité d'un schéma cryptographique utilisant comme primitive sous-jacente une fonction de hachage, on a besoin de formaliser la sécurité des fonctions de hachage pour pouvoir la prendre en compte dans la preuve de sécurité. Usuellement, ces preuves de sécurité sont des preuves par réduction. Elles énoncent que s'il existe un attaquant capable de remettre en cause la sécurité du protocole, alors il existe un attaquant capable de remettre en cause la sécurité d'une primitive sous-jacente. [121] détaille un problème lié à la formalisation de la résistance en collision des fonctions de hachage, primitive n'utilisant pas de clé. En effet, du fait de la taille des ensembles de départ et d'arrivée d'une fonction de hachage, l'existence d'une collision est assurée. Il **existe** donc un attaquant capable de produire une collision, un tel attaquant possédant dans son code une collision de la fonction de hachage. Les preuves de sécurité réduisant la remise en cause de la sécurité d'un protocole à l'existence d'un adversaire produisant une collision pour une fonction de hachage n'apporte donc pas de garantie de sécurité. Les cryptographes ont proposés deux méthodes pour lever cette difficulté.

Méthode 1 : Ajout d'une clé. Afin de faciliter la formalisation des propriétés de sécurité, Damgård considère des fonctions de hachage ayant une entrée supplémentaire correspondant à une clé [52]. L'objectif de l'attaquant cherchant à remettre en cause la résistance en collision d'une fonction de hachage est alors, **étant donné** une clé K, de trouver $M \neq M'$ tel que $h(K, M) = h(K, M')$. L'attaquant ne maîtrisant pas le choix de la clé K, l'espace des clés étant supposé suffisamment grand et la taille du code de l'attaquant étant limitée par son temps d'exécution, le code d'un attaquant efficace ne peut contenir une collision pour toute clé possible. La portée de cette première méthode de formalisation est cependant discutable, dans la mesure où les fonctions de hachage sont généralement définies sans clé. Quelques méthodes classiques permettent de transformer une fonction de hachage sans clé en fonction de hachage avec clé, comme par exemple l'utilisation d'un mode tel que HMAC [88]. Cependant l'usage de fonctions de hachage avec clé reste limité dans les contextes requérant la résistance en collision, notamment dans celui de la signature électronique [99][Chapitre 11].

Méthode 2 : Preuve constructive. Une deuxième méthode permettant de formaliser la résistance en collision des fonctions de hachage a été proposée par Rogaway [121] et consiste à imposer une contrainte supplémentaire des preuves de sécurité faisant reposer la sécurité d'un schéma sur la résistance en collision d'une fonction de hachage : la preuve doit construire explicitement la réduction. Ainsi un attaquant contre le schéma cryptographique pourra être utilisé pour construire un algorithme permettant de calculer une collision pour la fonction de hachage sous-jacente. Backes et Unruh montrent dans [9] l'existence de schémas cryptographiques reposant sur une fonction de hachage pouvant être prouvée sûre de manière existentielle, mais non de manière constructive.

La formalisation des propriétés de résistance en préimage et en seconde préimage est plus naturelle. Le jeu de sécurité consiste à fournir à l'attaquant une valeur de l'image de la fonction de hachage ou un message de son domaine. L'attaquant doit alors produire un message (nouveau dans le cas de l'attaque en seconde préimage) ayant même image par la fonction de hachage. Ces notions de sécurité peuvent également être étendues dans le cas où on considère une fonction de hachage à clé. On peut donner plus de moyens à l'attaquant en lui laissant le choix du message ou de l'image à inverser, ou celui de la clé. [122] définit ces différentes notions de sécurité.

4.2.4 Autres propriétés de sécurité

Nous avons vu que la sécurité des fonctions de hachage peut s'évaluer vis-à-vis de trois propriétés classiques, résistance en collision, seconde préimage et préimage. D'autres notions de sécurité ont été définies ultérieurement. Elles cherchent à rapprocher la fonction de hachage d'une primitive idéalisée.

PRF. La première de ces notions, celle de fonction pseudo-aléatoire, cherche à comparer une fonction de hachage à clé à une famille de fonctions aléatoires. Cette notion permet de décrire la sécurité des fonctions de hachage avec clé et est utilisée quand on cherche à démontrer qu'une telle fonction constitue un bon algorithme d'authentification de message (MAC).

PRO. La deuxième de ces notions, celle de pseudo oracle aléatoire, cherche à comparer la fonction de hachage à une fonction aléatoire. Intuitivement, elle suppose qu'il n'est pas possible d'exhiber de relation entre entrées et sorties de la fonction de hachage autre que l'application de la fonction. Cette notion doit son succès à son utilisation pour prouver de nombreux schémas cryptographiques asymétriques reposant sur une fonction de hachage.

4.3 Historique

L'histoire de l'étude des fonctions de hachage peut être grossièrement répartie en trois périodes.

4.3.1 Avènement des fonctions de hachage

L'étude publique des fonctions de hachage démarre au milieu des années 1980 et une première période va être consacrée à définir les propriétés de sécurité attendues des fonctions de hachage et à chercher à définir des fonctions réalisant ces propriétés de sécurité. Cette période va voir l'émergence d'un principe de conception de fonctions de hachage qui va se généraliser et aboutir à la normalisation d'un grand nombre de fonctions de hachage similaires, notamment MD4 [117] et MD5 [118], SHA-0 [110] et SHA-1 [111]. Étant donné les besoins exprimés l'emploi de ces fonctions va se généraliser très rapidement.

Les principes de conception communs entre ces fonctions permet de décomposer leur construction en trois niveaux :

- le cœur de la fonction de hachage est une permutation paramétrée. Dans le cas des algorithmes des familles MD et SHA, des permutations paramétrées spécifiques sont définies. Elles ont pour particularité des tailles de clés et de blocs plus importantes que les tailles usuellement rencontrées pour les algorithmes de chiffrement par bloc. Par exemple pour la fonction SHA-256, la taille de clé est de 512 bits, la taille de bloc de 256 bits.
- un *mode de compression* permet de transformer cette permutation paramétrée en une *fonction de compression*, qui peut être vue comme une fonction de hachage traitant des messages de taille fixe.
- une *extension de domaine* permet de transformer cette fonction de compression en une fonction de hachage, pouvant traiter des messages de longueur arbitraire.

4.3.2 Cryptanalyses

À partir du milieu des années 1990, une activité de cryptanalyse de ces fonctions de hachage commence à se développer. Les analyses se concentrent principalement sur la résistance en collision de ces fonctions. Les premiers résultats marquants sont obtenus par Dobbertin qui calcule une collision pour MD4 [57] et une collision pour la fonction de compression de MD5 [58].

En 1998, Chabaud et Joux proposent une attaque contre SHA-0 [37]. Outre le fait d'expliquer pourquoi cette fonction a été corrigée en 1995, définissant ainsi la fonction SHA-1, cette attaque expose des principes généraux pour réaliser des attaques en collision sur des fonctions de hachage. L'idée centrale est de réaliser une attaque différentielle. Il s'agit d'identifier des différences entre deux messages susceptibles de suivre une propagation bien établie au cours des calculs et finissant par s'annuler à l'issue du calcul. La séquence des différences parcourues au cours du calcul est désignée sous le terme de *chemin différentiel*. Une fois un chemin différentiel découvert, on cherche une paire de messages pour laquelle le chemin différentiel est suivi lors des calculs, ce qui conduit à une collision.

Durant la compétition AES, qui s'est déroulée entre 1997 et 2001 avec pour objectif la normalisation d'un algorithme de chiffrement par bloc successeur du DES, l'activité de cryptanalyse des fonctions de hachage est quelque peu mise entre parenthèses. Il faut attendre les années 2003-2004 pour voir émerger de nouveaux résultats comme la réalisation du calcul d'une collision de la fonction SHA-0 [20]. L'année 2005 voit la publication par Wang, Yu et Yin d'attaques en collision contre les fonctions MD5 [138] et SHA-1 [137]. Ces nouvelles attaques reprennent le principe des attaques différentielles, mais introduisent de nouvelles techniques pour accélérer la phase de recherche de paires de messages conformes au chemin différentiel identifié, comme les techniques de *modification de messages* et les techniques de *bits neutres*. La construction de chemins différentiels est également adaptée à ces techniques.

4.3.3 SHA-3

Suite à la publication de ces résultats, la majorité des fonctions de hachage normalisées a été mise en défaut. Seule la famille de fonctions de hachage SHA-2 semble résister aux attaques. Cependant, cette famille partage les principes de conception de la famille MD4 et sa sécurité est entachée de suspicion. Le NIST décide de lancer une compétition permettant de définir un nouveau standard de fonction de hachage susceptible de remplacer SHA-2 en cas de défaut.

Le lancement de cette compétition fait l'objet de travaux préparatoires afin de déterminer un cahier des charges correspondant aux attentes de l'industrie et au point de vue de la communauté académique sur ce qui constitue une bonne fonction de hachage. Une liste d'exigences est publiée fin 2007 [112]. Les principaux points notables sont les suivants :

- fonctionnellement, les candidats doivent présenter la même interface que la famille SHA-2 et pouvoir être utilisés dans les mêmes contextes d'emploi ;
- d'un point de vue sécurité, pour la variante ayant des sorties sur ℓ_h bits, on attend une résistance en collision en $\mathcal{O}\left(2^{\frac{\ell_h}{2}}\right)$ et une résistance en (seconde) préimage en $\mathcal{O}\left(2^{\ell_h}\right)$. La résistance aux attaques en extension de messages (cf section 4.4.2), est également requise.
- d'un point de vue performance, même si aucune exigence n'est formulée, une attente de voir des gains notables de performance est mentionnée.

À l'instar de la compétion AES et du projet eSTREAM, le déroulement de la compétition se découpe en phases successives au cours desquelles les candidats encore en course sont analysés. Le passage d'une phase à la phase suivante opère une sélection des candidats les plus prometteurs afin de concentrer les efforts des cryptanalystes et des implémenteurs sur ces candidats, dans

le but de faire émerger un bon niveau de confiance. Pour une bibliographie complète sur les candidats, les analyses de sécurité qui s'y rapportent et les attaques dont ils ont fait l'objet, on se réfère au SHA-3 Zoo [61].

Phase initiale. L'appel à candidature, se clôturant le 31 octobre 2008, a vu la soumission de 64 candidats, dont 51 ont été retenus pour prendre part à la compétition. De nombreuses attaques pratiques sont publiées pendant cette première phase. À l'issue de cette première phase, 14 candidats sont retenus pour participer à la phase suivante. La sélection s'est opérée principalement sur des critères de sécurité, élimination des candidats « cassés », et de performances, élimination des candidats notablement plus lents que les algorithmes de la famille SHA-2.

Deuxième phase. L'analyse des candidats restants, parfois désignés sous le terme de demi-finalistes, se fait plus subtile. Si quelques attaques théoriques sont encore découvertes [65, 31], l'essentiel des résultats pendant cette phase se concentrent sur des versions réduites des algorithmes ou l'identification de propriétés non-idéales des primitives sous-jacentes des algorithmes. Au terme de cette phase, cinq algorithmes sont retenus pour la phase finale. La sélection peut être vue comme conservatrice dans le sens où :

- Les algorithmes concourant peuvent être classés en différentes catégories selon leurs principes de conception, leur source de non linéarité, etc. La sélection du NIST ne prend pas parti pour l'un de ces principes de construction. Au contraire, elle conserve au moins un algorithme de chaque famille afin que tous les principes de conception continuent à être étudiés.
- En-dehors des considérations de performances, les algorithmes retenus sont ceux disposant des meilleures garanties de sécurité. En particulier, l'existence de propriétés non idéales sur les composants internes des fonctions de hachage a été considérée comme rédhibitoire. Au contraire, les fonctions de hachage disposant d'une bonne marge de sécurité ont été privilégiées. Cette marge de sécurité peut être estimée de la manière suivante : chaque fonction utilise une primitive sous-jacente (permutation, permutation paramétrée, etc) obtenue par itération d'une étape élémentaire ; on compare le nombre d'itérations pour lequel il est possible d'identifier une propriété non idéale sur la primitive sous-jacente et le nombre d'itérations utilisé dans la fonction de hachage. En règle générale, plus cet écart est important, plus on dispose d'une marge de sécurité importante.

On peut encore noter à propos de cette sélection que si les performances affichées par les candidats ont permis de les départager entre eux, elles n'ont pas favorisé quelques candidats plus rapides présentant moins de garanties. De plus les algorithmes retenus ont des débits du même ordre de grandeur que SHA-2 et sont plutôt moins rapides sur des plateforme à bas coût (processeur 8 ou 32 bits, cf [59]).

Phase finale. La phase finale est en cours actuellement. Elle devrait s'achever d'ici l'année prochaine, la publication du standard SHA-3 étant prévue pour 2012. Les algorithmes finalistes sont Blake, Grøstl, Keccak, JH et Skein.

Outre l'identification d'un recours au cas où des défauts viendraient à être identifiés contre la famille SHA-2, on peut d'ores et déjà affirmer que la compétition SHA-3 a permis des avancées notables de l'état de l'art du domaine des fonctions de hachage :

- Sur le plan des principes de conception, des familles ont été identifiées et des principes d'attaques spécifiques développés ;
- Sur le plan des modes opératoires, extensions de domaine, modes de compression, de nouvelles propositions ont été formulées. On peut citer à titre d'exemple :
  - une nouvelle extension de domaine où la méthode de compression est réduite à sa plus simple expression (ou exclusif) et où la fonction de tour est une permutation : les fonctions éponges [19], dont Keccak est un exemple ;
  - une extension de domaine basée sur un tweakable-block-cipher [91], mode proposé dans la définition de l'algorithme Skein.

4.4 Extension de domaine

Comme nous l'avons vu dans la section précédente, une fonction de hachage est une primitive cryptographique susceptible de traiter des messages très longs. Afin de faciliter la construction et l'évaluation de telles primitives cryptographiques, on procède usuellement en définissant une primitive ayant des entrées de taille fixe relativement petite et en la mettant en œuvre dans un mode opératoire. Un tel mode opératoire, prolongeant le domaine de définition de la primitive ayant des entrées de taille fixe est appelé *extension de domaine*. Historiquement, une extension de domaine relativement simple a émergé dès le début des années 1990 et son usage s'est répandu dans les fonctions de hachage définies à partir de cette période. On commence par présenter cette extension de domaine, attribuée à Merkle [100] et Damgård [52]. Des limitations découvertes sur cette extension de domaine ont entraîné un effort de recherche afin de déterminer les propriétés que doit satisfaire une bonne extension de domaine destinée à la construction d'une fonction de hachage et de déterminer de nouvelles constructions structurellement plus robustes. On réalise un panorama de ces propriétés et de ces constructions dans les sections suivantes.

4.4.1 Extension de domaine de Merkle-Damgård

4.4.1.1 Définition et propriétés de sécurité

L'extension de domaine de Merkle-Damgård a été adoptée par les fonctions de hachage de la famille de MD4 pour transformer une donnée de longueur finie arbitraire [1] en une empreinte de taille finie fixe. Le principe de Merkle-Damgård est similaire au principe du chiffrement par bloc : l'idée est de découper le message à hacher en blocs de taille fixe, et de hacher itérativement ces blocs au moyen d'une fonction de compression. On peut voir cette fonction de compression comme une fonction de hachage dont l'entrée est de taille fixe.

Une fonction de hachage h construite sur le principe de Merkle-Damgård se décrit au moyen d'un état de taille ℓ_h bits, une valeur initiale pour cet état notée IV et d'une fonction de compression $F : \{0,1\}^{\ell_h} \times \{0,1\}^m \to \{0,1\}^{\ell_h}$. Le principe de calcul de la fonction de hachage est alors le suivant :

- on commence par découper le message à hacher M en blocs de m bits : $M_0, \ldots, M_{k-1}$,
- on initialise l'état de la fonction de hachage avec la valeur initiale : $H_0 = IV$,
- pour $0 \leq i \leq k-1$, on calcule itérativement les hachés intermédiaires $H_{i+1} = F(H_i, M_i)$,
- la sortie de la fonction de hachage est la valeur finale de l'état : H_k.

Pour pouvoir utiliser ce schéma, il faut définir un padding, c'est-à-dire un moyen de transformer le message en un message dont la taille est multiple de la taille d'un bloc. Le choix de ce

1. en pratique la longueur de la donnée peut être limitée par une grande constante, 2^{64} dans le cas de MD4 par exemple. Cette limitation est introduite par le padding.

padding n'est pas anodin pour la sécurité (cf [99], commentaire de l'algorithme 9.25 et remarque 9.32). Un padding couramment employé consiste à concaténer au message un bit 1, suivi d'un nombre variable de bits à 0. Puis on concatène au message ainsi obtenu la valeur de la taille en bits du message initial. Cette taille est codée sur un nombre fixe de bits, 64 pour MD4. Le nombre de 0 ajoutés est choisi de telle sorte que le message final obtenu ait une taille multiple de la taille d'un bloc et soit le plus petit possible. Ce schéma de padding porte le nom de *MD-strengthening*.

La partie grisée du dernier bloc de message dans la figure 4.1 représente le padding. Avec le padding décrit ci-dessus, une partie de la chaîne de caractères ajoutée au message initial peut également être présente dans l'avant-dernier bloc, contrairement à ce que la figure semble impliquer.

Les fonctions de compression pouvant être considérées comme des fonctions de hachage réduites, il est naturel d'étendre les objectifs de sécurité des fonctions de hachage à ces fonctions de compression.

Définition 25 *Une collision de la fonction de compression F est un quadruplet $(IV, M, IV', M') \in \{0,1\}^{\ell_h} \times \{0,1\}^m \times \{0,1\}^{\ell_h} \times \{0,1\}^m, (IV, M) \neq (IV', M')$ tel que*

$$F(IV, M) = F(IV', M').$$

De manière formelle, on peut démontrer le théorème suivant dû à Merkle et Damgård :

Théorème 7 *Soit F une fonction de compression et h une fonction de hachage construite sur le principe de Merkle-Damgård utilisant F et utilisant le padding décrit plus haut. Alors on a :*

$$F \text{ est résistante en collision } \Rightarrow h \text{ est résistante en collision.}$$

Si une fonction de hachage utilise une "bonne" fonction de compression, il sera difficile de trouver des collisions sur la fonction de hachage. On ne sait cependant pas montrer l'équivalence des deux propositions. Ainsi, une collision pour la fonction de compression d'une fonction de hachage itérée ne remet pas pour autant en cause la résistance en collision de la fonction de hachage. Cependant, d'un point de vue académique, on rejette les fonctions de hachage contruites suivant le principe de Merkle-Damgård reposant sur des fonctions de compression pour lesquelles on peut trouver des collisions plus rapidement que par une attaque générique. L'importance de ce théorème découle du fait qu'il permette au concepteur d'une fonction de hachage de se concentrer sur la conception de la fonction de compression. L'extension de Merkle-Damgård lui fournit une manière naturelle de transformer une fonction de compression résistante en collision en une fonction de hachage résistante en collision.

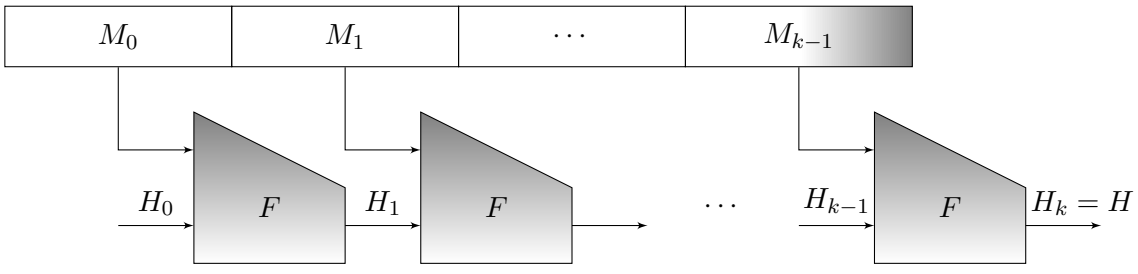


FIGURE 4.1 – Construction d'une fonction de hachage itérée

D'autre part, la notion de préimage pour une fonction de hachage se transpose également aux fonctions de compression :

Définition 26 *Soit $H_1 \in \{0,1\}^{\ell_h}$. Une préimage de H_1 par la fonction de compression F est un couple $(H_0, M) \in \{0,1\}^{\ell_h} \times \{0,1\}^m$ tel que*

$$F(H_0, M) = H_1.$$

De plus, on a le résultat de réduction de sécurité suivant :

Théorème 8 *Soit F une fonction de compression et h une fonction de hachage construite sur le principe de Merkle-Damgård utilisant F. Alors on a :*

$$F \text{ est résistante en préimage} \Leftrightarrow h \text{ est résistante en préimage.}$$

En d'autres termes, on peut attaquer la résistance en préimage d'une fonction de hachage itérée si on connait une attaque en préimage sur la fonction de compression (cf [116], Proposition 2.2).

4.4.1.2 Propriétés non idéales de l'extension de domaine de Merkle-Damgård

Si l'itération d'une fonction de compression permet de définir des fonctions de hachage utilisables en pratique, elle structure fortement les fonctions de hachage obtenues. Quelques résultats ont été obtenus sur le mode de Merkle-Damgård, résultats reposant notamment sur cette structure itérative. Elle induit un comportement qui distingue les fonctions construites suivant ce mode de fonctions aléatoires.

Nous présentons dans cette section les attaques tirant parti des modes opératoires utilisés par les fonctions de hachage usuelles. Ces attaques ne font intervenir que les propriétés des modes opératoires et ont des complexités meilleures que les attaques génériques contre les fonctions de hachage. Cependant, elles sont elles-mêmes génériques en un certain sens, car elles n'exploitent pas de faiblesse des primitives utilisées par les modes.

4.4.2 Attaque par extension de message

Une première propriété non idéale de l'extension de domaine de Merkle-Damgård est la possibilité de calculer à partir du haché d'un message M et de la longueur de ce message le haché de tout message $\mathsf{pad}(M)\|S$, où S est un suffixe quelconque, et ce sans connaître le message initial M. En effet, au cours du calcul du haché de $\mathsf{pad}(M)\|S$, M influence uniquement les premières itérations du calcul, et la valeur de $h(M)$ suffit à poursuivre le calcul et traiter les blocs de messages provenant de S. La longueur du message M intervient pour le calcul du padding à appliquer. Cette propriété empêche d'utiliser simplement une fonction de hachage mettant en œuvre le principe de Merkle-Damgård pour calculer des MACs en introduisant la clé comme valeur d'IV, car le MAC d'un message M permet dans ce cas de calculer les MACs de messages dont M est préfixe.

4.4.3 Multicollisions

Une autre propriété découverte par Joux sur les fonctions de hachage itérées est leur relative faiblesse vis-à-vis des multicollisions [79].

Définition 27 *Soit h une fonction de hachage, $k \in \mathbb{N}, k > 1$. Une k-multicollision de h est un k-uplet de messages distincts $(M_1, \ldots, M_k) \in (\{0,1\}^*)^k$, tel que*

$$\forall (i,j), h(M_i) = h(M_j).$$

Pour une fonction aléatoire, une variante du paradoxe des anniversaires permet de trouver une k-multicollision de h en $\mathcal{O}\left(2^{\frac{k-1}{k}\ell_h}\right)$ requêtes à h. Pour une fonction de hachage utilisant le mode de Merkle-Damgård, le nombre de requêtes nécessaires est beaucoup moins grand :

Proposition 20 *Soit $h : \{0,1\}^* \to \{0,1\}^{\ell_h}$ une fonction de hachage itérée. On peut trouver une 2^k-multicollision de h en $\mathcal{O}\left(k2^{\frac{\ell_h}{2}}\right)$ requêtes à h.*

Pour une fonction de hachage itérée, la complexité du calcul d'une 2^k-multicollision est donc très inférieure à la complexité de ce calcul pour une fonction de hachage idéale.

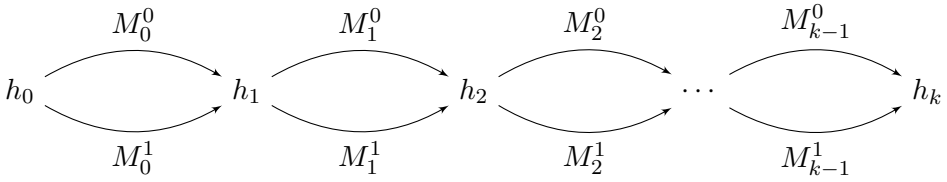


FIGURE 4.2 – Construction d'une 2^k-multicollision

La figure 4.2 donne une idée de la démonstration de la proposition 20. Pour construire une 2^k-multicollision sur la fonction de hachage, on construit k collisions pour la fonction de compression, en prenant pour IV la valeur précédente obtenue par la fonction de compression (complexité $\mathcal{O}\left(k2^{\frac{\ell_h}{2}}\right)$). Les 2^k messages obtenus en choisissant pour chaque bloc de message i l'un des deux blocs qui collisionnent à l'étape i, $M_0^{i_0}||M_1^{i_1}||\ldots||M_{k-1}^{i_{k-1}}, (i_0, \ldots, i_{k-1}) \in \{0,1\}^k$, collisionnent et sont distincts.

Depuis [79], d'autres publications ont exploité la possibilité d'obtenir des multicollisions pour attaquer les fonctions de hachage itérées. Les résultats les plus significatifs sont rappelés ci-dessous :

- Attaques sur des fonctions concaténées, variante initiale de Joux [79], extension par Hoch et Shamir [75],
- Accélération de l'attaque en seconde préimage de messages longs par Kelsey et Schneier [82],
- Attaque Nostradamus contre les schémas d'engagement [81].

4.4.4 Sécurité d'une extension de domaine

Dans la section précédente on vient de voir que l'attrait de l'extension de domaine de Merkle-Damgård provient notamment de sa capacité à transposer, préserver quelques propriétés de la fonction de compression sous-jacente à la fonction de hachage obtenue. [1, 12, 43] montrent que d'autres propriétés de sécurité ne sont cependant pas préservées par le mode de Merke-Damgård. En conjonction avec ses « faiblesses » identifiées, ceci a conduit les cryptographes à proposer des extensions de domaine plus robustes, dont on donne ici quelques exemples, en s'efforçant pour chacune d'elles d'exposer leurs motivations.

wide-pipe MD/Chop-MD. Une première idée proposée dès 2004 par Lucks [92] est l'utilisation d'un état interne plus grand que la taille de sortie de la fonction de hachage $w > \ell_h$. Dans une première phase, l'extension de domaine de Merkle-Damgård est appliquée avec une variable de chaînage de w bits. Dans une phase de finalisation la variable de chaînage sur w bits est transformée en une valeur de ℓ_h bits, soit à travers une fonction de compression, soit dans le cas de Chop-MD, par simple troncation. Coron et al. montrent que Chop-MD est indifférentiable d'un oracle aléatoire [43].

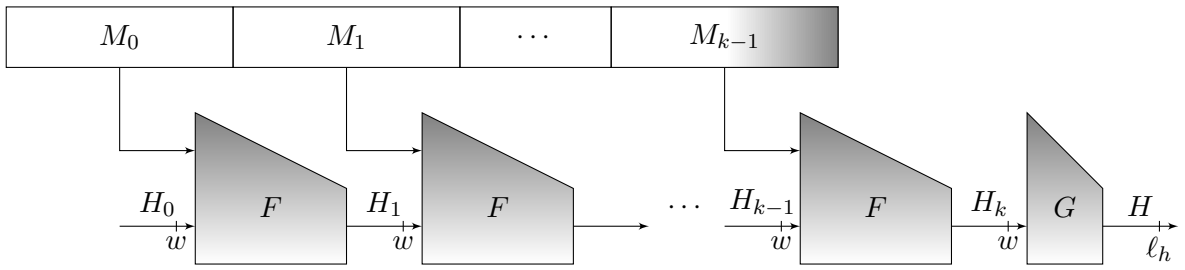


FIGURE 4.3 – Extension de domaine wide-pipe/Chop-MD

HAIFA. Biham et Dunkelman proposent en 2006 une extension de domaine, HAIFA [22], basée sur l'extension de domaine de Merkle-Damgård. Le but recherché est d'améliorer la résistance du mode de Merkle-Damgård contre les attaques qui l'affectent. La principale différence consiste en l'ajout d'entrées additionnelles à la fonction de compression :

- un sel, définissant de fait une fonction de hachage avec clé. Cet ajout permet notamment d'appliquer formellement les notions de sécurité définies sur les fonctions de hachage avec clé et d'éviter les compromis temps-mémoire.
- un compteur indiquant le nombre de bits de message déjà traités.

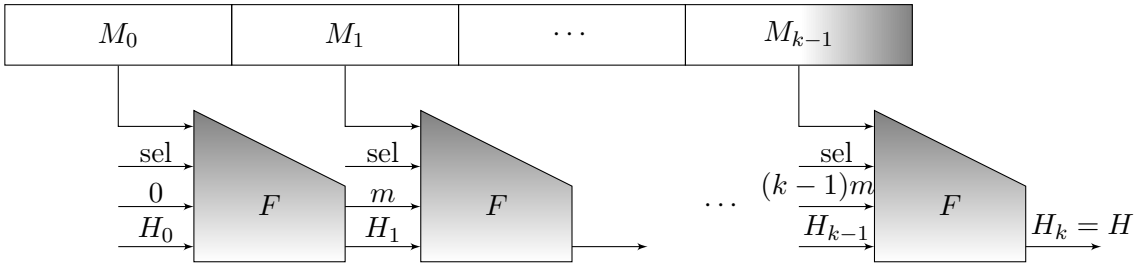


FIGURE 4.4 – Extension de domaine Haifa

EMD. EMD [12], pour *Enveloped Merkle-Damgård*, est une extension de domaine construite sur le même principe que le mode HMAC. Dans une première phase, le mode de Merkle-Damgård est appliqué. Le traitement du dernier bloc de message diffère. Sa concaténation avec la sortie du mode de Merkle-Damgård est utilisée comme bloc de message d'un dernier appel à la fonction de compression. [12] démontre que lorsque le padding utilisé est le MD-strengthening, ce mode préserve résistance en collision, ainsi que les propriétés PRF et PRO.

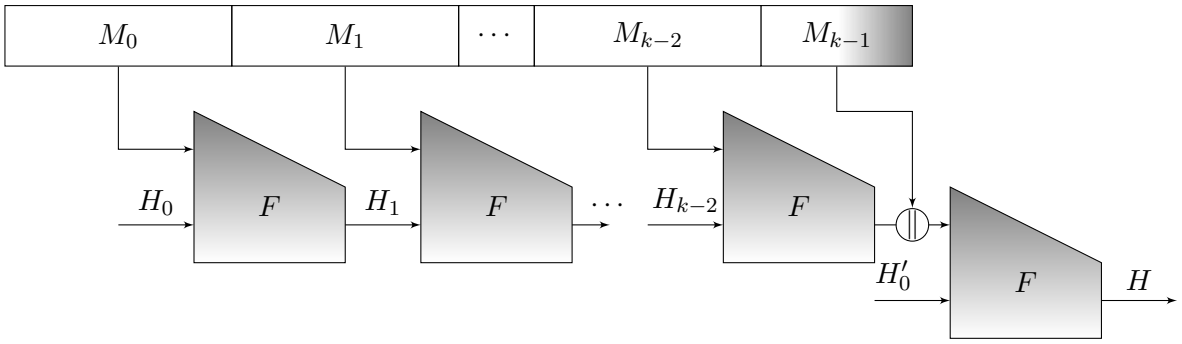


FIGURE 4.5 – Extension de domaine EMD

Fonctions éponges. On termine cette présentation en évoquant les fonctions éponges [19]. Si l'extension de domaine définie est toujours itérative, *i.e.* une fonction de tour traite bloc après bloc le message à hacher, cette construction diffère des autres en s'appuyant sur une primitive sous-jacente de taille égale à la taille de la variable de chaînage. Afin d'obtenir une fonction de tour réalisant une « compression », chaque bloc de message est inséré de manière élémentaire, par addition à une partie de l'état interne. La production d'un haché diffère également : l'extraction se fait bloc de message par bloc de message avec application de la primitive sous-jacente entre deux extractions. Les blocs de message extraits sont concaténés et le nombre de blocs extraits est tel qu'une valeur de ℓ_h bits est ainsi produite.

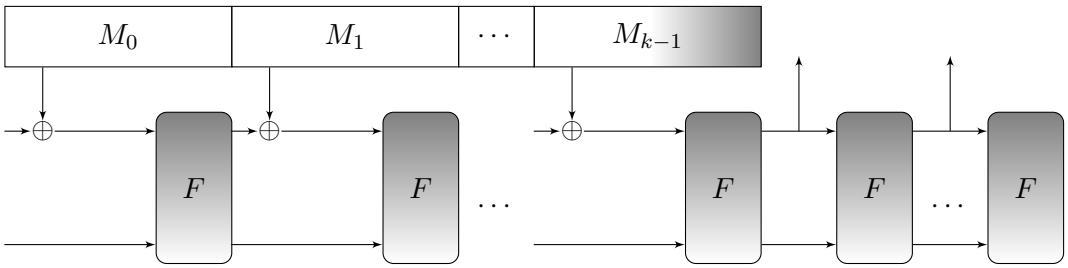


FIGURE 4.6 – Extension de domaine fonction éponge

Chapitre 5

Preuve d'indifférentiabilité dans le cas idéal

Sommaire

Nous présentons dans ce chapitre une preuve d'indifférentiabilité d'une extension de domaine générique. Cette extension de domaine généralise de nombreuses extensions de domaine, notamment Chop-MD et celle développée dans le cadre de la définition de la fonction Shabal. On dérive de cette preuve des bornes sur l'avantage d'un distingueur cherchant à distinguer une fonction idéale construite sur ce mode générique et un oracle aléatoire, pour différents encodages.

5.1 Un mode opératoire générique

Nous introduisons en Figure 5.1 une extension de domaine générique. Ce mode opératoire permet d'utiliser une primitive dont les entrées sont de taille finie et un encodage de message non-ambigu pour construire une fonction de hachage définie sur l'ensemble des messages de longueur finie arbitraire.

Ce mode est un mode itératif : à chaque tour, un bloc de message de ℓ_m bits est traité et un état interne[1] de s bits est mis à jour. Nous supposons donné un encodage de message non-ambigu, ou padding. Une fonction d'encodage, pad, permet de transformer un message de longueur finie arbitraire, $\mathcal{M} \in \{0,1\}^*$ en un message décomposable en un nombre entier de blocs, dit message paddé, $\mathsf{pad}(\mathcal{M}) \in \left(\{0,1\}^{\ell_m}\right)^+$. La fonction de décodage associée est notée unpad. Une fois le message paddé traité par la construction itérative, la valeur de haché, de longueur ℓ_h bits est définie comme une partie de l'état interne final. En d'autres termes, la fonction de finalisation retenue est une troncature de l'état interne final.

1. La terminologie *état interne* est ici préférée à celle de *variable de chaînage* traditionnellement employée pour la construction de Merkle-Damgård

Nous démontrons dans ce chapitre que la sécurité de cette construction dépend des propriétés du schéma de padding retenu. Après avoir considéré un schéma de padding arbitraire nous nous intéresserons au cas des padding sans préfixe.

La fonction utilisée pour mettre à jour l'état interne à chaque tour est décomposée en deux opérations élémentaires. Tout d'abord, une insertion de bloc de message $\mathsf{Insert}[M]$ consiste à appliquer à l'état interne une transformation inversible dépendant du bloc de message courant $M \in \{0,1\}^{\ell_m}$. Dans un deuxième temps, la primitive $\mathcal{F}$ sur laquelle repose la construction est appliquée. L'entrée de $\mathcal{F}$ est constituée de l'état interne et du bloc de message courant. La sortie de $\mathcal{F}$ met à jour tout ou partie de l'état interne, dont l'intégralité de la partie de l'état interne utilisée comme valeur de haché de l'état interne final. On peut donc décomposer l'état interne en trois parties disctinctes :

- la partie de l'état interne mise à jour par $\mathcal{F}$ mais non extraite à la fin du calcul de la construction est notée A et sa taille en bit est notée ℓ_a ;
- la partie de l'état interne mise à jour par $\mathcal{F}$ et extraite à la fin du calcul de la construction est notée B et sa taille en bit est notée ℓ_h ;
- la partie de l'état interne non mise à jour par $\mathcal{F}$ est notée C et sa longueur est notée ℓ_c.

La taille de l'état interne s vérifie $s = \ell_a + \ell_h + \ell_c$, et n la taille de sortie de la fonction $\mathcal{F}$ vérifie $n = \ell_a + \ell_h$. Pour un état interne x, on note $x|_A, x|_B, x|_C$ les parties A, B, C de x.

On note

$$\begin{array}{rcl} \mathcal{F} : \{0,1\}^{\ell_m} \times \{0,1\}^s & \to & \{0,1\}^n \\ (M, A, B, C) & \to & \mathcal{F}_{M,C}(A, B) = (A', B'). \end{array}$$

Lorsque $\mathcal{F}$ est modélisée par un chiffrement par bloc idéal, (M, C) joue le rôle de clé.

On note x_i la valeur de l'état interne après l'insertion de i blocs de message et M_i le bloc de message dont l'insertion produit l'état x_i. La valeur initiale de l'état interne est une constante $x_0 = (A_0, B_0, C_0)$. La construction est alors définie par la fonction de mise à jour suivante :

$$x_{i+1} = \mathrm{Round}[M_i](x_i) = (\mathcal{F}(M_i, \mathsf{Insert}[M_i](x_i)), \mathsf{Insert}[M_i](x_i)|_C)\,.$$

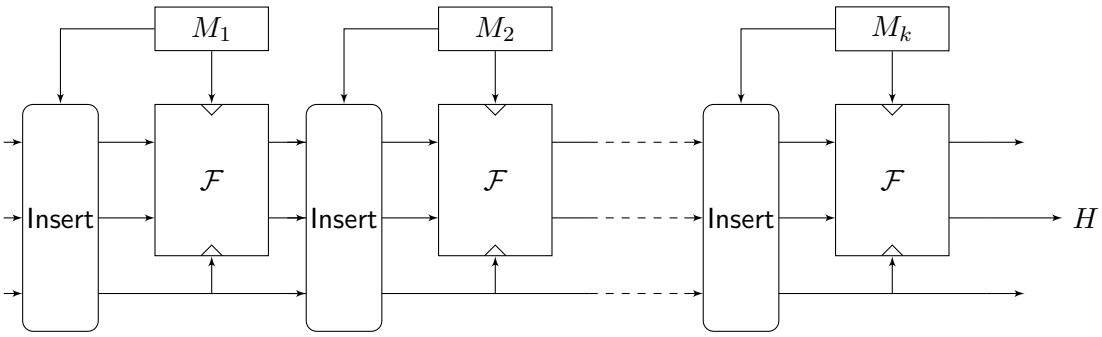


FIGURE 5.1 – Une extension de domaine générique.

Cette construction de fonction de hachage généralise les modes sous-jacents de plusieurs fonctions de hachage lorsque la fonction d'insertion varie, dont les modes suivants :

Chop-MD. Chop-MD est obtenu en choisissant pour fonction d'insertion la fonction identité

$$\mathsf{Insert}[M](A, B, C) = (A, B, C),$$

pour $\ell_c = 0$ (*i.e.* la primitive sous-jacente affecte l'intégralité de l'état interne). Dans le cas où l'état interne est le double de la taille des hachés produits (*i.e.* $n = s = 2\ell_h$), ce mode correspond à la construction dite *wide-pipe* [92].

Shabal. Le mode opératoire utilisé dans Shabal-512 peut être obtenu en utilisant la fonction d'insertion définie par

$$\mathsf{Insert}[M](A, B, C) = (A, C \boxplus M, B \boxminus M),$$

avec $\ell_h = \ell_c = \ell_m = 512$. En effet, dans la représentation originale du mode opératoire de Shabal, tel que décrit en Figure 5.2, l'insertion de message entre deux appels consécutifs à la permutation paramétrée $\mathcal{P}$ fait intervenir deux blocs de messages différents, puisque le bloc de message correspondant au premier appel à $\mathcal{P}$ est soustrait à C avant que le bloc de message correspondant au deuxième appel à $\mathcal{P}$ soit ajouté à B. Bien qu'il s'agisse là de le représentation naturelle de ce mode, particulièrement du point de vue de l'implémentation, elle alourdit les preuves de sécurité.

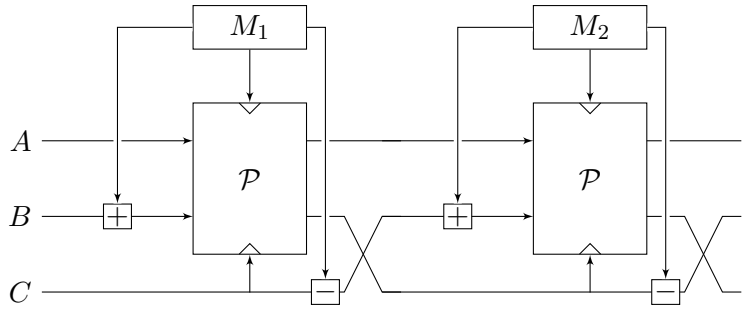


FIGURE 5.2 – Extension de domaine de Shabal.

On peut cependant ramener formellement le mode de Shabal à une instantiation du mode générique, c'est à dire tel qu'un bloc de message ne modifie pas l'état après l'appel à la primitive sous-jacente. Pour cela, il suffit de soustraire le bloc de message à C *avant* d'appliquer la primitive. Afin de conserver un mode opératoire équivalent, on remplace la permutation paramétrée originale $\mathcal{P}$ par la permutation paramétrée $\mathcal{Q}$ obtenue en composant l'addition de C et de M à $\mathcal{P}$:

$$\begin{array}{rcl} \mathcal{Q} : \{0,1\}^{\ell_m} \times \{0,1\}^s & \rightarrow & \{0,1\}^n \\ (M, A, B, C) & \rightarrow & \mathcal{P}(M, A, B, C \boxplus M). \end{array}$$

Cette représentation est donnée en Figure 5.3.

Pour les autres versions de Shabal, le mode opératoire correspond encore à une instanciation du mode générique. Cependant, dans la mesure où la partie B de l'état correspond pour le mode générique à la partie de l'état interne de laquelle est extraite la valeur de haché finale, la décomposition de l'état interne en parties A, B et C de l'état interne ne correspond pas aux registres A, B et C décrits dans la documentation de Shabal. Afin de faire correspondre les deux descriptions, les registre B, C et M de Shabal sont décomposés en deux parties de taille $\ell_m - \ell_h$ et ℓ_h bits, respectivement $(B_1, B_2), (C_1, C_2)$ et (M_1, M_2). La partie A de l'état interne de Shabal est obtenue en considérant le registre A (de taille s_a bits) et B_1. La fonction d'insertion s'écrit alors :

$$\begin{array}{rcl} \{0,1\}^{s_a+\ell_m-\ell_h} \times \{0,1\}^{\ell_h} \times \{0,1\}^{\ell_m} & \rightarrow & \{0,1\}^{s_a+\ell_m-\ell_h} \times \{0,1\}^{\ell_h} \times \{0,1\}^{\ell_m} \\ (A, B_1; B_2; C_1, C_2) & \rightarrow & (A, C_1 \boxplus M_1; C_2 \boxplus M_2; B_1 \boxminus M_1, B_2 \boxminus M_2). \end{array}$$

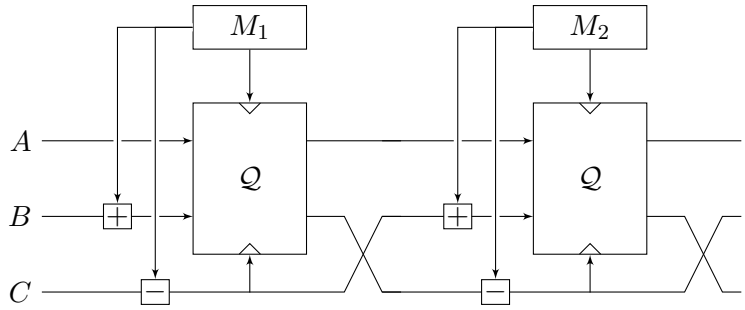


FIGURE 5.3 – Représentation équivalente de l'extension de domaine de Shabal.

5.2 Preuve d'indifférentiabilité dans le cas idéal

Dans cette section, nous donnons une preuve que le mode opératoire générique décrit dans la section précédente est indifférentiable d'un oracle aléatoire lorsque la primitive sous-jacente est idéale, *i.e.* lorsqu'elle est remplacée par un oracle aléatoire ou une permutation paramétrée idéale. Cette preuve, dont le résultat est énoncé dans les théorèmes 9 et 10 améliore les bornes précédement connues aussi bien pour Chop-MD [38] que pour Shabal [34]. Les bornes données pour le cas où l'encodage de message est arbitraire sont améliorées pour un encodage de message sans préfixe.

5.2.1 Concept d'indifférentiabilité

Le concept d'indifférentiabilité a été développé en section 4.1.4. Il spécifie un jeu de sécurité qui se déroule entre un système d'oracles S et un distingueur $\mathcal{D}$. S peut contenir plusieurs composants susceptibles d'interagir entre eux. Ainsi notre mode générique peut être modélisé par une primitive interne $\mathcal{F}$ et une construction $\mathcal{C}^{\mathcal{F}}$ faisant appel à cette primitive interne. La construction $\mathcal{C}$ est dite indifférentiable d'un oracle aléatoire si le système $\mathcal{Q} = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ peut être remplacé par un second système d'oracle $\mathcal{Q}' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$ ayant une interface identique avec le distingueur et tel que le distingueur ne puisse déterminer correctement le système avec lequel il interagit qu'avec une probabilité faible lorsque le nombre de ces interactions est borné (voir Figure 5.4). Ici $\mathcal{H}$ est un oracle aléatoire et $\mathcal{S}$ est un simulateur qui doit se comporter comme $\mathcal{F}$. Dans le cas où la primitive interne est une permutation paramétrée, $\mathcal{S}$ correspond à la fois à un simulateur de $\mathcal{F}$ et de $\mathcal{F}^{-1}$.

Au cours de ses interactions avec le système $\mathcal{Q}$ ou $\mathcal{Q}'$, le distingueur réalise des *requêtes à gauche* à $\mathcal{L}$, représentant soit $\mathcal{C}^{\mathcal{F}}$ soit $\mathcal{H}$, et des *requêtes à droite* à $\mathcal{R}$, représentant soit $\mathcal{F}$ soit $\mathcal{S}^{\mathcal{H}}$. Nous notons N le nombre total de requêtes à droite, *i.e.* le nombre de requêtes reçues par $\mathcal{F}$ lorsque $\mathcal{D}$ interagit avec S indépendamment de leur origine, soit le distingueur $\mathcal{D}$, soit la construction $\mathcal{C}^{\mathcal{F}}$. On définit l'avantage du distingueur $\mathcal{D}$ par

$$\mathsf{Adv}(\mathcal{D}) = \left| \Pr\left[\mathcal{D}^{\mathcal{Q}} = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{Q}'} = 1\right] \right|,$$

où les probabilités sont prises sur les jetons aléatoires des parties. Dans la preuve $\mathsf{Adv}(\mathcal{D})$ est borné en fonction de N. Chaque requête fournit en effet de l'information au distingueur, information susceptible de lui permettre d'identifier le système d'oracles auquel il est confronté. La preuve d'indifférentiabilité consiste par conséquent à construire un simulateur approprié pour $\mathcal{F}$ (ou $(\mathcal{F}, \mathcal{F}^{-1})$ dans le cas d'une permutation paramétrée) et à donner une borne supérieure à l'avantage d'un distingueur entre $\mathcal{Q}$ et $\mathcal{Q}'$.

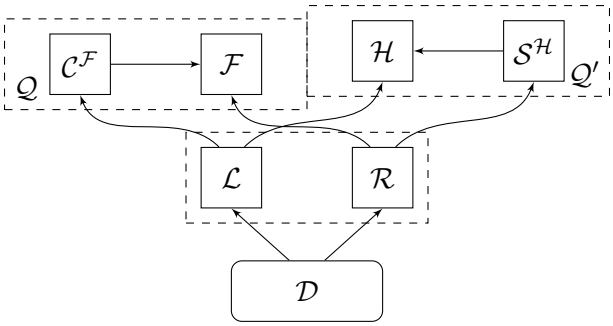


FIGURE 5.4 – La construction $\mathcal{C}^{\mathcal{F}}$ a accès à un oracle $\mathcal{F}$. Le simulateur $\mathcal{S}^{\mathcal{H}}$ a accès à l'oracle aléatoire $\mathcal{H}$. Le distingueur interagit soit avec $\mathcal{Q} = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, soit avec $\mathcal{Q}' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$ et doit les distinguer.

La vue du distingueur est donnée par la liste de ses requêtes au système d'oracles, associées à leur réponse. Dans la preuve qui suit,

- les requêtes à $\mathcal{L}$ sont des éléments de $\{0,1\}^*$ et les réponses de $\mathcal{L}$ sont des éléments de $\{0,1\}^{\ell_h}$;
- les requêtes à $\mathcal{R}$ sont des éléments de $\{0,1\}^{\ell_m} \times \{0,1\}^s$ et les réponses de $\mathcal{R}$ sont des éléments de $\{0,1\}^n$.

5.2.2 Principe de conception du simulateur et notations

Soit $\mathcal{X} = \{0,1\}^s$ l'ensemble des états internes possibles. Rappelons que la i-ème insertion de message exécute une sous-routine Round$[M_i]$, définie par

$$\text{Round}[M_i](x) = \mathcal{F}(M_i, \mathsf{Insert}[M_i](x)), \mathsf{Insert}[M_i](x)|_C,$$

sur l'état interne courant.

L'état interne initial est un état constant fixé par la spécification de la construction $x_0 = (A_0, B_0, C_0)$. Le simulateur de $\mathcal{F}$ (resp. de $(\mathcal{F}, \mathcal{F}^{-1})$) est obtenu en construisant dynamiquement un graphe $\mathcal{G} = (V, E) \subseteq \mathcal{X} \times \mathcal{X}^2$, où V est l'ensemble des sommets et E l'ensemble des arcs. V est formé de l'ensemble des états $\mathsf{Insert}^{-1}[M](A, B, C)$, où (M, A, B, C) est une requête à $\mathcal{F}$ reçue par le simulateur et de toutes les réponses aux requêtes reçues par le simulateur complétées par les parties C des requêtes pour former un état interne $\in \mathcal{X}$. Lorsque $\mathcal{F}$ est une permutation paramétrée, V contient également les états (A', B', C) correspondant aux requêtes (M, A', B', C) faites à la fonction inverse $\mathcal{F}^{-1}$, et $\mathsf{Insert}^{-1}[M](A, B, C)$ où (A, B) est la réponse correspondant du simulateur.

Un arc de x à y dans le graphe est étiqueté par un bloc de ℓ_m bits M et est noté $x \xrightarrow{M} y$. Il traduit le fait que les états x et y sont liés par l'application de la fonction de tour pendant le jeu, *i.e.* $y = \text{Round}[M](x)$. On note de plus X (resp. Y) le sous-ensemble de V constitué des origines (resp. des buts) des arcs dans E.

Un sommet $x \in V$ a un *chemin* étiqueté par une chaîne de bits μ s'il existe une liste (éventuellement vide) de blocs de message de ℓ_m bits $M_1, \ldots, M_k$ avec $\mu = M_1 \| \ldots \| M_k$ telle qu'il existe k arcs dans le graphe de la forme $x_{i-1} \xrightarrow{M_i} x_i$, $1 \leq i \leq k$ et $x_k = x$. On note alors $x_0 \overset{\mu}{\hookrightarrow} x$. Si besoin, on mentionne le graphe dans lequel le chemin existe $x_0 \xhookrightarrow{\mu\ [\mathcal{G}]} x$. Dans ce cas, les sommets traversés par le chemin et les arêtes qu'il emprunte appartiennent aux ensembles

des sommets et des arêtes de $\mathcal{G}$. Le chemin est dit *complet* si son étiquette est un message paddé, *i.e.* il existe $\mathcal{M} \in \{0,1\}^*$ tel que $\mu = \mathsf{pad}(\mathcal{M})$.

Au cours de la preuve de sécurité, nous allons être amenés à considérer des évènements se produisant au cours de la construction du graphe. Par conséquent nous considérerons l'état de certains objets à un instant donné du jeu de sécurité. Pour un objet O, on note $O(q)$ sa valeur avant la q-ème requête, la première requête étant indexée par 1. Soit un sommet (resp. un arc) présent dans le graphe à la fin d'un jeu. On note q_x (resp. q_a) la requête qui insère cet élément dans le graphe, *i.e.* $q_x = \max(q \geq 0 | x \notin V(q))$ (resp. $q_a = \max(q \geq 0 | x \notin E(q))$), avec la convention que la requête 0 correspond à l'initialisation du simulateur.

5.2.3 Indifférentiabilité dans le cas d'une fonction idéale

Nous considérons le mode générique de la Figure 5.1 pour un encodage de message quelconque. Nous présentons en Figure 5.5 un simulateur S. En utilisant ce simulateur, on obtient une borne précise sur l'avantage d'un distingueur cherchant à distinguer notre construction d'un oracle aléatoire dans le cadre de l'indifférentiabilité. Notre preuve consiste à définir une succession de jeux pour construire progressivement le simulateur $\mathcal{S}$. On montre ensuite que la différence des probabilités de voir le distingueur répondre 1 entre le jeu original où il interagit avec $Q = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ et le jeu final où il interagit avec $Q' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$ est bornée par les probabilités d'évènements d'échec qui sont susceptibles de permettre au distingueur de faire la différence entre la construction et la simulation.

`Initialisation de` $\mathcal{S}$

1. $V = \{x_0\}$ `et` $E = \emptyset$

`Simulation de` $\mathcal{F}$
`Entrée:` (M, A, B, C)
`Sortie:` (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. `Si` $\exists x \xrightarrow{M} y \in E$
   (a) `Retourner` (A', B')`, où` $y = (A', B', C)$
3. `Si` $\exists \mu, \mathcal{M}$ `tel que` $x_0 \overset{\mu}{\hookrightarrow} x$ `et` $\mu \| M = \mathsf{pad}(\mathcal{M})$
   (a) $B' = \mathcal{H}(\mathcal{M})$ `et` $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. `Sinon`
   (a) $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ `et` $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
5. $y = (A', B', C)$
6. $V = V \cup \{x, y\}$ `et` $E = E \cup \{x \xrightarrow{M} y\}$
7. `Retourner` (A', B')

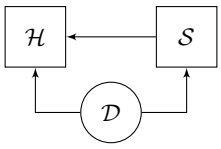


FIGURE 5.5 – Simulateur $\mathcal{S}$ de $\mathcal{F}$

5.2.3.1 Résumé de la preuve

La propriété d'indifférentiabilité est prouvée en construisant une séquence de jeux pour construire progressivement $\mathcal{S}$. Les jeux successifs sont représentés schématiquement en Figure 6.2 et peuvent être résumés de la manière suivante :

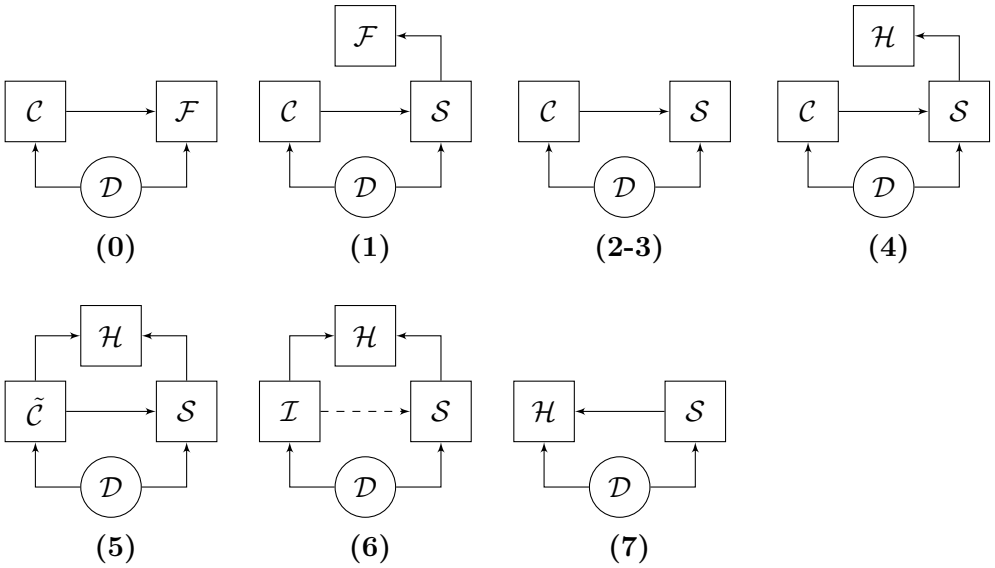


FIGURE 5.6 – Évolution des interactions entre oracles et simulateurs au cours de la preuve.

(0) On part du jeu original où le distingueur interagit avec le système $\mathcal{Q} = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, où $\mathcal{C}$ est une implémentation de la construction. [2]

(1) On insère un simulateur $\mathcal{S}$, qui remplace $\mathcal{F}$ auprès du distingueur et de la construction. Ce simulateur se contente de transmettre les requêtes à $\mathcal{F}$ et renvoie les réponses obtenues de $\mathcal{F}$ à l'appelant, $\mathcal{C}$ ou $\mathcal{D}$. $\mathcal{S}$ construit le graphe correspondant aux couples (requête, réponse) qu'il voit passer.

(2-3) $\mathcal{S}$ simule $\mathcal{F}$ au lieu de réaliser des appels à $\mathcal{F}$.

(4) Lorsque c'est nécessaire, $\mathcal{S}$ fait des appels à $\mathcal{H}$ pour définir ses valeurs de sorties.

(5) $\mathcal{C}$ est remplacée par une construction $\tilde{\mathcal{C}}$ qui exécute $\mathcal{C}^{\mathcal{F}}$ sur ses entrées, en faisant des appels à $\mathcal{S}$ lorsqu'une évaluation de $\mathcal{F}$ est nécessaire, mais ignore les réponses obtenues et réalise un appel final à $\mathcal{H}$ pour obtenir la réponse à la requête. Les réponses des requêtes à $\mathcal{L}$ ne dépendent plus du simulateur.

(6) $\tilde{\mathcal{C}}$ est supprimé et les requêtes à $\mathcal{L}$ sont directement transmises à l'oracle aléatoire. Un composant du distingueur enregistre les requêtes à $\mathcal{L}$ et les exécute en fin de jeu. Les réponses aux requêtes à $\mathcal{R}$ ne dépendent plus des appels à $\mathcal{L}$.

(7) L'exécution finale de la construction est supprimée. Le distingueur interagit à présent avec le système final $Q' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

On note W_i l'évènement $\mathcal{D}$ retourne 1 au jeu i, *i.e.* avant la $(i+1)$-ème transition. Lors de chaque transition entre les jeux i et $i+1$, on borne la différence entre les probabilités W_i et W_{i+1}. Pendant la séquence de jeux, nous allons définir des évènements d'échecs permettant d'évaluer dans quels cas les simulateurs dans deux jeux successifs produisent des sorties ayant des distributions différentes. Lorsqu'on considère simultanément un évènement Ev et W_i, la probabilité de l'évènement est considéré dans le jeu i. Si nécessaire, on indexe Ev par i pour éviter toute confusion : Ev^i.

Au cours de la preuve, on s'assure que chaque requête augmente d'au plus un le nombre d'arcs dans le graphe du simulateur.

2. On ne fait pas de distinction entre la construction $\mathcal{C}$ et un programme qui l'exécute.

Finalement, on ajoute deux informations supplémentaires aux requêtes effectuées auprès du simulateur :

- la provenance de la requête. Pour les requêtes adressées directement par le distingueur à l'interface $\mathcal{R}$, on note pour origine $\mathcal{D}$; pour les requêtes liées à l'exécution de la construction suite à un appel à $\mathcal{L}$, on note pour origine $\mathcal{C}$.
- l'indice de la requête du point de vue du distingueur. On numérote les requêtes du distingueur de manière croissante, indépendamment de l'interface, $\mathcal{L}$ ou $\mathcal{R}$ auxquelles elles sont adressées. Lorsqu'une construction reçoit une requête d'indice i adressée à $\mathcal{L}$, elle numérote par i toutes les requêtes qu'elle adresse au simulateur.

Ces informations permettent d'estimer les différences de probabilité entre des jeux successifs. Lorsque le simulateur n'en fait pas usage, on omet parfois de les mentionner dans sa description.

Jeu de départ (Jeu 0). Il s'agit du jeu original où $\mathcal{D}$ interagit avec $\mathcal{Q} = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$. Par définition du jeu 0, on a

$$\Pr[W_0] = \Pr\left[\mathcal{D}^{\mathcal{Q}} = 1\right].$$

Jeu 1. $\mathcal{F}$ est remplacé par un simulateur $\mathcal{S}$ avec la même interface que $\mathcal{F}$ qui transmet les requêtes à $\mathcal{F}$ et retourne les réponses de $\mathcal{F}$ l'appelant, soit à $\mathcal{C}$ soit à $\mathcal{D}$. Au cours du jeu, $\mathcal{S}$ construit les deux graphes

$$\mathcal{G}_{\mathcal{C}} = (V_{\mathcal{C}}, E_{\mathcal{C}}),\ \mathcal{G}_{\mathcal{D}} = (V_{\mathcal{D}}, E_{\mathcal{D}}),$$

en collectant les entrées et sorties de $\mathcal{F}$ correspondant aux requêtes provenant de $\mathcal{C}$ et de $\mathcal{D}$. $\mathcal{S}$ est explicité en Figure 5.7. L'action de $\mathcal{S}$ ne modifie pas la vue de $\mathcal{D}$, par conséquent $\Pr[W_1] = \Pr[W_0]$.

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Appeler $\mathcal{F}$: $(A', B') = \mathcal{F}(M, A, B, C)$
3. $y = (A', B', C)$
4. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
5. Retourner (A', B') à $\mathcal{O}$

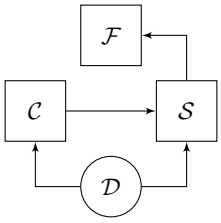


FIGURE 5.7 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 1

Jeu 2. On modifie légèrement notre simulateur pour éliminer la fonction aléatoire $\mathcal{F}$. Celle-ci est remplacée par une simulation parfaite réalisée par le simulateur. À chaque fois que $\mathcal{S}$ a besoin d'obtenir $\mathcal{F}(M, A, B, C)$ pour $(A, B, C) \in \mathcal{X}$, $\mathcal{S}$ sélectionne aléatoirement la réponse. On explicite le simulateur en Figure 5.8. Ceci ne modifie pas les distributions des sorties puisque $\mathcal{F}$ est un oracle aléatoire. Par conséquent $\Pr[W_2] = \Pr[W_1]$.

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$
Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A', B'), où $y = (A', B', C)$
3. $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $y = (A', B', C)$
5. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
6. Retourner (A', B') à $\mathcal{O}$

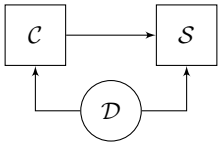


FIGURE 5.8 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 2

Jeu 3. L'étape suivante dans la séquence de jeux consiste à remplacer la simulation de la partie B des réponses par la réponse à un appel à l'oracle aléatoire lorsque cette partie correspond à une valeur pouvant être retournée comme valeur de sortie de la construction. Afin de pouvoir réaliser cette opération sans changer la distribution de la vue du distingueur, on doit s'assurer de propriétés de cohérence du graphe $\mathcal{G}$:

- Aucun sommet avec un chemin au moment de sa définition ne peut être atteint par deux chemins différents au cours du jeu. Dans le cas contraire l'oracle aléatoire ne sera pas en mesure de produire la collision attendue par la construction ;
- Aucun sommet n'ayant pas de chemin au moment de sa définition n'obtient de chemin au cours du jeu. Dans le cas contraire, l'oracle aléatoire ne sera pas en mesure de reproduire la valeur précédemment définie par le simulateur.

Nous modifions le simulateur $\mathcal{S}$ pour lever un $\mathsf{Drapeau}_1$ lorsque l'état $y \in \mathcal{X}$ nouvellement généré est déjà présent dans le graphe, *i.e.* correspond à $\tilde{x} \in V$. Si le $\mathsf{Drapeau}_1$ est levé à la fin du jeu, on dit que l'évènement $\mathsf{Échec}_1$ se produit. Le simulateur mis à jour est explicité en Figure 5.9.

Les modifications entre le jeu 2 et le jeu 3 ne modifient pas la vue de l'attaquant, par conséquent $\Pr[W_3] = \Pr[W_2]$.

On définit à présent une propriété du graphe impliquant les deux propriétés énoncées ci-dessus. Nous prouvons également que cette propriété est vérifiée lorsque $\mathsf{Échec}_1$ ne se produit pas.

Définition 28 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu de la suite de jeux décrite dans la preuve. On dit que le graphe est* stable *si pour tout sommet $x \in V$, le nombre de chemins de x n'augmente pas après la requête q_x.*

Proposition 21 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu. Si $\mathcal{G}$ est stable alors un sommet de V a au plus un chemin.*

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A', B'), où $y = (A', B', C)$
3. $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $y = (A', B', C)$
5. Si $y \in V_{\mathcal{C}} \cup V_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_1$
6. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
7. Retourner (A', B') à $\mathcal{O}$

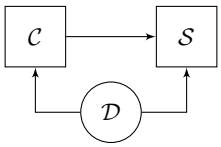


FIGURE 5.9 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 3

Démonstration : On démontre la proposition par contraposée. Soit x un sommet avec deux chemins distincts dans $\mathcal{G}$, $\mu_1 = M_1 \| \dots \| M_k$ et $\mu_2 = P_1 \| \dots \| P_\ell$. Alors il existe deux suites d'états, $u_1, \dots, u_k$ et $v_1, \dots, v_\ell$ telles que

$$x_0 \xrightarrow{M_1} u_1 \xrightarrow{M_2} u_2 \dots u_{k-1} \xrightarrow{M_k} u_k = x$$

et

$$x_0 \xrightarrow{P_1} v_1 \xrightarrow{P_2} v_2 \dots v_{\ell-1} \xrightarrow{P_\ell} v_\ell = x.$$

Sans perte de généralité, on suppose $k \leq \ell$. Considérons l'ensemble $I = \{1 \leq j \leq k | u_{k-j} \neq v_{\ell-j}\}$.

- Si I est vide, $x_0 = v_{\ell-k}$ et $\ell \neq k$ car les deux chemins sont distincts. Alors x_0 a un chemin $P_1 \| \dots \| P_{\ell-k}$ complété après l'insertion de x_0 à l'initialisation du simulateur. $\mathcal{G}$ n'est pas stable.
- Si I n'est pas vide, soit $i = \min(I)$. Soit $x' = u_{k-i+1}$ si $i > 1$ ou $x' = x$ si $i = 1$. Alors, les deux arcs $u_{k-i} \xrightarrow{M_{k-i+1}} x'$ et $v_{\ell-i} \xrightarrow{P_{\ell-i+1}} x'$ partagent le même but x' mais ont des sources différentes. Considérons à présent le sous-ensemble de E des arcs dont le but est x'. Étant donné qu'à chaque requête au plus un arc est défini, considérons la requête où le premier arc de ce sous-ensemble est défini. À la fin de cette requête, x' a au plus un chemin, alors que x' a au moins deux chemins à la fin du jeu. Ainsi $\mathcal{G}$ n'est pas stable.

□

Proposition 22 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu susceptible de lever le drapeau* $\mathsf{Drapeau}_1$. *Si* $\mathsf{Échec}_1$ *ne se produit pas, alors $\mathcal{G}$ est stable.*

Démonstration : On démontre la preuve par contraposée. Supposons que $\mathcal{G}$ n'est pas stable. Soit x un sommet avec un chemin $\mu = M_1 \| \dots \| M_k$ complété après l'insertion de x. Il existe une

suite d'états $x_1, \ldots, x_k$ tels que

$$x_0 \xrightarrow{M_1} x_1 \xrightarrow{M_2} x_2 \ldots x_{k-1} \xrightarrow{M_k} x_k = x.$$

Soit q l'indice de la requête pendant laquelle ce chemin a été complété. Ceci signifie qu'il existe un entier i, $1 \leq i \leq k$, tel que l'arc $x_{i-1} \xrightarrow{M_i} x_i$ a été ajouté au graphe pendant la q-ème requête alors que tous les autres arcs du chemin appartiennent à $E(q)$. Puisque q est minimal, $x_{i-1} \xrightarrow{M_i} x_i \notin E(q)$. Par définition de x et de μ, on a $q > q_x$. x_i est un élément de $V(q)$. En effet, puisque $x_{i-1} \xrightarrow{M_i} x_i$ est le seul arc n'appartenant pas à $E(q)$, on en déduit que pour $i \leq k-1$, $x_i \xrightarrow{M_{i+1}} x_{i+1} \in E(q)$, impliquant $x_i \in V(q)$ si $i < k$. De plus, on sait que $x_k = x \in V(q_x + 1) \subset V(q)$. Par conséquent, $\mathsf{Drapeau}_1$ est levé pendant la requête q car $x_i \in V(q)$. Donc $\mathsf{Échec}_1$ se produit. □

Jeu 4. On peut à présent insérer l'oracle aléatoire $\mathcal{H}$ dans le jeu. Au lieu de définir une réponse aléatoire pour une paire $x \in V, M \in \{0,1\}^{\ell_m}$, où x a un chemin μ dans le graphe, $\mathcal{S}$ réalise un appel à $\mathcal{H}$ pour laisser à $\mathcal{H}$ le soin de définir la partie B de la réponse à la requête, lorsque $\mu \| M$ est un message paddé. $\mathcal{S}$ complète alors la partie A manquante de la sortie en la simulant. Cette modification est bien définie si $\mathsf{Échec}_1$ ne se produit pas puisque, en raison de la stabilité du graphe, tout noeud x qui a un chemin, a alors un unique chemin dans $\mathcal{G}$ qui peut être extrait du graphe au moment où x est défini. Lorsque x n'a pas de chemin dans le graphe, ou lorsque le chemin de x complété avec M ne produit pas un message paddé, $\mathcal{S}$ définit la réponse à la requête par simulation comme dans le jeu précédent. Le nouveau simulateur est explicité en Figure 5.10. Dans le cas où $\mathsf{Échec}_1$ ne se produit pas, la distribution des sorties de $\mathcal{S}$ n'est pas modifiée et on a $\Pr\left[W_4 | \neg \mathsf{Échec}_1\right] = \Pr\left[W_3 | \neg \mathsf{Échec}_1\right]$. De plus la distribution des nouvelles valeurs de y, générées à l'étape 6 du simulateur, n'est pas modifiée entre ce jeu et le jeu précédent. La probabilité de l'évènement $\mathsf{Échec}_1$ reste donc identique. On a donc $\Pr\left[W_4 \wedge \neg \mathsf{Échec}_1\right] = \Pr\left[W_3 \wedge \neg \mathsf{Échec}_1\right]$.

Afin de terminer la preuve, il est nécessaire de remplacer complètement $\mathcal{C}$ par l'oracle aléatoire. Afin de pouvoir procéder à ce remplacement sans que le distingueur ne puisse le détecter nous devons nous assurer que :

- quand la construction ignore les réponses qu'elle obtient aux requêtes qu'elle soumet au simulateur, la vue du distingueur n'est pas significativement modifiée ;
- quand le simulateur ignore les requêtes provenant de la construction, la vue du distingueur n'est pas significativement modifiée.

Le premier point est pris en compte dans le jeu 5. Le deuxième point est pris en compte dans le jeu 6.

Dans le simulateur du jeu 4, on note la définition de deux nouveaux drapeaux, $\mathsf{Drapeau}_3$ et $\mathsf{Drapeau}'_3$, qui seront utiles pour analyser la différence entre les deux jeux suivants. On note $\mathsf{Échec}_3$ (resp. $\mathsf{Échec}'_3$) l'évènement « $\mathsf{Drapeau}_3$ (resp. $\mathsf{Drapeau}'_3$) est levé à la fin du jeu ». Ces deux drapeaux deviennent pertinents au jeu 6.

Jeu 5. La construction $\mathcal{C}$ est remplacée par une construction $\tilde{\mathcal{C}}$ avec une interface identique. Lorsque $\mathcal{D}$ effectue une requête $m \in \{0,1\}^*$ à $\mathcal{L}$, $\tilde{\mathcal{C}}$ procède en deux étapes : $\tilde{\mathcal{C}}$ commence par exécuter la construction $\mathcal{C}^{\mathcal{F}}$ sur m en effectuant des requêtes à $\mathcal{S}$ à chaque fois qu'une évaluation

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
i indice de la requête du point de vue du distingueur
Sortie : (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\mathcal{O} = \mathcal{D}$ et $\exists z \xrightarrow{M'} x \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ avec $x_0 \xhookrightarrow{\mu} z$, et $\mu \| M'$ est préfixe d'un message paddé, Lever $\mathsf{Drapeau}_3$
3. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A', B'), où $y = (A', B', C)$
4. Si $\exists \mu, \mathcal{M}$ tel que $x_0 \xhookrightarrow{\mu\ [\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{C}}]} x$ et $\mu \| M = \mathsf{pad}(\mathcal{M})$
   (a) $B' = \mathcal{H}(\mathcal{M})$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
5. Sinon
   (a) $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
6. $y = (A', B', C)$
7. Si $y \in V_{\mathcal{C}} \cup V_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_1$
8. Si $\mathcal{O} = \mathcal{C}$ et $x \xrightarrow{M} y \notin E_{\mathcal{D}}$ et $\exists y \xrightarrow{M'} z \in E_{\mathcal{D}}$ d'indice $j > i$ et $\mu \| M$ est préfixe d'un message paddé avec $x_0 \xhookrightarrow{\mu} x$, Lever $\mathsf{Drapeau}'_3$
9. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
10. Retourner (A', B') à $\mathcal{O}$

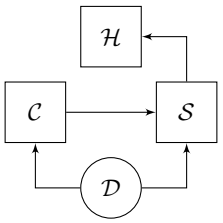


FIGURE 5.10 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 4

de $\mathcal{F}$ est requise ; puis $\tilde{\mathcal{C}}$ ignore le résultat de l'exécution de la construction et effectue un appel à $\mathcal{H}$ pour obtenir $h = \mathcal{H}(m)$ et retourne h à $\mathcal{D}$. Le simulateur $\mathcal{S}$ n'est pas modifié.

Dans le cas général, les appels de $\tilde{\mathcal{C}}$ définissent étape par étape un chemin complet dans $\mathcal{G}$ et la requête finale du simulateur $\tilde{\mathcal{C}}$ à l'oracle aléatoire $\mathcal{H}$ retourne une valeur précédemment définie par le traitement de la dernière requête de la construction auprès du simulateur $\mathcal{S}$. Dans ce cas, la vue du distingueur n'est pas modifiée.

Cependant, le comportement de $\tilde{\mathcal{C}}$ diffère du comportement de $\mathcal{C}$ dans le cas particulier suivant. Si l'arc correspondant à la dernière requête de la construction a été défini précédemment par le simulateur et que cette définition n'a pas fait intervenir l'oracle aléatoire $\mathcal{H}$, le dernier appel de $\tilde{\mathcal{C}}$ à $\mathcal{H}$ définit une nouvelle valeur qui n'est pas liée à la valeur définie par le simulateur $\mathcal{S}$. Plaçons-nous dans ce cas et considérons $x \xrightarrow{M} y$ l'arc correspondant à la dernière requête, notée requête q, de la construction au simulateur $\mathcal{S}$. Puisqu'il s'agit de la dernière requête de la construction, x a un chemin μ dans $\mathcal{G}$. $x \xrightarrow{M} y \in E$ a été défini par une requête q' précédente au simulateur, $q' < q$. Soit x avait un chemin μ' dans $\mathcal{G}$ avant la requête q' et $\mu' \| M$ n'est pas un chemin complet, ou x n'avait pas de chemin avant la requête q'. Dans les deux cas, le nombre de chemin de x a changé entre la requête q' et la requête q ($\mu' \neq \mu$

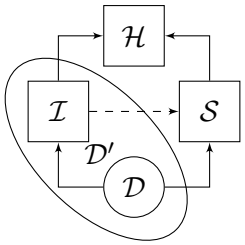


FIGURE 5.11 – Interactions entre les composants du jeux 6.

puisque $\mu \| M$ est un chemin complet). Si $\mathsf{Échec}_1$ ne se produit pas, $\mathcal{G}$ est stable et ce cas de figure est impossible. De plus $\Pr\left[\neg \mathsf{Échec}_1\right]$ n'est pas modifié entre les deux jeux. On a donc $\Pr\left[W_5 \wedge \neg \mathsf{Échec}_1\right] = \Pr\left[W_4 \wedge \neg \mathsf{Échec}_1\right]$.

Jeu 6. On modifie à présent le jeu de sécurité de telle sorte que les appels de la construction au simulateur sont retardés, et ne sont exécutés qu'à la fin du jeu. Informellement, tout se passe comme si on forçait le distingueur à exécuter à la fin du jeu la construction initiale sur chacune des requêtes qu'il a soumise à $\mathcal{L}$. Plus formellement, la construction $\tilde{C}$ est supprimée. Le distingueur $\mathcal{D}$ est utilisé pour construire un distingueur $\mathcal{D}'$ qui :

- transmet à $\mathcal{S}$ les requêtes-réponses de $\mathcal{D}$ à $\mathcal{R}$ sans modification ;
- transmet à $\mathcal{H}$ les requêtes-réponses de $\mathcal{D}$ à $\mathcal{L}$ en enregistrant la liste des requêtes ;
- à la fin du jeu, exécute les requêtes correspondant à l'exécution de la construction sur la liste des requêtes de $\mathcal{D}$ à $\mathcal{L}$;
- renvoie la sortie de $\mathcal{D}$.

On peut modéliser les actions supplémentaires de $\mathcal{D}'$ par l'introduction d'un intercepteur $\mathcal{I}$ par lequel passent toutes les requêtes de $\mathcal{D}$ à $\mathcal{L}$ et qui exécute, en fin de jeu, la construction sur ces requêtes, en les traitant dans l'ordre où il les a reçues, en faisant appel à $\mathcal{S}$. On donne en Figure 5.11 une représentation du jeu 6. Le simulateur est inchangé par rapport aux jeux 4 et 5. La définition de W_6 est modifiée pour tenir compte de la définition de ce jeu : $W_6 = \Pr\left[{\mathcal{D}'}^{\mathcal{H},\mathcal{S}^{\mathcal{H}}} = 1\right]$.

Les réponses des requêtes à $\mathcal{L}$ sont distribuées de la même manière dans les jeux 5 et 6. La manière de définir les réponses des requêtes faites au simulateur peut être modifiée. Pour rappel, le simulateur, définit au jeu 4, peut répondre de 3 manières différentes aux requêtes qu'il reçoit :

Méthode (1) : si la requête correspond à un arc précédemment défini, il répond en utilisant l'information contenue dans cet arc ;

Méthode (2) : si la requête définit un nouvel arc, le simulateur simule $\mathcal{F}$ pour générer les parties A et B de la réponse et complète avec la partie C appropriée ;

Méthode (3) : si cet arc complète un chemin complet, la partie B est remplacée par la réponse à une requête à l'oracle aléatoire.

On montre que les réponses des requêtes faites au système sont **identiques** si l'évènement $\mathsf{Échec}_1 \vee \mathsf{Échec}_3$ ne se produit pas dans le jeu 5.

On commence par analyser les requêtes effectuées par le distingueur et par la construction et comment elles diffèrent entre le jeu 5 et le jeu 6.

Considérons une requête provenant du distingueur. On analyse ce qui se passe suivant la manière de répondre dans le jeu 5.

1. Si la réponse provient de la répétion d'un arc précédent, on distingue les cas suivants.
   (a) Si cet arc a été défini en réponse au distingueur, il est défini de même dans le jeu 6.
   (b) Si cet arc a été défini uniquement en réponse à la construction, un changement de distribution peut apparaître entre le jeu 5 et le jeu 6. En effet, l'origine de l'arc rejoué dans le jeu 5 possède nécessairement un chemin μ au moment de sa définition. Sous l'hypothèse que $\mathsf{Échec}_1$ ne se produise pas, en notant M le bloc de message de la requête, la manière de simuler l'arc est fonction du fait que $\mu \| M$ soit un message paddé. Dans le jeu 6, cet arc n'a pas encore été défini au moment de l'exécution de la requête et est donc défini en réponse au distingueur.
      i. Si l'origine du nouvel arc a un chemin constitué uniquement d'arcs définis en réponse au distingueur, la distribution n'est pas modifiée.
      ii. Dans le cas contraire, l'origine du nouvel arc n'a pas de chemin dans le jeu 6, et par conséquent l'arc est nécessairement défini suivant la méthode (2). La distribution est modifiée si l'arc rejoué dans le jeu 5 était précédemment défini suivant la méthode (3). En résumé, en supposant $\neg\mathsf{Échec}_1$, la distribution est modifiée si dans le jeu 5 le distingueur soumet une requête (M, A, B, C) telle que $x = \mathsf{Insert}^{-1}[M](A, B, C)$ a un chemin μ dans $\mathcal{G}_\mathcal{C} \cup \mathcal{G}_\mathcal{D}$ mais pas dans $\mathcal{G}_\mathcal{D}$ et $\mu \| M$ est un chemin complet. Si une telle requête est soumise au cours du jeu, on dit que l'évènement $\mathsf{PredictionArc}$ se produit. On montre plus bas dans le lemme 3 que cet évènement ne se produit pas si $\neg\mathsf{Échec}_1 \wedge \neg\mathsf{Échec}_3$ à la fin du jeu 5.
2. Si la réponse est générée selon la méthode (2) dans le jeu 5, elle est générée de même dans le jeu 6. En effet, les arcs de $\mathcal{G}_\mathcal{C}$ non définis ne changent pas le fait que l'arc n'a pas encore été défini auparavant, ni qu'il n'existe pas de chemin complet que le nouvel arc complèterait.
3. Si la réponse est générée selon la méthode (3) dans le jeu 5, la distribution peut être modifiée dans le jeu 6. En effet, le chemin conduisant à utiliser l'oracle aléatoire pour générer la partie B peut ne pas exister dans le jeu 6 au moment de la définition de cet arc, conduisant à définir l'arc suivant la méthode (2).
   (a) Par conséquent, en supposant $\neg\mathsf{Échec}_1$, la distribution est modifiée si $\mathsf{PredictionArc}$ se produit. On remarque ici que $\mu \| M$ est nécessairement un chemin complet du fait de la méthode de génération de l'arc dans le jeu 5.
   (b) Si $\neg\mathsf{Échec}_1 \wedge \neg\mathsf{Échec}_3$, l'arc est généré de la même manière.

Considérons à présent une requête provenant de la construction. Pour rappel, l'exécution d'une telle requête dans le jeu 6 n'est plus réalisée immédiatement au moment de la requête correspondante de $\mathcal{D}$ à $\mathcal{L}$ mais est retardée à la fin du jeu. La chronologie des requêtes de $\mathcal{C}$ entre elles n'est cependant pas modifiée. On note (M, A, B, C) cette requête et $x = \mathsf{Insert}^{-1}[M](A, B, C)$. Du fait de la définition de la construction, x a un chemin au moment de sa définition. En supposant que $\mathsf{Échec}_1$ ne se produise pas, le graphe est stable. On note μ le chemin défini dans le jeu 5.

1. Si la réponse à cette requête est obtenue selon la méthode (1) dans le jeu 5, elle est obtenue de la même manière dans le jeu 6. En effet, l'arc répété dans le jeu 5 existe également dans le jeu 6 au moment de la requête de la construction car dans le jeu 6 les requêtes provenant du distingueur précèdent les requêtes provenant de la construction et car l'ordre des requêtes provenant de la construction n'est pas modifié.

2. Si la réponse à cette requête est obtenue selon la méthode (2) dans le jeu 5, cela signifie que $\mu \| M$ n'est pas un message paddé. Ceci est également vrai dans le jeu 6 car les blocs de messages du chemin μ ne sont pas modifiés et la distribution n'est donc pas modifiée dans ce cas.
3. Si la réponse à cette requête est obtenue selon la méthode (3) dans le jeu 5, la distribution peut être modifiée. En effet, dans le jeu 6, la réponse de la requête est obtenue soit selon la méthode (1), soit selon la méthode (3). La méthode (2) n'est pas possible car par construction x a un chemin et $\mu \| M$ est complet.
   (a) Si la méthode (3) est employée pour répondre dans le jeu 6, l'arc est défini de la même manière que dans le jeu 5.
   (b) Si la méthode (1) est employée, nécessairement l'arc répété dans le jeu 6 a été défini par le distingueur, car s'il avait été défini par la construction il aurait également été généré par la méthode (1) dans le jeu 5 par conservation de la chronologie des requêtes de la construction. De plus l'arc repété dans le jeu 6 correspond à une requête $j > i$ du distingueur, car dans le cas contraire il aurait été également généré par la méthode (1) dans le jeu 5.
      i. Si x a un chemin dans $\mathcal{G}_{\mathcal{D}}$ et que $\neg \mathsf{Échec}_1$, le chemin de x est μ par stabilité du graphe et l'arc est donc défini comme dans le jeu 5.
      ii. Si au contraire x n'a pas de chemin dans $\mathcal{G}_{\mathcal{D}}$ l'arc est défini selon la méthode (2) et la distribution est donc modifiée. À nouveau on a montré que la distribution n'est pas modifiée si $\neg \mathsf{Échec}_1 \wedge \neg \mathsf{PredictionArc}$ se produit dans le jeu 5 (lors de la requête j du distingueur).

Si pour des jetons aléatoires des oracles et du distingueur $\neg \mathsf{Échec}_1 \wedge \neg \mathsf{Échec}_3$ est vrai à la fin du jeu 5, l'exécution du jeu 6 avec ces mêmes jetons aléatoires conduit le simulateur à produire le même graphe en fin de jeu 6. Par conséquent, $\mathsf{Échec}_1$ ne se produit pas non plus dans le jeu 6 car il n'existe pas de noeud but de deux arêtes dans le graphe du jeu 6. De plus $\mathsf{Échec}_3^{'}$ ne se produit pas non plus dans le jeu 6 car cela entrainerait l'occurence de $\mathsf{Échec}_3$ dans le jeu 5. On a donc montré que $\neg(\mathsf{Échec}_1)^5 \wedge \neg(\mathsf{Échec}_3)^5 \subset \neg(\mathsf{Échec}_1)^6 \wedge \neg(\mathsf{Échec}_3^{'})^6$. On peut montrer de même l'inclusion inverse en partant d'une exécution du jeu 6. On obtient donc

$$\neg(\mathsf{Échec}_1)^5 \wedge \neg(\mathsf{Échec}_3)^5 = \neg(\mathsf{Échec}_1)^6 \wedge \neg(\mathsf{Échec}_3^{'})^6,$$

et les graphes définis par le simulateur dans les deux jeux sont identiques quand cet évènement se produit. Par conséquent, la vue du distingueur est identique dans les deux jeux et on a

$$\Pr\left[W_6 \wedge \neg \mathsf{Échec}_1 \wedge \neg \mathsf{Échec}_3^{'}\right] = \Pr\left[W_5 \wedge \neg \mathsf{Échec}_1 \wedge \neg \mathsf{Échec}_3\right].$$

On applique à présent le lemme de différence entre les jeux 5 et 6. On a

$$\begin{aligned} |\Pr[W_6] - \Pr[W_5]| &\leq \max\left(\Pr\left[(\mathsf{Échec}_1)^5 \vee (\mathsf{Échec}_3)^5\right], \Pr\left[(\mathsf{Échec}_1)^6 \vee (\mathsf{Échec}_3^{'})^6\right]\right), \\ &\leq \Pr\left[(\mathsf{Échec}_1)^5 \vee (\mathsf{Échec}_3)^5\right], \\ &\leq \Pr\left[(\mathsf{Échec}_1)^5\right] + \Pr\left[(\mathsf{Échec}_3)^5\right]. \end{aligned}$$

De plus

$$\Pr\left[W_5 \wedge \neg \mathsf{Échec}_1\right] = \Pr\left[W_4 \wedge \neg \mathsf{Échec}_1\right] = \Pr\left[W_3 \wedge \neg \mathsf{Échec}_1\right].$$

On peut donc appliquer une nouvelle fois le lemme de différence entre les jeux 3 et 5 pour obtenir

$$|\Pr[W_5] - \Pr[W_3]| \leq \max\left(\Pr\left[(\mathsf{Échec}_1)^3\right], \Pr\left[(\mathsf{Échec}_1)^5\right]\right).$$

On finit l'analyse de ce jeu en prouvant le lemme suivant.

Lemme 3 *Si* PredictionArc *se produit pour la première fois dans le jeu 5 pendant la requête q, il existe un arc $x \xrightarrow{M} x' \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ tel que :*

- *x a un chemin μ dans $\mathcal{G}_{\mathcal{C}}$,*
- *x' est la source d'un arc défini pour répondre à une requête provenant du distingueur, requête faisant intervenir un bloc de message M', et $\mu\|M\|M'$ est le préfixe d'un message paddé.*

De plus, si $\mathsf{Échec}_1$ *ne se produit pas, la requête du distingueur correspondant à la source x' se produit après la définition de l'arc $x \xrightarrow{M} x'$.*

Démonstration : Supposons que PredictionArc se produise pendant la requête q. Soit x_f l'état défini à l'étape 1 et M_f le bloc de message soumis par la requête. PredictionArc se produit pendant cette requête, on a donc $\mathcal{O} = \mathcal{D}$ et x_f a un chemin dans $\mathcal{G}$. Ainsi, il existe une suite d'états $x_1, \ldots, x_k = x_f$ et une suite de blocs de message $M_1, \ldots, M_k$ tels que

$$x_0 \xrightarrow{M_1} x_1 \xrightarrow{M_2} x_2 \ldots x_{k-1} \xrightarrow{M_k} x_k = x_f.$$

De plus, $M_1\| \ldots \|M_k\|M_f$ est un message paddé. x_f n'a pas de chemin dans $\mathcal{G}_{\mathcal{D}}$, donc il existe $1 \leq i \leq k$ tel que l'arc $a_i = x_{i-1} \xrightarrow{M_i} x_i$ appartiennent à $E_{\mathcal{C}} \setminus E_{\mathcal{D}}$. Soit I le plus grand entier vérifiant cette propriété. Par définition de la construction, pour $1 \leq j \leq I$, $q_{a_j} \leq q_{a_I}$ et l'arc a_j appartient à $E_{\mathcal{C}}$. Ainsi, x_{I-1} a un chemin $\mu = M_1 \ldots M_{I-1}$ dans $\mathcal{G}_{\mathcal{C}}$. On a également $a_I \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$. De plus, x_I correspond à une requête du distingueur, soit la requête $q_{a_{I+1}}$ si $I < k$, car par définition de I $x_I \xrightarrow{M_{I+1}} x_{I+1} \in E_{\mathcal{D}}$, auquel cas on pose $M' = M_{I+1}$, ou à la requête q auquel cas on pose $M' = M_f$. Finalement, $M_1\| \ldots \|M_k\|M_f$ est un message paddé, d'où $\mu\|M_I\|M'$ est le préfixe d'un message paddé, ce qui prouve la première partie du lemme.
Pour finir la preuve du lemme supposons que l'arc dont l'origine est x' est définie avant l'arc $x \xrightarrow{M} x'$. Alors, à l'étape 7 de la requête q_{a_I}, la condition est vérifiée puisque x' a été inséré dans V lors d'une requête précédente. Donc $\mathsf{Drapeau}_1$ est levé à l'étape q_{a_I}. □

Jeu 7. On modifie à présent le jeu de sécurité en supprimant à la fin du jeu la soumission au simulateur des arcs définis par la construction. Le changement de comportement entre les jeux 6 et 7 se produit après que la dernière réponse ait été fournie au distingueur. Sa vue n'est donc pas modifiée entre les deux jeux et on a

$$\Pr[W_6] = \Pr[W_7].$$

Dans le jeu 7, le graphe des appels provenant de $\mathcal{C}$ reste vide et on peut donc le supprimer. Toutes les requêtes provenant directement du distingueur, on peut supprimer dans le simulateur la mention de l'origine des requêtes et le code spécifique au graphe $\mathcal{C}$. L'indice des requêtes n'est plus utilisé, on peut également omettre de le préciser. On obtient ainsi le simulateur présenté en Figure 5.5. Ce jeu correspond au jeu final, donc $\Pr[W_7] = \Pr\left[\mathcal{D}^{\mathcal{Q}'}\right]$. En mettant bout à bout les résultats obtenus on a montré

$$\left|\Pr\left[\mathcal{D}^{Q}=1\right]-\Pr\left[\mathcal{D}^{Q'}=1\right]\right| \leq \max\left(\Pr\left[(\text{Échec}_1)^3\right],\Pr\left[(\text{Échec}_1)^5\right]\right) + \Pr\left[(\text{Échec}_1)^5\right]+\Pr\left[(\text{Échec}_3)^5\right].$$

On borne ces probabilités dans la section 5.3.

5.2.4 Indifférentiabilité dans le cas d'une permutation paramétrée idéale

On étend à présent la séquence de jeux au cas où $\mathcal{F}$ est une permutation paramétrée, ce qui permet de prouver dans ce cas l'indistinguabilité de la construction à un oracle aléatoire dans le modèle du chiffrement par bloc idéal. Le simulateur doit simuler non seulement les requêtes à la permutation paramétrée $\mathcal{F}$, mais également les requêtes à son inverse $\mathcal{F}^{-1}$. Cela ne modifie pas profondément les simulateurs intervenant dans les jeux 0 à 7 précédemment définis et la preuve peut être étendue en ajoutant une simulation de $\mathcal{F}^{-1}$ de la manière suivante.

Jeu 1. $\mathcal{S}$ redirige simplement les requêtes au chiffrement par bloc idéal $\mathcal{F}$ et retourne la sortie sans modification. Il stocke dans le graphe l'arc défini par la réponse de $\mathcal{F}^{-1}$. On donne le simulateur en Figure 5.12. La simulation des requêtes à $\mathcal{R}$ n'est pas modifiée. La distribution des réponses à $\mathcal{D}$ n'étant pas altérée, on a $\Pr[W_1]=\Pr[W_0]$.

Simulation de $\mathcal{F}^{-1}$
Entrée: (M,A',B',C), origine $\mathcal{O}=\mathcal{D}$
Sortie: (A,B)

1. $y=(A',B',C)$
2. Appeler $\mathcal{F}^{-1}$: $(A,B)=\mathcal{F}^{-1}(M,A',B',C)$
3. $x=\mathsf{Insert}^{-1}[M](A,B,C)$
4. $V_{\mathcal{O}}=V_{\mathcal{O}}\cup\{x,y\}$ et $E_{\mathcal{O}}=E_{\mathcal{O}}\cup\{x\xrightarrow{M}y\}$
5. Retourner (A,B) à $\mathcal{O}$

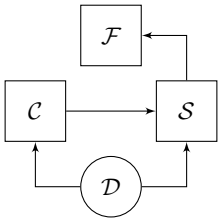


FIGURE 5.12 – Simulateur $\mathcal{S}$ de $\mathcal{F}^{-1}$ dans le jeu 1

Jeu 2. Dans le jeu 2, $\mathcal{S}$ simule $\mathcal{F}$ et $\mathcal{F}^{-1}$ parfaitement, *i.e.* la simulation assure que pour tout paramètre (M,C), $\mathcal{F}_{M,C}$ et $\mathcal{F}^{-1}_{M,C}$ se comportent comme des permutations inverses l'une de l'autre. Cette simulation est obtenue en répétant les sorties de $\mathcal{F},\mathcal{F}^{-1}$ précédemment définies et en sélectionnant les nouvelles valeurs uniformément parmi les valeurs qui ne correspondent pas à une réponse pour une entrée différente. Afin de préparer les jeux suivants on introduit deux drapeaux d'erreurs et on formule la simulation comme indiqué en Figure 5.13. Étant donné que la distribution des réponses du simulateur ne sont pas modifiées, on a $\Pr[W_2]=\Pr[W_1]$.

Jeu 3. Dans le jeu 3, $\mathcal{S}$ simule $\mathcal{F}$ et $\mathcal{F}^{-1}$ en choisissant la réponse aléatoirement. Le simulateur est obtenu à partir du simulateur du jeu 2 en supprimant deux instructions de type « **Aller à** », comme décrit en Figure 5.13. Ceci modifie la distribution des réponses de $\mathcal{R}$ et $\mathcal{R}^{-1}$. En effet, des collisions inattendues de la part d'une permutation paramétrée sont susceptibles de se produire dans les simulations de $\mathcal{F}$ et $\mathcal{F}^{-1}$ dans le jeu 3. Ces incohérences surviennent quand :

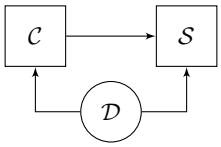
Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A', B'), où $y = (A', B', C)$
3. $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $y = (A', B', C)$
5. Si $y \in V_{\mathcal{D}} \cup V_{\mathcal{C}}$
   (a) Lever $\mathsf{Drapeau}_1$
   (b) Si $\exists x' \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
      i. Lever $\mathsf{Drapeau}_2$
      ii. (Jeu 2 uniquement) Aller à l'étape 3
6. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
7. Retourner (A', B') à $\mathcal{O}$



Simulation de $\mathcal{F}^{-1}$

Entrée: (M, A', B', C), origine $\mathcal{O} = \mathcal{D}$
Sortie: (A, B)

1. $y = (A', B', C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A, B), où $\mathsf{Insert}[M](x) = (A, B, C)$
3. $B \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
5. $y = (A', B', C)$
6. Si $x \in V_{\mathcal{D}} \cup V_{\mathcal{C}}$
   (a) Lever $\mathsf{Drapeau}_1$
   (b) Si $\exists x \xrightarrow{M} y' \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
      i. Lever $\mathsf{Drapeau}_2$
      ii. (Jeu 2 uniquement) Aller à l'étape 3
7. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
8. Retourner (A, B) à $\mathcal{O}$

FIGURE 5.13 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ et $\mathcal{F}^{-1}$ dans les jeux 2 et 3

- La réponse (A, B) assignée à la requête (M, A', B', C) faite à la permutation admet une définition antérieure par le simulateur pour la même paire (M, C). Comme pour tout M, $\mathsf{Insert}[M]$ est une bijection, ceci est équivalent à l'existence dans le graphe d'un arc $x \xrightarrow{M} y$

avec $x = \mathsf{Insert}^{-1}[M](A, B, C), y_C = C$ et $y_{A,B} \neq (A', B')$.

- La réponse (A', B') assignée à la requête (M, A, B, C) faite à la permutation inverse admet une définition antérieure par le simulateur pour la même paire (M, C). Ceci est équivalent à l'existence dans le graphe d'un arc $x \xrightarrow{M} y$, avec $y = (A', B', C)$ et $x \neq \mathsf{Insert}^{-1}[M](A, B, C)$.

On remarque que ces incohérences se produisent si $\mathsf{Drapeau}_2$ est levé pendant la simulation et que les jeux 2 et 3 se comportent de la même manière si ce drapeau n'est pas levé au cours du jeu. Afin de préparer la suite de la preuve, on considère le drapeau $\mathsf{Drapeau}_1$. On remarque que si $\mathsf{Drapeau}_2$ est levé, alors le drapeau $\mathsf{Drapeau}_1$ a déjà été levé précédemment. On note $\mathsf{Échec}_1$ l'évènement « $\mathsf{Drapeau}_1$ est levé au cours du jeu ». Comme les jeux 2 et 3 se comportent de la même manière quand le drapeau 2 n'est pas levé, on a $\Pr\left[W_3 \wedge \neg\mathsf{Échec}_1\right] = \Pr\left[W_2 \wedge \neg\mathsf{Échec}_1\right]$.

Dans les jeux suivants on cesse de considérer $\mathsf{Drapeau}_2$. Ceci ne change pas le comportement du distingueur quand $\mathsf{Échec}_1$ ne se produit pas.

On souhaite également préparer le remplacement de la simulation de la partie B des réponses au requête faite à la permutation par un appel à l'oracle aléatoire quand le noeud correspond à un noeud final d'une exécution de la construction. On affirme que $\mathsf{Échec}_1$ permet de détecter les configurations qui entraînent une incohérence de l'oracle aléatoire. On a besoin de s'assurer que les deux conditions énoncées dans le cas fonction idéale sont satisfaites. On doit également considérer la condition additionnelle suivante :

- Aucun noeud avec un chemin complet n'est inséré dans le graphe par un appel à la permutation inverse car dans le cas contraire, l'oracle aléatoire produira avec faible probabilité la partie B de la requête en question.

On veut s'assurer, dans le cas permutation, de la stabilité du graphe et de la satisfaction de cette propriété supplémentaire.

Proposition 23 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur au terme d'un jeu de la séquence de jeux susceptible de lever le drapeau $\mathsf{Drapeau}_1$. Si $\mathsf{Échec}_1$ ne se produit pas, le graphe est stable.*

Démonstration :
On démontre la preuve par contraposée.
Supposons que $\mathcal{G}$ n'est pas stable. Soit x un sommet avec un chemin $\mu = M_1 \| \ldots \| M_k$ complété après l'insertion de x. Il existe une suite d'états $x_1, \ldots, x_k$ tels que

$$x_0 \xrightarrow{M_1} x_1 \xrightarrow{M_2} x_2 \ldots x_{k-1} \xrightarrow{M_k} x_k = x.$$

Soit q l'indice de la requête pendant laquelle ce chemin a été complété. Ceci signifie qu'il existe un entier i, $1 \leq i \leq k$, tel que l'arc $x_{i-1} \xrightarrow{M_i} x_i$ a été ajouté au graphe pendant la q-ème requête (effectuée auprès de la simulation de $\mathcal{F}$ ou de $\mathcal{F}^{-}1$) alors que tous les autres arcs du chemin appartiennent à $E(q)$.
Puisque q est minimal, $x_{i-1} \xrightarrow{M_i} x_i \notin E(q)$. Par définition de x et de μ, on a $q > q_x$. x_{i-1} et x_i sont des éléments de $V(q)$. En effet, puisque $x_{i-1} \xrightarrow{M_i} x_i$ est le seul arc n'appartenant pas à $E(q)$, on en déduit que pour $i \geq 2$, $x_{i-2} \xrightarrow{M_{i-1}} x_{i-1} \in E(q)$, et donc $x_{i-1} \in V(q)$ pour $i > 1$. De même, pour $i \leq k-1$, $x_i \xrightarrow{M_{i+1}} x_{i+1} \in E(q)$, impliquant $x_i \in V(q)$ si $i < k$. De plus, on sait que $x_0 \in V(q)$ car ce sommet est ajouté à V lors de l'initialisation du simulateur, et $x_k = x \in V(q_x + 1) \subset V(q)$.

Par conséquent, $x_{i-1} \xrightarrow{M_i} x_i$ a été généré pendant une requête à la simulation de $\mathcal{F}$ et $\mathsf{Drapeau}_1$ est levé à pendant la requête q car $x_i \in V(q')$, ou il a été généré pendant une requête à la simulation de $\mathcal{F}^{-1}$ et $\mathsf{Drapeau}_1$ est levé à car $x_{i-1} \in V(q')$. Donc $\mathsf{Échec}_1$ se produit.

□

De plus, il est immédiat que si $\mathsf{Échec}_1$ ne se produit pas, la condition additionnelle, c'est à dire qu'aucun chemin complet n'est inséré dans le graphe par un appel à la permutation inverse, est satisfaite.

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$
i indice de la requête du point de vue du distingueur
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\mathcal{O} = \mathcal{D}$ et $\exists z \xrightarrow{M'} x \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ avec $x_0 \overset{\mu}{\hookrightarrow} z$, et $\mu \| M'$ est préfixe d'un message paddé, Lever $\mathsf{Drapeau}_3$
3. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) Si $\mathcal{O} = \mathcal{C}$ et $x \xrightarrow{M} y \in E_{\mathcal{D}}$ a été défini par un appel à $\mathcal{R}^{-1}$, Lever $\mathsf{Drapeau}'_4$
   (b) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (c) Retourner (A', B'), où $y = (A', B', C)$
4. Si $\exists \mu, \mathcal{M}$ tel que $x_0 \xrightarrow{\mu\,[\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{C}}]} x$ et $\mu \| M = \mathsf{pad}(\mathcal{M})$
   (a) $B' = \mathcal{H}(\mathcal{M})$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
5. Sinon
   (a) $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
6. $y = (A', B', C)$
7. Si $x_0 \xrightarrow{\mu\,[\mathcal{G}]} x$ et $y \in V_{\mathcal{C}} \cup V_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_1$
8. Si $\mathcal{O} = \mathcal{C}$ et $x \xrightarrow{M} y \notin E_{\mathcal{D}}$ et $\exists y \xrightarrow{M'} z \in E_{\mathcal{D}}$ d'indice $j > i$ et $\mu \| M$ est préfixe d'un message paddé avec $x_0 \overset{\mu}{\hookrightarrow} x$, Lever $\mathsf{Drapeau}'_3$
9. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
10. Retourner (A', B') à $\mathcal{O}$

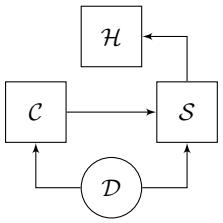


FIGURE 5.14 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 4

Jeu 4. L'oracle aléatoire $\mathcal{H}$ remplace la partie B de la simulation de $\mathcal{F}$ pour les noeuds situés à l'extrémité de chemins complets. Le simulateur est décrit en Figure 5.14. On a montré que si $\mathsf{Échec}_1$ ne se produit pas, le graphe est stable et donc la vue de l'attaquant n'est pas modifiée par l'introduction de l'oracle aléatoire. De plus la probabilité de $\mathsf{Échec}_1$ n'est pas modifiée, d'où $\Pr\left[W_4 \wedge \neg\mathsf{Échec}_1\right] = \Pr\left[W_3 \wedge \neg\mathsf{Échec}_1\right]$. On introduit un nouveau drapeau $\mathsf{Drapeau}_4$ et

Simulation de $\mathcal{F}^{-1}$
Entrée: (M, A', B', C), origine $\mathcal{O} = \mathcal{D}$
Sortie: (A, B)

1. $y = (A', B', C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) Si $x \xrightarrow{M} y \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_4$
   (b) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (c) Retourner (A, B), où $x = (A, B, C)$
3. $B \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
5. Si $x \in V_{\mathcal{D}} \cup V_{\mathcal{C}}$ Lever $\mathsf{Drapeau}_1$
6. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
7. Retourner (A, B) à $\mathcal{O}$

FIGURE 5.14 – Simulateur $\mathcal{S}$ de $\mathcal{F}^{-1}$ dans le jeu 4

l'évènement $\mathsf{Échec}_4$ défini par « $\mathsf{Drapeau}_4$ a été levé au cours du jeu », qui seront utiles pour analyser la différence de probabilité entre les jeux suivants.

Jeu 5. La construction est remplacée par une construction $\tilde{\mathcal{C}}$ qui exécute la construction mais ignore son résultat et obtient sa réponse de l'oracle aléatoire. Le simulateur n'est pas modifié par rapport au jeu 4. L'analyse du cas fonction reste valide dans le cas permutation, ainsi $\Pr\left[W_5 \wedge \neg\mathsf{Échec}_1\right] = \Pr\left[W_4 \wedge \neg\mathsf{Échec}_1\right]$.

Jeu 6. Dans le jeu 6, les appels de la construction au simulateur sont retardés et ne sont exécutés qu'à la fin du jeu. L'analyse faite dans le cas fonction idéale doit être adaptée pour prendre en compte le nouveau type de requêtes, faite par le distingueur à la permutation inverse. Pour ces requêtes, seules les méthodes de génération (1), répétition d'un arc précédent, et (2), simulation de $\mathcal{F}$, peuvent être utilisées pour déterminer la réponse, cf jeu 6 en section 5.2.3.1. L'oracle aléatoire $\mathcal{H}$ n'intervient jamais pour répondre à une telle requête.

Dans le cas où $\neg\mathsf{Échec}_1 \wedge \neg\mathsf{Échec}_3$ le cas des requêtes réalisées par le distingueur à $\mathcal{R}$ n'est pas modifiée.

Considérons à présent une requête destinée à $\mathcal{R}^{-1}$, dans le cas où $\neg\mathsf{Échec}_1 \wedge \neg\mathsf{Échec}_3$.

1. Si la requête correspond à un arc $x \xrightarrow{M} y$ défini précédemment on distingue deux cas :
   (a) Si l'arc a été défini par le distingueur, il est également défini par le distingueur dans le jeu 6.
   (b) Si l'arc a été défini par la construction dans le jeu 5, x a un chemin μ au moment de sa définition. Si cet arc est répété au cours du jeu 5 par le distingueur avec la requête considérée, il sera répété également dans le jeu 6 lors de la requête considérée, et l'analyse de la première requête du distingueur qui répète cet arc. Si cette requête est une requête à $\mathcal{R}$ on se ramène à l'analyse ci-dessus. Si cette requête est une requête à $\mathcal{R}^{-1}$, alors on a $x \xrightarrow{M} y \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ à l'étape 2a lors de cette requête. De plus, dans ce cas la distribution est nécessairement modifiée puisque y généré précédemment

aléatoirement est à présent donné par l'attaquant. La distribution est donc modifiée si $\mathsf{Échec}_4$ se produit.

2. Si la réponse est obtenue par simulation dans le jeu 5, elle sera également obtenue par simulation dans le jeu 6.

Il reste à étudier le cas des requêtes réalisées par la construction.

1. Le cas d'une réponse obtenue par répétition dans le jeu 5 ne change pas.
2. Le cas d'une réponse obtenue par simulation dans le jeu 5 ne change pas non plus, car si la réponse est définie par un appel à la permutation inverse (nouveau cas) elle est également obtenue par simulation.
3. Si la réponse à la requête est obtenue selon la méthode (3) dans le jeu 5, la distribution peut être modifiée. Dans le jeu 6, la réponse de la requête peut être obtenue soit selon la méthode (1), soit selon la méthode (3).
   (a) Si la méthode (3) est employée pour répondre dans le jeu 6, l'arc est défini de la même manière que dans le jeu 5.
   (b) Si la méthode (1) est employée dans le jeu 6, nécessairement l'arc a été défini par le distingueur et $j > i$, cf cas fonction idéale.
      i. Si l'appel j du distingueur a été effectué auprès de la permutation, on reprend l'analyse du cas fonction.
      ii. Si l'appel j du distingueur a été effectuée auprès de la permutation inverse, la distribution est modifiée. Dans ce cas, la réponse à l'appel j du distingueur dans le jeu 5 déclenche $\mathsf{Échec}_4$.

En résumé, on a montré que sous l'hypothèse $\neg\mathsf{Échec1} \wedge \neg\mathsf{Échec}_3 \wedge \neg\mathsf{Échec}_4$ dans le jeu 5, pour un même choix de jetons aléatoires le graphe obtenu est identique dans les jeux 5 et 6. Ceci permet de manière similaire au cas où $\mathcal{F}$ est une fonction de montrer que

$$\neg(\mathsf{Échec}_1)^5 \wedge \neg(\mathsf{Échec}_3)^5 \wedge \neg(\mathsf{Échec}_4)^5 \subset \neg(\mathsf{Échec}_1)^6 \wedge \neg(\mathsf{Échec}_3')^6 \wedge \neg(\mathsf{Échec}_4')^6.$$

L'inclusion inverse est obtenue en partant du jeu 6. On a donc

$$\neg(\mathsf{Échec}_1)^5 \wedge \neg(\mathsf{Échec}_3)^5 \wedge \neg(\mathsf{Échec}_4)^5 = \neg(\mathsf{Échec}_1)^6 \wedge \neg(\mathsf{Échec}_3')^6 \wedge \neg(\mathsf{Échec}_4')^6,$$

et les graphes définis par le simulateur dans les deux jeux sont identiques quand cet évènement se produit. Par conséquent la vue du distingueur est identique dans les deux jeux et on a

$$\Pr\left[W_5 \wedge \neg(\mathsf{Échec}_1)^5 \wedge \neg(\mathsf{Échec}_3)^5 \wedge \neg(\mathsf{Échec}_4)^5\right] = \Pr\left[W_6 \wedge \neg(\mathsf{Échec}_1)^6 \wedge \neg(\mathsf{Échec}_3')^6 \wedge \neg(\mathsf{Échec}_4')^6\right].$$

En appliquant le lemme de différence entre les jeux 5 et 6 on obtient

$$|\Pr[W_6] - \Pr[W_5]| \leq \Pr\left[(\mathsf{Échec}_1)^5\right] + \Pr\left[(\mathsf{Échec}_3)^5 \vee (\mathsf{Échec}_4)^5\right].$$

De plus

$$\Pr\left[W_5 \wedge \neg\mathsf{Échec}_1\right] = \Pr\left[W_4 \wedge \neg\mathsf{Échec}_1\right] = \Pr\left[W_3 \wedge \neg\mathsf{Échec}_1\right].$$

On peut donc appliquer une nouvelle fois le lemme de différence entre les jeux 3 et 5 pour obtenir

$$|\Pr[W_5] - \Pr[W_3]| \leq \max\left(\Pr\left[(\mathsf{Échec}_1)^3\right], \Pr\left[(\mathsf{Échec}_1)^5\right]\right).$$

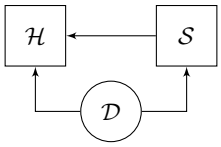
Jeu 7. Le jeu 7 est identique au jeu 7 du cas fonction auquel on ajoute le simulateur de $\mathcal{F}^{-1}$. Le graphe $\mathcal{G}_{\mathcal{C}}$ restant vide, on peut supprimer le code du simulateur qui s'y rapporte. Aucune décision n'étant prise à partir de $\mathsf{Drapeau}_1$ on supprime également le test susceptible de le lever. Le simulateur de la permutation est décrit en Figure 5.15.

Initialisation de $\mathcal{S}$

1. $V = \{x_0\}$ et $E = \emptyset$

Simulation de $\mathcal{F}$
Entrée: (M, A, B, C)
Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\exists x \xrightarrow{M} y \in E$
   (a) Retourner (A', B'), où $y = (A', B', C)$
3. Si $\exists \mu, \mathcal{M}$ tel que $x_0 \overset{\mu}{\hookrightarrow} x$ et $\mu \| M = \mathsf{pad}(\mathcal{M})$
   (a) $B' = \mathcal{H}(\mathcal{M})$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. Sinon
   (a) $B' \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A' \xleftarrow{\$} \{0,1\}^{\ell_a}$
5. $y = (A', B', C)$
6. $V = V \cup \{x, y\}$ et $E = E \cup \{x \xrightarrow{M} y\}$
7. Retourner (A', B')



Simulation de $\mathcal{F}^{-1}$
Entrée: (M, A', B', C)
Sortie: (A, B)

1. $y = (A', B', C)$
2. Si $\exists x \xrightarrow{M} y \in E$ Retourner (A, B) où $x = (A, B, C)$
3. $B \xleftarrow{\$} \{0,1\}^{\ell_h}$ et $A \xleftarrow{\$} \{0,1\}^{\ell_a}$
4. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
5. $V = V \cup \{x, y\}$ et $E = E \cup \{x \xrightarrow{M} y\}$
6. Retourner (A, B)

FIGURE 5.15 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ et $\mathcal{F}^{-1}$ dans le jeu final

En mettant bout à bout les résultats obtenus, on a montré

$$\begin{aligned}\left|\Pr\left[\mathcal{D}^{Q} = 1\right] - \Pr\left[\mathcal{D}^{Q'=1}\right]\right| \;\leq\; & \max\left(\Pr\left[(\mathsf{Échec}_1)^3\right], \Pr\left[(\mathsf{Échec}_1)^5\right]\right) \\ & + \Pr\left[(\mathsf{Échec}_1)^5\right] + \Pr\left[(\mathsf{Échec}_3 \vee \mathsf{Échec}_4)^5\right].\end{aligned}$$

5.3 Bornes d'indifférentiabilité

Dans les deux sections précédentes, on a détaillé une séquence de jeux permettant de borner l'avantage maximum d'un distingueur cherchant à faire la différence entre le mode général proposé en section 5.1 instancié avec une primitive idéalisée, et d'un oracle aléatoire utilisé avec une

simulation de la primitive. Afin de borner cet avantage maximal, on doit borner la probabilité d'occurence d'évènements définis dans la séquence de jeux. La probabilité de certains de ces évènements peut être bornée indépendamment de l'encodage de message utilisé. On qualifie de *générales* ces bornes. La probabilité d'autres évènements peut être affectée par les propriétés de l'encodage de message. Pour ces évènements, on donne des bornes en considérant deux types de fonctions d'encodage de message :

- un encodage de message générique, par exemple un padding non ambigu,
- un encodage sans préfixe, *i.e.* un encodage tel que tout message de l'image de cet encodage ne contient pas de préfixe strict appartenant également à cette image.

Les preuves des résultats présentés sont calculatoires et reportées en annexe A.

5.3.1 Bornes générales d'indifférentiabilité

Dans les bornes présentées, on note $n = \ell_a + \ell_h$ la taille de sortie de $\mathcal{F}$ et N le nombre maximum de requêtes faites au simulateur dans le jeu où le distingueur est confronté à la construction et à la fonction idéalisée.

Borne de $\mathsf{Échec}_1$. On énonce des bornes sur la probabilité d'occurence de l'évènement $\mathsf{Échec}_1$.

Lemme 4 *Pour $\mathcal{F}$ fonction ou permutation paramétrée, pour tout jeu où l'évènement $\mathsf{Échec}_1$ est considéré*

$$\Pr\left[\mathsf{Échec}_1\right] \leq N^2 2^{-n}.$$

L'évènement $\mathsf{Échec}_1$ correspond à l'occurence d'une collision sur l'état interne de la construction qui peut conduire à une collision sur les sorties du système qui ne se produirait qu'avec une probabilité beaucoup plus faible pour un oracle aléatoire.

Borne de $\mathsf{Échec}_3$ et $\mathsf{Échec}_4$. Afin de borner les probabilités de $\mathsf{Échec}_3$ et $\mathsf{Échec}_4$, qui correspondent à la probabilité de succès d'une attaque en extension de message, on commence par établir le lemme général suivant qui décrit la probabilité d'occurence de t-collisions.

Lemme 5 *Pour $1 \leq t \leq N$, soit $t\mathsf{Coll}(N,k)$ l'évènement « il existe au moins t occurences identiques dans un ensemble de N valeurs tirées aléatoirement dans un ensemble $\mathcal{E}$ selon une distribution D satisfaisant pour tout $\tilde{x} \in \mathcal{E}, \Pr_{x \xleftarrow{\$} D}[x = \tilde{x}] \leq 2^{-k}$ ». Alors, on a*

$$\sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] \leq 2k + (1+e)N2^{-k} ,$$

où $e = \exp(1) \approx 2.71828$. De plus si $N \leq 2^\omega$ pour un entier ω, on a

$$\sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] \leq \left\lceil \frac{k + \omega - \log_2(\sqrt{2\pi})}{k - \omega - \log_2 e} \right\rceil .$$

Le lemme suivant relie $\mathsf{Échec}_4$ aux multicollisions.

Lemme 6 *Pour $\mathcal{F}$ permutation paramétrée, la probabilité de l'évènement $\mathsf{Échec}_4$ après au plus N appels à $(\mathcal{R}, \mathcal{R}^{-1})$ satisfait*

$$\Pr\left[\mathsf{Échec}_4\right] \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}\right] ,$$

où $t\mathsf{Coll}$ *est l'évènement « il existe au moins* t *occurrences identiques dans un ensemble de* N *élements tirés de manière indépendante et uniformément distribués dans* $\{0,1\}^{\ell_h}$ *».*

5.3.2 Bornes pour un encodage de message arbitraire

Lorsqu'on borne la probabilité de l'évènement $\mathsf{Échec}_3$, les propriétés de l'encodage de message peuvent jouer un rôle. On énonce les bornes suivantes sur la probabilité de $\mathsf{Échec}_3$ quand au plus N requêtes sont soumises à $(\mathcal{R}, \mathcal{R}^{-1})$ au cours du jeu, pour un encodage de message arbitraire.

Le lemme suivant relie $\mathsf{Échec}_3$ aux multicollisions pour un encodage de message arbitraire.

Lemme 7 *Lorsque* $\mathcal{F}$ *est une fonction ou une permutation paramétrée, la probabilité de* $\mathsf{Échec}_3$ *après au plus* N *appels à* $\mathcal{R}, \mathcal{R}^{-1}$ *satisfait*

$$\Pr\left[\mathsf{Échec}_3\right] \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}\right] \ ,$$

où $t\mathsf{Coll}$ *est défini comme au lemme 6.*

De plus dans le cas permutation paramétrée, comme $\mathsf{Drapeau}_3$ ne peut être levé que lors d'une requête à $\mathcal{R}$ et $\mathsf{Drapeau}_4$ ne peut être défini pendant une requête à $\mathcal{R}^{-1}$, on a

$$\Pr\left[\mathsf{Échec}_3 \vee \mathsf{Échec}_4\right] \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}\right].$$

En utilisant tous les résultats précédents, en utilisant la borne précédente sur la probabilité des t collisions avec $k = \ell_h$ et $\omega = \ell_a$, on obtient le théorème suivant sur l'indifférentiabilité à un oracle aléatoire de notre construction de fonction de hachage dans le modèle de l'oracle aléatoire ou de la fonction de chiffrement idéale.

Théorème 9 *Considérons le mode décrit en Figure 5.1 et le simulateur* $\mathcal{S}$ *défini dans le dernier jeu de la séquence de jeux. Alors pour tout distingueur* $\mathcal{D}$ *interagissant avec* $\mathcal{Q}$ *et faisant au plus* N *appels à* $\mathcal{F}$ *si* $\mathcal{F}$ *est une fonction ou* N *appels à* $(\mathcal{F}, \mathcal{F}^{-1})$ *si* $\mathcal{F}$ *est une permutation paramétrée, on a*

$$\mathsf{Adv}(\mathcal{D}) \leq (3+e)2^{-(\ell_h+\ell_a)}N^2 + \ell_h 2^{-(\ell_a-1)}N \ . \tag{5.1}$$

De plus, pour $\ell_h > \ell_a$*, on a*

$$\mathsf{Adv}(\mathcal{D}) \ \leq \ 2^{-(\ell_h+\ell_a-1)}N^2 + 2^{-\ell_a}N\left\lceil \frac{\ell_h + \ell_a - \log_2(\sqrt{2\pi})}{\ell_h - \ell_a - \log_2 e} \right\rceil \tag{5.2}$$

et $\mathsf{Adv}(\mathcal{D})$ *est la borne la plus petite entre* (5.1) *et* (5.2)*.*

Il est intéressant de remarquer que pour notre mode d'opération, les bornes exposées ci-dessus ne font pas intervenir ℓ_c. Cette partie de l'état interne n'améliore donc pas la sécurité de la construction dans le contexte de l'indifférentiabilité. D'autres contextes, comme la résistance en seconde-préimage, ont été étudiés dans un modèle idéalisé dans le document de soumission de Shabal [34] et voient leur borne de sécurité améliorée par ℓ_c.

Le corollaire qui suit se concentre sur le cas de la construction Chop-MD, construction importante de par sa diffusion et son usage dans les fonctions de hachage les plus récentes :

Corollaire 1 *Pour $\mathcal{F}$ une fonction, le mode général décrit en Figure 5.1 avec $\ell_c = 0$ et une fonction d'insertion égale à l'identité correspond à la construction Chop-MD avec une variable de chaînage de n bits et une taille de haché de ℓ_h bits. Pour tout distingueur $\mathcal{D}$ qui réalise au plus N appels à la fonction $\mathcal{F}$, soit directement, soit au travers de la construction, on a*

$$\mathsf{Adv}(\mathcal{D}) \leq (3+e)2^{-n}N^2 + \ell_h 2^{-(n-\ell_h-1)}N \; . \tag{5.3}$$

De plus, lorsque $\ell_h > n/2$,

$$\mathsf{Adv}(\mathcal{D}) \;\; \leq \;\; 2\cdot 2^{-n}N^2 + 2^{-(n-\ell_h)}N\left\lceil \frac{n-\log_2(\sqrt{2\pi})}{2\ell_h - n - \log_2 e}\right\rceil \; . \tag{5.4}$$

et la borne supérieure sur l'avantage de $\mathcal{D}$ est donnée par $\min((5.3),(5.4))$.

Notre résultat peut être comparé à la borne d'indifférentiabilité obtenue pour Chop-MD par Chang et Nandi [38]

$$\mathsf{Adv}(\mathcal{D}) \leq 2\cdot 2^{-n}N^2 + (3\ell_h+1)2^{-(n-\ell_h)}N + 2^{-(\ell_h-1)}N \; .$$

Cette borne n'est pertinente que dans le cas $\ell_h \geq n/2$, *i.e.* c'est à dire qu'au plus la moitié des bits de la variable de chaînage sont tronqués. Notre résultat donne une borne pour toute valeur de ℓ_h et améliore la borne de Chang-Nandi dans le cas $\ell_h \geq n/2$.

5.3.3 Bornes pour un encodage de message sans préfixe

Le lemme suivant donne une borne sur la probabilité de $\mathsf{Échec}_3$ pour un encodage de message sans préfixe.

Lemme 8 *Pour $\mathcal{F}$ fonction et un encodage de message sans préfixe, la probabilité de $\mathsf{Échec}_3$ après N appels à $\mathcal{R}$ est*

$$\Pr\left[\mathsf{Échec}_3\right] \leq 2^{-n}N^2.$$

En résumant les résultats précédents, en utilisant la borne générale sur la probabilité des multicollisions avec $k = \ell_h$ et $\omega = \ell_a$, on obtient le théorème suivant sur l'indifférentiabilité de notre mode générique lorsque la primitive $\mathcal{F}$ est une fonction idéale ou un chiffrement par bloc idéal, dans le cas d'un encodage de message sans préfixe :

Théorème 10 *Considérons le mode générique décrit en Figure 5.1 et le simulateur $\mathcal{S}$ défini dans le dernier jeu de nos séquences de jeux. Supposons de plus que l'encodage de message est sans-préfixe. Alors pour tout distingueur $\mathcal{D}$ interagissant avec $\mathcal{S}$ et réalisant au plus N appels à $\mathcal{F}$, soit directement, soit au travers de l'exécution de la construction, pour $\mathcal{F}$ fonction on a*

$$\mathsf{Adv}(\mathcal{D}) \leq 3\cdot 2^{-n}N^2 \; . \tag{5.5}$$

Pour tout distingueur $\mathcal{D}$ interagissant avec $\mathcal{S}$ et réalisant au plus N appels à $(\mathcal{F},\mathcal{F}^{-1})$, soit directement, soit au travers de la construction, pour $\mathcal{F}$ permutation paramétrée, on a

$$\mathsf{Adv}(\mathcal{D}) \leq (3+e)2^{-(\ell_h+\ell_a)}N^2 + \ell_h 2^{-(\ell_a-1)}N \; . \tag{5.6}$$

De plus, dans ce cas, lorsque $\ell_h > \ell_a$, on a

$$\mathsf{Adv}(\mathcal{D}) \;\; \leq \;\; 2\cdot 2^{-(\ell_h+\ell_a)}N^2 + 2^{-\ell_a}N\left\lceil \frac{\ell_h+\ell_a-\log_2(\sqrt{2\pi})}{\ell_h-\ell_a-\log_2 e}\right\rceil \tag{5.7}$$

et l'avantage de $\mathcal{D}$ est borné par la plus petite valeur entre (5.6) *et* (5.7).

Chapitre 6

Preuve d'indifférentiabilité dans le cas biaisé

Sommaire

Dans le chapitre précédent, nous avons donné une preuve d'indifférentiabilité de l'oracle aléatoire d'un mode opératoire basé sur une fonction de compression ou une permutation paramétrée lorsque cette primitive sous-jacente est considérée comme idéale, *i.e.* choisie comme une fonction ou une permutation paramétrée aléatoire. Nous étudions dans ce chapitre l'impact sur la sécurité de la fonction de hachage de la découverte de propriétés écartant la primitive sous-jacente d'une primitive idéale, en donnant une modélisation de ces propriétés et en montrant comment on peut en tenir compte dans une preuve de sécurité dans un modèle idéalisé adapté. Les résultats obtenus permettent de démontrer la robustesse d'extensions de domaine vis-à-vis de l'existence de distingueurs contre les fonctions sous-jacentes.

6.1 Fonctions non idéales

Dans cette section, nous commençons par présenter quelques attaques en distingueur contre des fonctions internes à des fonctions de hachage candidates à la compétition SHA-3. La mise en lumière de tels distingueurs participe à l'effort de cryptanalyse des candidats. Dans leur quête d'attaques contre les fonctions de hachage proposées, les cryptanalystes s'efforcent de trouver des comportements non-idéaux et de les exploiter afin de remettre en cause une des propriétés de sécurité attendues d'une fonction de hachage. La compétition SHA-3 a ceci de particulier

que de nombreux distingueurs contre des primitives sous-jacentes ont été exposés **sans** que cela soit suivi dans les faits par une attaque contre la fonction de hachage complète. La mise en évidence de distingueurs et de propriétés non idéales devient une fin en soi, sans que l'impact sur la sécurité ne soit systématiquement analysé. Nous montrons dans la section suivante que ces distingueurs peuvent être incorporés à l'analyse de sécurité. Au prix d'un abaissement de la borne de sécurité, on peut étendre les preuves du cas idéal pour prendre en compte les distingueurs de la fonction sous-jacente.

6.1.1 Attaques en distingueur sur fonction de tour

On présente ici quelques distingueurs contre des composants de fonctions de hachage candidates à la compétition SHA-3. Ces distingueurs permettent d'identifier une différence de comportement entre une fonction de compression ou une permutation paramétrée notée $\mathcal{F} : (M, X) \rightarrow \mathcal{F}(M, X) = \mathcal{F}_M(X)$. On identifie deux parties distinctes dans l'entrée de ces fonctions, une partie notée M correspondant à un bloc de message, contrôlée directement par l'attaquant, et une partie X correspondant à un état interne [1], sur lequel l'attaquant n'a qu'un contrôle indirect, au travers de le fonction $\mathcal{F}$.

Points fixes. Knudsen, Matusiewicz et Thomsen [86] ont montré que, pour certaines valeurs X de l'entrée de la permutation paramétrée interne utilisée par Shabal, il est possible de trouver un paramètre M_X (fonction de X) tel que $\mathcal{F}_{M_X}(X)$ et X sont partiellement égaux.

Distingueurs différentiels. La fonction de compression de Hamsi-256 a des propriétés différentielles vérifiées avec probabilité 1. Par exemple, il est montré dans [35] que, pour tout M, les sorties Y_0, Y_1 correspondant à deux entrées X_0, X_1 différant uniquement aux positions 71 et 199 prennent les mêmes valeurs aux positions 228 et 230.

Distingueurs à somme nulle. Lorsqu'une fonction est de bas degré ou définie par itération de fonctions élémentaires de bas degré, il est possible d'identifier des distingueurs *à somme nulle* [5]. Une somme nulle de taille 2^k est un ensemble de 2^k entrées $\{X_1, \ldots, X_{2^k}\}$ tel que $\bigoplus_i X_i = 0$ et $\bigoplus_i \mathcal{F}(M, X_i) = 0$. Les distingueurs à somme nulle sont donc des distingueurs différentiels d'ordre supérieur. La valeur de $2^k - 1$ sorties de la fonction $\mathcal{F}$ détermine complètement une sortie additionnelle. De telles sommes nulles existent pour la fonction de compression de Hamsi-256 [4] et pour la permutation interne de Keccak [30, 32].

États symétriques. Pour des raisons de performance, afin notamment de pouvoir paralléliser facilement les calculs, les primitives sous-jacentes de certains candidats SHA-3 sont très fortement structurées. Ceci peut permettre d'identifier des sous-ensembles d'états, possédant une propriété de symétrie qui sont stables par la primitive considérée. On peut citer comme exemples CubeHash [63] et SIMD [29].

Distingueurs rotationnels. Les distingueurs rotationnels sont apparentés aux distingueurs différentiels. Au lieu d'être reliés par une différence donnée, les entrées étudiées se déduisent l'une de l'autre par une rotation donnée. Ce type de distingueur est particulèrement adapté aux fonctions qui s'appuient sur la composition d'opérations de type rotation, addition modulaire et xor, comme Skein et Shabal.

1. ou une partie d'un état interne, selon le mode d'extension de domaine utilisé

6.1.2 Modélisation de fonctions biaisées

On souhaite étendre le modèle de l'indifférentiabilité pour prendre en compte, dans la mesure du possible, les distingueurs évoqués dans la section précédente. Pour ce faire, on commence par formaliser les distingueurs d'une primitive sous-jacente $\mathcal{F}$ comme relations implicites de probabilité significative. On qualifie d'ordre du distingueur le nombre de couples (entrée, sortie) nécessaire pour le décrire.

Définition 29 *Soit $\mathcal{F}$ une primitive sous-jacente d'une fonction de hachage et k un entier ≥ 1. On appelle distingueur de $\mathcal{F}$ d'ordre k et de probabilité p toute relation « implicite » $\mathcal{R}$ faisant intervenir k entrées de $\mathcal{F}$ $\{(M_i, X_i)\}_{i\in[\![0,k-1]\!]}$ et les sorties correspondantes $\{Y_i = \mathcal{F}(M_i, X_i)\}_{i\in[\![0,k-1]\!]}$ vérifiée avec probabilité p supérieure à la probabilité obtenue lorsque $\mathcal{F}$ est remplacée par une fonction aléatoire.*

La notion de relation « implicite » n'est pas définie formellement. Intuitivement, pour que de tels distingueurs puissent être pris en compte dans une preuve en indifférentiabilité, la fonction $\mathcal{F}$ doit pouvoir être idéalisée dans un certain sens. Des relations dont les définitions reposent sur la définition de la fonction $\mathcal{F}$, par exemple la relation $Y = \mathcal{F}(M, X)$ d'ordre 1 vérifiée avec probabilité 1, sont incompatibles avec les preuves d'indifférentiabilité.

Afin de pouvoir adapter nos techniques de preuves, nous verrons dans les sections suivantes que nous ne prendrons pas en compte tous les distingueurs suivant la définition ci-dessus. On se restreindra pour des raisons techniques aux distingueurs d'ordre inférieur ou égal à deux. Ceci prend en compte une bonne partie des distingueurs publiés dans le cadre de la compétition SHA-3 :

- **Distingueur d'ordre 1.** Les distingueurs de type « point fixe » et « états symétriques » entrent dans cette catégorie. Dans les deux cas on peut identifier des ensembles distingués d'entrées pour lesquelles la fonction a un comportement non-aléatoire. Pour les points fixes ce comportement non aléatoire est une égalité entre la sortie et une partie de l'entrée lorsque l'entrée est prise dans l'ensemble distingué.

  $$\mathcal{R}((X, M), Y) : X \in \tilde{\mathcal{X}} \Rightarrow X = Y \ [p].$$

  Pour les états symétriques, le comportement non aléatoire est la stabilité de l'ensemble distingué par la fonction $\mathcal{F}$.

  $$\mathcal{R}((X, M), Y) : X \in \tilde{\mathcal{X}} \Rightarrow Y \in \tilde{\mathcal{X}} \ [p].$$

- **Distingueur d'ordre 2.** Les distingueurs de type « différentiel » et « rotationnel » entrent dans cette catégorie.

  $$\mathcal{R}((X_0, M_0), Y_0, (X_1, M_1), Y_1) : X_0 \oplus X_1 = \delta_X \wedge M_0 \oplus M_1 = \delta_M \Rightarrow Y_0 \oplus Y_1 = \delta_Y \ [p].$$

  $$\mathcal{R}((X_0, M_0), Y_0, (X_1, M_1), Y_1) : X_1 = X_0 \ggg \alpha \wedge M_1 = M_0 \ggg \beta \Rightarrow Y_1 = Y_0 \ggg \gamma \ [p].$$

Lorsqu'il existe plusieurs distingueurs contre une fonction $\mathcal{F}$, il est également possible de les composer pour en déduire de nouveaux distingueurs. Ceci peut rendre fastidieux l'énumeration de tous les distingueurs possibles. On présente dans la section suivante une abstraction de la notion de distingueur qui

- est adaptée au cadre des preuves d'indifférentiabilité ;
- identifie les grandeurs caractéristiques des ensembles de distingueurs considérés.

6.2 Modèle de la fonction biaisée idéale

On étudie dans cette section la possibilité d'adapter les preuves d'indifférentiabilité du mode générique étudié au chapitre précédent quand la primitive sous-jacente est l'objet de distingueurs.

6.2.1 Problématique

Considérons une fonction de hachage $\mathcal{C}^{\mathcal{F}}$, obtenue par la combinaison d'une construction $\mathcal{C}$ réalisant des appels à une primitive sous-jacente $\mathcal{F}$. On suppose de plus que la construction $\mathcal{C}$ peut être prouvée indifférentiable d'un oracle aléatoire dans un modèle idéalisé, c'est-à-dire que $\mathcal{C}^{\mathcal{F}}$ se comporte de manière idéale quand $\mathcal{F}$ se comporte de manière idéale. Lorsqu'on construit une fonction de hachage basée sur la construction $\mathcal{C}$, on « instancie » la construction, c'est à dire on définit explicitement la fonction $\mathcal{F}$ qui est utilisée. Supposons à présent que cette fonction ne se comporte pas de manière idéale et que l'on puisse mettre en évidence des distingueurs tels que définis dans la section 6.1. À première vue, on ne peut plus tirer aucune garantie de sécurité de la preuve de la construction puisque l'hypothèse de base, le caractère idéal, ou du moins idéalisable, de $\mathcal{F}$ n'est plus respectée. La question de la sécurité de la fonction de hachage $\mathcal{C}^{\mathcal{F}}$ est donc posée.

Cette question s'inscrit dans une préoccupation plus générale, celle de construire des fonctions sûres à partir de briques de base ayant des défauts, ou encore d'affaiblir autant que possible les propriétés attendues de primitives sous-jacentes à des constructions cryptographiques. Il est effet désirable d'un point de vue conception de mettre en œuvre des modes assurant que la fonction obtenue est proche de l'idéal même quand la primitive sous-jacente s'en écarte. Cette robustesse de la construction permet

- soit d'accroître la marge de sécurité de la fonction, en la rendant résistante à la découverte de défauts sur ses composants internes ;
- soit d'obtenir de meilleures performances en mettant en œuvre une primitive interne moins coûteuse mais plus éloignée d'une primitive idéale.

Que ce soit à des fins de sécurité ou de performance, la question de l'impact sur la sécurité d'une construction cryptographique de distingueurs sur la primitive sous-jacente est une question primordiale pour le concepteur de fonctions cryptographiques.

6.2.2 Fonction biaisée idéale

Afin de simplifier l'exposé on se restreint au cas où la primitive sous-jacente est une fonction de compression $\mathcal{F} : \{0,1\}^{\ell_m+n} \to \{0,1\}^n$. On a vu dans la section précédente que l'existence de distingueurs rend problématique l'idéalisation d'une fonction de compression. Cette idéalisation consiste à remplacer la fonction de compression mise en œuvre par une fonction aléatoire, tirée uniformément parmi l'ensemble FUNC des fonctions de compression ayant les mêmes paramètres. Afin de prendre en compte les distingueurs tout en restant dans un cadre idéalisé on considère la proposition suivante : dans le modèle idéalisé, au lieu de tirer aléatoirement la fonction de compression parmi toutes les fonctions de mêmes paramètres, on tire dorénavant $\mathcal{F}$ parmi un sous-ensemble de fonctions $\text{FUNC}' \subset \text{FUNC}$ correspondant à l'ensemble des fonctions exhibant un biais donné. On cherche à appliquer ce formalisme pour adapter la preuve de sécurité donnée au chapitre précédent dans le cas idéal. Dans la suite de cette section, on identifie les points qu'il est nécessaire d'adapter dans la preuve et on donne une nouvelle représentation des distingueurs, en termes des distributions d'échantillonage qu'ils induisent.

6.2.3 Borne directe d'indifférentiabilité d'un oracle aléatoire

Lorsqu'on cherche à adapter les preuves d'indifférentiabilité d'un oracle aléatoire dans le cas biaisé, une première idée est de réutiliser le même simulateur que dans le cas idéal. La même séquence de jeux est utilisée. Cependant, entre le jeu 1, où le simulateur redirige les requêtes à la fonction $\mathcal{F}$, et le jeu 2, où le simulateur simule parfaitement les réponses aux requêtes, on se retrouve à présent confronté à une transition de type « changement de distribution ». Dans le cas idéal, les distributions ne sont pas modifiées à cause du caractère idéal de la fonction $\mathcal{F}$, c'est-à-dire que ces images sont tirées uniformément dans $\{0,1\}^n$. Une différence de taille entre le cas idéal et le cas biaisé est que dans le cas biaisé la distribution de la réponse de $\mathcal{F}$ à une requête n'est plus a priori indépendante des réponses antérieures. Il suffit pour s'en convaincre de considérer l'existence d'un distingueur d'ordre 2. La distribution utilisée lors de la requête q peut donc être écrite

$$D(u, \mathcal{L}(q)) = \{(A', B') = \mathcal{F}(u), \mathcal{F} \xleftarrow{\$} \mathrm{FUNC}'[\mathcal{L}(q)]\},$$

où $\mathcal{L}(q)$ désigne la liste des requêtes réalisées auprès du simulateur accompagnées des réponses produites par le simulateur avant la requête q et $\mathrm{FUNC}'[\mathcal{L}(q)]$ désigne le sous-ensemble des fonctions $\mathcal{F} \in \mathrm{FUNC}'$ vérifiant de plus $\mathcal{F}(u) = v$ pour tout $(u, v) \in \mathcal{L}(q)$. Dans ce cas la distance entre les deux jeux peut être bornée par N fois la distance statistique entre la distribution uniforme sur $\{0,1\}^n$ et la distributions $\mathcal{D}$, où N est le nombre de définitions de sortie de $\mathcal{F}$, comme vu en section 4.1.5.

Proposition 24 *Considérons le mode générique défini en section 5.1. Supposons que $\mathcal{F}$ soit une fonction aléatoire sélectionnée uniformément dans $\mathrm{FUNC}' \subset \mathrm{FUNC}$ et soit $\mathcal{H}$ un oracle aléatoire. On a*

$$\mathsf{Adv}(\mathcal{D}) \leq \mathsf{Adv}_{\mathrm{id}}(\mathcal{D}) + N \max \|D - U\|,$$

où $\mathsf{Adv}_{\mathrm{id}}(\mathcal{D})$ est la borne supérieure de l'avantage du distingueur quand $\mathcal{F}$ est une fonction de compression idéale et

$$\max \|D - U\| = \max_{u, \mathcal{L}, |\mathcal{L}| \leq N} \|D(u, \mathcal{L}) - U\|.$$

Cette borne, pouvant être obtenue de manière directe, n'apporte des garanties de sécurité que dans le cas où la distance statistique $\max \|D - U\|$ est petite. En particulier, elle n'apporte aucune garantie quand elle est utilisée dans des cas où la fonction interne possède des distingueurs vérifiés avec des probabilités proches ou égales à 1. Nous avons vu plus haut que ce cas de figure ne peut être ignoré. Afin d'obtenir des preuves de sécurité prenant en compte ces distingueurs et apportant des garanties non triviales, il faut donc adapter le simulateur au cas biaisé. Une telle adaptation est susceptible de réussir car

- pour les distingueurs d'ordre 1, la proportion d'états auxquels ils s'appliquent est en général faible ;
- pour les distingueurs d'ordre quelconque, l'utilisation d'un distingueur pour mettre en défaut la fonction de hachage nécessite un évènement de type collision interne, ce point apparaîtra plus clairement par la suite.

6.2.4 Représentation algorithmique d'une fonction biaisée idéalisée

On rappelle la forme de la fonction de compression utilisée par le mode générique défini en section 5.1 :

$$\begin{aligned} \mathcal{F} : \{0,1\}^{\ell_m} \times \{0,1\}^s &\rightarrow \{0,1\}^n \\ (M, A, B, C) &\mapsto \mathcal{F}_{M,C}(A, B) = (A', B') \ . \end{aligned}$$

On rappelle que M est le bloc de message, B la partie de l'état interne mise à jour par la fonction $\mathcal{F}$ susceptible d'être extraite comme valeur de haché, A le reste de la partie de l'état interne mise à jour par $\mathcal{F}$ et C la partie de l'état interne non mise à jour par $\mathcal{F}$. Si $\mathcal{F}_{M,C}(A, B) = (A', B')$, on note $\mathcal{F}^a_{M,C}(A, B) = A', \mathcal{F}^b_{M,C}(A, B) = B'$ les projections de $\mathcal{F}_{M,C}$ sur les parties A et B de l'état interne.

Lorsqu'on considère une fonction idéalisée parfaite, il y a deux manières équivalentes de voir la fonction :

- la fonction est tirée aléatoirement, uniformément dans l'ensemble FUNC de toutes les fonctions de même paramètre ;
- pour chaque entrée de la fonction, la sortie est tirée aléatoirement, uniformément parmi toutes les sorties possibles.

La deuxième manière d'envisager la fonction idéalisée permet de l'évaluer de manière « paresseuse » : on ne définit que les sorties correspondant aux entrées déduites des requêtes du distingueur $\mathcal{D}$. Elle est utilisée de manière implicite dans les simulateurs définis dans le chapitre 5. En fait on utilise de manière implicite les deux propriétés suivantes :

- on peut simuler les réponses de la fonction idéalisée indépendamment des requêtes précédentes ;
- on peut simuler séparément les parties A et B de la fonction idéalisée.

Dans le cas de fonctions biaisées idéalisées, la première manière de voir la fonction se transpose facilement, en remplaçant FUNC par un sous-ensemble FUNC$'$. Cependant, il n'existe plus de manière triviale d'évaluer de manière paresseuse la fonction. De plus les deux propriétés utilisées de manière implicite ne sont plus vérifiées :

- de par l'existence de distingueurs sur la primitive $\mathcal{F}$, la définition de sorties de $\mathcal{F}$ peut impacter la définition de paires (entrée, sortie) additionnelles ;
- fixer la partie B d'une sortie est susceptible d'introduire un biais sur la partie A correspondante.

Afin de pouvoir écrire des simulateurs similaires aux simulateurs définis dans le cas de fonctions non biaisées, on a besoin de pouvoir simuler les réponses de la fonction $\mathcal{F}$. On suppose que l'on dispose d'un algorithme efficace SimulerAB, qui étant donné une entrée (M, A, B, C) et un historique $\mathcal{L}$ génère aléatoirement la valeur $\mathcal{F}_{M,C}(A, B)$ sachant l'historique. La distribution des valeurs produites par SimulerAB est telle que, pour tout historique compatible avec la distibution de $\mathcal{F}$, on ait

$$\Pr\left[\mathsf{SimulerAB}(M, A, B, C, \mathcal{L}) = (A', B')\right] = \Pr_{\mathcal{F} \xleftarrow{\$} \mathrm{FUNC}'}\left[\mathcal{F}_{M,C}(A, B) = (A', B') | \mathcal{L}\right] .$$

En d'autres termes, SimulerAB évalue paresseusement la fonction $\mathcal{F}$.

Dans le cadre des preuves dans le cas idéal, on a également besoin de pouvoir simuler la partie A de la sortie, la partie B étant fixée par l'oracle aléatoire. On postule donc également l'existence d'un algorithme efficace SimulerA, qui étant donné une entrée (M, A, B, C), une partie B de sortie B' et un historique, génère aléatoirement la partie A de la sortie $\mathcal{F}_{M,C}(A, B)$ sachant

l'historique et sachant $\mathcal{F}^b_{M,C} = B'$. La distribution des valeurs produites par SimulerA est telle que, pour tout historique et toute valeur B' compatible avec la distribution de $\mathcal{F}$, on ait

$$\Pr\left[\mathsf{SimulerA}(M, A, B, C, B', \mathcal{L}) = A'\right] = \Pr_{\mathcal{F} \xleftarrow{\$} \mathrm{FUNC}'}\left[\mathcal{F}_{M,C}(A, B) = (A', B') | \mathcal{L} \wedge \mathcal{F}^b_{M,C}(A, B) = B'\right].$$

6.2.5 Quantification du biais d'une fonction idéalisée biaisée

Dans le cadre du cas idéal, on a bien identifié que la partie A et la partie B des sorties de $\mathcal{F}$ ne jouent pas le même rôle. Ceci est d'autant plus vrai dans le cas biaisé. En effet, la partie B d'une sortie de $\mathcal{F}$ est susceptible de servir de valeur de haché lorsque la construction est utilisée. Par conséquent, le biais sur cette partie de la sortie doit rester très faible, car le moindre biais statistique sur les valeurs de hachés conduit à un distingueur sur la construction. En fait, seules les parties B des requêtes intervenant dans le calcul d'un haché potentiel ne doivent pas présenter de biais. Il est donc possible de tolérer un biais important sur la partie B d'une fraction des entrées de $\mathcal{F}$, tant que ces entrées n'interviennent pas dans le calcul d'un haché. La partie A d'une sortie de $\mathcal{F}$ peut également être utilisée par un attaquant pour essayer de provoquer des évènements dont la probabilité est plus élevée que pour une fonction idéale. Cependant, ces évènements portent sur l'état interne de la construction qui est surdimensionné par rapport à la taille de sortie de la fonction de hachage et donc par rapport au paramètre de sécurité. On peut donc tolérer un biais plus important sur cette partie.

On quantifie ces biais au travers de deux paramètres, notés ε et τ. On s'efforcera de garantir dans les simulateurs que la distance statistique entre la distribution des parties B des sorties de $\mathcal{F}$ et la distribution uniforme de la partie B reste inférieure à ε, et que le biais sur la partie A ne perturbe pas trop la distribution de la sortie de $\mathcal{F}$, soit

$$\Pr\left[\mathcal{F}_{M,C}(A, B) = (A', B')\right] \leq 2^{-n+\tau}.$$

6.2.6 Requêtes biaisées

L'existence de distingueurs sur la fonction de compression ayant une probabilité de succès importante, parfois égale à 1, est courante pour des primitives sous-jacentes rencontrées dans des fonctions de hachage soumises à la compétition SHA-3. Cependant, elle ne menace habituellement pas la sécurité des fonctions de hachage basées sur ces primitives. Ces distingueurs permettent de dériver de l'information sur des paires (entrée, sortie) de la primitive sous-jacente à partir de paires (entrée, sortie) obtenues par ailleurs. Par exemple, pour un distingueur différentiel de probabilité 1, *i.e.* il existe (α, β) tel que $\mathcal{F}(x+\alpha) = F(x)+\beta$ pour tout x, la connaissance d'une paire $x, y = \mathcal{F}(x)$ permet de déduire $\mathcal{F}(x+\alpha) = y+\beta$. Les requêtes pour lesquelles la sortie est fortement biaisée par la connaissance de l'historique des requêtes antérieures doivent être traitées de manière spécifique. Ces requêtes, désignées sous le terme de *relatives* aux entrées de $\mathcal{L}$, sont définies par les relations ci-dessous, paramétrées par les biais acceptables ε et τ.

Définition 30 ((ε, τ)-relatif à un historique) *Soit ε un réel dans $[0, 1]$, τ un réel dans $[0, n]$ et $\mathcal{L}$ un ensemble de couples entrée-sortie de $\mathcal{F}$. L'ensemble des (ε, τ)-relatifs de $\mathcal{L}$ est défini comme*

$$\begin{aligned}\mathsf{Rel}(\mathcal{L}, \varepsilon, \tau) \quad = \quad & \{u = (M, A, B, C) \notin \mathcal{L}|_X \textit{ tels que } \|D_B(u, \mathcal{L}) - U\| > \varepsilon \\ & \textit{ou } \exists (A', B') \in \{0,1\}^n \textit{ tels que } \Pr\left[\mathcal{F}_{M,C}(A, B) = (A', B')\right] > 2^{-n+\tau}, \}\end{aligned}$$

où U est la distribution uniforme sur $\{0, 1\}^n$ et $D_B(u, \mathcal{L})$ est la distribution de la partie B des sorties de $\mathcal{F}$ sachant que $\mathcal{F}$ est tirée uniformément dans FUNC' *et étant donné l'historique $\mathcal{L}$.*

On a vu plus haut que dans une grande majorité des cas, les distingueurs peuvent être écrits comme une relation telle qu'une propriété sur les entrées de $\mathcal{F}$ implique une relation sur ses sorties. Dans ce cas de figure, les requêtes relatives d'un historique ne dépendent que des valeurs d'entrée présentes dans l'historique, *i.e.* de l'ensemble $V = \mathcal{L}|_X$. On se restreint aux distingueurs de cette forme et on note $\mathsf{Rel}(V, \varepsilon, \tau)$ l'ensemble des requêtes relatives à un historique constitué de paires (entrée, sortie) dont les entrées forment l'ensemble V.

L'ensemble $\mathsf{Rel}(\emptyset, \varepsilon, \tau)$ joue un rôle particulier. Il correspond aux entrées de $\mathcal{F}$ qui conduisent à un biais excessif, même si aucune information additionnelle n'est connue sur $\mathcal{F}$, en dehors des distingueurs qu'elle vérifie. Par exemple, dans le cas de la fonction Shabal, cet ensemble contient les « points fixes » exhibés dans [86].

Définition 31 (requêtes (ε, τ)-atypiques) *Soit ε un réel dans $[0, 1]$, τ un réel dans $[0, n]$. L'ensemble des requêtes (ε, τ)-atypiques est défini comme*

$$\mathtt{AT}(\varepsilon, \tau) \quad = \quad \mathsf{Rel}(\emptyset, \varepsilon, \tau)$$

Dans la construction du simulateur dans le cas biaisé, on a également besoin de déterminer de manière incrémentale les requêtes relatives de l'historique. Pour ce faire, un historique étant donné, on distingue les requêtes relatives introduites par une nouvelle requête.

Définition 32 (requêtes (ε, τ)-relatives à une requête x) *Soit ε un réel dans $[0, 1]$, τ un réel dans $[0, n]$. Soit X un ensemble d'entrées de $\mathcal{F}$ et x une entrée de $\mathcal{F}$. L'ensemble des requêtes (ε, τ)-relatives de x, conformément à X, noté $\mathcal{R}(x, X, \varepsilon, \tau)$, est défini comme*

$$\mathcal{R}(x, X, \varepsilon, \tau) = \{x\} \cup (\mathsf{Rel}(\{x\} \cup X, \varepsilon, \tau) \setminus \mathsf{Rel}(X, \varepsilon, \tau)) \cup \left\{x' | x \in \mathsf{Rel}(\{x'\} \cup X, \varepsilon, \tau) \setminus \mathsf{Rel}(X, \varepsilon, \tau)\right\}.$$

Ceci définit une relation symétrique.

Dans la suite on se limitera à l'étude de distingueurs tels que l'ensemble des requêtes relatives à une entrée est indépendant de l'historique, *i.e.* pour tous ensembles d'entrées (X, X'), pour toute entrée x, on a

$$\mathcal{R}(x, X, \varepsilon, \tau) = \mathcal{R}(x, X', \varepsilon, \tau) = \mathcal{R}(x, \varepsilon, \tau).$$

C'est en particulier le cas de distingueurs d'ordre 2 comme les distingueurs rotationnels et différentiels évoqués précédemment.

Dans nos preuves d'indifférentiabilité, les entrées de $\mathcal{F}$ sont décomposées en deux parties, une correspondant à la variable de chaînage, l'autre correspondant aux blocs de message. On postule l'existence d'algorithmes efficaces permettant de réaliser des tests d'existence de messages, ou couple de messages, qui permettent de compléter une variable de chaînage de manière à obtenir des entrées atypiques ou relatives :

- (ε, τ)-$\mathsf{EstAtypique}(y)$ renvoie vrai si et seulement si il existe M tel que $M, \mathsf{Insert}[M](y) \in \mathtt{AT}(\varepsilon, \tau)$. Par extension un noeud y pour lequel cet algorithme renvoie vrai sera qualifié d'*atypique*.
- (ε, τ)-$\mathsf{EstAutoRelatif}(y)$ renvoie vrai si et seulement si il existe $M \neq M'$ tels que $(M, \mathsf{Insert}[M](y)) \in \mathcal{R}((M', \mathsf{Insert}[M'](y)), \varepsilon, \tau)$. Par extension un noeud y pour lequel cet algorithme renvoie vrai sera qualifié d'*auto-relatif*.
- (ε, τ)-$\mathsf{SontRelatifs}_1(y, y')$ renvoie vrai si et seulement si il existe M, M' tels que $(M, \mathsf{Insert}[M](y)) \in \mathcal{R}((M', \mathsf{Insert}[M'](y')), \varepsilon, \tau)$. Si cet algorithme renvoie vrai, les noeuds y et y' sont qualifiés de *relatifs*.

- (ε,τ)-$\mathsf{SontRelatifs}_2(y,y',M')$ renvoie vrai si et seulement si il existe M tels que $(M,\mathsf{Insert}[M](y)) \in \mathcal{R}((M',\mathsf{Insert}[M'](y')),\varepsilon,\tau)$. Si cet algorithme renvoie vrai, on dit que y est *relatif* à la requête $M',\mathsf{Insert}[M'](y')$.

Finalement on définit quelques paramètres caractéristiques du biais qui seront utile pour borner l'avantage d'un distingueur dans la section suivante. On suppose ε et τ fixés.

- On note $\mathcal{N}_{\mathcal{R}}$ le nombre maximum de noeuds relatifs d'un noeud.

$$\mathcal{N}_{\mathcal{R}} = \max_x |\{y \text{ tels que } \mathsf{SontRelatifs}_1 x, y = \text{ Vrai}\}| .$$

- On note $\mathcal{N}_{\mathsf{atypique}}$ le nombre de noeuds atypiques

$$\mathcal{N}_{\mathsf{atypique}} = |\{x \text{ tels que } \mathsf{EstAtypique} x = \text{ Vrai}\}| .$$

- On note $\mathcal{N}_{\mathsf{autorel}}$ le nombre de noeuds auto-relatifs

$$\mathcal{N}_{\mathsf{autorel}} = |\{x \text{ tels que } \mathsf{EstAutoRelatif} x = \text{ Vrai}\}| .$$

6.3 Preuve d'indifférentiabilité dans le modèle de la fonction biaisée idéale

On étend dans cette section la preuve d'indifférentiabilité du chapite 5 pour prendre en compte les distingueurs d'une fonction de compression $\mathcal{F}$. On commence par expliquer la « stratégie » de construction du simulateur et la manière dont la preuve du cas idéal est complétée. On donne ensuite un simulateur de la primitive $\mathcal{F}$ et la borne d'indifférentiabilité associée. Enfin, on détaille la preuve d'indifférentiabilité en présentant la séquence de jeux permettant le calcul de la borne d'indifférentiabilité.

6.3.1 Modifications de la preuve du cas idéal

On doit adapter le simulateur du cas idéal car la borne d'indifférentiabilité que l'on obtient à partir de ce simulateur est triviale, du fait de l'existence de distingueurs. Une première modification correspond à simuler $\mathcal{F}$ non plus de manière aléatoire, mais en utilisant des algorithmes d'échantillonage $\mathsf{SimulerAB}$ et $\mathsf{SimulerA}$ qui rendent compte de la distribution de ses sorties. Une deuxième modification consiste à commencer par définir, pour chaque requête faite au simulateur, les requêtes relatives prolongeant un chemin. De cette manière, on peut mieux garantir la cohérence du simulateur avec l'oracle aléatoire.

6.3.2 Simulateur et borne d'indifférentiabilité

On considère le simulateur $\mathcal{S}$ de la fonction $\mathcal{F}$ défini en figure 6.1.

Pour ce distingueur, on peut obtenir une variante du théorème 9.

Proposition 25 *Soit $\mathcal{F}$ une fonction aléatoire de* FUNC′ *et $\mathcal{H}$ un oracle aléatoire. Soit $\varepsilon = 0$ et $\tau \leq \ell_a$ les valeurs des deux paramètres déterminant les biais excessifs, x_0 un noeud non atypique et non auto-relatif pour la fonction $\mathcal{F}$ et ces paramètres. On suppose enfin que pour des requêtes non-excessivement biaisées, la simulation de la partie A de $\mathcal{F}$ est indépendante de l'historique. Alors le simulateur $\mathcal{S}$ défini en figure 6.1 est tel que pour tout distingueur $\mathcal{D}$ réalisant au plus N requêtes à $\mathcal{F}$*

$$\mathsf{Adv}(\mathcal{D}) \leq 2(\Pr\left[\mathsf{Échec}^3\right] + 2N\mathcal{N}_{\mathcal{R}}\varepsilon) + \Pr\left[(\mathsf{Échec}_3)^5\right] ,$$

Initialisation de $\mathcal{S}$

1. $V = \{x_0\}$ et $E = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C)

Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Pour $\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C} \in \mathcal{R}((M, x))$ tel que $\exists x_0 \overset{\tilde{\mu}}{\hookrightarrow} \tilde{x}$, où $\tilde{x} = \mathsf{Insert}^{-1}[\tilde{M}](\tilde{A}, \tilde{B}, \tilde{C})$
   (a) Si $\nexists \tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \in E$
      i. Si $\exists \mathcal{M}$ tel que $\tilde{\mu} \| \tilde{M} = \mathsf{pad}(\mathcal{M})$
         A. $B' = \mathcal{H}(\mathcal{M})$
         B. $A' = \mathsf{SimulerA}(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C}, B', E)$
      ii. Sinon $(A', B') = \mathsf{SimulerAB}(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C}, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
      iii. $y = (A', B', \tilde{C})$
      iv. $V = V \cup \{x, y\}$ et $E = E \cup \{x \xrightarrow{M} y\}$
3. Si $\exists x \xrightarrow{M} y \in E$ sélectionner y
4. Sinon
   (a) $(A', B') = \mathsf{SimulerAB}(M, A, B, C, E)$
   (b) $y = (A', B', \tilde{C})$
5. $V = V \cup \{x, y\}$ et $E = E \cup \{x \xrightarrow{M} y\}$
6. Retourner (A', B') à $\mathcal{O}$

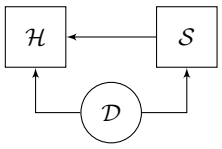


FIGURE 6.1 – Cas biaisé – Simulateur $\mathcal{S}$ de $\mathcal{F}$.

avec pour tout encodage de message

$$\Pr\left[\text{Échec}^3\right] \leq \left[\mathcal{N}_{\mathcal{R}}(\mathcal{N}_{\mathsf{atypique}} + \mathcal{N}_{\mathsf{autorel}} + 1 + \mathcal{N}_{\mathcal{R}})N + (\max(2, \mathcal{N}_{\mathcal{R}}) + 2\mathcal{N}_{\mathcal{R}}^2)\mathcal{N}_{\mathcal{R}}^2 N^2\right] 2^{-(n-\tau)}$$

pour un encodage de message arbitraire

$$\Pr\left[(\text{Échec}_3)^5\right] \leq \mathcal{N}_{\mathcal{R}} 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}\right] ,$$

où $t\mathsf{Coll}$ *est défini comme au lemme 6, et un encodage de message sans préfixe*

$$\Pr\left[(\text{Échec}_3)^5\right] \leq \mathcal{N}_{\mathcal{R}}^2 2^{-(n-\tau)} N^2.$$

On remarque que ce théorème ne s'applique que pour des formes de biais particulières, à savoir

- on ne tolère aucun biais sur la partie B ;
- pour les noeuds non excessivement biaisé, la simulation de la partie A est indépendante de l'historique.

Ces deux contraintes sont utilisées dans la preuve au moment où la dépendance du simulateur aux requêtes effectuées par la construction est retirée. Bien que $\varepsilon = 0$, on donne des bornes de manière générique pour les premiers jeux où cette contrainte n'est pas utilisée.

6.3.3 Résumé de la preuve

La propriété d'indifférentiabilité est prouvée en construisant une séquence de jeux pour construire progressivement $\mathcal{S}$. Les jeux successifs sont représentés schématiquement en figure 6.2 et peuvent être résumés de la manière suivante :

(0) On part du jeu original où le distingueur interagit avec le système $Q = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, où $\mathcal{C}$ est une implémentation de la construction. [2]

(1) La fonction aléatoire $\mathcal{F}$ est remplacée par un simulateur $\mathcal{S}$ qui se contente de transmettre les requêtes à $\mathcal{F}$ et renvoie les réponses obtenues de $\mathcal{F}$ à l'appelant, $\mathcal{C}$ ou $\mathcal{D}$. $\mathcal{S}$ construit le graphe correspondant aux couples (requête, réponse) qu'il voit passer.

(2-3) $\mathcal{S}$ simule $\mathcal{F}$ au lieu de réaliser des appels à $\mathcal{F}$. Dans le jeu 3, on peut éventuellement simuler plusieurs valeurs de $\mathcal{F}$ pour chaque requête à $\mathcal{S}$ pour tenir compte des valeurs excessivement biaisées.

(4) Lorsque c'est nécessaire, $\mathcal{S}$ fait des appels à $\mathcal{H}$ pour définir ses valeurs de sortie.

(5) $\mathcal{C}$ est remplacée par une construction $\tilde{\mathcal{C}}$ qui exécute $\mathcal{C}^{\mathcal{F}}$ sur ses entrées, en faisant des appels à $\mathcal{S}$ lorsqu'une évaluation de $\mathcal{F}$ est nécessaire, mais ignore les réponses obtenues et réalise un appel final à $\mathcal{H}$ pour obtenir la réponse à la requête. Les réponses des requêtes à $\mathcal{L}$ ne dépendent plus du simulateur.

(6) $\tilde{\mathcal{C}}$ est supprimée et les requêtes à $\mathcal{L}$ sont directement transmises à l'oracle aléatoire. Un composant du distingueur enregistre les requêtes à $\mathcal{L}$ et les exécute en fin de jeu. Les réponses aux requêtes à $\mathcal{R}$ ne dépendent plus des appels à $\mathcal{L}$.

(7) L'exécution finale de la construction est supprimée. Le distingueur interagit à présent avec le système final $Q' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

On reprend les conventions de la preuve dans le cas idéal (indexation d'évènement, levée de drapeau, sortie du jeu de sécurité, etc.)

2. On ne fait pas de distinction entre la construction $\mathcal{C}$ et un programme qui l'exécute.

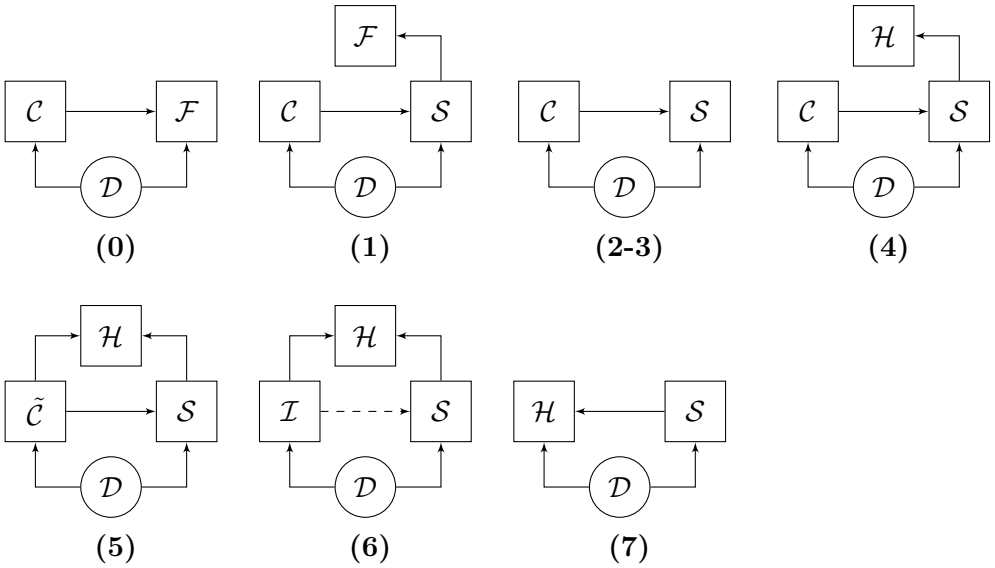


FIGURE 6.2 – Évolution des interactions entre oracles et simulateurs au cours de la preuve.

6.3.4 Séquence de jeux de la preuve

On adapte à présent la séquence de jeux présentée au chapitre 5. Pour toute la séquence de jeux, on fixe deux paramètres ε et τ, définis en section 6.2.5, permettant de séparer les requêtes dont les réponses sont excessivement biaisées et les requêtes pouvant être traitées de manière usuelle. Les ensembles $\mathcal{R}$ et $\mathtt{AT}$ définis en section 6.2.6 sont considérés pour ces paramètres ε et τ. Pour simplifier les notations, on omettra de les mentionner dans les simulateurs.

Jeu 0. Il s'agit du jeu de départ où $\mathcal{D}$ interagit avec $\mathcal{C}^{\mathcal{F}}$ et la fonction $\mathcal{F}$, qui est tirée aléatoirement, uniformément dans FUNC′. Par définition du jeu 0, on a

$$\Pr[W_0] = \Pr\left[\mathcal{D}^{\mathcal{Q}} = 1\right] .$$

Jeu 1. $\mathcal{F}$ est remplacée par un simulateur $\mathcal{S}$ présentant la même interface que $\mathcal{F}$, redirigeant les requêtes reçues vers $\mathcal{F}$ et retournant les réponses de $\mathcal{F}$ à l'appelant, soit $\mathcal{C}$ soit $\mathcal{D}$. Au cours du jeu, $\mathcal{S}$ construit progressivement les graphes $\mathcal{G}_{\mathcal{C}}$ et $\mathcal{G}_{\mathcal{D}}$ déduits des requêtes effectuées respectivement par $\mathcal{C}$ et $\mathcal{D}$. $\mathcal{S}$ est décrit en figure Fig. 6.3. Il est clair que $\Pr[W_1] = \Pr[W_0]$.

Jeu 2. On remplace l'appel à $\mathcal{F}$ par une simulation parfaite. Lorsque $\mathcal{S}$ a besoin de définir la valeur $\mathcal{F}(M, A, B, C)$ pour $(A, B, C) \in \mathcal{X}$, $\mathcal{S}$ appelle l'algorithme $\mathsf{SimulerAB}$. Par définition, cet algorithme prend pour entrée la requête courante (M, A, B, C) et la liste $\mathcal{L}$ de toutes les paires (requête, réponse) générées par le simulateur. Comme ces paires sont toutes représentées par un arc dans le graphe $\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{C}}$, on utilise l'ensemble d'arcs $E_{\mathcal{D}} \cup E_{\mathcal{C}}$ comme paramètre de $\mathsf{SimulerAB}$. [3] L'historique correspondant est défini par

$$\mathcal{L} = \{(M, A, B, C; A', B'),\ \mathsf{Insert}^{-1}[M](A, B, C) \xrightarrow{M} (A', B', C) \in E_{\mathcal{D}} \cup E_{\mathcal{C}}\} .$$

On décrit le nouveau simulateur en figure 6.4. Par définition de $\mathsf{SimulerAB}$, ceci ne modifie pas la distribution de la vue de l'attaquant. On a donc $\Pr[W_2] = \Pr[W_1]$.

3. et plus tard de $\mathsf{SimulerA}$.

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$

Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Appeler $\mathcal{F}$: $(A', B') = \mathcal{F}(M, A, B, C)$
3. $y = (A', B', C)$
4. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
5. Retourner (A', B') à $\mathcal{O}$

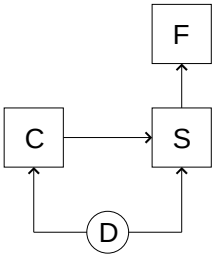


FIGURE 6.3 – Cas biaisé – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 1.

Initialization of $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$

Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
   (a) $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
   (b) Retourner (A', B'), où $y = (A', B', C)$
3. $(A', B') = \mathsf{SimulerAB}(M, A, B, C, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
4. $y = (A', B', C)$
5. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
6. Retourner (A', B') à $\mathcal{O}$

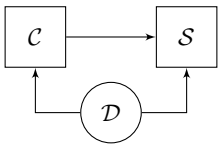


FIGURE 6.4 – Cas biaisé – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 2.

Jeu 3. On prépare à présent le simulateur à l'insertion de l'oracle aléatoire. Pour ce faire on détecte deux types d'évènements problématiques.

- Comme pour le cas idéal, on doit s'assurer que le nombre de chemins de chaque noeud ne change pas au cours du jeu pour garantir la consistance des définitions faites par l'oracle aléatoire. Dans le cas idéal, on vérifiait l'absence de collision interne entre noeud nouvellement défini et noeud déjà présent dans le graphe. Dans le cas biaisé, il faut ajouter un test additionnel. En effet, on n'insère pas systématiquement tous les relatifs d'une requête lors de sa définition. Cependant, cette définition engendre un biais sur la distribution des sorties des requêtes relatives. Elle « définit » indirectement les requêtes relatives. Une telle requête ne doit donc pas intervenir dans le calcul d'un haché. On s'assure que l'origine d'une telle requête n'acquiert pas de chemin au cours du jeu.
- Il faut de plus s'assurer que la distribution de sortie des arcs susceptibles de définir des valeurs de haché n'est pas excessivement biaisée. Pour ce faire, lors de l'introduction de chaque noeud susceptible de figurer comme avant dernier noeud d'un chemin complet, on s'assure que la distribution utilisée pour répondre à une requête postérieure ne présentera pas de biais excessif. On lève un drapeau d'erreur $\mathsf{DrapeauBiais}_i$ en cas de problème.

Pour pouvoir remplir ce deuxième objectif, on modifie également le simulateur pour donner la priorité à la simulation de requête prolongeant des chemins. En effet, une stratégie envisageable pour mettre en défaut le distingueur est de construire un chemin dans le graphe en s'arrêtant avant la dernière requête permettant d'obtenir un haché, puis de réaliser une requête relative de la requête complétant le chemin. On termine en réalisant un appel complétant le chemin. Si on simule directement la réponse de l'avant dernière requête, la réponse imposera une contrainte sur la réponse de la dernière requête, contrainte qui ne pourra pas être prise en compte par l'oracle aléatoire. Il faut donc commencer lors de l'avant dernière requête par compléter le chemin avant de définir, en accord avec la réponse à cette simulation la réponse à la requête adressée au simulateur.

Le simulateur de ce jeu est décrit en figure 6.5.

Les modifications entre le jeu 2 et le jeu 3 ne modifient pas la distribution de la vue de l'attaquant par définition de $\mathsf{SimulerAB}$. Elles ne changent éventuellement que le nombre de points et l'ordre des points où $\mathcal{F}$ est évalué. Ainsi $\Pr[W_3] = \Pr[W_2]$.

La notion de stabilité de graphe, définie dans le chapitre précédente, doit être adaptée au cas idéal pour prendre en compte l'existence de relatifs.

Proposition 26 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu de la suite de jeux décrite dans la preuve. On dit que le graphe est* stable *si :*

- *pour toute requête (M, A, B, C) au simulateur, $x = \mathsf{Insert}^{-1}[M](A, B, C)$, le nombre de chemins de x et de tout noeud relatif à (x, M) n'augmente pas après la requête en question ;*
- *pour tout noeud y inséré avec un chemin dans le graphe du simulateur, le nombre de chemins de y n'augmente pas après la requête.*

Si on compare cette définition à la propriété de stabilité du cas idéal, on constate qu'on distingue à présent les noeuds insérés comme origine et comme but d'arc, en imposant des contraintes plus fortes pour les origines d'arcs permettant de traiter les relatifs.Si le graphe est stable, aucun arc complétant un chemin n'a été défini avant le chemin qu'il complète, ni explicitement par simulation, ni implicitement comme relatif d'un arc présent dans le graphe.

On montre à présent que $\mathsf{Drapeau}_1$ permet de déterminer la non stabilité du graphe.

Proposition 27 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu susceptible de lever le drapeau $\mathsf{Drapeau}_1$. Si $\mathsf{Échec}_1$ ne se produit pas, alors $\mathcal{G}$ est stable.*

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$

Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Pour $\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C} \in \mathcal{R}((M, x))$ tel que $\exists x_0 \overset{\tilde{\mu}}{\hookrightarrow} \tilde{x}$, où $\tilde{x} = \mathsf{Insert}^{-1}[\tilde{M}](\tilde{A}, \tilde{B}, \tilde{C})$
   (a) Si $\exists \tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
      i. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{\tilde{x}, \tilde{y}\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}\}$
   (b) Sinon
      i. $(A', B') = \mathsf{SimulerAB}(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C}, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
      ii. $y = (A', B', \tilde{C})$
      iii. Si $y \in V_{\mathcal{C}} \cup V_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_1$
      iv. Si $\exists z \xrightarrow{M'} z' \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$ tel que $\mathsf{SontRelatifs}_2(y, z, M')$ est vrai Lever $\mathsf{Drapeau}_1$
      v. Si $\mathsf{EstAtypique}(y)$ Lever $\mathsf{DrapeauBiais}_1$
      vi. Si $\mathsf{EstAutoRelatif}(y)$ Lever $\mathsf{DrapeauBiais}_2$
      vii. Si $\exists x_0 \overset{\tilde{\mu}}{\hookrightarrow} \tilde{y}$ tel que $\mathsf{SontRelatifs}_1(y, \tilde{y})$ Lever $\mathsf{DrapeauBiais}_3$
      viii. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
3. $(A', B') = \mathsf{SimulerAB}(M, A, B, C, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
4. $y = (A', B', \tilde{C})$
5. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
6. Retourner (A', B') à $\mathcal{O}$

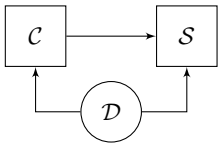


FIGURE 6.5 – Cas biaisé – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 3.

Démonstration : On démontre la preuve par contraposée. Supposons que $\mathcal{G}$ ne soit pas stable. On distingue 2 cas :

- le nombre de chemins vers un noeud relatif d'une requête augmente après le traitement de cette requête. Considérons l'entrée $(\tilde{M}, \tilde{x})$ pour laquelle un nouveau chemin vers une entrée relative d'une requête antérieure est créé. Le traitement de cette entrée crée un chemin, $\tilde{x}$ a donc un chemin. Considérons la valeur y définie à l'étape 2(b)ii. Un nouveau chemin vers une entrée relative d'une requête antérieure étant créé, soit y est dans le graphe et le traitement de l'entrée $\tilde{M}, \tilde{x}$ lève $\mathsf{Drapeau}_1$ à l'étape 2(b)iii, soit il existe un arc $x \xrightarrow{M} *$ dans le graphe et le traitement de l'entrée $(\tilde{M}, \tilde{x})$ lève $\mathsf{Drapeau}_1$ à l'étape 2(b)iv.
- le nombre de chemins vers un noeud $\tilde{y}$ ayant un chemin augmente après l'insertion de y dans le graphe. Le cas où $\tilde{y}$ correspond à une entrée de $\mathcal{F}$ a déjà été traitée. Si $\tilde{y}$ correspond à une sortie de $\mathcal{F}$, comme il possède un chemin, il a été inséré dans le graphe. Lors du traitement de l'entrée ajoutant un chemin supplémentaire vers $\tilde{y}$, $\tilde{y} \in V$ et $\mathsf{Drapeau}_1$ est levé à l'étape 2(b)iii.

□

On termine en prouvant des propriétés relatives à l'apparition d'entrées de requêtes biaisées avec un chemin dans le graphe.

Proposition 28 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu susceptible de lever le drapeau* $\mathsf{DrapeauBiais}_1$. *Si x_0 n'est pas un noeud atypique et si* $\mathsf{ÉchecBiais}_1$ *ne se produit pas, alors il n'y a pas de noeud atypique avec un chemin dans $\mathcal{G}$.*

Démonstration : On procède par contraposée. Soit un noeud atypique x ayant un chemin. Supposons $x \neq x_0$. Considérons la requête créant un chemin vers x. À l'étape 2(b)v, le drapeau $\mathsf{DrapeauBiais}_1$ est levé. □

Proposition 29 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu susceptible de lever le drapeau* $\mathsf{DrapeauBiais}_2$. *Si x_0 n'est pas un noeud auto-relatif et si* $\mathsf{ÉchecBiais}_2$ *ne se produit pas, alors il n'y a pas de noeud auto-relatif avec un chemin dans $\mathcal{G}$.*

La preuve est analogue à celles du cas noeuds atypiques.

Proposition 30 *Soit $\mathcal{G} = (V, E)$ le graphe construit par le simulateur à la fin d'un jeu susceptible de lever le drapeau* $\mathsf{DrapeauBiais}_3$. *Si* $\mathsf{ÉchecBiais}_3$ *ne se produit pas, alors pour toute paire de noeuds x, x' ayant un chemin dans le graphe,* $\mathsf{SontRelatifs}(x, x')$ *est faux.*

Cette proposition découle directement du test réalisé à l'étape 2(b)vii.

Jeu 4. On peut à présent insérer l'oracle aléatoire $\mathcal{H}$ dans le jeu. Lorsque $x_0 \overset{\mu}{\hookrightarrow} x$ et $\mu \| M$ est un message paddé, on répond à la requête $(M, \mathsf{Insert}[M](x))$ en appelant l'oracle aléatoire $\mathcal{H}$ pour définir la partie B et en complétant la réponse par un appel à l'algorithme $\mathsf{SimulerA}$. Cette modification est bien définie si $\mathsf{Échec}_1$ ne se produit pas car la stabilité du graphe garantit que x a au plus un chemin qui peut être extrait au moment où la requête $M, \mathsf{Insert}[M](x)$ est traitée. Le simulateur modifié est explicité en figure 6.6.

On note $\mathsf{Échec}$ l'évènement « l'un des drapeaux définis par ce simulateur, à l'exception du drapeau $\mathsf{Drapeau}_3$, est levé à la fin du jeu » et $\mathsf{Échec}_3$ l'évènement le drapeau $\mathsf{Drapeau}_3$ est levé à la fin du jeu.

Initialisation de $\mathcal{S}$

1. $V_{\mathcal{C}} = \{x_0\}$ et $E_{\mathcal{C}} = \emptyset$
2. $V_{\mathcal{D}} = \{x_0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée: (M, A, B, C), origine $\mathcal{O} = \mathcal{C}$ ou $\mathcal{D}$

Sortie: (A', B')

1. $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. Pour $\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C} \in \mathcal{R}((M, x))$ tel que $\exists x_0 \overset{\tilde{\mu}}{\hookrightarrow} \tilde{x}$, où $\tilde{x} = \mathsf{Insert}^{-1}[\tilde{M}](\tilde{A}, \tilde{B}, \tilde{C})$
   (a) Si $\mathcal{O} = \mathcal{D}$ et $\exists z \xrightarrow{M'} \tilde{x} \in E_{\mathcal{C}} \setminus E_{\mathcal{D}}$ avec $x_0 \overset{\mu}{\hookrightarrow} z$, et $\mu \| M'$ est préfixe d'un message paddé, Lever $\mathsf{Drapeau}_3$
   (b) Si $\exists \tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$
      i. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{\tilde{x}, \tilde{y}\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}\}$
   (c) Sinon
      i. Si $\exists \mathcal{M}$ tel que $\tilde{\mu} \| \tilde{M} = \mathsf{pad}(\mathcal{M})$
         A. $B' = \mathcal{H}(\mathcal{M})$
         B. $A' = \mathsf{SimulerA}(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C}, B', E_{\mathcal{D}} \cup E_{\mathcal{C}})$
      ii. Sinon $(A', B') = \mathsf{SimulerAB}(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{C}, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
      iii. $y = (A', B', \tilde{C})$
      iv. Si $y \in V_{\mathcal{C}} \cup V_{\mathcal{D}}$ Lever $\mathsf{Drapeau}_1$
      v. Si $\exists z \xrightarrow{M'} z' \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$ tel que $\mathsf{SontRelatifs}_2(y, z, M')$ est vrai Lever $\mathsf{Drapeau}_1$
      vi. Si $\mathsf{EstAtypique}(y)$ Lever $\mathsf{DrapeauBiais}_1$
      vii. Si $\mathsf{EstAutoRelatif}(y)$ Lever $\mathsf{DrapeauBiais}_2$
      viii. Si $\exists x_0 \overset{\tilde{\mu}}{\hookrightarrow} \tilde{y}$ tel que $\mathsf{SontRelatifs}_1(y, \tilde{y})$ Lever $\mathsf{DrapeauBiais}_3$
      ix. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
3. Si $\exists x \xrightarrow{M} y \in E_{\mathcal{D}} \cup E_{\mathcal{C}}$ sélectionner y
4. Sinon
   (a) $(A', B') = \mathsf{SimulerAB}(M, A, B, C, E_{\mathcal{D}} \cup E_{\mathcal{C}})$
   (b) $y = (A', B', \tilde{C})$
5. $V_{\mathcal{O}} = V_{\mathcal{O}} \cup \{x, y\}$ et $E_{\mathcal{O}} = E_{\mathcal{O}} \cup \{x \xrightarrow{M} y\}$
6. Retourner (A', B') à $\mathcal{O}$

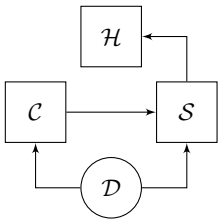


FIGURE 6.6 – Cas biaisé – Simulateur $\mathcal{S}$ de $\mathcal{F}$ dans le jeu 4.

On borne à présent la différence de distribution entre les sorties de $\mathcal{F}$ générées par le simulateur dans les jeux 3 et 4. Le seul cas de figure où la simulation est modifiée entre ces deux jeux est le cas de la définition d'une sortie par l'oracle aléatoire dans le jeu 4. Plaçons-nous à un instant où une telle requête est traitée et supposons que l'historique des requêtes et des simulations a été identique dans les deux jeux, on note E cet historique commun. Pour borner la distance statistique de la sortie de $\mathcal{F}$ dans chacun des jeux on estime la somme suivante

$$\begin{aligned}
& \sum_{A',B'} \left|\Pr\left[\mathsf{SimulerAB}(M,A,B,C,E)=(A',B')\right] - \Pr\left[\mathcal{H}(\mathcal{M})=B'\right]\Pr\left[\mathsf{SimulerA}(M,A,B,C,B',E)=A'\right]\right|, \\
\leq & \sum_{A',B'} \left|\Pr_{\mathcal{F}\xleftarrow{\$}\mathrm{FUNC}'}\left[\mathcal{F}_{M,C}(A,B)=(A',B')|E\right] - 2^{-\ell_h}\Pr_{\mathcal{F}\xleftarrow{\$}\mathrm{FUNC}'}\left[\mathcal{F}_{M,C}(A,B)=(A',B')|E\wedge\mathcal{F}^b_{M,C}(A,B)=B'\right]\right|, \\
\leq & \sum_{B'}\left(\left|\Pr_{\mathcal{F}\xleftarrow{\$}\mathrm{FUNC}'}\left[\mathcal{F}^b_{M,C}(A,B)=B'|E\right] - 2^{-\ell_h}\right|\sum_{A'}\Pr_{\mathcal{F}\xleftarrow{\$}\mathrm{FUNC}'}\left[\mathcal{F}^a_{M,C}(A,B)=A'|E\wedge\mathcal{F}^b_{M,C}(A,B)=B'\right]\right), \\
\leq & \sum_{B'}\left|\Pr_{\mathcal{F}\xleftarrow{\$}\mathrm{FUNC}'}\left[\mathcal{F}^b_{M,C}(A,B)=B'|E\right] - 2^{-\ell_h}\right|.
\end{aligned}$$

On peut donc borner la distance statistique entre les deux jeux par la distance statistique entre la partie B de $\mathcal{F}$ et la distribution uniforme. Sous l'hypothèse qu'aucun drapeau n'est levé, l'entrée de $\mathcal{F}$ n'est pas un noeud atypique, autorelatif ou relatif d'un noeud précédemment défini. Cette distance est donc bornée par ε, par définition des ensembles de relatifs.

Lors de chaque requête au simulateur, au plus $\mathcal{N}_\mathcal{R}$ requêtes sont simulées. La distance statistique entre les distributions des graphes obtenus après N requêtes au simulateur dans les jeux 3 et 4 est donc majorée par $\mathcal{N}_\mathcal{R}N\varepsilon$. Les tests de levée de drapeau s'appliquant de manière déterministe à partir du graphe, on en déduit

$$\left|\Pr\left[W_4\wedge\neg\mathsf{Échec}\right] - \Pr\left[W_3\wedge\neg\mathsf{Échec}\right]\right| \leq \mathcal{N}_\mathcal{R}N\varepsilon$$

et

$$\left|\Pr\left[\mathsf{Échec}^4\right] - \Pr\left[\mathsf{Échec}^3\right]\right| \leq \mathcal{N}_\mathcal{R}N\varepsilon.$$

Comme dans la preuve dans le cas idéal, on doit à présent supprimer les dépendances entre construction et simulateur. Le jeu 5 supprime la dépendance de la construction en le simulateur, le jeu 6 supprime la dépendance du simulateur en les requêtes réalisées auprès de la construction.

Jeu 5. Comme dans le cas idéal, la construction $\mathcal{C}$ est remplacée par une construction $\tilde{\mathcal{C}}$ avec une interface identique. Lorsque $\mathcal{D}$ effectue une requête $m\in\{0,1\}^*$ à $\mathcal{L}$, $\tilde{\mathcal{C}}$ procède en deux étapes : $\tilde{\mathcal{C}}$ commence par exécuter la construction $\mathcal{C}^\mathcal{F}$ sur m en effectuant des requêtes à $\mathcal{S}$ à chaque fois qu'une évaluation de $\mathcal{F}$ est requise ; puis $\tilde{\mathcal{C}}$ ignore le résultat de l'exécution de la construction et effectue un appel à $\mathcal{H}$ pour obtenir $h=\mathcal{H}(m)$ et retourne h à $\mathcal{D}$. Le simulateur $\mathcal{S}$ n'est pas modifié. L'analyse faite dans le cas idéal se transpose dans le cas biaisé. Un changement de distribution est susceptible de se produire entre les jeux 4 et 5 si le dernier appel d'une construction de haché a été défini précédemment par le simulateur sans faire intervenir directement l'oracle aléatoire. Si $\mathsf{Échec}_1$ ne se produit pas, un tel cas de figure n'est pas possible. L'exécution du simulateur n'étant pas modifiée, on a $\Pr\left[W_5\wedge\neg\mathsf{Échec}\right] = \Pr\left[W_4\wedge\neg\mathsf{Échec}\right]$.

Jeu 6. Comme dans le cas idéal, on modifie à présent le jeu de sécurité de telle sorte que les requêtes réalisées auprès du simulateur par la construction sont retardées et ne sont exécutées qu'à la fin du jeu. On renvoie au jeu 6 du cas idéal pour une description formelle des modifications.

On analyse à présent la différence de comportement entre les jeux 5 et 6. On montre que si l'évènement $\mathsf{Échec}\vee\mathsf{Échec}_3$ ne se produit pas dans le jeu 5, et qu'une hypothèse supplémentaire sur le biais de la fonction $\mathcal{F}$, à savoir que ce biais est indépendant de l'historique, est vérifiée,

alors la vue de l'attaquant n'est pas modifiée. Comme dans le cas idéal, l'évènement $\mathsf{Échec}_3$ peut être réexprimé dans le jeu 6 en terme de contrainte sur l'ordre des requêtes. On obtient alors

$$\Pr\left[W_6 \wedge \neg\mathsf{Échec} \wedge \neg\mathsf{Échec}_3\right] = \Pr\left[W_5 \wedge \neg\mathsf{Échec} \wedge \neg\mathsf{Échec}'_3\right].$$

On se contente de montrer ici que si $\mathsf{Échec}$ et $\mathsf{Échec}_3$ ne se produisent pas, alors le graphe produit dans le jeu 5 contient le graphe produit dans le jeu 6, sous l'hypothèse que les mêmes jetons aléatoires sont employés par le distingueur et la fonction $\mathcal{F}$.

On procède par récurrence sur le nombre d'arcs dans le graphe $\mathcal{G}_6$ obtenu en fin de jeu 6. Les requêtes de définition de $\mathcal{F}$ sont traitées dans l'ordre où elles apparaissent dans le jeu 6. On rappelle que les requêtes de définition de $\mathcal{F}$ provenant de $\mathcal{D}$ restent ordonnées, ainsi que les requêtes provenant de $\mathcal{C}$. Seul l'ordre relatif de requêtes provenant de $\mathcal{C}$ et de $\mathcal{D}$ est susceptible d'être modifié.

Si $\mathcal{G}_6$ est vide, aucune requête n'a été effectuée auprès du simulateur ni par le distingueur ni par la construction. On a donc $\mathcal{G}_5$ est vide et par conséquent la propriété est vraie. Supposons la propriété vérifiée pour les i premiers arcs de $\mathcal{G}_6$ et considérons le $i+1$-ème arc défini par $\mathcal{G}_6$. $x^* \xrightarrow{M^*} y^*$. On souligne qu'une sortie de $\mathcal{F}$ peut être définie par le simulateur de trois façons différentes :

- par répétition d'une valeur antérieure, dans ce cas la valeur a été définie précédemment d'une des deux manières qui suivent ;
- par simulation de A et B ;
- par appel à l'oracle aléatoire pour obtenir B et simulation de A.

On distingue les cas suivants :

1. l'arc $i+1$ provient d'une requête de la construction. Par définition de la construction, x^* a un chemin μ^* dans le graphe $\mathcal{G}_6$ au moment de la requête $i+1$. Par récurrence, ce chemin a été généré de la même manière dans $\mathcal{G}_5$. On distingue à présent les cas suivants :
   (a) dans le jeu 6, l'arc est traité par redéfinition d'un arc découlant d'une requête de la construction. Dans ce cas, comme l'ordre des requêtes provenant de la construction est conservé entre le jeu 5 et le jeu 6, l'arc est également obtenu par répétition dans le jeu 5. Par récurrence, il est défini de la même manière dans le jeu 5 et le jeu 6.
   (b) dans le jeu 6, l'arc est traité par redéfinition d'un arc découlant d'une requête du distingueur. On distingue deux sous-cas :
      i. si la requête du distingueur est antérieure à la requête de la construction dans le jeu 5, l'arc est également obtenu par répétition dans le jeu 5 et par récurrence, l'arc est généré de la même manière dans les jeux 5 et 6.
      ii. si la requête du distingueur est postérieure dans le jeu 5, l'arc n'est pas défini par rejeu mais par simulation totale ou en utilisant l'oracle aléatoire et une différence de comportement peut survenir si $\mu^*\|M^*$ est un message paddé. Or on constate que dans ce cas le traitement dans le jeu 5 de la requête provenant du distingueur entraînera la levée de $\mathsf{Drapeau}_3$ dans ce cas.
   (c) dans le jeu 6, l'arc est traité par simulation. Comme il a chemin, ceci signifie que $\mu^*\|M^*$ n'est pas un message paddé. μ^* étant également un chemin de x^* dans le jeu 5 au moment du traitement de cette requête, la manière de générer n'est pas modifiée
   (d) dans le jeu 6, l'arc est traité en utilisant l'oracle aléatoire. Pour les mêmes raisons, la manière de générer n'est pas modifiée.

2. l'arc $i+1$ provient d'une requête du distingueur. On distingue alors les cas suivants :
   (a) le noeud x^* a un chemin μ^* dans $\mathcal{G}_6$ au moment de la requête $i+1$. Par récurrence les noeuds de ce chemin sont générés de la même manière dans le jeu 5 et le jeu 6. On peut même remarquer que par définition de la construction retardée, tous les arcs constituant μ^* ont été définis par des appels provenant du distingueur. On distingue les sous-cas suivants :
      i. l'arc est traité par rejeu dans le jeu 6. Il s'agit nécessairement du rejeu d'un arc découlant d'une requête du distingueur. L'ordre des requêtes provenant du distingueur n'étant pas modifié, on en déduit par récurrence que la manière de générer l'arc n'est pas modifiée.
      ii. l'arc est traité par simulation dans le jeu 6. Comme il a un chemin, cela signifie que $\mu^* \| M^*$ n'est pas un message paddé. μ^* étant également un chemin de x^* au moment de traiter la requête dans le jeu 5, la manière de générer n'est pas modifiée.
      iii. l'arc est traité par l'oracle aléatoire dans le jeu 6. De la même manière que le sous-cas précédent, la manière de générer n'est pas modifiée.
   (b) le noeud x^* n'a pas de chemin dans $\mathcal{G}_6$. Dans ce cas on distingue les sous-cas suivants :
      i. s'il n'a pas de chemin dans $\mathcal{G}_5$, la réponse est générée de la même manière, soit par rejeu, soit par simulation.
      ii. si x^* a un chemin μ^* dans $\mathcal{G}_5$ au moment du traitement de cette requête dans le jeu 5, la distribution peut changer si $\mu^* \| M$ est un chemin complet. La condition x^* n'a pas de chemin dans $\mathcal{G}_6$ mais a un chemin dans $\mathcal{G}_5$ peut être réécrite dans le jeu 5 uniquement : au moment où l'on considère cette requête dans le jeu 5, x^* n'a pas de chemin dans $\mathcal{G}_D$ mais a un chemin dans G_C. On a vu dans la preuve du cas idéal que ceci entrainait l'existence d'un arc de $\mathcal{E}_C \setminus \mathcal{E}_\mathcal{D}$ dont x^* est le but. Le cas problématique déclenche donc la levée de $\mathsf{Drapeau}_3$ dans le jeu 5.

On a montré par cette étude de cas que la manière de générer une requête ne change pas si $\mathsf{Drapeau}_3$ n'est pas levé et si $\neg$Échec, condition suffisante pour assurer que les chemins utilisés lors de la détermination de la méthode de génération sont les mêmes dans les jeux 5 et 6. Il reste à s'assurer que la distribution de sortie n'est pas modifiée. On utilise ici les contraintes supplémentaires sur la nature du biais de $\mathcal{F}$. En effet, si la manière utilisée pour générer la réponse de $\mathcal{F}$ est identique, l'historique courant au moment de cette génération varie. Comme Échec ne se produit pas dans le jeu 5, il ne se produit pas dans le jeu 6. La requête est donc générée de manière non biaisée et dans ce cas la simulation est indépendante de l'historique. Si les mêmes jetons aléatoires sont utilisés, on obtient donc la même réponse, ce qui conclut la preuve par récurrence.

Jeu 7. On termine la preuve comme dans le cas idéal en supprimant l'exécution retardée de la construction. Celle-ci n'affectant pas la distribution de la vue de l'attaquant, on a

$$\Pr[W_6] = \Pr[W_7].$$

En supprimant du simulateur du jeu 7 toutes les mentions au graphe traçant les appels de la construction et les tests de levée de drapeau qui n'affectent pas la vue de l'attaquant, on obtient bien le simulateur présenté en figure 6.1. Ce jeu correspond au jeu final. En mettant bout à bout les résultats obtenus, en appliquant le lemme de différence entre les jeux 5 et 6 , on a montré

$$\left|\Pr\left[\mathcal{D}^{Q}=1\right]-\Pr\left[\mathcal{D}^{Q'=1}\right]\right| \leq 2(\Pr\left[(\mathsf{Échec})^{3}\right]+2N\mathcal{N}_{\mathcal{R}}\varepsilon)+\Pr\left[(\mathsf{Échec}_{3})^{5}\right].$$

On borne ces probabilités dans la section suivante.

6.3.5 Borne d'indifférentiabilité

On borne à présent les évènements intervenant au cours des jeux précédents et permettant de borner l'avantage d'un distingueur pour différencier la construction d'un oracle aléatoire pour le simulateur défini. À nouveau on distingue les bornes générales, indépendantes de l'encodage de message utilisé des bornes faisant intervenir l'encodage de message. Les preuves sont reportées en appendice A.

6.3.5.1 Bornes générales d'indifférentiabilité

Dans les bornes présentées, on note $n = \ell_a + \ell_h$ la taille de sortie de $\mathcal{F}$ et N le nombre maximum de requêtes faites au simulateur dans le jeu où le distingueur est confronté à la construction et à la fonction idéalisée. On reprend également les paramètres pertinents dérivés du biais de la fonction et les paramètres ε et τ choisi pour la preuve.

Borne de Échec. On énonce des bornes sur la probabilité d'occurrence de l'évènement $\mathsf{Échec}_1$.

Lemme 9 *Pour $\mathcal{F}$ fonction de compression, pour tout jeu où l'évènement $\mathsf{Échec}_1$ est considéré*

$$\Pr\left[\mathsf{Échec}_1\right] \leq \left[\mathcal{N}_{\mathcal{R}}(\mathcal{N}_{\mathsf{atypique}}+\mathcal{N}_{\mathsf{autorel}}+1+\mathcal{N}_{\mathcal{R}})N+(\max\left(2,\mathcal{N}_{\mathcal{R}}\right)+2\mathcal{N}_{\mathcal{R}}^{2})\mathcal{N}_{\mathcal{R}}^{2}N^{2}\right]2^{-(n-\tau)}$$

L'évènement $\mathsf{Échec}_1$ correspond à l'occurrence de collisions sur l'état interne de la construction qui peuvent conduire à des collisions sur les sorties du système qui ne se produiraient qu'avec une probabilité beaucoup plus faible pour un oracle aléatoire ou à la possibilité d'apparition au cours du calcul d'un haché d'une entrée dont le biais trop important permet de distinguer la construction d'un oracle aléatoire.

6.3.5.2 Bornes pour un encodage de message arbitraire

Lorsqu'on borne la probabilité de l'évènement $\mathsf{Échec}_3$, les propriétés de l'encodage de message peuvent jouer un rôle. On énonce les bornes suivantes sur la probabilité de $\mathsf{Échec}_3$ quand au plus N requêtes sont soumises à $(\mathcal{R})$ au cours du jeu, pour un encodage de message arbitraire.

Le lemme suivant relie $\mathsf{Échec}_3$ aux multicollisions pour un encodage de message arbitraire.

Lemme 10 *Lorsque $\mathcal{F}$ est une fonction de compression ou une permutation paramétrée, la probabilité de $\mathsf{Échec}_3$ après au plus N appels à $\mathcal{R}$ satisfait*

$$\Pr\left[\mathsf{Échec}_3\right] \leq \mathcal{N}_{\mathcal{R}}2^{-(\ell_a-\tau)}N\sum_{t=1}^{N}\Pr\left[t\mathsf{Coll}\right],$$

où tColl est défini comme au lemme 6.

6.3.5.3 Bornes pour un encodage de message sans préfixe

Quand l'encodage de message est sans préfixe, on peut améliorer la borne donnée dans la section précédente.

Lemme 11 *Pour $\mathcal{F}$ fonction de compression et un encodage de message sans préfixe, la probabilité de* $\mathsf{Échec}_3$ *après N appels au simulateur est*

$$\Pr\left[\mathsf{Échec}_3\right] \leq \mathcal{N}_{\mathcal{R}}^2 2^{-(n-\tau)} N^2.$$

6.3.5.4 Interprétation

Les bornes précédentes permettent d'identifier deux grandes catégories de distingueurs, en fonction de leur impact sur la *perte* au niveau de la borne d'indifférentiabilité :

- Les distingueurs basés sur des entrées particulières de $\mathcal{F}$. La perte au niveau de la borne d'indifférentiabilité intervient uniquement sur le terme linéaire de la borne et est linéaire en la grandeur caractéristique du distingueur. Elle correspond à l'apparition d'un état particulier au cours d'un calcul de haché. Tant que la taille de cet ensemble ($\mathcal{N}_{\mathsf{atypique}} + \mathcal{N}_{\mathsf{autorel}}$) reste faible devant le nombre de sorties de la primitive sous-jacente, les bornes obtenues restent intéressantes. Dans le cas de SIMD, l'existence d'états symétriques est prise en compte dans la preuve développée dans [29], qui obtient des résultats analogues.
- Les ditingueurs basés sur des relations entre différentes entrées de $\mathcal{F}$. On rappelle que la preuve développée dans cette section se restreint aux distingueurs basés sur les entrées. Les bornes que nous obtenons ici montrent que le nombre de requêtes relatives d'une requête donnée impacte le terme quadratique, et que la perte n'est pas linéaire en la grandeur caractéristique du distingueur, comme c'était le cas dans le cas des distingueurs basés sur des entrées particulières. Si $\mathcal{N}_{\mathcal{R}}$ devient important, en raison de l'existence de distingueurs pouvant être combinés de diverses manières, la borne de sécurité obtenue peut ne plus apporter de garantie pour un nombre de requêtes raisonnable.

Dans les deux cas, pour que les garanties apportées par la preuve restent suffisantes, il est nécessaire de disposer d'une marge de sécurité, *i.e.* que la primitive sous-jacente soit surdimensionnée par rapport au niveau de sécurité visé. L'emploi de preuves de sécurité dans le modèle de la fonction idéalisée biaisée permet d'évaluer si cette marge de sécurité est suffisante pour prendre en compte l'existence de distigueurs.

Bibliographie

[1] E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Seven-Property-Preserving Iterated Hashing : ROX. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2007. 80

[2] F. Armknecht, C. Carlet, P. Gaborit, S. Künzli, W. Meier, and O. Ruatta. Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks. In Vaudenay [136], pages 147–164. 38, 41, 42, 47

[3] G. Ars and J.-C. Faugère. Algebraic Immunities of functions over finite fields. Technical report, INRIA, Rapport de Recherche n°5532, 2005. 38, 41

[4] J.-P. Aumasson, E. Käsper, L.R. Knudsen, K. Matusiewicz, R. Odegaard, T. Peyrin, and M. Schläffer. Differential Distinguishers for the Compression Function and Output Transformation of Hamsi-256. Cryptology ePrint Archive, Report 2010/091, 2010. 110

[5] J.-P. Aumasson and W. Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. presented at the rump session of Cryptographic Hardware and Embedded Systems - CHES 2009, 2009. 110

[6] S. Babbage. A Space/Time Tradeoff in Exhausive Search Attacks on Stream Ciphers. In *European Convention on Security and Detection*, volume 408. IEE Conference Publication, mai 1995. 14

[7] S. Babbage, C. De Cannière A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen, and M. Robshaw. The eSTREAM Portfolio. eSTREAM, ECRYPT Stream Cipher Project, 2005. `http://www.ecrypt.eu.org/stream`. 12

[8] S. Babbage and M. Dodd. The stream cipher MICKEY (version 1). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/015, 2005. `http://www.ecrypt.eu.org/stream`. 10

[9] M. Backes and D. Unruh. Limits of Constructive Security Proofs. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2008. 73

[10] E. Barkan, E. Biham, and N. Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *J. Cryptology*, 21(3) :392–429, 2008. 8, 11, 18

[11] M. Bellare, A. Boldyreva, and A. Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2004. 66

[12] M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In X. Lai and K. Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006. 80, 81

[13] M. Bellare and P. Rogaway. Random Oracles are Practical : A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993. 66

[14] C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. Decim - A New Stream Cipher for Hardware Applications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/004, 2005. `http://www.ecrypt.eu.org/stream`. 10

[15] C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In Robshaw [120], pages 15–29. 20

[16] C. Berbain, H. Gilbert, and J. Patarin. QUAD : A Practical Stream Cipher with Provable Security. In Vaudenay [136], pages 109–128. 11

[17] D.J. Bernstein. Salsa20. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/025, 2005. `http://www.ecrypt.eu.org/stream`. 12

[18] D.J. Bernstein. Understanding Brute Force. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/036, 2005. `http://www.ecrypt.eu.org/stream`. 17

[19] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the Indifferentiability of the Sponge Construction. In N.P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. 77, 82

[20] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005. 75

[21] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In E.F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992. 22

[22] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`. 81

[23] A. Biryukov. A new 128 bit key stream cipher : LEX. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/013, 2005. `http://www.ecrypt.eu.org/stream`. 52

[24] A. Biryukov. The Design of a Stream Cipher LEX. In E. Biham and A. Youssef, editors, *Selected Areas in Cryptography – SAC 2006*, Lecture Notes in Computer Science. Springer, 2006. to appear. 52

[25] A. Biryukov and A. Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In T. Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000. 15

[26] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM J. Comput.*, 15(2) :364–383, 1986. 11

[27] D. Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003. 134, 138

[28] D. Boneh, A. Joux, and P. Nguyen. Why Textbook ElGamal and RSA Encryption are Insecure. In T. Okamoto, editor, *Advances in Cryptology — Proceedings of ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2000. 59

[29] C. Bouillaguet, P.-A. Fouque, and G. Leurent. Security Analysis of SIMD. In A. Biryukov, G. Gong, and D.R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010. 110, 130

[30] C. Boura and A. Canteaut. A Zero-Sum property for the Keccak-f Permutation with 18 Rounds. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT 2010)*, 2010. 110

[31] C. Boura and A. Canteaut. Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256. In A. Biryukov, G. Gong, and D.R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2010. 76

[32] C. Boura, A. Canteaut, and C. De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In A. Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011. 110

[33] A. Braeken, J. Lano, N. Mentens, B. Preneel, and I. Verbauwhede. SFINKS : A Synchronous Stream Cipher for Restricted Hardware Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/026, 2005. `http://www.ecrypt.eu.org/stream`. 10

[34] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Shabal, a submission to NIST'cryptographic hash algorithm competition. Submission to the NIST Hash competition, 2008. 86, 107

[35] C. Calik and M.S. Turan. Message Recovery and Pseudo-Preimage Attacks on the Compression Function of Hamsi-256. Cryptology ePrint Archive, Report 2010/057, 2010. 110

[36] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited (Preliminary Version). In *Symposium on Theory of Computing - STOC '98*, pages 209–218. ACM, 1998. Full version available at `http://arxiv.org/abs/cs.CR/0010019`. 66

[37] F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998. 50, 54, 75

[38] D. Chang and M. Nandi. Improved Indifferentiability Security Analysis of chopMD Hash Function. In K. Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2008. 86, 108

[39] P. Chose, A. Joux, and M. Mitton. Fast Correlation Attacks : An Algorithmic Point of View. In L.R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2002. 21, 22

[40] C. Cid and G. Leurent. An Analysis of the XSL Algorithm. In B.K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 333–352. Springer, 2005. 22

[41] D. Coppersmith, H. Krawczyk, and Y. Mansour. The Shrinking Generator. In *CRYPTO*, pages 22–39, 1993. 10

[42] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.*, 9(3) :251–280, 1990. 39

[43] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited : How to Construct a Hash Function. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005 : 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005. 80, 81

[44] N. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In P.J. Lee and C.H. Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2002. 22

[45] N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Boneh [27], pages 176–194. 22, 38, 46

[46] N. Courtois. Cryptanalysis of Sfinks. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/002, 2006. `http://www.ecrypt.eu.org/stream`. 38

[47] N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003. 38, 40, 41

[48] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Y. Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002. 22

[49] D.A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (3. ed.)*. Undergraduate texts in mathematics. Springer, 2007. 39

[50] J. Daemen, R. Govaerts, and J. Vandewalle. A Practical Approach to the Design of High Speed Self Synchronizing Stream Ciphers. In *Singapore ICCS/ISITA '92. 'Communications on the Move'*, volume 1, pages 279–283, 1992. 7

[51] J. Daemen and P. Kitsos. The Self-Synchronizing Stream Cipher Moustique. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018, 2005. `http://www.ecrypt.eu.org/stream`. 7

[52] I. Damgård. On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 17–27. Springer, 1989. 73, 77

[53] D.E. Denning. *Cryptography and Data Security*, page 100. Addison-Wesley, 1982. 17

[54] F. Didier. Using Wiedemann's Algorithm to Compute the Immunity Against Algebraic and Fast Algebraic Attacks. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2006. 38, 41, 42, 47

[55] F. Didier and J.-P. Tillich. Computing the Algebraic Immunity Efficiently. In Robshaw [120], pages 359–374. 38, 41, 42, 47

[56] distributed.net. Project RC5. `http://www.distributed.net/RC5`. 13

[57] H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996. 75

[58] H. Dobbertin. Cryptanalysis of MD5 compress, 1996. Eurocrypt 1996, Rump Session. 75

[59] ECRYPT. eBASH : ECRYPT Benchmarking of All Submitted Hashes. `//http://bench.cr.yp.to/ebash.html`. 76

[60] ECRYPT. eSTREAM : ECRYPT Stream Cipher Project, IST-2002-507932. `http://www.ecrypt.eu.org/stream`. 12, 38, 49

[61] ECRYPT. The Sha-3 Zoo. `http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo`. 76

[62] J.-C. Faugère and G. Ars. An Algebraic Cryptanalysis of Nonlinear Filter Generators using Gröbner Bases. Technical report, INRIA, Rapport de Recherche n°4739, 2003. 39, 40

[63] N. Ferguson, S. Lucks, and K.A. McKay. Symmetric States and their Structure : Improved Analysis of CubeHash. Cryptology ePrint Archive, Report 2010/273, 2010. 110

[64] S.R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In S. Vaudenay and A.M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001. 11

[65] T. Fuhr. Finding Second Preimages of Short Messages for Hamsi-256. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2010. 76

[66] B. Gammel, R. Göttfert, and O. Kniffler. The Achterbahn Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002, 2005. `http://www.ecrypt.eu.org/stream`. 10

[67] F.D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs. Dismantling MIFARE Classic. In S. Jajodia and J. López, editors, *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008. 10

[68] S. Goldwasser and Y.T. Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 102–113. IEEE Computer Society, 2003. 66

[69] J.Dj. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997. 14

[70] G. Gong and Y. Nawaz. The WG Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/033, 2005. `http://www.ecrypt.eu.org/stream`. 10

[71] M. Hasanzadeh, S. Khazaei, and A. Kholosha. On IV Setup of Pomaranch. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/082, 2005. `http://www.ecrypt.eu.org/stream`. 23

[72] P. Hawkes and G.G. Rose. Rewriting Variables : The Complexity of Fast Algebraic Attacks on Stream Ciphers. In M.K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2004. 22, 38, 46, 47

[73] M. Hell and T. Johansson. Breaking the F-FCSR-H Stream Cipher in Real Time. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 557–569. Springer, 2008. 12

[74] M.E. Hellman. A Cryptanalytic Time-Memory Tradeoff. *IEEE Transactions on Information Theory*, 26(4) :401–406, 1980. 14, 15

[75] J.J. Hoch and A. Shamir. Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In M. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006. 80

[76] T. Isobe, T. Ohigashi, H. Kuwakado, and M. Morii. A Chosen-IV Key Recovery Attack on Py and Pypy. *IEICE Transactions*, 92-D(1) :32–40, 2009. 23

[77] É. Jaulmes and F. Muller. Cryptanalysis of the F-FCSR Stream Cipher Family. In B. Preneel and S.E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2005. 23

[78] T. Johansson and F. Jönsson. Fast Correlation Attacks through Reconstruction of Linear Polynomials. In M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 2000. 21, 22

[79] A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. Franklin, editor, *Advances in Cryptology — Proceedings of CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004. 79, 80

[80] A. Joux and J.-R. Reinhard. Overtaking VEST. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2007. 23

[81] J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006. 80

[82] J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005. 80

[83] E.L. Key. An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators. *IEEE Transactions on Information Theory*, 22(6) :732–736, 1976. 42

[84] S. Kiyomoto, T. Tanaka, and K. Sakurai. K2 : A Stream Cipher Algorithm using Dynamic Feedback Control. In J. Hernando, E. Fernández-Medina, and M. Malek, editors, *SECRYPT*, pages 204–213. INSTICC Press, 2007. 10

[85] A. Klein. Attacks on the RC4 stream cipher. *Des. Codes Cryptography*, 48(3) :269–286, 2008. 11

[86] L. Knudsen, K. Matusiewicz, and S.S. Thomsen. Observations on the Shabal keyed permutation. Public comment on the NIST Hash competition, 2009. 110, 116

[87] D.E. Knuth. *The Art of Computer Programming, vol. II : Seminumerical Algorithms*. Addison-Wesley, 1969. 72

[88] H. Krawczyk, M. Bellare, and R. Canetti. HMAC : Keyed-Hashing for Message Authentication. RFC 2104 (Informational), 1997. `http://www.ietf.org/rfc/rfc2104.txt`. 73

[89] S. Lang. *Algebra*. Springer, 2002. 29

[90] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 2008. 26

[91] M. Liskov, R.L. Rivest, and D. Wagner. Tweakable Block Ciphers. *J. Cryptology*, 24(3) :588–613, 2011. 77

[92] S. Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive, Report 2004/253, 2004. `http://eprint.iacr.org/`. 81, 85

[93] J.L. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15(1) :122–127, 1969. 9, 33

[94] U.M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004. 68, 69

[95] W. Meier, E. Pasalic, and C. Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer, 2004. 38

[96] W. Meier and O. Staffelbach. Fast Correltaion Attacks on Stream Ciphers (Extended Abstract). In C.G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 1988. 20

[97] W. Meier and O. Staffelbach. Fast Correlation Attacks on Certain Stream Ciphers. *J. Cryptology*, 1(3) :159–176, 1989. 20

[98] W. Meier and O. Staffelbach. The Self-Shrinking Generator. In A. De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 1994. 10

[99] A. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. 9, 10, 18, 71, 73, 78

[100] R.C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989. 77

[101] B. Mourrain and O. Ruatta. Relations Between Roots and Coefficients, Interpolation and Application to System Solving. *J. Symb. Comput.*, 33(5) :679–699, 2002. 41

[102] F. Muller. Differential Attacks and Stream Ciphers. In *State of the Art in Stream Ciphers*. ECRYPT Network of Excellence in Cryptology, 2004. Workshop Record. 23

[103] National Institute of Standards and Technology. FIPS PUB 197 : Advanced Encryption Standard, 2001. 6, 22

[104] National Institute of Standards and Technology. FIPS Special Publication 800-38A, Recommandation for Block Cipher Modes of Operation, 2001. 6

[105] National Institute of Standards and Technology. FIPS Special Publication 800-38A, Recommandation for Block Cipher Modes of Operation : The XTS-AES Mode for Confidentiality on Storage Devices, 2010. 6

[106] K. Nohl and C. Paget. GSM - SRSLY ? 26th Chaos Communication Congress, 2009. 8, 11, 17

[107] P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In Boneh [27], pages 617–630. 14, 15

[108] National Bureau of Standards. FIPS PUB 46 : Data Encryption Standard, 1977. 6

[109] National Institute of Standards and Technology. FIPS PUB 81 : DES Modes of Operations, 1977. 6

[110] National Institute of Standards and Technology. Secure Hash Standard. FIPS Publication 180, 1993. 74

[111] National Institute of Standards and Technology. Secure Hash Standard. FIPS Publication 180-1, 1995. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`. 74

[112] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register, volume 72, no 212, 11 2007. 75

[113] P.J. Olver. On Multivariate Interpolation. Technical report, IN STUD. APPL. MATH, 2004. 36, 41

[114] S. O'Neil, B. Gittins, and H. Landman. VEST – Hardware-Dedicated Stream Ciphers. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/032, 2005. `http://www.ecrypt.eu.org/stream`. 49

[115] S. O'Neil, B. Gittins, and H. Landman. VEST Ciphers. eSTREAM, ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream/p2ciphers/vest/vest_p2.pdf`. 8, 49, 50, 51, 52, 55, 56, 61

[116] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. 79

[117] R. Rivest. The MD4 Message-Digest Algorithm. RFC 1320 (Informational), 1992. `http://www.ietf.org/rfc/rfc1320.txt`. 74

[118] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), 1992. `http://www.ietf.org/rfc/rfc1321.txt`. 74

[119] P. Rizomiliotis. Remarks on the New Attack on the Filter Generator and the Role of High Order Complexity. In S.D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*, pages 204–219. Springer, 2007. 42

[120] M.J.B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006. 132, 134, 140

[121] P. Rogaway. Formalizing Human Ignorance. In P.Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006. 73

[122] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics : Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In B.K. Roy and W. Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004. 73

[123] S. Rønjom, G. Gong, and T. Helleseth. On Attacks on Filtering Generators Using Linear Subspace Structures. In S.W. Golomb, G. Gong, T. Helleseth, and H.-Y. Song, editors, *SSC*, volume 4893 of *Lecture Notes in Computer Science*, pages 204–217. Springer, 2007. 38, 42

[124] S. Rønjom and T. Helleseth. A New Attack on the Filter Generator. *IEEE Transactions on Information Theory*, 53(5) :1752–1758, 2007. 38, 42, 46

[125] G. Rose, P. Hawkes, M. Paddon, and M. Wiggers de Vries. Primitive Specification for SSS. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/028, 2005. `http://www.ecrypt.eu.org/stream`. 7

[126] R. Schroeppel and A. Shamir. A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3) :456–464, 1981. 59

[127] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and Exploitation of New Biases in RC4. In A. Biryukov, G. Gong, and D.R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2010. 11

[128] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28 :656–715, 1949. 6

[129] Victor Shoup. Sequences of games : a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`. 69

[130] A. Sidorenko and B. Schoenmakers. Concrete Security of the Blum-Blum-Shub Pseudorandom Generator. In N.P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 355–375. Springer, 2005. 11

[131] T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Transactions on Computers*, 34(1) :81–85, 1985. 18

[132] L.R. Simpson, E. Dawson, J.Dj. Golić, and W. Millan. LILI Keystream Generator. In D.R. Stinson and S.E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2000. 10

[133] D.R. Stinson. disponible à `http://www.cacr.math.uwaterloo.ca/~dstinson/papers/polemic.ps`, 2001. 66

[134] V. Strassen. Gaussian Elimination is not Optimal. *Journal of Numerical Mathematics*, 13 :354–356, 1969. 39

[135] E. Tews, R.-P. Weinmann, and A. Pyshkin. Breaking 104 Bit WEP in Less Than 60 Seconds. In S. Kim, M. Yung, and H.-W. Lee, editors, *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2007. 8, 11, 18

[136] S. Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006. 131, 132

[137] X. Wang, H. Yu, and Y. Yin. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005 : 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005. 75

[138] X. Wang, H. Yu, and Y. Yin. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. 75

[139] D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/020, 2005. `http://www.ecrypt.eu.org/stream`. 8

[140] D.H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1) :54–62, 1986. 42

[141] H. Wu and B. Preneel. Resynchronization Attacks on WG and LEX. In Robshaw [120], pages 422–432. 23

[142] H. Wu and B. Preneel. Differential Cryptanalysis of the Stream Ciphers Py, Py6 and Pypy. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2007. 23

Annexe A

Calculs des bornes des preuves d'indifférentiabilité

A.1 Preuves des bornes d'indifférentiabilité des preuves dans le cas idéal

A.1.1 Preuve du lemme 4

Pour $1 \leq q \leq N$, On note $V(q)$ et $E(q)$ les ensembles V et E quand le simulateur reçoit la q-ième requête (de son point de vue) à $\mathcal{F}$ ou $(\mathcal{F}, \mathcal{F}^{-1})$. Il est clair que $|V(q)| \leq 2q-1$ et $|E(q)| \leq q-1$. On note $\Pr\left[\mathsf{Échec}_1(q)\right]$ la probabilité que $\mathsf{Drapeau}_1$ soit levé pendant le traitement de la q-ième requête. On commence par étudier le cas où $\mathcal{F}$ est une fonction de compression. Soit $u = (M, A, B, C)$ la requête considérée. On considère l'ensemble

$$D(u) = \{(A', B', C) \mid (A', B') \in \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h}\} .$$

Dans tout jeu de la séquence de jeu, le noeud y [1] défini pendant la simulation de la requête q avec pour entrée u est soit extrait du graphe $\mathcal{G}$, soit tiré uniformément dans l'ensemble $D(u)$. Dans le premier cas, $\Pr\left[\mathsf{Échec}_1(q)\right] = 0$. Dans le second cas, on considère $\tilde{y} = (\tilde{A}, \tilde{B}, \tilde{C})$ un élément fixé de $\mathcal{X}$. En prenant la probabilité sur le tirage uniforme de y dans $D(u)$ on a

$$\begin{aligned} \Pr[y = \tilde{y}] &= \Pr\left[(A', B') = (\tilde{A}, \tilde{B}) \wedge C = \tilde{C}\right] \\ &\leq 2^{-(\ell_a + \ell_h)} = 2^{-n}, \end{aligned}$$

du fait de l'indépendance de la partie C de y et des parties A et B. On en déduit que pour tout $\tilde{y} \in \mathcal{X}$,

$$\max_u \Pr[y = \tilde{y}] \leq 2^{-n}$$

d'où

$$\begin{aligned} \Pr\left[\mathsf{Échec}_1(q)\right] &\leq \max_u \Pr[\exists\, \tilde{y} \in V(q) : y = \tilde{y}] \\ &\leq 2^{-n}(2q-1) \end{aligned}$$

puisque $|V(q)| \leq 2q-1$. Ainsi on a

$$\begin{aligned} \Pr\left[\mathsf{Échec}_1\right] &= \sum_{q=1}^{N} \Pr\left[\mathsf{Échec}_1(q)\right] \\ &\leq 2^{-n} \sum_{q=1}^{N} (2q-1) \\ &\leq 2^{-n}\left(N(N+1) - N\right) \\ &\leq 2^{-n} N^2. \end{aligned}$$

1. la première valeur de y dans le jeu 2 du cas permutation paramétrée

On s'intéresse à présent au cas permutation paramétrée. Si la requête q est adressée à $\mathcal{R}$ la probabilité de $\mathsf{Échec}_1(q)$ est évaluée de la même manière. Supposons que la requête q soit adressée à $\mathcal{R}^{-1}$. Soit $v = (M, A', B', C)$ la requête effectuée. On considère l'ensemble

$$D'(v) = \{\mathsf{Insert}^{-1}[M](A, B, C) \mid (A, B) \in \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h}\} .$$

Le noeud x [2] défini pendant le traitement de la requête q est soit extrait de $\mathcal{G}$ soit tiré uniformément dans $D'(v)$. Dans le premier cas, $\Pr\left[\mathsf{Échec}_2(q)\right] = 0$. Dans le deuxième cas, on considère $\tilde{x}$ un élément de $\mathcal{X}$. Comme $\mathsf{Insert}[M]$ est une permutation pour tout M fixé, il existe un unique état $(\tilde{A}, \tilde{B}, \tilde{C})$ tel que $\mathsf{Insert}[M](\tilde{x}) = (\tilde{A}, \tilde{B}, \tilde{C})$. En prenant les probabilités sur la sélection uniforme de x dans $D'(v)$, on a

$$\begin{aligned} \Pr[x = \tilde{x}] &= \Pr\left[(A, B) = (\tilde{A}, \tilde{B}) \wedge C = \tilde{C}\right] \\ &\leq 2^{-(\ell_a + \ell_h)} = 2^{-n}, \end{aligned}$$

du fait de l'indépendance de la partie C et des parties A et B de $\mathsf{Insert}[M](x)$. On en déduit que pour tout $\tilde{x} \in \mathcal{X}$,

$$\max_v \Pr[x = \tilde{x}] \leq 2^{-n}$$

d'où

$$\begin{aligned} \Pr\left[\mathsf{Échec}_1(q) \mid \text{ requête } q \text{ est adressée à } \mathcal{R}^{-1}\right] &\leq \max_v \Pr[\exists\, \tilde{x} \in V(q) : x = \tilde{x}] \\ &\leq 2^{-n}(2q-1) \end{aligned}$$

puisque $|V(q)| \leq 2q - 1$.

Comme $\mathsf{Échec}_1(q)$ peut être borné de la même manière quand la requête q est adressée à $\mathcal{R}$ où $\mathcal{R}^{-1}$, on a

$$\Pr\left[\mathsf{Échec}_1(q)\right] \leq (2q-1)2^{-n}$$

et finalement

$$\begin{aligned} \Pr\left[\mathsf{Échec}_1\right] &= \sum_{q=1}^{N} \Pr\left[\mathsf{Échec}_1(q)\right] \\ &\leq 2^{-n} \sum_{q=1}^{N} (2q-1) \\ &\leq 2^{-n} N^2. \end{aligned}$$

A.1.2 Preuve du lemme 5

- **Borne générale.** Pour tout $t > 1$, on a

$$\begin{aligned} \Pr[t\mathsf{Coll}(N, k)] &\leq \binom{N}{t} \max_{\{i_1,\ldots,i_t\} \subset \{1,\ldots,N\}} \Pr_{x_{i_1},\ldots,x_{i_t} \xleftarrow{\$} D}[x_{i_1} = x_{i_2} = \ldots = x_{i_t}] \\ &\leq \binom{N}{t} 2^{-k(t-1)} \\ &\leq \frac{N^t}{\sqrt{2\pi t}} \left(\frac{e}{t}\right)^t 2^{-k(t-1)} . \end{aligned}$$

2. la première valeur de x dans le jeu 2 dans le cas permutation paramétrée

On pose

$$t_0 = 2k + \frac{eN}{2^k} ,$$

et on prouve que pour tout $t \geq t_0$ et tout N,

$$\frac{N^t}{\sqrt{2\pi t}} \left(\frac{e}{t}\right)^t 2^{-k(t-1)} \leq 2^{-k}. \tag{A.1}$$

En effet, (A.1) est équivalent à

$$t\left(\log_2 t + k - \log_2 N - \log_2 e\right) + \frac{1}{2}\log_2 t \geq 2k - \frac{1}{2}\log_2(2\pi).$$

Donc, pour $t \geq t_0$,

$$\log_2 t \geq \log_2\left(\frac{eN}{2^k}\right) + \log_2\left(1 + \frac{k2^{k+1}}{eN}\right) = \log_2 e + \log_2 N - k + \log_2\left(1 + \frac{k2^{k+1}}{eN}\right) ,$$

d'où

$$\begin{aligned} t\left(\log_2 t + k - \log_2 N - \log_2 e\right) &\geq 2k\left(1 + \frac{eN}{k2^{k+1}}\right)\log_2\left(1 + \frac{k2^{k+1}}{eN}\right) \\ &\geq 2k \end{aligned}$$

où la dernière égalité est déduite du fait que pour $x \geq 0$,

$$f : x \mapsto \left(1 + \frac{1}{x}\right)\log_2(1+x)$$

est croissante et $\lim_{x\to 0} f(x) = 1/\ln 2 > 1$. En utilisant $t \geq 2k \geq 1$, on déduit

$$t\left(\log_2 t + k - \log_2 N - \log_2 e\right) + \frac{1}{2}\log_2 t \geq 2k .$$

Par conséquent, on obtient

$$\begin{aligned} \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] &= \sum_{t=1}^{\lceil t_0\rceil - 1} \Pr\left[t\mathsf{Coll}(N,k)\right] + \sum_{t=\lceil t_0\rceil}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] \\ &\leq \lceil t_0\rceil - 1 + N2^{-k} , \end{aligned}$$

conduisant à

$$\sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] \leq 2k + (1+e)N2^{-k} .$$

- **Borne améliorée pour $N \leq 2^\omega$.**
  Lorsque $N \leq 2^\omega$, une autre borne peut-être obtenue en posant

$$t_1 = \frac{k + \omega - \log_2(\sqrt{2\pi})}{k - \omega - \log_2 e}.$$

Pour tout $t \geq \lceil t_1\rceil$ et tout $N \leq 2^\omega$, on a alors

$$\Pr\left[t\mathsf{Coll}(N,k)\right] \leq \frac{1}{N}.$$

Il suffit de prouver

$$t\,(\log_2 t + k - \log_2 N - \log_2 e) \geq \log_2 N + k - \log_2(\sqrt{2\pi}).$$

Par définition de t_1, on a pour tout $t \geq t_1$ et tout $N \leq 2^\omega$

$$\begin{aligned} t\,(\log_2 t + k - \log_2 N - \log_2 e) &\geq t\,(k - \omega - \log_2 e) \\ &\geq k + \omega - \log_2(\sqrt{2\pi}) \\ &\geq \log_2 N + k - \log_2(\sqrt{2\pi}). \end{aligned}$$

On en déduit que pour tout $N \leq 2^\omega$,

$$\begin{aligned} \sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] &\leq (\lceil t_1 \rceil - 1) + \sum_{t=\lceil t_1 \rceil}^{N} \frac{1}{N} \\ &\leq (\lceil t_1 \rceil - 1) + 1 = \lceil t_1 \rceil\,. \end{aligned}$$

On obtient finalement

$$\sum_{t=1}^{N} \Pr\left[t\mathsf{Coll}(N,k)\right] \leq \left\lceil \frac{k + \omega - \log_2(\sqrt{2\pi})}{k - \omega - \log_2 e} \right\rceil.$$

A.1.3 Preuve du lemme 6

On considère le graphe construit par notre simulateur après qu'il ait répondu à N requêtes. Soit r un entier et $r\mathsf{Coll}$ l'évènement correspondant à la situation suivante : il existe r arcs $x_i \xrightarrow{M_i} y_i$ dans $E_\mathcal{D} \cup E_\mathcal{C}$, pour $1 \leq i \leq r$, tels que tous les y_i, $1 \leq i \leq r$, ont la même partie B. De plus, on définit l'évènement $t\mathsf{Collmax}$ comme « t est la plus grande valeur r telle que $r\mathsf{Coll}$ est vraie ». Alors :

$$\begin{aligned} \Pr\left[\mathsf{Échec}_4\right] &= \sum_{t=1}^{N} \Pr\left[t\mathsf{Collmax}\right] \Pr\left[\mathsf{Échec}_4 \mid t\mathsf{Collmax}\right] \\ &= \sum_{t=1}^{N} \big(\Pr\left[t\mathsf{Coll}\right] - \Pr\left[(t+1)\mathsf{Coll}\right]\big) \Pr\left[\mathsf{Échec}_4 \mid t\mathsf{Collmax}\right] \end{aligned}$$

On calcule à présent une borne supérieure à la probabilité $\Pr\left[\mathsf{Échec}_4 \mid t\mathsf{Collmax}\right]$. En supposant que $(t+1)\mathsf{Coll}$ ne se produit pas après N requêtes, on calcule $\Pr\left[\mathsf{Échec}_4(q) \mid t\mathsf{Collmax}\right]$, la probabilité que $\mathsf{Drapeau}_4$ soit levé pour la première fois pendant la q-ième requête au simulateur.

Considérons l'ensemble $Y^\mathcal{C}$ des buts des arcs $\tilde{x} \xrightarrow{\tilde{m}} \tilde{y}$ de $E_\mathcal{C} \setminus E_\mathcal{D}$. On doit évaluer la probabilité avec laquelle une entrée choisie par le distingueur $y = (A', B', C)$ est dans l'ensemble $Y^\mathcal{C}(q)$. On partitionne $Y^\mathcal{C}(q)$ en 2^{ℓ_h} parties, selon la valeur de la partie B de ses éléments $\tilde{y}$. $\mathcal{D}$ peut avoir une connaissance partielle de $Y^\mathcal{C}(q)$. La partie B de $\tilde{y}$ est connue par $\mathcal{D}$ quand $\tilde{y}$ correspond à l'état final dans le calcul d'un haché par la construction. La partie C est connue si la partie B de l'état précédent est connue. On note $Y^\mathcal{C}(q, H)$ l'ensemble des états $\tilde{y}$ dans $Y^\mathcal{C}(q)$ où la partie B de $\tilde{y}$ est égale à H. On a alors

$$Y^\mathcal{C}(q) = \bigcup_{H \in \{0,1\}^{\ell_h}} Y^\mathcal{C}(q, H).$$

Comme on effectue l'analyse sachant que l'évènement $(t+1)\mathsf{Coll}$ ne se produit pas, pour tout H, la taille de $Y^{\mathcal{C}}(q,H)$ est majorée par t.

Pour un état fixé $z=(A_z,H,C_z)\in\mathcal{X}$ la probabilité qu'un noeud $\tilde{y}=(A'_{\tilde{y}},B'_{\tilde{y}},C_{\tilde{y}})$ dans $Y^{\mathcal{C}}(q)$ soit égal à z est

$$\begin{aligned}\Pr\left[y=z\right] &= \Pr\left[A'_{\tilde{y}}=A_z\wedge B'_{\tilde{y}}=H\wedge C_{\tilde{y}}=C_z\right],\\ &\leq \delta\left[B'_{\tilde{y}}=H\right]\Pr\left[A'_{\tilde{y}}=A_z\right],\\ &\leq \delta\left[B'_{\tilde{y}}=H\right]2^{-\ell_a},\end{aligned}$$

où la première égalité découle de l'indépendance des parties A et B des noeuds définis par le simulateur et la seconde inégalité découle de la sélection uniforme des parties A.

Pour un noeud $y=(A',B',C)$ choisi par le distingueur par le biais d'une requête q avec pour entrée $u=(M,A',B',C)$, la probabilité qu'un noeud $\tilde{y}=(A'_{\tilde{y}},H,C_{\tilde{y}})$ dans $Y^{\mathcal{C}}(q,H)$ partage la même partie A est

$$\begin{aligned}\Pr\left[A'=A'_{\tilde{y}}\right] &= \sum_{a\in\{0,1\}^{\ell_a}}\Pr\left[A'=a\wedge A'_{\tilde{y}}=a\right],\\ &= \sum_{a\in\{0,1\}^{\ell_a}}\Pr\left[A'=a\right]\Pr\left[A'_{\tilde{y}}=a\right],\\ &= 2^{-\ell_a}\sum_{a\in\{0,1\}^{\ell_a}}\Pr\left[A'_{\tilde{y}}=a\right],\\ &= 2^{-\ell_a},\end{aligned}$$

où la seconde égalité découle de l'indépendance des parties A de y et de $\tilde{y}$, et la troisième égalité de la sélection uniforme de la partie A de y. On est à présent prêt à borner $\Pr\left[\mathsf{Échec}_4(q)\mid t\mathsf{Collmax}\right]$:

$$\begin{aligned}\Pr\left[\mathsf{Échec}_4(q)\mid t\mathsf{Collmax}\right] &\leq \max_u \Pr\left[y\in Y^{\mathcal{C}}(q)\right],\\ &\leq \max_u \sum_{H\in\{0,1\}^{\ell_h}}\Pr\left[y\in Y^{\mathcal{C}}(q,H)\right],\\ &\leq \max_u \sum_{H\in\{0,1\}^{\ell_h}}\sum_{\tilde{y}\in Y^{\mathcal{C}}(q,H)}\Pr\left[y=\tilde{y}\right],\\ &\leq \max_u \sum_{H\in\{0,1\}^{\ell_h}}\sum_{\tilde{y}\in Y^{\mathcal{C}}(q,H)}\delta\left[y_B=H\right]\Pr\left[y_A=\tilde{y}_A\right],\\ &\leq \max_u \sum_{H\in\{0,1\}^{\ell_h}}|Y^{\mathcal{C}}(q,H)|\delta\left[y_B=H\right]2^{-\ell_a},\\ &\leq t2^{-\ell_a}.\end{aligned}$$

On en déduit immédiatement $\Pr\left[\mathsf{Échec}_4\mid t\mathsf{Collmax}\right]\leq tN2^{-\ell_a}$ et donc

$$\begin{aligned}\Pr\left[\mathsf{Échec}_4\right] &\leq \sum_{t=1}^{N}(\Pr\left[t\mathsf{Coll}\right]-\Pr\left[(t+1)\mathsf{Coll}\right])Nt2^{-\ell_a},\\ &\leq 2^{-\ell_a}N\sum_{t=1}^{N}t(\Pr\left[t\mathsf{Coll}\right]-\Pr\left[(t+1)\mathsf{Coll}\right]),\\ &\leq 2^{-\ell_a}N\sum_{t=1}^{N}\Pr\left[t\mathsf{Coll}\right],\end{aligned}$$

puisque la probabilité d'occurence d'une $N+1$-collision est exactement 0.

A.1.4 Preuve du lemme 7

La preuve du lemme est identique à la preuve du lemme 6 : on remplace simplement les mentions de $y = (A', B', C)$ provenant de requêtes $u = (M, A', B', C)$ par des mentions de $x = \mathsf{Insert}^{-1}[M](A, B, C)$ provenant de requêtes $u = (M, A, B, C)$.

A.1.5 Preuve du lemme 8

Pour un encodage de message arbitraire, lorsqu'on borne la probabilité d'occurrence de $\mathsf{Échec}_3$, on considère uniquement le fait que le noeud soumis par le distingueur est le but d'un arc de $E_{\mathcal{C}} \setminus E_{\mathcal{D}}$. Quand l'encodage de message est sans préfixe, la condition additionnelle qui détermine l'occurence de $\mathsf{Échec}_3$ peut être mise à contribution pour améliorer la borne. Considérons l'ensemble $Y^{\mathcal{C}}$ des buts des arcs de $E_{\mathcal{C}} \setminus E_{\mathcal{D}}$. Par définition de la construction, $y \in Y$ a un chemin μ_y. On note $Y^{\mathcal{C}}_{\mathsf{pf}}$ le sous-ensemble de Y tel qu'il existe un bloc de message M_y telle que $\mu_y || M_y$ est prefixe d'un message paddé. Il nous reste à évaluer la probabilité avec laquelle une entrée donnée $x = \mathsf{Insert}^{-1}[M](A, B, C)$ est dans $Y^{\mathcal{C}}_{\mathsf{pf}}(q)$. Considérons un élement y de $Y^{\mathcal{C}}_{\mathsf{pf}}(q)$. $\mu_y | M_y$ est préfixe d'un message paddé. Comme l'encodage de message est sans préfixe, μ_y n'est pas un message paddé. Par conséquent, le distingueur n'a pas d'information sur la partie B de y. Ainsi, la distribution des parties A et de B des noeuds sur lesquels portent les requêtes de $\mathcal{D}$ est indépendante de celle des parties A et B des noeuds de $Y^{\mathcal{C}}_{\mathsf{pf}}(q)$.

Pour un noeud $x = (A, B, C)$ choisi par le distingueur à travers la requête q avec pour entrée $u = (M, \mathsf{Insert}[M](x))$, la probabilité qu'un noeud $y = (A', B', C)$ de $Y^{\mathcal{C}}_{\mathsf{pf}}(q)$ partage les mêmes parties A et B est

$$\begin{aligned}
\Pr\left[x_A = y_A \wedge x_B = y_B\right] &= \sum_{a,b \in \{0,1\}^n} \Pr\left[A = a \wedge B = b \wedge A' = a \wedge B' = b\right], \\
&= \sum_{a,b \in \{0,1\}^n} \Pr\left[A = a \wedge B = b\right] \Pr\left[A' = a \wedge B' = b\right], \\
&= 2^{-n} \sum_{a,b \in \{0,1\}^n} \Pr\left[A = a \wedge B = b\right], \\
&= 2^{-n},
\end{aligned}$$

où la seconde égalité est déduite de l'indépendance des parties A et B de x et de y.
On est prêt à borner $\Pr\left[\mathsf{Échec}_3\right]$:

$$\begin{aligned}
\Pr\left[\mathsf{Échec}_3(q)\right] &= \max_u \Pr\left[x \in Y^{\mathcal{C}}_{\mathsf{pf}}(q)\right], \\
&\leq \max_u \sum_{y \in Y^{\mathcal{C}}_{\mathsf{pf}}(q)} \Pr\left[x = y\right], \\
&\leq \max_u \sum_{y \in Y^{\mathcal{C}}_{\mathsf{pf}}(q)} 2^{-n}, \\
&\leq (q-1)2^{-n},
\end{aligned}$$

car $|Y^{\mathcal{C}}_{\mathsf{pf}}(q)| \leq q - 1$. Par conséquent, on a

$$\begin{aligned}\Pr\left[\mathsf{Échec}_3\right] &\leq \sum_{q=1}^{N}\Pr\left[\mathsf{Échec}_3(q)\right]\\ &\leq 2^{-n}\sum_{q=1}^{N}(q-1)\\ &\leq 2^{-n}\left(\frac{N(N+1)}{2}-N\right)\\ &\leq 2^{-n}\frac{N^2}{2}.\end{aligned}$$

A.2 Preuves des bornes d'indifférentiabilité des preuves dans le cas biaisé

A.2.1 Preuve du lemme 9

Supposons que $\mathsf{Échec}$ ne se soit pas produit lors des q premières définitions de valeurs de sorties de $\mathcal{F}$. Considérons une nouvelle définition de valeurs de sorties de $\mathcal{F}$ susceptible de lever un des drapeaux définissant $\mathsf{Échec}$. Comme aucun des drapeaux n'a été levé, on a pour tout A^*, B^*

$$\Pr\left[(A',B')=(A^*,B^*)\right]\leq 2^{-(n-\tau)}.$$

Un des drapeaux est levé si (A', B') appartient un ensemble de noeuds constitué :

- des noeuds atypiques, au nombre de $\mathcal{N}_{\mathsf{atypique}}$;
- des noeuds auto-relatifs, au nombre de $\mathcal{N}_{\mathsf{autorel}}$;
- des noeuds figurant dans le graphe, au nombre borné par $1+q\max(2,\mathcal{N}_{\mathcal{R}})$;
- des noeuds relatifs d'une requête dans le graphe, au nombre borné par $q\mathcal{N}_{\mathcal{R}}^2$;
- des noeuds relatifs d'un noeud possédant un chemin dans le graphe, au nombre borné par $\mathcal{N}_{\mathcal{R}}(1+q\mathcal{N}_{\mathcal{R}})$.

On a donc

$$\Pr\left[\mathsf{Échec}(q)\right]\leq\left[\mathcal{N}_{\mathsf{atypique}}+\mathcal{N}_{\mathsf{autorel}}+1+\mathcal{N}_{\mathcal{R}}+(\max(2,\mathcal{N}_{\mathcal{R}})+2\mathcal{N}_{\mathcal{R}}^2)q\right]2^{-(n-\tau)}.$$

Pour N requêtes aux simulateurs, il y a au plus $N\mathcal{N}_{\mathcal{R}}$ définitions d'entrées de $\mathcal{F}$. La probabilité que $\mathsf{Échec}$ se produise au cours de N requêtes est bornée par

$$\begin{aligned}\Pr\left[\mathsf{Échec}\right] &\leq \left[\sum_{q=1}^{N\mathcal{N}_{\mathcal{R}}}\mathcal{N}_{\mathsf{atypique}}+\mathcal{N}_{\mathsf{autorel}}+1+\mathcal{N}_{\mathcal{R}}+(\max(2,\mathcal{N}_{\mathcal{R}})+2\mathcal{N}_{\mathcal{R}}^2)q\right]2^{-(n-\tau)},\\ &\leq \left[\mathcal{N}_{\mathcal{R}}(\mathcal{N}_{\mathsf{atypique}}+\mathcal{N}_{\mathsf{autorel}}+1+\mathcal{N}_{\mathcal{R}})N+(\max(2,\mathcal{N}_{\mathcal{R}})+2\mathcal{N}_{\mathcal{R}}^2)\mathcal{N}_{\mathcal{R}}^2N^2\right]2^{-(n-\tau)}.\end{aligned}$$

A.2.2 Preuve du lemme 10

On reprend la preuve du lemme 7. $\mathsf{Drapeau}_3$ est désormais levé si l'attaquant réalise une requête relative d'un noeud but d'un arc défini par la construction. Si $\mathsf{Échec}$ ne se produit pas, pour les requêtes provenant de la construction, il y a au plus un arc défini par requête. Le nombre d'arcs correspondant au bout de la q-ème requête reste donc q. Par contre, en soumettant une requête, le distingueur teste jusqu'à $\mathcal{N}_{\mathcal{R}}$ valeurs de $Y^{\mathcal{C}}$. Par ailleurs, la condition $\varepsilon = 0$ permet d'assurer que la probabilité pour l'attaquant de choisir A permettant de compléter un noeud

pour trouver un relatif d'un noeud de $Y^{\mathcal{C}}$ est majorée par $2^{\ell_a - \tau}$. On en déduit la borne de la preuve.

A.2.3 Preuve du lemme 11

Comme la preuve de la section précédente, la preuve de ce lemme consiste à adapter la preuve du cas idéal en tenant compte :

- de la borne modifiée sur la probabilité de tirage d'une sortie (A', B') donnée, prise à $2^{-(n-\tau)}$ en supposant que $\mathsf{Échec}$ ne se produise pas ;
- du but modifié de l'attaquant qui doit trouver non plus un élément de $Y^{\mathcal{C}}$ mais un relatif d'un tel élément, ce qui conduit à multiplier la borne de sa probabilité de succès par un facteur $\mathcal{N}_{\mathcal{R}}$;
- de la taille inchangée de $Y^{\mathcal{C}}$.