ANSSI Agence nationale de la sécurité des systèmes d'information

TELECOM ParisTech

ANSSI — Télécom ParisTech

Laboratoire de Cryptographie — Département Informatique et Réseaux

EDITE de Paris

THÈSE de DOCTORAT

Conception, preuves et analyse de fonctions de hachage cryptographiques

présentée pour obtenir le grade de

Docteur de Télécom ParisTech

Spécialité : Informatique et Réseaux

soutenue publiquement par

Thomas Fuhr

Le 3 octobre 2011

Composition du jury :

Rapporteurs :	Jean-Sébastien Coron	-	Université de Luxembourg
	Willi Meier	-	FNHW
Directeurs de thèse :	Henri Gilbert	-	ANSSI
	Hugues Randriambololona	-	Télécom ParisTech
Examinateurs :	Anne Canteaut	-	INRIA Paris-Rocquencourt
	Gérard Cohen	-	Télécom ParisTech
	Antoine Joux	-	Université de Versailles et DGA
	Pascal Paillier	-	CryptoExperts

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

Remerciements

Je tiens à remercier les personnes qui m'ont apporté leur aide et leur soutien au cours de ces quatre années de thèse.

Tout d'abord, merci à Henri Gilbert pour m'avoir encadré pendant ma thèse. Ses conseils avisés et ses remarques pertinentes ont été très précieux. Je remercie également Hugues Randriam, qui a accepté de co-encadrer ma thèse.

Je voudrais aussi remercier Jean-Sébastien Coron et Willi Meier pour avoir accepté la lourde tâche d'être mes rapporteurs de thèse, ainsi qu'Anne Canteaut, Gérard Cohen, Antoine Joux et Pascal Paillier pour avoir accepté de faire partie de mon jury.

Merci encore à Henri Gilbert, Pascal Paillier et Anne Canteaut, mais aussi à Marion Videau, Aurélie Bauer et Yannick Seurin pour leurs commentaires sur ce mémoire et pour leur traque des inévitables coquilles qu'il a pu contenir.

La DCSSI puis l'ANSSI m'ont apporté un cadre de travail très épanouissant, grâce aux qualités humaines et à la compétence des personnes qui y travaillent. Je tiens à en remercier particulièrement mes collègues passés et présents : Éliane Jaulmes, Jean-René Reinhard, Marion Videau, Mathieu Baudet, Emmanuel Bresson, Benoît Chevallier-Mames, Karim Khalfallah, Joana Treger-Marim, Aurélie Bauer, Henri Gilbert, Yannick Seurin, Thomas Roche, Victor Lomné, Jean-Claude Bourrée et Guillaume Stehlin.

Merci à Florent Chabaud et Loïc Duflot, qui ont toujours considéré la recherche comme une activité primordiale pour la sous-direction ACE et qui m'ont permis de dégager du temps à y consacrer. Je remercie aussi la direction de l'ANSSI qui offre ce cadre de travail.

Une bonne partie de mes travaux de thèse ont été réalisés au sein des projets collaboratifs SAPHIR et Saphir2. La conception de la fonction de hachage Shabal a été une belle aventure, j'en remercie ceux et celles avec qui je l'ai partagée : Benoît Chevallier-Mames, Emmanuel Bresson, María Naya Plasencia, Aline Gouget, Christophe Clavier, Jean-François Misarsky, Céline Thuillet, Thomas Icart et Thomas Pornin, et surtout Marion Videau, Jean-René Reinhard, Anne Canteaut et Pascal Paillier avec qui j'ai continué à travailler sur les preuves de sécurité. Je remercie aussi mes autres co-auteurs de ces quatre années, Henri Gilbert et Thomas Peyrin.

Ces dernières années ont été marquées par d'heureux évènements sur le plan personnel. Je tiens à remercier mes parents et mes beaux-parents pour toute l'aide qu'ils nous ont apportée. Enfin, merci à Corinne, pour son soutien constant pendant les moments de doute, pour sa patience pendant la rédaction de ce manuscrit et pour tous les moments partagés au quotidien. Quelques mots ne sauraient suffire à exprimer ce que je lui dois.

Résumé : Dans ce mémoire nous étudions le domaine des fonctions de hachage, qui sont utilisées par de nombreux mécanismes cryptographiques. Les travaux présentés ici abordent à la fois la conception et l'analyse de la sécurité de ces fonctions.

La première partie de ce mémoire est une introduction générale au domaine des fonctions de hachage. Nous décrivons la manière dont elles sont utilisées en la cryptographie et la manière de formaliser leur sécurité. Nous exposons également les principes de conception sur lesquels les fonctions de hachage les plus utilisées sont fondées. Enfin, nous évoquons la situation actuelle. La cryptanalyse différentielle a donné lieu à des attaques contre les principales fonctions de hachage. Le NIST organise actuellement une compétition de conception de fonctions de hachage (la compétition SHA-3), dans le but de définir une nouvelle norme de hachage.

Dans la deuxième partie nous présentons nos travaux liés à la conception d'un candidat à cette compétition : Shabal. Nous commençons par décrire cette fonction, ainsi que les différentes évaluations de sa sécurité. Nous montrons la sécurité de l'algorithme d'extension de domaine que Shabal utilise, dans le modèle de l'indifférenciabilité d'un oracle aléatoire et en considérant la fonction de compression comme idéale. Nous présentons ensuite un modèle permettant l'étude de la sécurité d'un algorithme d'extension de domaine en cas de découverte de vulnérabilités de la fonction de compression utilisée.

Enfin, dans la troisième partie, nous abordons le domaine de la cryptanalyse de fonctions de hachage. Nous présentons la meilleure attaque connue contre RadioGatún, qui a été définie avant la compétition SHA-3 ainsi qu'une attaque contre `Hamsi`-256, qui a été la première attaque contre une des fonctions de hachage sélectionnées pour le deuxième tour de la compétition SHA-3.

Mots clés : Fonctions de hachage, conception, preuves de sécurité, cryptanalyse, Shabal, RadioGatún, `Hamsi`, indifférenciabilité, modèle de sécurité étendu, cryptanalyse différentielle symétrique, cryptanalyse conditionnelle algébrique.

Abstract : This Thesis focuses on hash functions, which are used in numerous cryptographic mechanisms. We present various results, that belong to the fields of design and security analysis of hash functions.

In the first part we introduce the field of hash functions. We describe how they are used in cryptography, and how to formalize security notions that guarantee their robustness in their various use cases. We also display the design rationale on which the most widely used hash functions are based. Then, we summarize the current situation of hash functions. A few years ago, differential cryptanalysis gave way to attacks on most of the recent hash algorithms. The adoption process for a new standard is under way, it will be the winner of the SHA-3 competition run by the NIST.

In the second part we present the results that are related to the design of Shabal, a candidate to the SHA-3 competition. Firstly, we describe this function. We display some security analysis of the Shabal compression function and we give a security proof of the Shabal domain extender in the indifferentiability model, when it is combined with an idealized version of the compression function. We then describe a new model that enables the security analysis of domain extenders under weaker assumptions on the compression function.

In the third part, we focus on cryptanalysis of hash functions. We present attacks on two different algorithms : the best known attack on RadioGatún, which was defined before the SHA-3 competition ; and an attack against Hamsi-256, which was the first complete attack on a second-round SHA-3 candidate.

Keywords : Hash functions, design, security proofs, cryptanalysis, Shabal, RadioGatún, Hamsi, indifferentiability, extended security model, symmetric differential cryptanalysis, conditional algebraic cryptanalysis.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Notations et Définitions

$\mathbb{F}_q$	Corps fini à q éléments
$\overline{x}$	Complémentation à 1 bit par bit du vecteur x
$\oplus$	OU EXCLUSIF, addition dans $\mathbb{F}_2$
$\boxplus$	Addition dans $\mathbb{F}_{2^{32}}$
$\boxminus$	Soustraction dans $\mathbb{F}_{2^{32}}$
$\circ$	Composition de fonctions
$L\|\|R$	Concaténation de la chaîne L et de la chaîne R
$x \lll a$	Rotation de a positions vers la gauche appliquée au registre x
$x \ggg a$	Rotation de b positions vers la droite appliquée au registre x
$x \wedge y$	Opération de ET LOGIQUE appliquée aux vecteurs (ou booléens) x et y
$x \vee y$	Opération de OU LOGIQUE appliquée aux vecteurs (ou booléens) x et y

Première partie

Introduction Générale

Chapitre 1

Introduction

Sommaire

1.1 Introduction à la cryptologie

1.1.1 Un peu d'histoire

La cryptologie est un domaine de recherche scientifique qui tire ses origines des techniques mises en œuvre pour protéger la confidentialité des communications militaires. Pour rendre un message inintelligible pour l'ennemi, son émetteur lui applique une transformation appelée *chiffrement*. Lors de la réception du message, le destinataire applique l'opération inverse, appelée *déchiffrement*. On parle de *décryptement* lorsqu'un attaquant parvient à retrouver le message clair (c'est-à-dire le message avant l'opération de chiffrement) à partir du chiffré.

De telles techniques ont vu le jour dès l'antiquité, comme par exemple la scytale utilisée par les Spartiates dès le cinquième siècle avant Jésus-Christ, ou le chiffrement de César. D'abord très rudimentaires, les mécanismes cryptologiques se sont progressivement complexifiés et répandus dans le domaine militaire. De ce fait, le caractère secret des mécanismes employés est devenu de plus en plus difficile à garantir. En 1883, Auguste Kerckhoffs énonce un principe fondateur de la cryptologie moderne : les mécanismes de chiffrement et de déchiffrement doivent pouvoir être rendus publics, la confidentialité des messages doit être garantie uniquement par le secret d'une clé. Lors des deux guerres mondiales, la cryptographie a été beaucoup utilisée et la capacité à décrypter les communications ennemies a joué un grand rôle dans l'issue de ces conflits.

Dans la deuxième moitié du vingtième siècle, avec l'essor de l'informatique et des télécommunications, la cryptologie a trouvé des applications dans le domaine civil et est devenue un domaine de recherche académique à part entière. Elle permet de protéger des transactions bancaires, des communications téléphoniques, de stocker des données de manière sécurisée, ou encore d'authentifier les utilisateurs d'un système informatique. Si la signification étymologique de *cryptologie* est *science du secret*, les problématiques traitées par ce domaine vont aujourd'hui bien au-delà de la confidentialité des communications.

Dans le domaine de la cryptologie, on distingue la *cryptographie*, qui correspond à la conception de mécanismes cryptologiques, de la *cryptanalyse*, qui correspond à l'analyse de leur sécurité. Cette branche contient en particulier les attaques contre les mécanismes cryptographiques. Une autre distinction s'opère entre cryptographie *symétrique* et cryptographie *asymétrique*, que nous allons maintenant expliciter.

1.1.2 La cryptographie symétrique

La cryptographie symétrique, aussi appelée cryptographie à clé secrète, regroupe des mécanismes reposant sur la connaissance d'une clé secrète par deux personnes ou entités. Il s'agit de la branche la plus ancienne de la cryptographie. En effet, les systèmes cryptographiques historiques mettent en œuvre une même clé pour les opérations de chiffrement et de déchiffrement. La cryptographie à clé secrète remplit différentes fonctionnalités, dont les suivantes.

Chiffrement symétrique. Le chiffrement symétrique permet de protéger la confidentialité d'une donnée en la chiffrant avec une clé K. Un mécanisme de déchiffrement utilisant la même clé K permet à quiconque la possède de retrouver la donnée à partir du chiffré.

Intégrité d'une donnée. Pour certaines applications, l'intégrité d'une donnée, c'est-à-dire le fait qu'elle n'ait pas été modifiée après sa création, est au moins aussi importante que sa confidentialité. Par exemple, lors d'une transaction bancaire, il est indispensable que la somme d'argent mise en jeu ne soit pas modifiée. Certains mécanismes de la cryptographie à clé secrète permettent de garantir cette propriété. Ces mécanismes consistent à adjoindre à la donnée M à protéger un *code d'authentification de message* ou MAC (pour *Message Authentication Code*), qui dépend de M et d'une clé secrète K. La connaissance de M et K permet la vérification du motif d'intégrité. La validité d'un MAC garantit à la fois l'intégrité du message (il n'a pas été modifié après le calcul du MAC) et une forme d'authenticité appelée authenticité de message (le MAC a été calculé par un détenteur de K).

Authentification d'une entité. Certains mécanismes permettant à un utilisateur d'un système d'information de s'authentifier reposent sur la connaissance commune d'une clé secrète K et sur l'emploi de primitives de la cryptographie symétrique.

La cryptographie à clé secrète repose principalement sur l'emploi de trois types d'algorithmes de base, ou *primitives*.

Les algorithmes de chiffrement par blocs. Un algorithme de chiffrement par blocs est une transformation inversible d'un bloc de taille fixe n paramétrée par une clé de taille k. En étant utilisé selon un *mode opératoire*, ce type de primitives permet notamment d'assurer des fonctionnalités de chiffrement symétrique ou de calcul de MACs. L'algorithme AES, conçu par Daemen et Rijmen [DR98, NIS01], est actuellement la primitive la plus utilisée pour les applications de chiffrement symétrique. Cet algorithme a remplacé l'ancienne norme DES en 2000.

Les algorithmes de chiffrement à flot. Ces algorithmes génèrent une *suite chiffrante* de longueur variable à partir d'une clé de k bits et d'une valeur d'initialisation publique. La suite

chiffrante est ensuite combinée au message à chiffrer par une opération de OU EXCLUSIF. Les algorithmes de ce type présentent l'avantage d'offrir de très bons débits, mais malheureusement ils présentent souvent des failles de sécurité. C'est le cas notamment des algorithmes A5/1 et DSC, qui étaient respectivement utilisés pour le chiffrement de communications GSM et des communications par téléphone sans fil. Plus récemment, la compétition eSTREAM a permis la définition d'algorithmes de chiffrement à flot plus robustes.

Sécurité. En cryptographie, la sécurité des mécanismes utilisés n'est pas inconditionnelle, à de très rares exception près. Par exemple, en cryptographie à clé secrète, la taille de la clé étant fixe, il est toujours théoriquement envisageable pour un attaquant de tester toutes les valeurs possibles de la clé, jusqu'à identifier la bonne. Cette attaque est appelée recherche exhaustive. Les tailles de clés doivent donc être choisies de manière à éviter que cette attaque puisse être réalisée dans la pratique, c'est-à-dire en un temps de calcul réaliste.

Selon les primitives utilisées, d'autres attaques peuvent s'avérer plus performantes. La confiance dans les primitives de la cryptographie à clé secrète repose sur leur résistance aux tentatives de cryptanalyse à travers le temps.

1.1.3 La cryptographie asymétrique

La cryptographie asymétrique repose sur une idée exposée par Diffie et Hellman en 1976 dans [DH76] : le chiffrement et le déchiffrement sont deux opérations fonctionnellement différentes. Il n'y a donc aucune nécessité pour que les clés servant au chiffrement et au déchiffrement soient les mêmes. Dès lors que l'on utilise deux clés différentes pour le chiffrement et pour le déchiffrement et que la clé de déchiffrement ne peut pas être déduite de la clé de chiffrement, cette dernière peut être publique. De ce fait, la cryptographie asymétrique est également appelée cryptographie à clé publique. La clé de déchiffrement (ou clé privée) n'est connue que du destinataire. Ce dernier est l'unique détenteur de la clé privée.

Différentes fonctions de sécurité peuvent être obtenues en utilisant des outils asymétriques.

Échange de clés. L'application découverte par Diffie et Hellman dans leur article était un mécanisme d'échange de clé secrète. En effet, un des problèmes posés par la cryptographie symétrique réside dans l'établissement de clés partagées entre deux entités. Le protocole d'échange de clés de Diffie et Hellman est encore le plus utilisé aujourd'hui.

Chiffrement asymétrique et chiffrement hybride. La découverte d'outils mathématiques permettant d'instancier l'idée de Diffie et Hellman est arrivée un peu plus tard. En 1978, Rivest, Shamir et Adleman ont inventé l'algorithme RSA [RSA78]. D'autres algorithmes ont suivi, comme celui d'ElGamal [Gam84]. Pour des raisons de débit, le chiffrement à clé publique n'est généralement pas utilisé pour chiffrer directement des données. On utilise plutôt de telles primitives pour chiffrer des clés symétriques tirées aléatoirement. Ces clés servent ensuite au chiffrement de données à l'aide d'un algorithme de chiffrement symétrique. On parle alors de chiffrement hybride.

Signature électronique. Comme la signature manuscrite, la signature électronique permet de lier un document à un signataire. Dans les deux cas, les signatures peuvent être vérifiées publiquement. Les schémas de signature électronique sont donc fondamentalement asymétriques : la clé privée du signataire intervient dans l'algorithme de signature, et la clé publique correspondante

sert à la vérification. La primitive RSA peut également être utilisée pour des applications de signatures, notamment en utilisant le format RSA-PSS [RSA02]. D'autres normes telles que DSA [NIS09] ou sa variante utilisant des courbes elliptiques ECDSA, décrite dans le même document, sont aussi fréquemment rencontrées.

Sécurité. Les mécanismes de la cryptographie à clé publique utilisent des outils mathématiques utilisant des fonctions fondamentalement asymétriques. Ces fonctions sont faciles à calculer et considérées comme difficiles à inverser. Leur sécurité est liée à celle de problèmes mathématiques conjecturés difficiles. Citons par exemple la factorisation de grands entiers pour RSA, ou le calcul de logarithmes discrets dans des groupes multiplicatifs pour DSA, ElGamal ou encore l'échange de clés Diffie-Hellman. La résolution de tels problèmes permet de retrouver la clé privée à partir de la clé publique. Les meilleures algorithmes permettant de résoudre ces problèmes sont plus efficaces qu'une recherche exhaustive sur la clé privée. A tailles de clés égales, la sécurité potentiellement offerte par les algorithmes symétriques est donc meilleure que la sécurité des algorithmes asymétriques. Réciproquement, pour un niveau de sécurité équivalent, les clés asymétriques sont donc plus longues que les clés symétriques.

1.2 Les fonctions de hachage en cryptographie

Les systèmes cryptographiques sont obtenus en combinant l'utilisation de primitives, parmi lesquelles on trouve souvent des fonctions de hachage.

1.2.1 Définition et origines

Définition. Une fonction de hachage est un algorithme permettant de calculer une empreinte de taille fixe à partir d'une donnée de taille quelconque.

$$\begin{aligned} H : \{0,1\}^* &\rightarrow \{0,1\}^n \\ M &\mapsto H(M) \end{aligned}$$

A l'origine, les fonctions de hachage ont été créées pour faciliter la gestion de base de données. Plutôt que de manipuler des données de taille variable et potentiellement grande, on associe à ces données une empreinte de taille fixe, sur lesquelles des comparaisons sont plus rapides à effectuer. L'utilisation de fonctions de hachage permet par conséquent d'accélerer des opérations comme le tri, l'insertion ou l'accès à un élément.

Propriétés. Les fonctions de hachage ne sont pas des systèmes cryptographiques : elle ne permettent pas à elles seules d'assurer une propriété de sécurité. En particulier, elles ne constituent pas un algorithme de chiffrement. Nous verrons en section 1.2.3 quelques exemples d'utilisation de fonctions de hachage en cryptographie, mais nous pouvons d'ores et déjà souligner les propriétés suivantes des fonctions de hachage.

- **Absence de clé.** La définition des fonctions de hachage ne fait pas intervenir de clé. Pour certaines applications, la donnée hachée est entièrement publique, et un attaquant éventuel peut effectuer lui-même les calculs de hachés.
- **Absence d'algorithme d'inversion.** Contrairement aux primitives de chiffrement par blocs ou de chiffrement asymétrique, les fonctions de hachage n'ont pas besoin de pouvoir être

inversées. L'impossibilité de calculer un antécédent par une fonction de hachage est même une propriété de sécurité fondamentale de ces primitives.

1.2.2 Sécurité des fonctions de hachage cryptographiques

1.2.2.1 Notions de sécurité

Lorsque les fonctions de hachage sont utilisées dans les bases de données, le seul écueil à éviter réside dans des collisions accidentelles : si deux entrées d'une même base s'avèrent avoir la même empreinte, elles seront considérées comme identiques. Pour éviter que cela se produise, les fonctions de hachage sont construites de manière à ce que les empreintes soient réparties uniformément sur l'ensemble d'arrivée de la fonction.

Dans le domaine de la cryptographie, les fonctions de hachage sont utilisées comme brique de base de différents mécanismes. La robustesse de ces mécanismes repose en partie sur la résistance de la fonction de hachage qu'ils utilisent à différentes attaques. Bien que d'autres chemins d'attaque soient possibles, la sécurité des fonctions de hachage est surtout évaluée dans les trois scénarios suivants.

- Recherche de collisions : l'attaquant doit trouver deux messages différents M et M' tels que $H(M) = H(M')$.
- Recherche de secondes préimages : étant donné un message M, l'attaquant doit trouver M' différent de M tel que $H(M) = H(M')$.
- Recherche de préimages : étant donné une valeur x de l'ensemble des hachés, l'attaquant doit trouver M tel que $H(M) = x$.

Problème de définition de la résistance à la recherche de collisions. Les fonctions de hachage sont utilisées pour calculer des empreintes de taille fixe à partir de données de taille variable et potentiellement grande. Le domaine de définition d'une fonction de hachage est donc plus grand que l'ensemble des empreintes. Ceci implique l'existence de collisions. La sécurité repose donc sur la difficulté de trouver ces collisions dans la pratique.

De plus, pour n'importe quelle fonction de hachage H, il existe des attaquants qui renvoient des collisions de manière immédiate. Si M et M' sont deux messages différents tels que $H(M) = H(M')$, alors l'algorithme qui renvoie simplement M et M' permet de trouver une collision pour H. La résistance à la recherche de collisions ne peut donc pas être définie comme l'impossibilité de l'existence d'un tel attaquant, mais comme la difficulté à en trouver un dans la pratique.

1.2.2.2 Attaques génériques

Pour chacun de ces scénarios, il existe des techniques permettant d'attaquer n'importe quelle fonction de hachage. On appelle ces techniques *attaques génériques*.

Recherche de collisions. La meilleure attaque générique en recherche de collisions repose sur le *paradoxe des anniversaires*. Cette expression désigne une propriété contre-intuitive : parmi un groupe d'au moins 23 personnes prises au hasard, il y a au moins une chance sur deux pour que deux d'entre elles aient leur anniversaire le même jour. Cette propriété peut être expliquée par le raisonnement heuristique suivant.

Considérons k valeurs $x_1, \dots x_k$ tirées indépendamment et uniformément sur un ensemble de taille N. Il y a $\frac{k^2-k}{2}$ manières de choisir une paire (x_i, x_j) parmi $\{x_1, \dots, x_k\}$. Pour chacune de ces paires, la probabilité pour que $x_i = x_j$ vaut $1/N$. Si $k^2 \geq N$, c'est-à-dire si $k \geq \sqrt{N}$, la somme

$$\sum_{1 \leq i < j \leq k} \Pr[x_i = x_j] \approx \frac{k^2 - k}{2N} \approx \frac{1}{2} .$$

Ces probabilités ne sont pas tout à fait indépendantes, néanmoins il existe i et j différents tels que $x_i = x_j$ avec bonne probabilité.

Le calcul de hachés de messages aléatoires correspond à un tirage aléatoire sur l'espace des hachés $\{0,1\}^n$, de taille 2^n. D'après le raisonnement précédent, le nombre de hachés qu'il faut calculer pour trouver une collision est de l'ordre de $\sqrt{2^n} = 2^{n/2}$.

Recherche de secondes préimages et de préimages. La meilleure attaque générique pour la recherche de secondes préimages ou de préimages est la *recherche probabiliste* : étant donné une empreinte x, l'attaquant calcule des hachés pour des messages aléatoires jusqu'à ce que l'un des hachés soit égal à x. Chacune des empreintes calculées est égale à x avec probabilité proche de 2^{-n}, donc le nombre moyen de hachés à calculer pour trouver une préimage de x est de l'ordre de 2^n.

Résumé. La complexité des meilleures attaques génériques est résumée dans le Tableau 1.1

Notion de sécurité	Attaque	Complexité
Résistance aux collisions	Paradoxe des anniversaires	$2^{n/2}$
Résistance aux 2^{ndes} préimages	Recherche probabiliste	2^n
Résistance aux préimages	Recherche probabiliste	2^n

TABLE 1.1 – Complexité des meilleures attaques génériques sur les fonctions de hachage

1.2.3 Utilisations en cryptographie

Les fonctions de hachage sont utilisées par de nombreux systèmes cryptographiques, à la fois en cryptographie symétrique et en cryptographie asymétrique. Elles en ont hérité le surnom de "couteau suisse" de la cryptographie. Nous pouvons citer les exemples suivants.

1.2.3.1 Code d'authentification de message

En cryptographie, les codes d'authentification de message (ou MAC) sont des empreintes d'une donnée qui dépendent d'une clé K, et qui ne peuvent être calculées qu'en connaissant K. En ce sens, ils peuvent être vus comme des fonctions de hachage à clé. Il existe des constructions de MAC à partir de fonctions de hachage, la plus répandue étant HMAC [BCK96]. HMAC repose sur l'utilisation de deux constantes *ipad* et *opad*, et sur une fonction de hachage H. Le MAC d'un message M avec une clé K est défini par :

$$MAC_K(M) = H(K \oplus opad || H(K \oplus ipad || M)) .$$

1.2.3.2 Signature électronique

Les primitives de signature telles RSA, ElGamal ou DSA imposent la manipulation de données représentées sous la forme d'entiers modulaires. L'utilisation de ECDSA requiert leur expression sous la forme de points sur des courbes elliptiques. La taille des données à signer est donc contrainte. Il est théoriquement possible de construire des mécanismes de signature permettant de signer des données de taille quelconque, cependant de telles constructions seraient très lentes. Pour pouvoir signer des données de taille quelconque, la solution la plus répandue consiste à leur appliquer une fonction de hachage, puis d'appliquer la fonction de signature σ sur l'empreinte obtenue. La signature d'un message M est alors $\sigma(H(M))$.

Cette utilisation des fonctions de hachage permet d'illustrer la nécessité de la résistance à la recherche de collisions. En effet, supposons qu'Alice détienne une clé privée de signature, et que Charlie puisse lui transmettre des données à signer. Si Charlie sait générer deux données M et M' ayant la même empreinte par H et fait signer M à Alice, Alice calcule $S = \sigma(H(M))$. Or, comme $H(M) = H(M')$, $\sigma(H(M')) = \sigma(H(M)) = S$. S est donc également une signature valide pour M'. En trouvant une collision sur H, Charlie peut connaître une signature valide de M' sans qu'Alice n'ait signé ce message. De ce fait, en attaquant la fonction de hachage H, Charlie parvient à casser le schéma de signature sans avoir attaqué la primitive de signature utilisée.

1.2.3.3 Dérivation de clés symétriques

Les fonctions de hachage cryptographiques peuvent être utilisées pour dériver des clés symétriques à partir de différents types de secrets.

Protocoles d'échange de clés. Pour que les estimations des complexités des attaques génériques données plus haut soient valides, une condition nécessaire est que les empreintes par H de messages pris au hasard soient réparties uniformément sur l'espace des hachés. Cette propriété permet d'apporter une solution à la problématique suivante. Les algorithmes de la cryptographie symétriques reposent sur l'utilisation d'une clé K choisie aléatoirement sur $\{0,1\}^k$. Dans le cas de la protection de communications, cette clé doit être connue des deux entités qui communiquent. Or les protocoles d'échange de clés, comme celui de Diffie et Hellman, permettent d'établir un secret S dans un format particulier (un élément d'un groupe dans le cas de Diffie-Hellman). Pour transformer ce secret S en une clé de k bits, on utilise souvent une construction reposant sur une fonction de hachage. Par exemple, le NIST recommande dans [NIS07a] de dériver les clés symétriques en concaténant les valeurs $(h_1||\ldots||h_\ell)$, avec

$$h_i = H(i||S||\mathsf{Info}) \ ,$$

où i est représenté sur 32 bits et Info contient de l'information additionnelle sur les paramètres Diffie-Hellman. ℓ est choisi de manière à ce que $(h_1||\ldots||h_\ell)$ soit assez longue pour en extraire les clés dont on a besoin. On dérive généralement une clé pour un algorithme de chiffrement et une clé pour un algorithme de MAC.

Dérivation à partir d'un secret maître. Certains protocoles de communication reposent entièrement sur la cryptographie symétrique. Les équipements qui communiquent possèdent dans ce cas un secret partagé S, défini avant leur déploiement. Afin de limiter l'impact de la compromission des clés utilisées pour protéger des communications, il est nécessaire de renouveler régulièrement

leurs valeurs. Une manière de procéder consiste à calculer la $i^{ème}$ clé K_i comme le haché du secret S et d'un compteur i.

Mots de passe. Les clés utilisées par les algorithmes de chiffrement modernes sont des chaînes aléatoires d'au moins 128 bits, difficiles à retenir pour un humain. Pour pallier ce problème, elles peuvent être calculées en hachant un mot de passe. Ce mot de passe doit idéalement être suffisamment long pour contenir 128 bits d'entropie.

1.2.3.4 Génération de nombres aléatoires.

De façon similaire, les propriétés statistiques des fonctions de hachage et leur caractère à sens unique peuvent être exploités dans des contextes de génération d'aléa. En effet, de nombreux mécanismes cryptographiques nécessitent la génération de nombres aléatoires : génération de clés, signature électronique, chiffrement asymétrique, génération de valeurs d'initialisation pour le chiffrement symétrique... Les nombres ainsi générés doivent être tirés uniformément et être imprédictibles pour un attaquant. Dans certains cas, les nombres aléatoires générés doivent rester confidentiels bien après leur génération.

Les nombres aléatoires sont générés à partir de secrets stockés en mémoire et/ou de phénomènes physiques aléatoires. La traduction en chaînes de bits de phénomènes physiques aléatoires peut conduire à des chaînes non uniformément distribuées. Dans ce cas l'utilisation de fonctions de hachage permet d'extraire l'entropie de ces chaînes, c'est-à-dire d'obtenir des suites de bits indépendants et uniformément distribués.

D'autre part, le caractère non inversible des fonctions de hachage permet de dériver des nombres pseudo-aléatoires à partir de secrets sans les compromettre. Ces fonctions permettent également de mettre à jour les valeurs secrètes, afin que leur compromission éventuelle n'affecte pas leurs valeurs passées. Le schéma défini par Barak et Halevi dans [BH05], et normalisé par le NIST dans [NIS07b], fait un tel usage des fonctions de hachage.

1.2.3.5 Mots de passe

De nombreux systèmes informatiques authentifient leurs utilisateurs grâce à la connaissance de mots de passe. Un inconvénient potentiel de cette méthode est que si les mots de passe des utilisateurs sont gardés en mémoire sur le système et si un attaquant parvient à accéder à ces mots de passe, il peut s'authentifier sous l'identité de n'importe quel utilisateur. Ce problème peut être résolu à l'aide de fonctions de hachage. Au lieu de stocker la valeur des mots de passe, le système ne conserve que leur empreinte par une fonction de hachage H. Si un attaquant parvient à accéder à ces valeurs, il doit encore trouver une préimage de l'une d'entre elles pour pouvoir s'authentifier.

Remarque. Il n'est théoriquement pas nécessaire à un attaquant de retrouver le mot de passe d'un utilisateur pour s'authentifier, mais un mot de passe ayant la même empreinte que le mot de passe d'un utilisateur. Cependant, dans la pratique, les mots de passe choisis par les utilisateurs ne contiennent pas suffisamment d'entropie. Certaines techniques permettant de tester des valeurs probables de mots de passe sont en pratique plus efficaces que la recherche probabiliste. Cette vulnérabilité potentielle des mots de passe peut être prise en compte en itérant plusieurs fois la fonction de hachage sur le mot de passe.

1.2.3.6 Schémas d'engagement

Les schémas d'engagement sont des protocoles permettant à une entité de s'engager sur le choix d'une valeur sans la donner immédiatement, puis de la révéler plus tard. Ces schémas sont utilisés notamment dans les preuves sans transfert de connaissance. Supposons qu'Alice veut s'engager sur une valeur x, sans la transmettre à Bob. Elle choisit un nombre aléatoire r, et transmet $H(x||r)$ à Bob. De cette manière, Bob ne peut pas retrouver x. L'utilisation d'un nombre aléatoire r empêche Bob de tester un petit nombre de valeurs de x qu'il estime plausibles. Plus tard dans le protocole, Alice peut révéler x et r. Si Alice ne sait pas générer de collision pour H, Bob a la garantie que x est bien la valeur qu'avait choisie Alice.

1.2.3.7 Dimensionnement des fonctions de hachage

Pour garantir une certaine marge de sécurité par rapport aux puissances de calcul actuellement accessibles, on cherche aujourd'hui à se protéger contre des attaquants pouvant effectuer jusqu'à 2^{100} opérations. Lorsque la résistance à la recherche de collisions est requise, cela impose d'employer des fonctions permettant de calculer des empreintes d'au moins 200 bits. Lorsque la résistance à la recherche de préimages suffit, des empreintes d'au moins 100 bits peuvent suffire.

Chapitre 2

Extension de domaine

Sommaire

Dans ce chapitre nous nous intéressons à la structure des fonctions de hachage cryptographiques. Dans la pratique, les opérations accessibles pour calculer les empreintes prennent des entrées de taille fixe. De ce fait, les fonctions de hachage sont généralement conçues à partir de deux mécanismes internes : une *fonction de compression*, qui permet le traitement de données de taille fixe, et un *algorithme d'extension de domaine*, qui constitue une manière d'utiliser la fonction de compression pour calculer des empreintes de messages de taille variable.

2.1 Autour de la construction de Merkle-Damgård

La plupart des fonctions de hachage traditionnelles utilisent l'algorithme d'extension de domaine de Merkle-Damgård, que nous allons décrire maintenant. Cet algorithme a été découvert indépendamment par Merkle [Mer89] et Damgård [Dam89].

2.1.1 Algorithme de Merkle-Damgård

L'algorithme de Merkle-Damgård permet de calculer des empreintes pour des données de taille variable en utilisant une fonction de compression

$$\begin{aligned} \mathcal{F} : \{0,1\}^n \times \{0,1\}^m &\to \{0,1\}^n \\ (C, M) &\mapsto \mathcal{F}(C, M)\ . \end{aligned}$$

L'idée consiste à diviser la donnée à hacher M en blocs de taille fixe $M^1, \ldots, M^k$ et à traiter les blocs M^i itérativement. On note m la taille des blocs de message. Le calcul du haché de M se déroule de la manière suivante.

Padding. Afin d'obtenir une donnée dont la taille est un nombre entier de blocs, on applique au message une transformation injective pad appelée *padding*. Le caractère injectif du padding est nécessaire dans le but d'éviter les collisions sur la fonction de hachage. Bien que plusieurs transformations soient utilisables avec l'algorithme de Merkle-Damgård, la plupart des fonctions de hachage traditionnelles utilisent un padding commun. Celui-ci consiste à concaténer à M un bit valant 1, puis le plus petit nombre de bits valant 0 tel que la longueur λ totale de la donnée obtenue vérifie $\lambda = m - v \bmod m$, où v est une constante. On concatène ensuite à cette donnée la longueur ℓ de M représentée sur v bits. On peut donc écrire :

$$\mathsf{pad}(M) = M||100\ldots0||\ell\ .$$

Notons qu'avec ce padding, appelé *renforcement MD*, la longueur totale des messages est limitée à 2^v bits.

Itération de la fonction de compression. Après l'application du padding, on peut diviser $\mathsf{pad}(M)$ en k blocs de m bits $M^1, \ldots, M^k$. La définition d'une fonction de hachage requiert également la définition d'une valeur constante $IV \in \{0,1\}^n$, appelée valeur d'initialisation. Le calcul de l'empreinte de M est obtenu en itérant $\mathcal{F}$ de la manière suivante. On définit $h^0 = IV$, et pour tout i de $\{1, \ldots, k\}$, on calcule

$$h^i = \mathcal{F}(h^{i-1}, M^i)\ .$$

L'empreinte de M est $H(M) = h^k$, la sortie de la dernière application de la fonction de compression. Elle est donc obtenue en calculant de manière itérative $\mathcal{F}$ sur la sortie de l'iteration précédente et un nouveau bloc de message. Les valeurs h^i sont généralement appelées *variables de chaînage* ou *hachés intermédiaires*. L'algorithme de Merkle-Damgård est représenté sur la Figure 2.1.

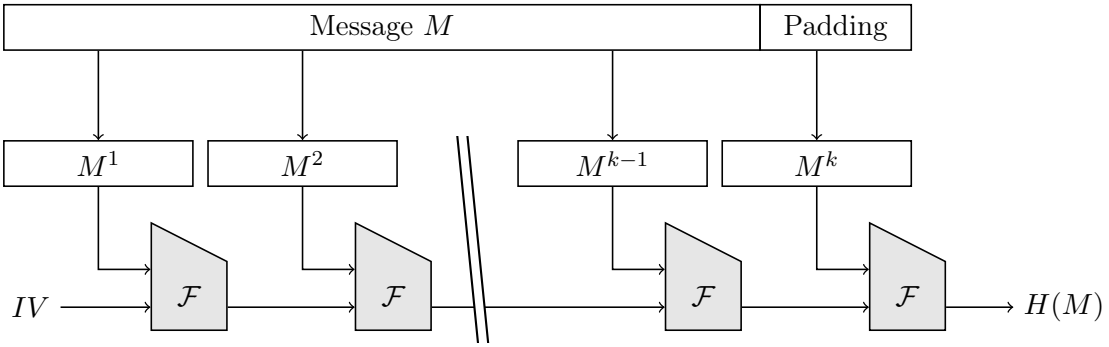


FIGURE 2.1 – Algorithme d'extension de domaine de Merkle-Damgård

L'algorithme de Merkle-Damgård a longtemps fait figure de référence en matière d'extension de domaine. Il est notamment utilisé par les fonctions MD5, SHA-1 et les fonctions de la famille SHA-2, qui sont les fonctions de hachage les plus utilisées à ce jour et que nous évoquerons plus longuement au chapitre 3. Cependant, des découvertes récentes révèlent certaines imperfections de cette construction. Il s'agit d'attaques génériques de la construction de Merkle-Damgård, c'est-à-dire de propriétés valables pour toutes les fonctions de hachage fondées sur cette construction.

2.1.2 Extension de messages

Les fonctions de hachage construites selon l'algorithme de Merkle-Damgård ont la propriété suivante. Soient M et M' deux messages tels que $\mathsf{pad}(M)$ soit un préfixe de $\mathsf{pad}(M')$. Il existe deux entiers j et k, et k blocs de m bits $M^1, \ldots, M^k$ tels que

$$\begin{aligned} M &= M^1||\ldots||M^j \\ M' &= M^1||\ldots||M^j||M^{j+1}||\ldots||M^k \ . \end{aligned}$$

Si H est une fonction de hachage construite selon l'algorithme de Merkle-Damgård, on peut alors calculer le haché de M' à partir du haché de M et de $M^{j+1}, \ldots, M^k$ en définissant $h^j = H(M)$ et en calculant pour tout i de $\{j+1, \ldots, k\}$:

$$h^i = \mathcal{F}(h^{i-1}, M^i) \ .$$

On vérifie facilement que $h^k = H(M')$.

Une conséquence immédiate de cette propriété est une attaque contre une construction naïve de MAC à partir d'une fonction de hachage. En effet, l'idée la plus simple pour ajouter une clé à une fonction de hachage est de définir

$$MAC_K(M) = H(K||M) \ .$$

Avec l'algorithme de Merkle-Damgård et le padding défini plus haut, on peut facilement construire deux messages M et M' tels que $\mathsf{pad}(K||M)$ soit un préfixe de $\mathsf{pad}(K||M')$, sans connaître K. D'après la propriété d'extension de messages, la connaissance de M, M' et de $MAC_K(M)$ permet de retrouver $MAC_K(M')$ sans connaître la clé.

2.1.3 Multicollisions

Une autre conséquence de la structure itérative de la construction de Merkle-Damgård est la possibilité de générer des multicollisions avec une complexité anormalement faible. Ce résultat a été découvert par Joux en 2004 [Jou04].

Une r-multicollision sur une fonction de hachage H est un ensemble de r messages deux à deux distincts $M_1, \ldots, M_r$ tels que toutes les valeurs $H(M_i)$ soient égales. En particulier, une collision est une 2-multicollision.

De manière générique, si on calcule des hachés de messages choisis sans propriétés particulières, une r-multicollision se produit après le calcul d'environ $2^{\frac{(r-1)n}{r}}$ calculs de haché. Ce résultat est une conséquence d'une généralisation du paradoxe des anniversaires. Dans le cas de la construction de Merkle-Damgård, Joux a découvert la méthode suivante, représentée sur la Figure 2.2, qui permet de générer des 2^k-multicollisions avec une complexité moyenne de $k2^{n/2}$ calculs de la fonction de compression.

On commence par chercher deux blocs de messages M_0^1, M_1^1 tels que $\mathcal{F}(IV, M_0^1) = \mathcal{F}(IV, M_1^1) = h^1$. D'après le paradoxe des anniversaires, cette étape requiert $2^{n/2}$ évaluations de $\mathcal{F}$. A partir de h^1, on cherche ensuite deux blocs M_0^2, M_1^2 tels que $\mathcal{F}(h^1, M_0^2) = \mathcal{F}(h^1, M_1^2) = h^2$, ce qui coûte là encore $2^{n/2}$ opérations. On itère cette opération k fois, jusqu'à trouver deux blocs M_0^k, M_1^k tels que $\mathcal{F}(h^{k-1}, M_0^k) = \mathcal{F}(h^{k-1}, M_1^k) = h^k$. Supposons pour fixer les idées que H utilise le padding usuel défini plus haut. On définit $M^{k+1} = 100\ldots0||(mk)$ où (mk) est la représentation du produit $m \times k$ sur v bits, et $h = \mathcal{F}(h^k, M^{k+1})$.

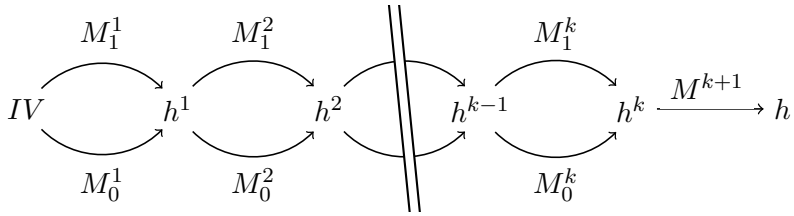


FIGURE 2.2 – Multicollision sur une fonction de hachage utilisant la construction de Merkle-Damgård

Soit $B = (b_1, \ldots, b_k)$ une suite de k bits quelconque. On définit alors

$$M_B = M_{b_1}^1 || \ldots || M_{b_k}^k \ .$$

On vérifie facilement que $H(M) = h$. Il y a 2^k valeurs possibles pour B, qui définissent 2^k messages différents ayant la même empreinte. Cette méthode permet donc de générer des 2^k-multicollisions. Sa complexité réside dans les k étapes de recherche de collisions pour la fonction de compression. Elle nécessite donc en moyenne de l'ordre de $k2^{n/2}$ applications de $\mathcal{F}$.

2.1.4 Recherche de secondes préimages

De manière générale, la construction de Merkle-Damgård ne permet pas d'obtenir une résistance à la recherche de secondes préimages en 2^n opérations. Il existe en effet une attaque générique décrite dans [MVO96], qui permet de trouver des secondes préimages pour un message M en $2^n/(\ell+1)$ opérations, où ℓ est la longueur de $\mathsf{pad}(M)$ en nombre de blocs.

Soit H une fonction de hachage utilisant la construction de Merkle-Damgård, $\mathcal{F}$ sa fonction de compression, et IV la valeur initiale de la variable de chaînage. Notons $M = M_1 || \ldots || M_\ell$, et h_i, $0 \leq i \leq \ell$ les valeurs successives de la variable de chaînage au cours du calcul de l'empreinte de M. Pour un bloc de message μ aléatoire, $\mathcal{F}(IV, \mu)$ est égal à l'un des h_i avec probabilité $(\ell+1)/2^n$. En effectuant de l'ordre de $2^n/(\ell+1)$ calculs de $\mathcal{F}$, un attaquant peut donc trouver μ et j tels que $\mathcal{F}(IV, \mu) = h_j$.

Supposons alors qu'il existe un message M' tel que $\mathsf{pad}(M') = \mu || M_{j+1} || \ldots || M_\ell$. On vérifie facilement que $H(M') = h_\ell = H(M)$: l'attaquant a trouvé une seconde préimage pour M.

La vulnérabilité d'une fonction de hachage à cette attaque est néanmoins conditionnée par l'existence d'un tel message M', qui dépend du padding utilisé. Dans le cas du renforcement MD, le dernier bloc M_ℓ contient la longueur du message. L'existence de M' n'est alors possible que si $j = 1$, et on retrouve une complexité en 2^n calculs de $\mathcal{F}$.

Dans un article publié à Eurocrypt 2005 [KS05], Kelsey et Schneier montrent comment utiliser l'algorithme de recherche de multicollisions de Joux pour étendre cette attaque aux fonctions de hachage utilisant la construction de Merkle-Damgård avec renforcement MD. Pour y parvenir, ils utilisent des *messages extensibles*.

Un (p, q)-message extensible pour une fonction de hachage $\mathcal{H}$ est un ensemble de $(q - p + 1)$ messages $(\mu_p, \ldots, \mu_q)$ tels que :

1. $\mathcal{H}(\mu_p) = \mathcal{H}(\mu_{p+1}) = \ldots = \mathcal{H}(\mu_q) = h$.
2. $\forall i \in \{p, \ldots, q\}$, $\mathsf{pad}(\mu_i)$ contient exactement i blocs.

Dans le cas des fonctions de hachage utilisant la construction de Merkle-Damgård, la technique de Joux pour générer des multicollisions permet également de trouver des messages extensibles. Nous utilisons maintenant la définition récursive suivante des fonctions $\mathcal{F}^i$:

$$\forall C \in \{0,1\}^n, \qquad \mathcal{F}^0(C,\varepsilon) = C$$
$$\forall i \in \mathbb{N}^*, \forall M \in \{0,1\}^{im}, \forall \mu \in \{0,1\}^m, \forall C \in \{0,1\}^n, \qquad \mathcal{F}^{i+1}(C, M||\mu) = \mathcal{F}(\mathcal{F}^i(C,M),\mu)\ .$$

A chaque étape de la recherche de multicollisions, au lieu de chercher deux blocs M_0^i et M_1^i tels que $\mathcal{F}(h^i, M_0^i) = \mathcal{F}(h^i, M_1^i)$, on cherche maintenant un bloc M_0^i et une suite de $2^{i-1}+1$ blocs $M_1^i = B_0^i||\ldots||B_{2^{i-1}}^i$ tels que $\mathcal{F}(h^i, M_0^i) = \mathcal{F}^{2^{i-1}+1}(h^i, M_1^i)$. La 2^k-multicollision ainsi obtenue contient 2^k messages de longueurs toutes différentes, qui constitue un $k, k+2^k-1$-message extensible.

Soit M un message tel que $\mathsf{pad}(M) = M^1||\ldots||M^\ell$ contienne $\ell \geq k+2^k$ blocs. Notons $h_i = F^i(IV, M^1||\ldots||M^i)$ les valeurs intermédiaires de la variable de chaînage au cours du calcul de $H(M)$. La recherche de secondes préimages fonctionne de la manière suivante.

1. L'attaquant génère un $(k, k+2^k-1)$-message extensible $M_k, \ldots, M_{k+2^k-1}$, tel que pour tout $i \in \{k, \ldots, k+2^k-1\}$, $\mathcal{F}^i(IV, M_i) = h$.
2. Il cherche ensuite un bloc μ tel que $\mathcal{F}(h,\mu) = h_t$, pour une certaine valeur de t entre $k+1$ et $k+2^k$.
3. L'attaquant renvoie $M' = \mathsf{unpad}(M_{t-1}||\mu||M^{t+1}||\ldots||M^\ell)$, où unpad est l'opération inverse du padding

On vérifie facilement que $H(M') = H(M)$. Si $\ell = k+2^k$, les valeurs de μ testées par l'attaquant sont choisies de manière à ce que les derniers bits contiennent la longueur du message final. En effet, dans ce cas on peut avoir $t = \ell$.

La complexité moyenne de cette attaque est alors $k2^{n/2} + 2^{n-k}$ calculs de $\mathcal{F}$.

Points fixes. Une autre manière de générer des messages extensibles, découverte par Dean [Dea99], est d'utiliser des points fixes de la fonction de compression, c'est-à-dire des valeurs de (C, M) telles que $\mathcal{F}(C, M) = C$. De tels points fixes peuvent être trouvés facilement pour la plupart des fonctions de hachage classiques, du fait de la structure de leur fonction de compression (voir chapitre 3).

Blocs de message courts. Certaines fonctions de hachage apparues récemment telles que Hamsi [Küç09] utilisent des blocs de messages courts. Dans ce cas, les attaques décrites ci-dessus ne peut pas être appliquées telles quelles. En effet, si $m < n$, un bloc de message ne peut pas prendre suffisamment de valeurs possibles pour que le succès des étapes 1 et 2 de la recherche de secondes préimages soit garanti. L'attaque de Kelsey et Schneier peut néanmoins être adaptée. Notons $p = \lceil n/2m \rceil$, $q = \lceil n/m \rceil$ et $r = \lceil v/m \rceil$, où v est la longueur de la chaîne codant la longueur du message dans le padding. L'attaque fonctionne maintenant de la manière suivante :

1. L'attaquant génère un $(kp, kp+2^k-1)$-message extensible $M_kp, \ldots, M_{kp+2^k-1}$-message extensible, tel que pour tout $i \in \{kp, \ldots, kp+2^k-1\}$, $\mathcal{F}^i(IV, M_i) = h$. A chaque étape, l'attaquant cherche une collision entre des messages de p et $p+2^{i-1}$ blocs.
2. Il cherche ensuite une chaîne μ de q blocs de messages telle que $\mathcal{F}(h,\mu) = h_t$, pour une certaine valeur de t entre $kp+q$ et $kp+2^k+q-1$.
3. L'attaquant renvoie $M' = \mathsf{unpad}(M_{t-1}||\mu||M^{t+1}||\ldots||M^\ell)$, où unpad est l'opération inverse du padding.

Afin de garantir la conformité du padding, il est nécessaire que $\ell \geq t + r$, l'attaque fonctionne donc pour des messages de $kp + 2^k + q + r - 1$ blocs.

2.2 Algorithmes d'extension de domaine récents

Les faiblesses identifiés de la construction de Merkle-Damgård ont conduit les cryptographes à proposer de nouvelles constructions. Parmi celles qui ont abouti à des propositions concrètes, on peut citer les fonctions éponges et la construction *wide-pipe*.

2.2.1 La construction *wide pipe*

La construction wide pipe, introduite par Lucks [Luc05] est une construction itérée qui généralise celle de Merkle-Damgård. Elle permet de calculer des hachés de taille $h < n$ à partir de deux fonctions de compression $\mathcal{F}$ et $\mathcal{G}$ vérifiant

$$\begin{aligned} \mathcal{F} : \{0,1\}^n \times \{0,1\}^m &\rightarrow \{0,1\}^n \\ \mathcal{G} : \{0,1\}^n &\rightarrow \{0,1\}^h . \end{aligned}$$

Le calcul de l'empreinte d'un message M consiste à appliquer la construction de Merkle-Damgård avec la fonction de compression $\mathcal{F}$, puis à appliquer $\mathcal{G}$ au résultat pour obtenir une empreinte de taille h. Cette construction est résumée sur la Figure 2.3.

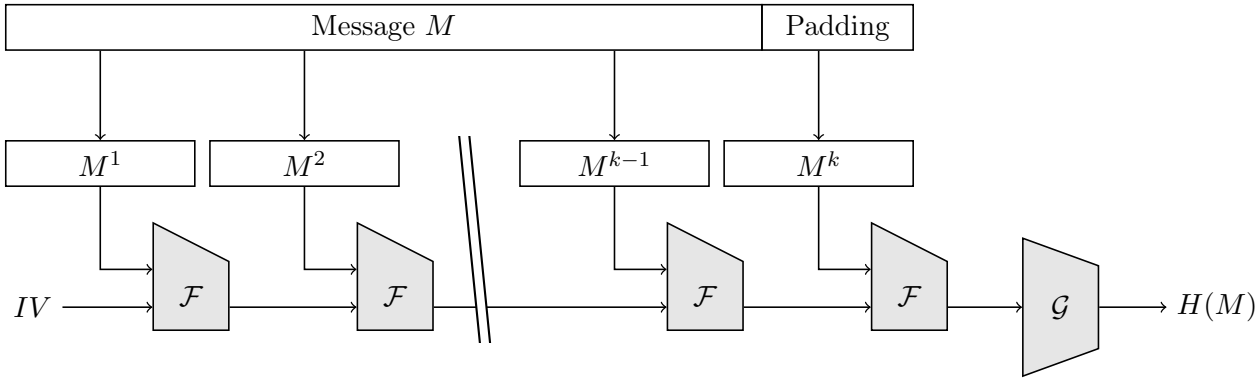


FIGURE 2.3 – Construction wide-pipe

La construction wide-pipe englobe plusieurs variantes, parmi lesquelles on peut noter les constructions double pipe, définie par $n = 2h$, et Chop-MD, pour laquelle la fonction $\mathcal{G}$ est une simple troncation. Cette dernière construction a été proposée par Coron *et al.* dans [CDMP05], sa sécurité est étudiée dans [CDMP05], [CN08] et aux chapitres 5 et 6.

Résistance aux attaques. Nous pouvons d'ores et déjà constater que la construction wide-pipe résiste aux attaques par extension de longueur et à la recherche de multicollisions décrites ci-dessus. En effet, étant donnée une valeur de haché y, il existe en moyenne 2^{n-h} valeurs de x telles que $\mathcal{G}(x) = y$. Si $n \geq 2h$, $2^{n-h} \geq 2^h$ et la connaissance de l'empreinte de M ne suffit pas à reconstituer avec certitude la valeur de la variable de chaînage après le traitement du message. Les attaques par extension de longueur ne peuvent donc pas s'appliquer.

L'algorithme de recherche de multicollisions de Joux peut être utilisé dans le cas de la fonction Chop-MD. Cependant, la complexité de chacune des étapes de recherche de collisions sur la variable de chaînage est $2^{n/2}$. Si $n \geq 2h$, $2^{n/2} \geq 2^h$ et la complexité de la recherche de multicollisions dépasse la complexité de l'attaque générique en recherche de préimages. Le fait d'utiliser des variables de chaînage de plus grande taille que les empreintes permet donc de résister à la recherche de multicollisions.

2.2.2 Fonctions éponges

Une autre construction de fonctions de hachage est due à Bertoni *et al.* [BDPA07]. Cette construction est radicalement différente de celle de Merkle-Damgård : au lieu de faire appel à une fonction non inversible de la variable de chaînage et d'un bloc de message, la fonction de compression repose sur une transformation fixe de la variable de chaînage. Cette transformation peut être instanciée par une permutation ou par une fonction non inversible. Les fonctions ainsi obtenues sont appelées *fonctions éponges*. Nous décrivons maintenant la construction de ces fonctions. Soient r et c deux entiers, nous considérons une fonction

$$\mathcal{F} : \{0,1\}^{r+c} \to \{0,1\}^{r+c} .$$

En notant h la longueur en bits des empreintes, nous définissons le paramètre u suivant :

$$u = \left\lceil \frac{h}{r} \right\rceil .$$

Les empreintes par une fonction éponge H sont obtenues en générant des vecteurs de r bits, et en tronquant le dernier pour garder h bits. u est donc le nombre de vecteurs à générer.

Le hachage d'un message M se déroule de la manière suivante. On commence par appliquer un padding injectif à M, tel que la longueur de $\mathsf{pad}(M)$ soit un multiple de r bits. On écrit alors $\mathsf{pad}(M) = m^1 || \ldots || m^\ell$. Dans l'article [BDPA08b] qui traite de la sécurité des fonctions éponges, le padding conseillé consiste à concaténer à M un bit valant "1" et suffisamment de bits valant "0" pour obtenir une taille multiple de r bits. Une variable de chaînage (R, C) de $r+c$ bits est initialisée à 0. Le calcul de haché se décompose en deux phases :

- Absorption : les ℓ blocs de message sont incorporés itérativement à l'état interne par les opérations suivantes. Pour i de 1 à ℓ,

$$\begin{aligned} (R, C) &\leftarrow (R \oplus M^i, C) \\ (R, C) &\leftarrow \mathcal{F}(R, C) . \end{aligned}$$

- Essorage : le haché est généré par concaténation des variables R après des itérations successives de $\mathcal{F}$. Pour i de 1 à u,

$$\begin{aligned} z_i &\leftarrow R \\ (R, C) &\leftarrow \mathcal{F}(R, C) . \end{aligned}$$

Le haché est alors

$$H(M) = (z_1 || \ldots || z_u)|_h .$$

Cette construction est résumée sur la Figure 2.4.

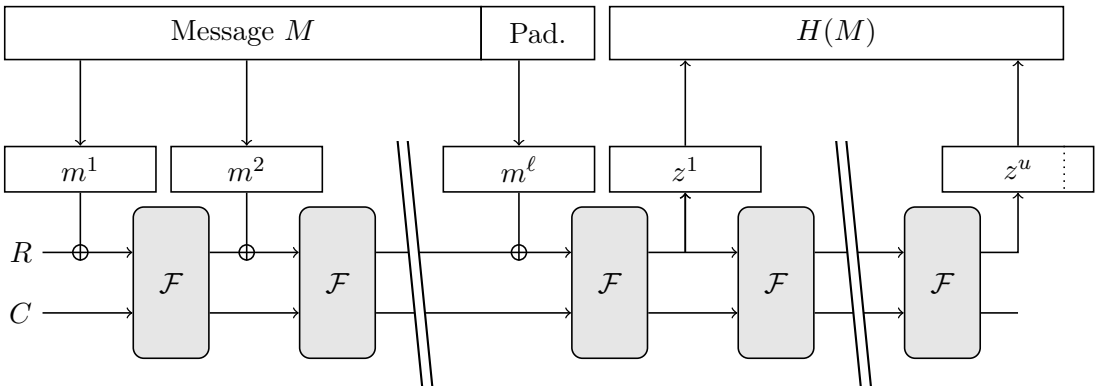


FIGURE 2.4 – Schéma général des fonctions éponges

Fonctions et permutations. L'étude théorique de la sécurité des fonctions éponges traite séparément deux cas de figure, selon que la fonction de compression interne $\mathcal{F}$ est générée comme une fonction non inversible ou comme une permutation facilement inversible (généralement notée $\mathcal{P}$). En pratique, les instanciations de cette construction utilisent des permutations internes.

Instances. Certaines fonctions de hachage récentes utilisent cette construction. RADIOGATÚN, publiée en 2006, et qui est étudiée au chapitre 7 ne rentre pas tout à fait dans ce cadre, mais présente certaines similitudes avec les fonctions éponges. Parmi les candidats à la compétition SHA-3 visant à définir une nouvelle norme en matière de fonctions de hachage, KECCAK et `cubehash` suivent la construction des fonctions éponges et `Luffa` et Fugue utilisent des constructions approchantes. La fonction de hachage à bas coût QUARK utilise également cette construction.

2.2.3 Encodage sans préfixe

Une autre manière de contourner les problèmes liés aux attaques par extension de longueur consiste à utiliser un encodage sans préfixe. Plutôt que d'utiliser un *padding* qui consiste à concaténer une certaine chaîne au message à hacher, on utilise un *encodage*, qui peut être une modification affectant tout le message. Un encodage est dit *sans préfixe* si deux messages convenablement encodés ne peuvent jamais être préfixes l'un de l'autre. La sécurité d'une telle construction a été étudiée dans [CDMP05], nous traitons également ce cas dans les chapitres 5 et 6.

Une manière spécifique de procéder est la construction HAIFA définie par Biham et Dunkelman dans [BD06]. Elle consiste à ajouter une entrée à la fonction de compression, qui représente le nombre de bits déjà traités dans les blocs de message précédents. Cette entrée peut être prise sur le bloc de message, on obtient alors un cas particulier d'encodage sans préfixe. Cette construction permet de plus de contrer les attaques en recherche de secondes préimages de Kelsey et Schneier, mais pas la recherche de multicollisions.

2.3 Arguments de sécurité

La sécurité offerte par les principaux algorithmes d'extension de domaine n'est pas uniquement liée à leur résistance à des attaques précises. Leurs concepteurs se sont attachés à montrer leur ré-

sistance dans certains modèles de sécurité. Dans cette section nous traitons des différentes manières d'apporter des arguments de sécurité pour un algorithme d'extension de domaine.

2.3.1 Conservation de propriétés de sécurité de la fonction de compression

Les garanties de sécurité apportées par la construction de Merkle-Damgård résident dans des preuves de conservation des propriétés de sécurité de la fonction de compression. Les notions de sécurité habituelles des fonctions de hachage se traduisent de manière naturelle sur les fonctions de compression. Considérons une fonction de compression $\mathcal{F}$ utilisée avec la construction de Merkle-Damgård :

$$\begin{aligned} \mathcal{F} : \{0,1\}^n \times \{0,1\}^m &\rightarrow \{0,1\}^n \\ (C, M) &\rightarrow \mathcal{F}(C, M) \ . \end{aligned}$$

Les notions de sécurité de $\mathcal{F}$ se déduisent naturellement des notions de sécurité pour une fonction de hachage.

- Résistance à la recherche de collisions : l'attaquant doit trouver deux entrées différentes (C, M) et (C', M') telless que $\mathcal{F}(C, M) = \mathcal{F}(C', M')$.
- Résistance à la recherche de secondes préimages : étant donné une entrée (C, M), l'attaquant doit trouver (C', M') différent de (C, M) tel que $\mathcal{F}(C, M) = \mathcal{F}(C', M')$.
- Résistance à la recherche de préimages : étant donné une valeur x de l'ensemble des sorties de $\mathcal{F}$, l'attaquant doit trouver (C, M) tel que $\mathcal{F}(C, M) = x$.

Pour les fonctions de compression, les attaques contre les notions définies ci-dessus sont parfois appelées recherche de *pseudo-collisions*, *pseudo-secondes préimages* et *pseudo préimages*. On rencontre également les termes de collisions, secondes préimages et préimages *libres*, cet adjectif caractérisant la liberté de l'attaquant dans le choix de la valeur de la variable de chaînage en entrée de $\mathcal{F}$.

Dans le cas de la construction de Merkle-Damgård avec le renforcement MD, on peut montrer les propriétés suivantes.

Théorème 1 *Soit H une fonction de hachage utilisant la construction de Merkle-Damgård et le renforcement MD, avec une fonction de compression $\mathcal{F}$. Les propriétés suivantes sont vérifiées :*

1. *La résistance à la recherche de collisions de $\mathcal{F}$ contre tout attaquant effectuant moins de N_{coll} calculs de $\mathcal{F}$ implique la résistance à la recherche de collisions de H contre tout attaquant effectuant moins de N_{coll} calculs de $\mathcal{F}$.*
2. *La résistance à la recherche de préimages de $\mathcal{F}$ contre tout attaquant effectuant moins de N_{pre} calculs de $\mathcal{F}$ implique la résistance à la recherche de préimages de H contre tout attaquant effectuant moins de N_{pre} calculs de $\mathcal{F}$.*

Ces résultats sont montrés dans les articles de Merkle [Mer89] pour les préimages, et de Damgård [Dam89] pour les collisions. L'étude de la robustesse des fonctions de hachage ainsi construites peut donc se ramener à l'étude de la sécurité des fonctions de compression sous-jacentes, pour les notions de résistance à la recherche de collisions ou de préimages.

La résistance d'une fonction de hachage n'est pas définie comme l'impossibilité de l'existence d'une attaque, mais comme l'ignorance d'une telle attaque dans la pratique. Dans le cas de la recherche de collisions, il est même certain qu'une telle attaque existe, comme mentionné au chapitre 1.

2.3.2 Indifférenciabilité d'un oracle aléatoire

Les spécificités de la construction de Merkle-Damgård, comme sa vulnérabilité contre les attaques en recherche de multicollisions ou par extension de longueur décrite au chapitre précédent, jettent un doute sur la robustesse de son usage comme primitive dans des mécanismes cryptographiques. D'autre part, les arguments de sécurité de ces mécanismes requièrent souvent des hypothèses sur la fonction de hachage H sous-jacente qui ne sont pas nécessairement vérifiées si H résiste aux attaques en collision, en seconde préimage ou en préimage.

Ces remarques ont conduit la communauté académique à développer de nouveaux arguments de sécurité pour les fonctions de hachage. Les algorithmes d'extension de domaine sont aujourd'hui étudiés dans le modèle de l'*indifférenciabilité de systèmes cryptographiques*, qui permet notamment de comparer un mécanisme réel avec une version idéalisée. Les fonctions de hachage sont comparées à des *oracles aléatoires*.

2.3.2.1 Oracle aléatoire

Les oracles aléatoires sont des objets très utilisés dans le domaine des preuves de sécurité de mécanismes cryptographiques. Comme nous l'avons vu au chapitre 1, de nombreux mécanismes cryptographiques font appel à des fonctions de hachage, c'est-à-dire à des fonctions $H : \{0,1\}^* \to \{0,1\}^n$.

Le modèle de l'oracle aléatoire donne un cadre à l'étude de tels mécanismes, indépendamment de la fonction de hachage sous-jacente. Il consiste à remplacer la fonction H réellement utilisée par un *oracle aléatoire*, c'est-à-dire une fonction $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ accessible uniquement en boîte noire, et dont les sorties sont indépendantes et identiquement distribuées. Le modèle de l'oracle aléatoire consiste donc à remplacer une fonction de hachage concrète H, dont les sorties sont fixes et peuvent êtres calculées par tous, par une fonction prise aléatoirement sur l'ensemble des fonctions ayant la même signature.

Ce modèle a été formalisé par Bellare et Rogaway dans [BR93]. D'autres travaux plus anciens, tels que l'heuristique de Fiat-Shamir qui permet de transformer un protocole d'authentification en schéma de signature électronique [FS86], faisaient déjà appel à ce type d'outils.

2.3.2.2 Modèle de l'indifférenciabilité

Le modèle de l'indifférenciabilité de systèmes cryptographiques a été défini par Maurer *et al.* dans [MRH04]. La définition générale de la notion d'indifférenciabilité a été adaptée par Coron *et al.* dans [CDMP05] pour l'étude d'algorithmes d'extension de domaine pour les fonctions de hachage. C'est cette dernière définition que nous utilisons dans ce mémoire.

Dans le domaine des fonctions de hachage, les preuves d'indifférenciabilité supposent l'instanciation d'un algorithme d'extension de domaine $\mathcal{C}$ avec une version idéalisée de la fonction de compression, que nous désignons également par $\mathcal{F}$ pour ne pas alourdir les notations. On cherche alors à évaluer la différence de comportement entre la fonction de hachage $\mathcal{C}^{\mathcal{F}}$ et un oracle aléatoire $\mathcal{H}$. Pour cela on considère un programme $\mathcal{D}$ appelé distingueur et ayant accès à deux boîtes noires. Ces deux boîtes noires contiennent soit la construction $\mathcal{C}^{\mathcal{F}}$ et une fonction de compression idéale $\mathcal{F}$, soit un oracle aléatoire $\mathcal{H}$ et un simulateur $\mathcal{S}$. $\mathcal{D}$ peut émettre des requêtes à ces deux boîtes sous la forme de messages M dont il obtient les empreintes, ou d'entrées de la fonction de compression pour lesquelles il reçoit les sorties. $\mathcal{D}$ retourne ensuite 0 ou 1. Informellement, la différence entre $\mathcal{C}^{\mathcal{F}}$

et $\mathcal{H}$ est caractérisée par la probabilité que $\mathcal{D}$ ait un comportement différent selon qu'il interagit avec $\mathcal{C}^{\mathcal{F}}$ ou $\mathcal{H}$.

Fonction de compression idéale. On appelle *fonction de compression idéale* de k bits vers n bits une fonction $\mathcal{F}$ tirée uniformément sur l'ensemble des fonctions

$$\{\mathcal{F} : \{0,1\}^k \to \{0,1\}^n\} .$$

En d'autres termes, une fonction de compression idéale peut être considérée comme une adaptation d'un oracle aléatoire pour des entrées de taille fixe k.

Simulateur. $\mathcal{F}$ est représentée par un oracle, auquel l'attaquant $\mathcal{D}$ et la construction $\mathcal{C}$ ont tous deux accès. Dans le cas où $\mathcal{D}$ interagit avec $\mathcal{C}^{\mathcal{F}}$, il a donc accès à deux oracles, l'un correspondant à la fonction de hachage et l'autre à la fonction de compression. $\mathcal{D}$ dispose donc d'une interface pour chacun de ces accès. Dans le scénario où $\mathcal{D}$ interagit avec $\mathcal{H}$, ce deuxième oracle est instancié par un *simulateur* $\mathcal{S}$, qui dispose lui-même d'un accès à l'oracle aléatoire.

Le simulateur a pour rôle d'émuler $\mathcal{F}$, c'est-à-dire de renvoyer des réponses aux requêtes de $\mathcal{D}$ dont la distribution de probabilités est aussi proche que possible de celle des réponses de $\mathcal{F}$. Il doit également assurer la compatibilité avec l'oracle aléatoire, c'est-à-dire faire en sorte qu'en calculant des hachés par application de la construction $\mathcal{C}$ aux réponses de $\mathcal{S}$, on retrouve les valeurs d'empreintes définies par l'oracle aléatoire.

Indifférenciabilité forte d'un oracle aléatoire. Un algorithme d'extension de domaine $\mathcal{C}$ faisant appel à une fonction de compression idéale $\mathcal{F}$ est dit $\varepsilon, N_{\mathcal{S}}, N_{\mathcal{D}}$*-fortement indifférenciable* d'un oracle aléatoire $\mathcal{H}$ s'il existe un simulateur $\mathcal{S}$ effectuant au plus $N_{\mathcal{S}}$ requêtes à $\mathcal{H}$ tel que pour tout distingueur $\mathcal{D}$ effectuant au plus $N_{\mathcal{D}}$ requêtes au total, la relation suivante est vérifiée :

$$\left|\Pr\left[\mathcal{D}(\mathcal{C}^{\mathcal{F}}, \mathcal{F}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{H}, \mathcal{S}^{\mathcal{H}}) = 1\right]\right| \leq \varepsilon .$$

Indifférenciabilité faible d'un oracle aléatoire. Une autre notion d'indifférenciabilité, plus faible que la précédente, est obtenue en permettant au simulateur de dépendre de $\mathcal{D}$. Un algorithme d'extension de domaine $\mathcal{C}$ faisant appel à une fonction de compression idéale $\mathcal{F}$ est dit $\varepsilon, N_{\mathcal{S}}, N_{\mathcal{D}}$-*faiblement indifférenciable* d'un oracle aléatoire $\mathcal{H}$ si pour tout distingueur $\mathcal{D}$ effectuant au plus $N_{\mathcal{D}}$ requêtes au total, il existe un simulateur $\mathcal{S}_{\mathcal{D}}$ effectuant au plus $N_{\mathcal{S}}$ requêtes à $\mathcal{H}$ tel que la relation suivante est vérifiée :

$$\left|\Pr\left[\mathcal{D}(\mathcal{C}^{\mathcal{F}}, \mathcal{F}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{H}, \mathcal{S}_{\mathcal{D}}^{\mathcal{H}}) = 1\right]\right| \leq \varepsilon .$$

Le scénario correspondant à la notion d'indifférenciabilité est représenté sur la Figure 2.5.

Résultats. Parmi les constructions usuelles, ChopMD et les fonctions éponges disposent de preuves d'indifférenciabilité.

Pour ChopMD, la borne initiale démontrée par Coron *et al.* dans [CDMP05] a été améliorée par Chang et Nandi dans [CN08]. L'avantage de tout distingueur $\mathcal{D}$, défini comme

$$\mathsf{Adv}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}(\mathcal{C}^{\mathcal{F}}, \mathcal{F}) = 1\right] - \Pr\left[\mathcal{D}(\mathcal{H}, \mathcal{S}_{\mathcal{D}}^{\mathcal{H}}) = 1\right]\right| ,$$

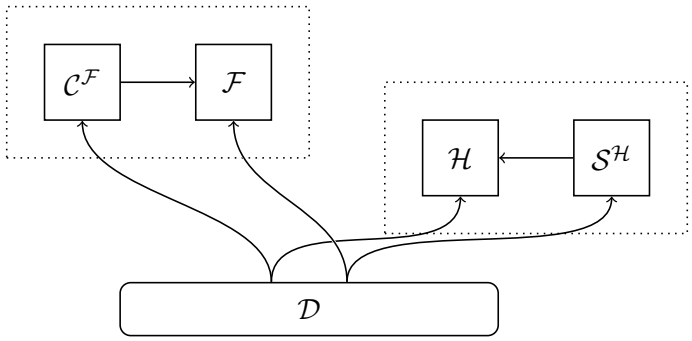


FIGURE 2.5 – Modèle de l'indifférenciabilité de l'oracle aléatoire pour un algorithme d'extension de domaine $\mathcal{C}$

est négligeable tant que le nombre total de calculs de la fonction de compression reste très inférieur à $\min(2^{n-h}, 2^{n/2})$. Nous retrouvons un résultat similaire au chapitre 5.

Dans le cas des fonctions éponges, Bertoni *et al.* montrent dans [BDPA08b] que l'avantage de l'attaquant est négligeable tant que le nombre total de calculs de la fonction de compression est inférieur à $2^{c/2}$.

Oracle aléatoire à usage public. D'après la définition de l'indifférenciabilité, la construction de Merkle-Damgård n'est pas indifférenciable d'un oracle aléatoire. En effet, la propriété d'extension de longueur, qui est vérifiée pour les paddings classiques, est propre à cette construction et permet d'établir une différence avec l'oracle aléatoire.

Les fonctions de hachage les plus répandues, notamment la famille SHA-2, utilisent la construction de Merkle-Damgård. Il est donc important d'obtenir le plus de garanties de sécurité possibles pour les mécanismes cryptographiques utilisant ces fonctions. Dans un article d'Eurocrypt 2009 [DRS09], Dodis *et al.* introduisent le concept d'*oracle aléatoire à usage public*. Un tel oracle $\mathcal{H}$ dispose de deux interfaces $(\mathcal{H}^{eval}, \mathcal{H}^{reval})$. Par l'interface $\mathcal{H}^{eval}$, $\mathcal{H}$ reçoit des requêtes de haché (c'est-à-dire des messages M) et renvoie les valeurs $\mathcal{H}(M)$ correspondantes. Par l'interface $\mathcal{H}^{reveal}$, $\mathcal{H}$ rend publiques toutes les valeurs de M reçues par l'interface $\mathcal{H}^{eval}$. Un tel objet permet de modéliser une fonction de hachage dans des usages pour lesquels les messages dont les empreintes sont calculées sont publics.

Les auteurs de cet article montrent que le mode de Merkle-Damgård est indifférenciable d'un oracle aléatoire public.

2.3.2.3 Discussion du modèle

Intuitivement, le fait qu'une construction soit indifférenciable de l'oracle aléatoire signifie qu'elle se comporte comme un oracle aléatoire lorsqu'elle est instanciée avec une fonction de compression idéale. Cependant, l'enjeu réel en terme de sécurité n'est pas la résistance d'un algorithme d'extension de domaine à un distingueur $\mathcal{D}$, mais la robustesse des cryptosystèmes utilisant des fonctions de hachage.

L'introduction de ce modèle pour évaluer la sécurité des algorithmes d'extension de domaine est motivée par Maurer *et al.* par un théorème de composabilité. Dans le cas des fonctions de hachage, ce théorème peut-être interprêté de la manière suivante. Soit $\mathcal{T}$ un cryptosystème utilisant une

fonction de hachage comme primitive interne et $\mathcal{C}$ un algorithme d'extension de domaine. Si $\mathcal{C}$ est indifférenciable de l'oracle aléatoire (au sens de l'indifférenciabilité faible), alors $\mathcal{T}(\mathcal{C}^{\mathcal{F}})$ est au moins aussi sûr que $\mathcal{T}(\mathcal{H})$.

Un résultat récent découvert par Ristenpart *et al.* [RSS11] montre que ce théorème ne s'applique pas lorsque le scénario d'attaque contre $\mathcal{T}$ comporte plusieurs phases. Le théorème s'applique notamment pour des notions de sécurité telles que l'indistinguabilité dans le cadre du chiffrement à clé publique, ou l'impossibilité de falsifier une signature électronique.

De nombreux exemples classiques de tels mécanismes cryptographiques sont prouvés sûrs lorsque la fonction de hachage sous-jacente est un oracle aléatoire. Le théorème de composabilité de Maurer *et al.* permet d'étendre ce résultat au cas où la fonction de hachage utilisé est construite à partir de $\mathcal{C}$, instancié avec une fonction de compression idéale. La robustesse des applications réelles est ainsi reliée à la sécurité des fonctions de compression.

Chapitre 3

La famille MD-SHA

Sommaire

Après avoir décrit les principaux algorithmes d'extension de domaine, nous abordons dans ce chapitre le thème des fonctions de compression. Contrairement aux algorithmes d'extension de domaine, chaque fonction de hachage utilise une fonction de compression qui lui est propre.

3.1 Constructions classiques de fonctions de compression

Pour des applications pratiques, il est souvent nécessaire de hacher un gros volume de données. La fonction de compression doit permettre le traitement rapide de ces données. La même contrainte pesant sur les primitives de chiffrement symétrique, les principes de conception des fonctions de compression sont souvent proches des principes de conception de ces primitives. Dans cette section, nous commençons par décrire quelques méthodes classiques permettant de construire une fonction de compression à partir d'un algorithme de chiffrement par blocs.

Les applications de chiffrement symétrique sont les plus anciennes du domaine de la cryptographie. Au moment de l'apparition du besoin de fonctions de hachage cryptographiques, les primitives de chiffrement par blocs étaient déjà des objets bien connus de la communauté académique. La norme DES (Data Encryption Standard), publiée par le National Bureau of Standards en 1977 [NIS99], n'a été cryptanalysée qu'une quinzaine d'années plus tard. Son obsolescence et son remplacement par l'AES sont au moins autant la conséquence d'une taille de clé et d'une taille de bloc devenues trop petites que d'attaques théoriques. Il est donc naturel d'essayer de construire des fonctions de hachage à partir de ce type de primitives. Nous montrons maintenant quelques constructions permettant d'y parvenir.

Fonctionnellement, une primitive de chiffrement par blocs est une famille $\mathcal{E}$ de permutations de $\{0,1\}^n$, paramétrées par une clé k de $\{0,1\}^m$. Pour toute permutation $\mathcal{E}_k$, il existe une permutation inverse $\mathcal{D}_k$ telle que pour tout bloc de message M, $\mathcal{D}_k(\mathcal{E}_k(M)) = \mathcal{E}_k(\mathcal{D}_k(M)) = M$. Pour les constructions comme l'algorithme de Merkle-Damgård, la fonction de compression a pour entrées un bloc de message M^i et une variable de chaînage H^{i-1}. Pour obtenir la non-inversibilité de la fonction de hachage, il est nécessaire qu'en connaissant (H^{i-1}, H^i), on ne puisse pas retrouver un bloc de message M^i tel que $\mathcal{F}(H^{i-1}, M^i) = H^i$. Pour obtenir cette propriété, une première idée serait d'utiliser le bloc M^i comme clé de $\mathcal{E}$, et la variable chaînage d'entrée H^{i-1} comme bloc de clair. Cela revient à définir :

$$\mathcal{F}(H^{i-1}, M^i) = \mathcal{E}_{M^i}(H^{i-1}) \ .$$

Une telle fonction de compression permet cependant de retrouver facilement H^{i-1} en connaissant H^i et M^i. En effet, connaissant $\mathcal{H}^i$, on peut choisir n'importe quelle valeur de M^i et calculer

$$H^{i-1} = \mathcal{D}_{M^i}(H^i) \ .$$

Cette attaque peut être étendue en une attaque en recherche de préimages à IV choisi sur la fonction de hachage. Elle peut également être combinée avec la technique de recherche au milieu pour trouver des (secondes) préimages avec une complexité de l'ordre de $2^{n/2}$ calculs de $\mathcal{E}$ ou $\mathcal{D}$.

Diverses solutions ont été définies pour pallier ce problème. Elles consistent à utiliser la primitive de chiffrement $\mathcal{E}$ avec une opération de rebouclage : une opération de OU EXCLUSIF est appliquée entre la sortie et une (ou les deux) entrée(s) de $\mathcal{E}$. La solution la plus utilisée est attribuée à Davies et Meyer dans [PGV93]. Elle consiste à chiffrer la variable de chaînage entrante avec le bloc de message, et à effectuer un rebouclage avec cette variable de chaînage. Nous avons donc

$$\mathcal{F}_{(DM)}(H^{i-1}, M^i) = \mathcal{E}_{M^i}(H^{i-1}) \oplus H^{i-1} \ .$$

Cette construction est utilisée dans les principales normes de hachage, comme les fonctions MD4, MD5, SHA-0, SHA-1 et la famille SHA-2. Elle est représentée en Figure 3.1.

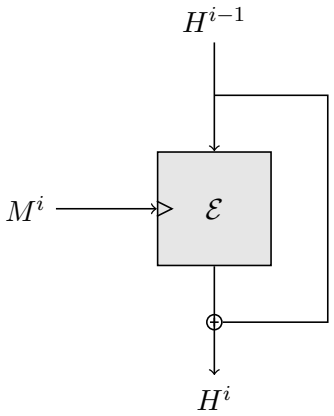


FIGURE 3.1 – Mode de Davies-Meyer pour construire une fonction de compression à partir d'une primitive de chiffrement par blocs $\mathcal{E}$.

Une autre construction connue est celle de Matyas, Meyer et Oseas [MMO85]. Par rapport à celle de Davies et Meyer, elle consiste à inverser les rôles de la variable de chaînage d'entrée et du bloc de message. Nous avons donc

$$\mathcal{F}_{(MMO)}(H^{i-1}, M^i) = \mathcal{E}_{H^{i-1}}(M^i) \oplus M^i \ .$$

Cette construction a ensuite été modifiée par Miyaguchi et Preneel [PGV93, MIO89], dont la construction impose le rebouclage avec $M^i \oplus H^{i-1}$, ce qui nous donne

$$\mathcal{F}_{(MP)}(H^{i-1}, M^i) = \mathcal{E}_{H^{i-1}}(M^i) \oplus M^i \oplus H^{i-1} \ .$$

Cette dernière construction est notamment utilisée par la fonction de hachage Whirlpool [BR03], définie par Barreto et Rijmen.

Dans [PGV93], Preneel *et al.* donnent une représentation générale de ces constructions, et étudient leur sécurité. Chacune des entrées de $\mathcal{E}$, ainsi que le deuxième opérande du rebouclage, vaut soit une constante, soit H^{i-1}, soit M^i, soit $H^{i-1} \oplus M^i$. Cette représentation englobe 64 constructions, dont seules 49 font intervenir à la fois le bloc de message et la variable de chaînage. Parmi ces 49 constructions, seules 12 sont conjecturées sûres, dont les trois citées précédemment. Dans [BRS02], Black *et al.* montrent la sécurité de ces constructions, en modélisant $\mathcal{E}$ comme un algorithme de chiffrement par blocs idéal. Comme un oracle aléatoire, un algorithme de chiffrement par blocs idéal est une boîte noire, qui instancie une famille de permutations tirée uniformément sur l'ensemble des familles de permutations paramétrées.

Dimensionnement des fonctions de compression. Une idée naturelle consiste à utiliser directement ces modes avec des algorithmes de chiffrement par blocs bien étudiés. Une telle démarche ne permet cependant pas d'obtenir une sécurité suffisante. Supposons en effet qu'une fonction de hachage soit construite à partir de l'AES en mode de Davies-Meyer, et de l'algorithme de Merkle-Damgård. On peut ainsi générer des empreintes de 128 bits, et une attaque en recherche de collisions tirant avantage du paradoxe des anniversaires nécessite en moyenne environ 2^{64} évaluations de $\mathcal{F}$, ce qui est insuffisant au regard des capacités de calcul actuellement disponibles.

Pour contrer ce problème, deux approches sont possibles. La première consiste à utiliser des modes qui permettent de construire des fonctions de compression à partir d'algorithmes de chiffrement par blocs et qui définissent des variables de chaînage de taille supérieure à la taille de bloc. La seconde consiste à définir des permutations paramétrées spécifiques, dont les longueurs des blocs et des clés sont choisies pour pouvoir être utilisées avec la construction de Davies-Meyer. C'est cette approche qui a été adoptée par les concepteurs des fonctions de hachage les plus utilisées actuellement.

D'autre part, de nouveaux modèles d'attaques contre les primitives de chiffrement par blocs ont été développés récemment, comme les attaques à clés corrélées ou les distingueurs à clé connue ou à clé choisie. Ces modèles supposent que l'attaquant dispose d'une certaine maîtrise sur l'entrée de clé de $\mathcal{E}$, ce qui n'est pas pertinent dans le cas des constructions usuelles de modes de chiffrement ou de MAC à partir de $\mathcal{E}$. En revanche, de telles attaques peuvent avoir des conséquences pratiques si elles sont appliquées à des fonctions de hachage construites sur des algorithmes de chiffrement par blocs.

3.2 La famille MD-SHA

Nous nous intéressons maintenant à la famille MD-SHA, qui contient les principales normes de hachage cryptographique.

3.2.1 Les premières fonctions

MD2. La première fonction de hachage cryptographique à avoir été utilisée est MD2, conçue par Rivest en 1989 et normalisée par l'IETF dans [Kal92]. On en trouve aujourd'hui encore la trace dans les certificats de certaines infrastructures de gestion de clés. Rogier et Chauvaud ont montré dans [RC97] comment générer des collisions pour la fonction de compression. Plus tard, des attaques en recherche de collisions [KM05, KMMT10] et de préimages [Mul04, KMMT10] sur la fonction de hachage ont été découvertes.

MD4. Plus que MD2, MD4 peut être considérée comme l'origine de la famille MD-SHA. En effet, les principes ayant présidé à la conception de sa fonction de compression sont largement repris par les fonctions suivantes. Toutefois, cette fonction n'a été que peu utilisée, étant très vite remplacée par MD5. Elle a été conçue par Rivest en 1990 et normalisée par l'IETF dans [Riv92a]. MD4 utilise la construction de Merkle-Damgård avec des variables de chaînage de 128 bits et des blocs de message de 512 bits, qui sont divisés en variables de 32 bits. La fonction de compression est construite selon le mode de Davies-Meyer, l'opération de OU EXCLUSIF du rebouclage étant remplacée par une addition modulo 2^{32} des 4 mots de 32 bits qui composent la variable de chaînage. La fonction de compression consiste à modifier alternativement chacun des 4 registres de la variable de chaînage, par une opération faisant intervenir l'un des 16 registres du message. 48 étapes sont ainsi effectuées. Den Boer et Bosselaers ont découvert les premières faiblesses de MD4 en 1991 [dBB91], ce qui a conduit à l'abandon de MD4 au profit de MD5. La première attaque en recherche de collisions est due à Dobbertin en 1996 [Dob96]. En 2005, Wang *et al.* ont publié une série d'attaques contre des fonctions de hachage, dont MD4, pour laquelle la recherche de collisions est instantanée [WLF+05]. La résistance à la recherche de préimages de MD4 a ensuite été cassée par Leurent en 2008 [Leu08].

MD5. MD5 a été conçue par Rivest en 1991 [Riv92b]. Cette fonction a été très utilisée, elle est toujours disponible aujourd'hui dans de nombreux produits et protocoles. Elle résulte d'une modification de MD4 : 16 étapes sont ajoutées à la fonction de compression, l'étape élémentaire en est légèrement modifiée. Comme pour MD4, les premiers résultats marquants de cryptanalyse de MD5 ont été dus à Den Boer et Bosselaers [dBB93]. Leur attaque permet la recherche de collisions sur la fonction de compression. L'attaque de Wang et Yu est la première à avoir permis la recherche de collisions sur MD5 [WY05]. Ces attaques ont été ensuite améliorées, et permettent aujourd'hui de trouver des collisions sur MD5 instantanément. Sasaki et Aoki ont découvert une attaque en recherche de préimages contre MD5 [SA09].

SHA-0. SHA-0 est la première des fonctions de la famille SHA. Elle a été conçue par la NSA en 1993 et normalisée par le NIST, avant d'être retirée peu après et remplacée par SHA-1. Les choix effectués pour sa conception n'ont pas été expliqués publiquement, mais les principes généraux reprennent dans les grandes lignes ceux de MD4 et MD5. SHA-0 permet de calculer des empreintes de 160 bits, ce qui représente un gain substantiel quant à la complexité des attaques génériques

par rapport aux fonctions précédentes. Dès 1995, le NIST a publié la fonction SHA-1, que nous détaillons ci-dessous et qui représente une correction mineure par rapport à SHA-0.

Les premières faiblesses de SHA-0 ont été découvertes par Chabaud et Joux dans [CJ98], qui sont parvenus à montrer une attaque théorique contre SHA-0. La première collision a été montrée par Joux en 2004. Les attaques de Wang *et al.* s'appliquent également à SHA-0 et ont permis d'accélérer fortement la recherche de collisions pour cette fonction [WYY05b]. Une attaque de Peyrin et Manuel permet de trouver des collisions pour SHA-0 en pratique, avec une puissance de calcul raisonnable [MP08]. Du fait de la proximité entre la publication de SHA-0 et celle de SHA-1, SHA-0 n'a que rarement été utilisée dans la pratique.

3.2.2 SHA-1

SHA-1 a été publiée en 1995 par le NIST pour pallier une faiblesse de SHA-0 [NIS95]. Nous décrivons maintenant les détails de sa fonction de compression. Ce choix est motivé par deux raisons :

- elle constitue une cible représentative des différentes techniques de cryptanalyse mises en œuvre contre la famille MD-SHA ;
- malgré ses vulnérabilités connues qui conduisent à son remplacement progressif par SHA-2, elle reste probablement aujourd'hui la fonction la plus utilisée dans la pratique.

SHA-1 est construite avec l'algorithme d'extension de domaine de Merkle-Damgård, et le padding utilisé est le renforcement MD décrit dans le chapitre 2. Sa fonction de compression permet le traitement de blocs de message de 512 bits et de variables de chaînage de 160 bits. Elle est construite à partir d'une permutation paramétrée en mode de Davies-Meyer, où le rebouclage est remplacé par 5 additions modulo 2^{32} entre les 5 registres de 32 bits d'entrée de la variable de chaînage et les 5 registres de sortie de la permutation.

Le bloc de message, divisé en 16 mots de 32 bits $(M_0, \ldots, M_{15})$ passe par une expansion linéaire (dans $GF(2)$) permettant d'obtenir 80 mots de 32 bits, de la manière suivante.

$$\begin{aligned} \forall i \in \{0, \ldots, 15\}, W_i &= M_i \\ \forall i \in \{16, \ldots, 79\}, W_i &= (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 \ . \end{aligned}$$

Cette rotation de 1 bit vers la gauche dans l'expansion de message constitue l'unique différence entre SHA-0 et SHA-1. A partir de la variable de chaînage d'entrée, on initialise 5 registres A, B, C, D et E.

$$A_0 = H_0^{i-1},\ B_0 = H_1^{i-1},\ C_0 = H_2^{i-1},\ D_0 = H_3^{i-1},\ E_0 = H_4^{i-1}$$

La permutation paramétrée de SHA-1 se décompose en 4 tours de 20 étapes. Chaque étape est définie par les opérations suivantes :

$$\begin{aligned} A_{i+1} &= (A_i \lll 5) \boxplus \Phi_i(B_i, C_i, D_i) \boxplus E_i \boxplus W_i \boxplus k_i \\ B_{i+1} &= A_i \\ C_{i+1} &= B_i \lll 30 \\ D_{i+1} &= C_i \\ E_{i+1} &= D_i \ , \end{aligned}$$

où k_i est une constante dépendant du numéro de l'étape et Φ_i est la juxtaposition de 32 fonctions identiques de 3 bits vers 1 bit, définies par

$$\Phi_i(x,y,z) = \left\{ \begin{array}{lcll} ITE(x,y,z) & = & xy \oplus \overline{x}z & \forall i \in \{0,\dots 19\} \\ XOR(x,y,z) & = & x \oplus y \oplus z & \forall i \in \{20,\dots 39\} \\ MAJ(x,y,z) & = & xy \oplus xz \oplus yz & \forall i \in \{40,\dots 59\} \\ XOR(x,y,z) & = & x \oplus y \oplus z & \forall i \in \{60,\dots 79\} \end{array} \right)$$

Une étape de la fonction de compression de SHA-1 est décrite en Figure 3.2.

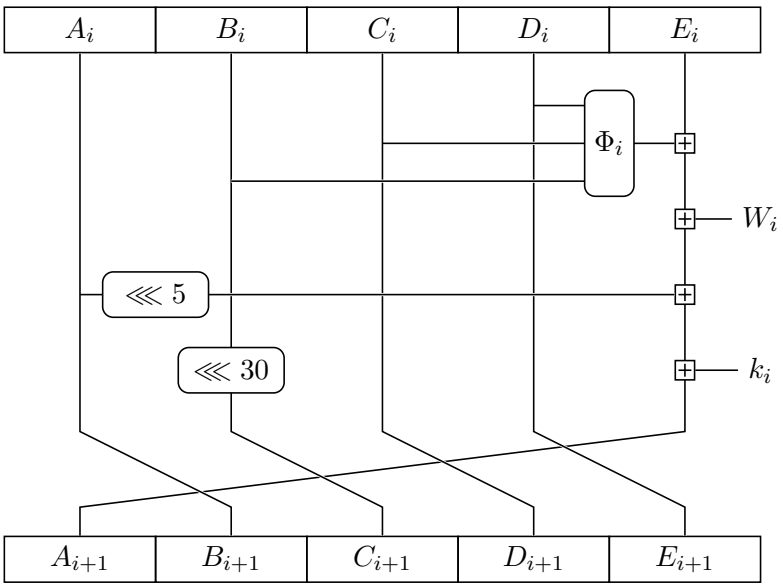


FIGURE 3.2 – Une étape de la fonction de compression de SHA-1

La nouvelle valeur de la variable de chaînage est définie par

$$\begin{array}{rcl} H_0^i & = & A_0 \boxplus A_{80} \\ H_1^i & = & B_0 \boxplus B_{80} \\ H_2^i & = & C_0 \boxplus C_{80} \\ H_3^i & = & D_0 \boxplus D_{80} \\ H_4^i & = & E_0 \boxplus E_{80} \ . \end{array}$$

Sécurité. SHA-1 est aujourd'hui la cible d'une attaque théorique en recherche de collisions, dont la complexité est estimée à la limite des puissances de calcul actuellement atteignables. Les techniques ayant permis d'aboutir à cette attaque sont évoquées en section 3.3.

3.2.3 La famille SHA-2

En 2002 le NIST a normalisé une nouvelle famille de fonctions de hachage, SHA-2 [NIS02]. Quatre versions de cet algorithme permettent de calculer des empreintes de 224, 256, 384 et 512 bits. Comme ses prédécesseurs, SHA-2 utilise l'algorithme d'extension de domaine de Merkle-Damgård

et la construction de Davies-Meyer. La taille des variables de chaînage est de 256 bits pour les deux premières versions et de 512 bits pour les deux dernières. Les blocs de message correspondants ont pour tailles respectives 512 et 1024 bits. De manière générale, SHA-224 et SHA-256 utilisent des registres de 32 bits, et SHA-384 et SHA-512 utilisent des registres de 64 bits. La structure générale des opérations effectuées est similaire entre ces quatre fonctions.

Le bloc de message passe par une expansion permettant de générer 64 mots pour les fonctions SHA-224 et SHA-256 et 80 mots pour SHA-384 et SHA-512. Cette expansion est définie par

$$\begin{aligned} \forall i \in \{0,\ldots,15\}, W_i &= M_i \\ \forall i \in \{16,\ldots,79\}, W_i &= (\sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16}) \lll 1 \ . \end{aligned}$$

Pour SHA-224 et SHA-256, σ_0 et σ_1 sont définies par

$$\begin{aligned} \sigma_0(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) \\ \sigma_1(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) \ . \end{aligned}$$

Pour SHA-384 et SHA-512, ces fonctions sont définies par

$$\begin{aligned} \sigma_0(x) &= (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7) \\ \sigma_1(x) &= (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6) \ . \end{aligned}$$

La variable de chaînage d'entrée permet d'initialiser 8 registres A, B, C, D, E, F, G et H.

$$\begin{gathered} A_0 = H_0^{i-1},\ B_0 = H_1^{i-1},\ C_0 = H_2^{i-1},\ D_0 = H_3^{i-1}, \\ E_0 = H_4^{i-1},\ F_0 = H_5^{i-1},\ G_0 = H_6^{i-1},\ H_0 = H_7^{i-1}. \end{gathered}$$

Les permutations paramétrées utilisées se composent de 64 étapes pour SHA-224 et SHA-256 et de 80 étapes pour SHA-384 et SHA-512. Ces étapes élémentaires sont identiques, la notion de tour disparaît. Elles sont définies par

$$\begin{aligned} T_1 &= ITE(E_i, F_i, G_i) \boxplus \Sigma_1(E_i) \\ T_2 &= MAJ(A_i, B_i, C_i) \boxplus \Sigma_0(A_i) \\ A_{i+1} &= H_i \boxplus T_1 \boxplus T_2 \boxplus W_i \boxplus k_i \\ B_{i+1} &= A_i \\ C_{i+1} &= B_i \\ D_{i+1} &= C_i \\ E_{i+1} &= H_i \boxplus T_1 \boxplus W_i \boxplus k_i \\ F_{i+1} &= E_i \\ G_{i+1} &= F_i \\ H_{i+1} &= G_i \end{aligned}$$

Pour SHA-224 et SHA-256, Σ_0 et Σ_1 sont définies par

$$\begin{aligned}\Sigma_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22)\\ \Sigma_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25)\ .\end{aligned}$$

Pour SHA-384 et SHA-512, ces fonctions sont définies par

$$\begin{aligned}\Sigma_0(x) &= (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39)\\ \Sigma_1(x) &= (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41)\ .\end{aligned}$$

L'étape élémentaire de ces fonctions de compression est représentée sur la Figure 3.3.

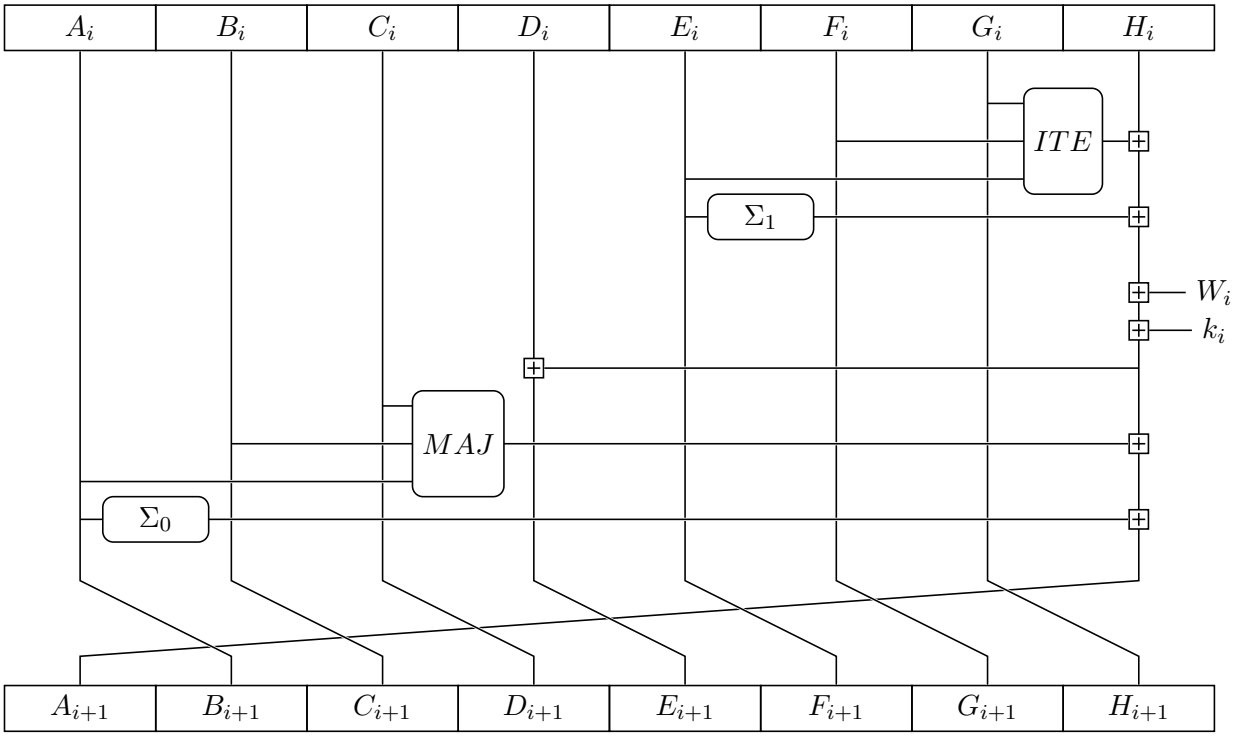


FIGURE 3.3 – Une étape de la fonction de compression de SHA-2

La nouvelle valeur de la variable de chaînage est obtenue par des additions modulo 2^{32} ou 2^{64} des valeurs initiales et finales des registres A à H.

Sécurité. Bien que la famille SHA-2 ait une structure très similaire à ses prédécesseurs, aucune attaque meilleure que les attaques génériques n'est aujourd'hui connue. Le caractère non linéaire de l'expansion de message qui s'ajoute aux non linéarités de la fonction d'étape rend difficile l'application des techniques habituelles. La mise à jour simultanée de deux registres introduit également de la complexité dans la fonction de compression.

3.3 La cryptanalyse différentielle

Les techniques ayant mis à mal la sécurité de MD4 et de ses successeurs décrits en section 3.2 utilisent toutes une base commune : la cryptanalyse différentielle. Dans cette section, nous décrivons

l'idée de base et les principales améliorations possibles à travers l'exemple de SHA-1.

3.3.1 Idée de la cryptanalyse différentielle

La cryptanalyse différentielle s'est avérée un outil remarquablement efficace pour trouver des collisions pour certaines fonctions de hachage. Dans le contexte des fonctions de hachage, l'idée générale de la cryptanalyse différentielle peut se résumer de la manière suivante. Plutôt que de chercher une collision sur les empreintes de messages pris au hasard, l'attaquant cherche une collision entre les empreintes de deux messages M et M' vérifiant une propriété particulière. Cette propriété est une valeur constante de la différence entre M et M'. Cette technique a été initialement développée par Biham et Shamir pour attaquer le DES [BS92] et plus généralement des algorithmes de chiffrement par blocs.

Notion de différence. Avant toute chose, il est nécessaire de définir ce que nous entendons par *différence* entre deux messages M et M'. La définition la plus naturelle est liée à leur représentation binaire : pour deux messages M et M' de même longueur, la différence entre M et M' correspond à $M \oplus M'$ et caractérise les positions pour lesquels les bits de M et les bits de M' diffèrent. Cette notion est souvent appelée *différence xor*.

Une autre définition intéressante pour les fonctions de la famille MD-SHA est la différence modulaire. Les fonctions de la famille MD-SHA utilisant beaucoup d'additions modulo 2^{32}, la différence entre deux blocs de message est représentée par la soustraction modulo 2^{32} des registres de 32 bits aux mêmes positions dans M et M'.

Les attaques de l'équipe de Wang tirent parti d'une troisième définition de la différence : la différence signée. La différence entre deux messages M et M' de m bits est représentée par un vecteur Δ de m valeurs prises dans $\{-1, 0, 1\}$, la $i^{ème}$ coordonnée Δ_i étant définie par

$$\Delta_i = M_i - M'_i \ ,$$

la soustraction étant prise dans $\mathbb{Z}$. Cette notion de différence a pour avantage de représenter sans équivoque toute valeur de différence xor et toute valeur de différence modulaire.

Chemins différentiels. La difficulté de l'attaque réside dans la recherche d'un *chemin différentiel*. L'idée de la cryptanalyse différentielle est la suivante. Une fonction de compression est construite comme une suite d'opérations élémentaires sur le bloc de message. Pour chacune de ces opérations, une modification sur un bit d'un des opérandes conduit probablement à une différence sur un petit nombre de bits du résultat. La cryptanalyse différentielle revient à chercher à contrôler la valeur de la différence entre les résultats d'une même opération au cours des calculs de hachés de M et M'. La succession des valeurs des différences qui sont visées est appelée *chemin différentiel*.

La deuxième étape de l'attaque consiste alors en une recherche d'une paire de messages M et M' qui suivent le chemin différentiel. Pour une paire de messages prise aléatoirement, chacune des opérations respecte le chemin différentiel avec une certaine probabilité. Cette phase de recherche est donc probabiliste.

Linéarisation de la fonction de compression. Dans l'attaque de Chabaud et Joux contre SHA-0 [CJ98], l'idée de linéariser la fonction de compression est développée. Au lieu de considérer la fonction de compression de SHA-0 (ou SHA-1), on remplace chacune des opérations élémentaires

par des opérations linéaires. Les additions modulaires et les fonctions Φ_i sont donc remplacées par des opérations de OU EXCLUSIF. Les valeurs de messages conduisant à des collisions sont donc les éléments de sous-espaces affines de $\{0,1\}^m$ qui peuvent être déterminés facilement. Le principe de l'attaque consiste alors à chercher des vecteurs dans l'espace des différences qui induisent peu de différences sur les résultats des calculs intermédiaires. Ces valeurs de différences sont utilisées comme chemin différentiel pour la fonction de compression réelle.

3.3.2 Application à SHA-1

Nous revenons maintenant à l'exemple spécifique de SHA-1.

Collisions locales. Les chemins différentiels pour SHA-1 reposent sur le phénomène de *collisions locales*, qui est défini par le raisonnement suivant. Supposons qu'au cours du calcul des fonctions de compression $\mathcal{F}(H, M)$ et $\mathcal{F}(H', M')$ on ait une collision sur les valeurs intermédiaires des registres A à E :

$$A_i = A'_i,\ B_i = B'_i,\ C_i = C'_i,\ D_i = D'_i,\ E_i = E'_i\ .$$

Supposons alors que les registres W_{i+1} dépendant de M et M' diffèrent uniquement sur le bit j. En notant e_j le vecteur de 32 bits ayant uniquement un 1 en position j, on a $W'_{i+1} = W_{i+1} \oplus e_j$. Après l'étape $i+1$, seul le registre A dépend de W_{i+1}. Ce registre est obtenu par addition modulo 2^{32} de W_{i+1} avec une valeur dépendant des autres registres. Lorsqu'on pose cette addition avec des écritures binaires, si la retenue ne se propage pas, seul le bit j du résultat diffère, et

$$A_{i+1} = A'_{i+1} \oplus e_j,\ B_{i+1} = B'_{i+1},\ C_{i+1} = C'_{i+1},\ D_{i+1} = D'_{i+1},\ E_{i+1} = E'_{i+1}\ .$$

A l'étape $i+2$, la différence sur A intervient à deux endroits. D'une part elle se propage directement sur B. D'autre part, elle est additionnée après une rotation de 5 vers la gauche à une valeur dépendant des autres registres et à W_{i+2} pour obtenir A_{i+2}. Si nous imposons une différence de signe inverse sur les bits $j+5$ de W_{i+2}, la différence s'annule sur le résultat de cette addition, et

$$A_{i+2} = A'_{i+2},\ B_{i+2} = B'_{i+2} \oplus e_j,\ C_{i+2} = C'_{i+2},\ D_{i+2} = D'_{i+2},\ E_{i+2} = E'_{i+2}\ .$$

A l'étape $i+3$, la différence sur le bit j du registre B se transforme en une différence sur le bit $j+30$ du registre C. Elle peut également donner lieu à une différence sur le bit j de $\Phi_{i+2}(B_{i+2}, C_{i+2}, D_{i+2})$. Dans ce cas, son influence sur A_{i+3} peut être annulée par une différence de signe inverse sur le bit j de W_{i+3}. Nous avons alors :

$$A_{i+3} = A'_{i+3},\ B_{i+3} = B'_{i+3},\ C_{i+3} = C'_{i+3} \oplus e_{j+30},\ D_{i+3} = D'_{i+3},\ E_{i+3} = E'_{i+3}\ .$$

De la même manière, la différence sur le bit $j+30$ de C avant l'étape $i+4$ passe sur le bit $j+4$ du registre D, et son influence sur A peut être annulée par une différence sur le bit $j+30$ de W_{i+4}. Il en résulte alors que

$$A_{i+4} = A'_{i+4},\ B_{i+4} = B'_{i+4},\ C_{i+4} = C'_{i+4},\ D_{i+4} = D'_{i+4} \oplus e_{j+30},\ E_{i+4} = E'_{i+4}\ .$$

Le même phénomène se produit à l'étape $i+5$. Une différence sur le bit $j+30$ de W_{i+5} se traduit par

$$A_{i+5} = A'_{i+5},\ B_{i+5} = B'_{i+5},\ C_{i+5} = C'_{i+5},\ D_{i+5} = D'_{i+5},\ E_{i+5} = E'_{i+5} \oplus e_{j+30}\ .$$

Etape	ΔA	ΔB	ΔC	ΔD	ΔE	ΔW
i	-	-	-	-	-	j
$i+1$	j	-	-	-	-	$j+5$
$i+2$	-	j	-	-	-	j
$i+3$	-	-	$j+30$	-	-	$j+30$
$i+4$	-	-	-	$j+30$	-	$j+30$
$i+5$	-	-	-	-	$j+30$	$j+30$
$i+6$	-	-	-	-	-	-

TABLE 3.1 – Masque de collision locale pour la fonction de compression de SHA-1

Enfin, après l'étape $i+6$, une différence sur le bit $j+30$ de W_{i+6} peut entraîner

$$A_{i+6} = A'_{i+6},\ B_{i+6} = B'_{i+6},\ C_{i+6} = C'_{i+6},\ D_{i+6} = D'_{i+6},\ E_{i+6} = E'_{i+6}\ .$$

La différence sur les registres a été complètement annulée après 6 étapes. Ce phénomène est appelé collision locale. Pour SHA-1, il peut être résumé dans la table 3.1.

Vecteur de perturbation. La stratégie permettant de trouver des chemins différentiels pour la fonction de compression de SHA-1 consiste à combiner de telles collisions locales. Pour y parvenir, le masque des différences introduites par les variables W_i doit être compatible avec l'expansion du bloc de message. Nous rappelons ici la relation de récurrence permettant de générer les mots W_{16} à W_{79} :

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1\ .$$

Les vecteurs de 80 mots de 32 bits résultant de cette opération sont les éléments d'un code linéaire. La différence entre deux mots du code vérifie donc également cette relation. Les équations ont les propriétés suivantes :

- si on applique une même rotation sur tous les W_i d'un mot du code, on obtient encore un mot du code ;
- la relation entre 17 valeurs de W_i consécutives ne dépend pas de i ;
- cette relation est inversible : on peut calculer toutes les valeurs de W_i à partir de 16 valeurs consécutives.

Choisissons alors 16 valeurs consécutives $W^0_0, \ldots, W^0_{15}$, et calculons les 64 valeurs $W^0_{16}, \ldots, W^0_{79}$ ainsi que 5 valeurs $W^0_{-5}, \ldots, W^0_{-1}$ en inversant la relation définissant l'expansion. Nous appelons $V^0 = (W^0_0, \ldots, W^0_{80})$ le bloc de message étendu ainsi obtenu. Ce vecteur est appelé *vecteur de perturbation*, il correspond à un vecteur de différences contenant le départ des collisions locales.

Si les mots $W^0_{-5}, \ldots, W^0_{-1}$ sont tous nuls, nous pouvons définir pour j de 1 à 5 les expansions de message $V^j = (W^j_0, \ldots, W^j_{80})$, avec pour tout i de 0 à 15 :

$$\begin{aligned} W^1_{i+1} &= W^0_i \lll 5 \\ W^2_{i+2} &= W^0_i \\ W^3_{i+3} &= W^0_i \lll 30 \\ W^4_{i+4} &= W^0_i \lll 30 \\ W^5_{i+5} &= W^0_i \lll 30 \end{aligned}$$

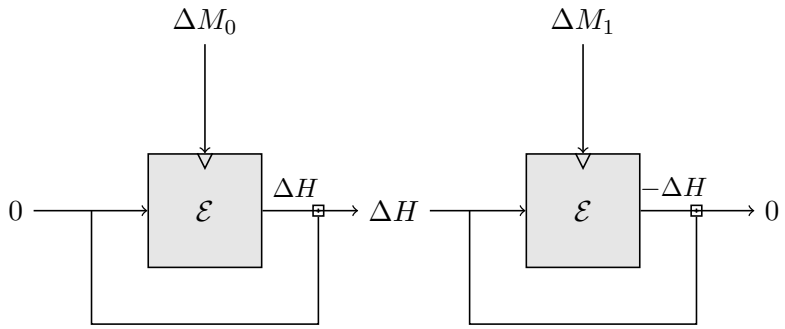


FIGURE 3.4 – Propagation possible de la différence sur deux itérations de la fonction de compression de SHA-1.

Ces relations s'étendent alors à toutes les valeurs de i de 0 à 79. En définissant

$$W = V^0 \oplus V^1 \oplus V^2 \oplus V^3 \oplus V^4 \oplus V^5 \ ,$$

nous voyons qu'à chaque bit valant 1 de V^0, un masque de collision locale est formé par un bit à 1 de chacun des V^i, pour i de 1 à 5. Comme les W^{-i} sont toutes nulles, les bits des premiers mots des V^i le sont également. Tous les bits valant 1 des V^i font donc partie d'une collision locale. Si de plus les 5 dernières valeurs $W^{75}, \ldots, W^{79}$ sont nulles, il n'y a pas d'introduction de collision locale dans les 5 dernières étapes.

En définissant une différence Δ avec les 16 premiers mots de W, pour tout bloc M, la différence entre M et $M \oplus \Delta$ est une succession de masques de collision locale. Lorsque l'on considère l'exécution des étapes de la fonction de compression, chaque collision locale se produit avec une certaine probabilité. Si toutes les collisions locales se produisent simultanément, nous obtenons une collision pour la fonction de compression. L'idée de l'attaque est donc de déterminer un vecteur de perturbation V^0 de poids faible, pour limiter le nombre de ces collisions locales. Cette technique a été introduite par Chabaud et Joux dans [CJ98].

Utilisation de la non linéarité du premier tour. Du fait des contraintes pesant sur $W^0_{-5}, \ldots, W^0_{-1}$ et $W^0_{75}, \ldots, W^0_{79}$, l'espace de recherche des vecteurs de perturbation est restreint. Une des caractéristiques de l'attaque de l'équipe de Wang [WYY05a] est de permettre de s'affranchir des premières conditions. Pour cela, au lieu de supposer que le comportement des premières étapes est identique à celui de la fonction linéarisée, l'attaquant utilise la non linéarité des premières étapes pour forcer les différences sur les registres A à E à une valeur voulue. Après ces premières étapes, le chemin différentiel dérivé de la version linéarisée de la fonction de compression s'applique.

Attaque multi-blocs. Il est également possible de s'affranchir des conditions sur les derniers blocs. Pour cela, on utilise un chemin différentiel sur plusieurs blocs. En partant de variables de chaînage identiques, l'attaquant cherche deux blocs de message $M_0, M_0 \oplus \Delta M_0$ de manière à obtenir une différence Δ après la permutation de la fonction de compression. Après le rebouclage, avec une bonne probabilité cette différence reste Δ. Il cherche ensuite deux messages $M_1, M_1 \oplus \Delta M_1$ tels que la différence en sortie de la permutation soit à nouveau Δ et que cette différence s'annule après le rebouclage, ce qui conduit à une collision. Cette technique est représentée en Figure 3.4.

3.3.3 Améliorations possibles

Une fois un chemin différentiel impliquant une différence Δ entre les blocs de message déterminé, la meilleure solution pour l'exploiter ne consiste pas à calculer des empreintes pour des paires de messages $\{M, M \oplus \Delta\}$ prises aléatoirement. Cependant, une fois fixée la différence Δ entre les deux blocs de message pour lesquels l'attaquant cherche à obtenir une collision, l'attaquant dispose encore de la liberté sur le choix de M. Nous décrivons maintenant certaines méthodes lui permettant d'améliorer sa probabilité de succès.

Jeu de conditions suffisantes. Une première étape de l'exploitation d'un chemin différentiel consiste à en dériver un jeu de conditions suffisantes sur M pour que le chemin soit suivi. Ces conditions portent sur les valeurs de bits des registres A à E et sur les mots W_i de l'expansion de message. Pour des valeurs de M aléatoires, on estime que ces conditions sont indépendantes et que chacune d'entre elles est vérifiée avec probabilité $1/2$. Les conditions sur les mots W_i s'expriment comme des relations linéaires sur les bits de M, elles peuvent donc être facilement satisfaites. Nous voyons maintenant comment remplir un maximum de conditions sur la valeur des registres A à E.

Modifications de messages. L'expression d'un chemin différentiel à l'aide d'un jeu de conditions suffisantes sur les bits des registres, utilisée par Wang *et al.*, implique le test de chacune des conditions dès que le registre en question est calculé. Au cours des 16 premières étapes, les valeurs W_i qui interviennent sont indépendantes des valeurs précédentes. Une modification de W_i permet de corriger la valeur d'un bit du registre A_{i+1} qui ne serait pas celle qui est imposée par le chemin différentiel. On parle alors de *modification de messages*. Au cours des étapes suivantes, il est possible d'appliquer des techniques de modification de messages dites multiples pour remplir les conditions du chemin différentiel de manière déterministe. La complexité de l'attaque, en termes de nombre de calculs de la fonction de compression, est en moyenne proche de $T = 2^c$, où c est le nombre de conditions que l'on ne peut pas satisfaire à l'aide de modifications de message.

Bits neutres. Une autre approche consiste, une fois trouvée une paire de messages conforme au chemin différentiel jusqu'à l'étape $i > 16$, à modifier M de manière à obtenir d'autres paires de messages satisfaisant ce même chemin différentiel. La technique de *bits neutres* découverte par Biham et Chen [BC04], permet de trouver de telles paires de messages. Une fois un chemin différentiel, cette technique consiste à chercher des bits de M dont la valeur n'affecte pas la valeurs des bits des registres sur lesquels existent des conditions du chemin différentiel jusqu'à l'étape i. Cette amplification du nombre de paires de messages satisfaisant le chemin jusqu'à l'étape i permet de faire décroître la complexité de l'attaque.

Boomerangs. Une technique similaire, décrite par Joux et Peyrin dans [JP07], consiste à utiliser des *boomerangs*, c'est-à-dire des collisions locales auxiliaires. Soient M et $M \oplus \Delta$ deux blocs de messages tels que le chemin différentiel soit suivi jusqu'à l'étape 15. Supposons maintenant que l'on modifie M de manière à introduire une différence sur un bit de W_{10}, et que l'on corrige l'effet de cette différence sur les valeurs des registres en appliquant des modifications sur $W_{11}, \ldots, W_{15}$ comme décrit dans la Table 3.1. Nous notons Δ_{aux} la différence induite. Pour des choix adéquats de M, des collisions locales sur les valeurs de registres se produisent après l'étape 15 des calculs de hachés de M et $M \oplus \Delta_{aux}$ d'une part, et $M \oplus \Delta$ et $M \oplus \Delta \oplus \Delta_{aux}$ d'autre part. Dans ce cas, $M \oplus \Delta_{aux}$ et $M \oplus \Delta \oplus \Delta_{aux}$ suivent le chemin différentiel jusqu'à l'étape 15. Un choix judicieux

des valeurs de Δ_{aux} permet d'étendre cette propriété de conservation du chemin différentiel jusqu'à des étapes ultérieures du calcul de la fonction de compression. De cette manière, on parvient à augmenter le nombre de paires de messages satisfaisant le chemin différentiel jusqu'à des étapes avancées.

3.3.4 Conséquences sur la famille SHA

L'ensemble des techniques décrites ci-dessus ont contribué à faire chuter la complexité de la recherche de collisions sur SHA-1. Du fait des techniques d'accélération de l'attaque, il est judicieux pour un attaquant de chercher un chemin différentiel induisant peu de conditions après l'étape 16, et surtout à partir de l'étape 20 où les techniques de modifications de message multiples s'avèrent inopérantes. La complexité de la meilleure attaque connue aujourd'hui est sujette à discussion. Dans leur article, Wang *et al.* annoncent une complexité de 2^{63} calculs de la fonction de compression. En automatisant certaines parties de la recherche de chemin différentiel, De Cannière et Rechberger sont parvenus à montrer une collision sur des versions de SHA-1 avec une fonction de compression réduite à 64 puis à 70 étapes. D'autres annonces plus récentes font état de complexités inférieures et atteignables dans la pratique. Cependant, aucune collision sur la fonction complète n'est aujourd'hui connue, ce qui viendrait confirmer ces études.

Du fait de ces attaques, la sécurité de SHA-1 est fortement remise en cause. Des principales fonctions de hachage utilisées, seule la famille SHA-2 résiste à ces attaques. Cependant, la proximité de ses principes de conception avec ceux des autres fonctions de la famille MD-SHA, ainsi que les vulnérabilités de l'algorithme d'extension de domaine de Merkle-Damgård décrites au chapitre précédent, a conduit le NIST à envisager une solution de repli.

3.4 La compétition SHA-3

La compétition SHA-3 est un concours organisé par le NIST et dont l'objectif est la définition d'une nouvelle norme de hachage cryptographique qui pourrait pallier une éventuelle défaillance de SHA-2.

3.4.1 Déroulement de la compétition

La compétition a commencé par la conception des fonctions candidates. Ces fonctions devaient permettre de calculer des hachés de longueur 224, 256, 384 et 512 bits, dans l'idéal avec une sécurité et des performances au moins équivalentes à celles de SHA-2. Les spécifications des candidats ont été transmises au NIST pour le 31 octobre 2008, qui les a publiées peu après.

Après cette phase de conception, une phase d'analyse des différents candidats par la communauté académique a été prévue. Cette phase se déroule en trois tours, un certain nombre de candidats étant sélectionnés par le NIST à l'issue de chacun d'entre eux. Cette sélection a pour but de permettre une concentration de l'effort sur un petit nombre de fonctions, à la fois en termes d'attaques, d'arguments de sécurité et d'implantations, pour le dernier tour de la compétition. Sur 64 soumissions reçues, 51 ont été retenues pour le premier tour de la compétition. Le 24 juillet 2009, le NIST a annoncé le choix de 14 fonctions pour le deuxième tour. Le choix des 5 finalistes a été annoncé le 10 décembre 2010. La nouvelle fonction SHA-3 devrait être choisie au deuxième trimestre 2012.

Afin d'améliorer le processus, le NIST a permis aux fonctions sélectionnées pour le deuxième, puis pour le troisième tour de la compétition, de procéder à des modifications mineures dans leurs

Fonction	Ext. Domaine	Compression
BLAKE	Merkle-Damgård	ARX
Blue Midnight Wish	Wide pipe	ARX
cubehash	Eponge	ARX
ECHO	Wide pipe	AES
Fugue	≈ Eponge	AES
grøstl	≈ Wide pipe	ARX
Hamsi	Merkle-Damgård	op. booléennes
JH	≈ Eponge	op. booléennes
Keccak	Eponge	op. booléennes
Luffa	≈ Eponge	op. booléennes
Shabal	≈ Eponge	ARX-op. booléennes
SHAvite-3	Merkle-Damgård	AES
SIMD	Wide pipe	ARX-op. booléennes
Skein	Merkle-Damgård	ARX

Table 3.2 – Fonctions de hachage sélectionnées pour le deuxième tour de la compétition SHA-3.

spécifications. Cette possibilité a été utilisée par certaines des fonctions.

3.4.2 Le deuxième tour

Le deuxième tour de la compétition SHA-3 a concerné 14 fonctions de hachage. Les fonctions éliminées à l'issue du premier tour l'ont été pour des raisons de performance ou de sécurité. Les 14 fonctions sélectionnées représentaient une bonne diversité en termes de principes de conception. Divers algorithmes d'extension de domaine ont été employés : Merkle-Damgård, la construction wide pipe et les fonctions éponges. Les approches utilisées pour la conception des fonctions de compression étaient également variées. Certaines des fonctions ont fait le choix d'employer des boîtes S, c'est-à-dire des fonctions fixes sur des ensembles de petite taille dont les valeurs peuvent être tabulées. Cette approche a souvent été combinée avec la réutilisation de certains principes de conception de l'AES. D'autres fonctions ont utilisé un mélange d'additions modulaires, de rotations et de ou exclusifs. Cette approche est dénotée ARX (pour Add-Rot-Xor). Enfin, une autre approche consiste à utiliser uniquement des opérations booléennes simples (ou logique, et logique, ou exclusif) sur des registres, combinées avec des rotations. Les approches utilisées par chacun des candidats du deuxième tour sont résumées dans la Table 3.2.

Parmi les fonctions retenues pour le deuxième tour de la compétition figurent Shabal, que nous avons conçue dans le cadre des projets ANR Saphir et Saphir2 et que nous décrivons au chapitre 4, et Hamsi, pour laquelle nous montrons une attaque en recherche de secondes préimages au chapitre 8.

3.4.3 Les cinq finalistes

La dernière phase de la compétition, de décembre 2010 au printemps 2012, permet de regrouper les efforts de recherche sur 5 finalistes. Nous les présentons maintenant, dans l'ordre alphabétique.

BLAKE. BLAKE a été conçue par Aumasson, Henzen, Meier et Phan [AHMP10]. Les fonctions BLAKE utilisent l'algorithme d'extension de domaine de Merkle-Damgård, mais la primitive interne de leurs fonctions de compression s'applique sur des états internes dont la taille est double de celle de la variable de chaînage. Cette augmentation de la taille permet entre autres de compliquer la recherche et l'exploitation de propriétés différentielles, ainsi que d'instancier la construction HAIFA [BD06], qui permet d'éviter les attaques par extension de longueur. La primitive interne est quant à elle adaptée de l'algorithme de chiffrement à flot ChaCha, conçu par Bernstein [Ber08]. Elle emploie les opérations d'addition, de rotation et de OU EXCLUSIF.

Les meilleures attaques contre des versions réduites de la fonction de compression sont des distingueurs tirant parti du phénomène de différentielles impossibles (c'est-à-dire de propriétés différentielles qui ne peuvent pas se produire). Ces attaques s'appliquent à 5 tours sur 14 pour les versions BLAKE-224 et BLAKE-256, et à 6 tours sur 16 pour BLAKE-384 et BLAKE-512 [AGK+10]. Pour la fonction de hachage, les meilleures attaques connues permettent théoriquement de trouver des préimages sur 2,5 tours [JL09].

Grøstl. Les fonctions `grøstl` ont été conçues par Gauravaram, Knudsen, Matusiewicz, Mendel, Rechberger, Schläffer et Thomsen [GKM+11]. L'algorithme d'extension de domaine utilisé est proche de la construction wide pipe avec une fonction de finalisation. La taille de la variable de chaînage est 512 bits pour les versions `grøstl`-224 et `grøstl`-256, et 1024 bits pour les versions `grøstl`-384 et `grøstl`-512. La fonction de compression est construite à partir de deux permutations $\mathcal{P}$ et $\mathcal{Q}$ dont la structure est dérivée de celle de l'AES, sur un état plus gros.

La fonction de tour de `grøstl` a été légèrement modifiée entre les deuxième et troisième tours. Pour la version du deuxième tour, les meilleures attaques par distingueur sur la fonction de compression s'appliquaient aux 10 tours des versions `grøstl`-224 et `grøstl`-256, et à 11 tours sur 14 pour `grøstl`-384 et `grøstl`-512 [Pey10, NP10]. Toutefois, les meilleures attaques sur les fonctions de la famille `grøstl` permettent théoriquement de trouver des collisions pour des versions réduites à 3 tours de la nouvelle version de la fonction [Sch11].

JH. La famille JH a été créée par Wu [Wu11]. Elle utilise un algorithme d'extension de domaine ad hoc, proche de celui des fonctions éponges. Le bloc de message est ajouté à la fois avant et après la fonction de compression, ce qui rapproche cet algorithme de celui utilisé dans Shabal que nous décrivons dans le chapitre 4. La fonction de compression de JH est une permutation qui utilise des boîtes S, cependant sa structure rend naturelle l'implantation des boîtes S comme une suite d'opérations logiques (et non pas à l'aide d'une table).

Les meilleures attaques connues [RTV10, NP10] permettent d'attaquer 22 des 42 tours de la fonction de compression et de trouver des collisions à IV choisi pour la fonction de hachage réduite à 16 tours.

Keccak. KECCAK est une famille de fonctions qui a été conçue par Bertoni, Daemen, Peeters et Van Assche [BDPA08a]. Il s'agit d'une famille de fonctions éponges. Les fonctions de compression de KECCAK sont des permutations qui se composent d'opérations booléennes élémentaires appliquées en parallèle sur tout l'état interne de 1600 bits.

Les meilleures attaques par distingueur sur la permutation interne s'appliquent à la version complète (24 tours), cependant elles nécessitent 2^{1579} applications de cette permutation [BCdC11, DL11]. Les meilleures attaques sur la fonction de hachage permettent une recherche de secondes

préimages sur 8 tours de KECCAK-512, avec une complexité légèrement inférieure à celle de l'attaque théorique [Ber11].

Skein. La famille Skein a été conçue par Ferguson, Lucks, Schneier, Whiting, Bellare, Kohno, Callas et Walker [FLS+10]. Elle utilise la construction de Merkle-Damgård. La fonction de compression dispose d'une entrée supplémentaire qui permet l'utilisation d'un mode proche de celui de HAIFA afin d'éviter les attaques par extension de longueur. Elle est construite à partir d'un algorithme de chiffrement par blocs dédié, Threefish, qui fait partie de la classe des fonctions ARX.

Les attaques par distingueur les plus efficaces contre la fonction de compression s'appliquent à 53 tours sur 72 pour Skein-224 et Skein-256, et à 57 tours sur 72 pour Skein-384 et Skein-512 [KNR10]. Cependant, ces attaques exploitaient des symétries de la fonction de compression, qui ont été corrigées par une modification mineure à l'issue du deuxième tour.

Deuxième partie

Conception et preuves de sécurité

Chapitre 4

Description et analyse de Shabal

Sommaire

Ce chapitre est consacré à la famille de fonctions de hachage Shabal, que nous avons conçue en collaboration avec Emmanuel Bresson, Anne Canteaut, Benoit Chevallier-Mames, Christophe Clavier, Aline Gouget, Thomas Icart, Jean-François Misarsky, María Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet et Marion Videau [BCCM+08]. Les travaux de conception de Shabal ont été initiés par les partenaires du projet RNRT Saphir, et la famille Shabal a été soumise à la compétition SHA-3. Le nom de la famille, Shabal, a été choisi en référence aux fonctions de la famille SHA et au rugbyman Sébastien Chabal, connu pour résister aux collisions et hacher ses adversaires. En juillet 2009, Shabal a été sélectionnée pour le deuxième tour de la compétition, mais n'a pas été choisie pour faire partie des 5 finalistes en décembre 2010.

Shabal repose sur des principes de conception novateurs, tant dans l'algorithme d'extension de domaine que de la primitive interne, qui est une permutation paramétrée. Cette particularité nous démarque d'autres candidats du deuxième tour de la compétition. En effet, certaines de ces fonctions réutilisent des parties d'algorithmes de hachage ou de chiffrement symétrique déjà existants tels que AES (ECHO, `grøstl`, SHAvite-3, Fugue), Serpent (`Hamsi`), ChaCha (BLAKE), ou les fonctions de la famille MD-SHA (SIMD).

Dans un premier temps, nous décrivons l'algorithme d'extension de domaine et la permutation paramétrée de Shabal. Nous décrivons également les idées générales ayant présidé à sa conception. Des analyses de sécurité plus précises justifiant les choix effectués peuvent être trouvées dans le document de soumission [BCCM+08]. Nous étudions ensuite les propriétés statistiques connues des mécanismes de Shabal. L'étude de la sécurité de l'algorithme d'extension de domaine est l'objet des chapitres 5 et 6.

4.1 Description de la fonction

4.1.1 Conventions et notations

Shabal est une famille de fonctions de hachage dont les variantes diffèrent uniquement par la longueur des empreintes produites et par la valeur de certaines constantes. En particulier, la structure interne est identique quelle que soit la taille du haché. Dans la suite du document, nous désignerons donc par Shabal la famille entière et par Shabal-x la variante produisant des sorties de longueur x. Pour désigner une variante quelconque de Shabal, nous employons la notation Shabal-ℓ_h.

Shabal est conçue pour traiter des séquences de bits de longueur finie arbitraire. Fonctionnellement, aucune limite n'est imposée sur la longueur des messages traités, cependant les analyses de sécurité effectuées supposent le traitement de messages de moins de 2^{73} bits.

Formellement, Shabal-ℓ_h est défini pour des tailles de sortie multiples de 32 bits, jusqu'à 512 bits. Les valeurs usuelles de la taille de sortie sont 224 bits, 256 bits, 384 bits et 512 bits, qui correspondent aux exigences du NIST pour les candidats SHA-3. Des sorties de 160 bits ou 192 bits peuvent également être considérées pour certaines applications (génération d'aléa, ECDSA-192...).

La structure de Shabal repose sur l'itération d'une fonction de compression, dont les entrées et sorties sont des vecteurs de mots de 32 bits. Dans la suite de ce chapitre, nous utilisons les notations suivantes :

- $X[i]$ représente le bit i du mot X.
- Y_j représente le mot j de la variable Y.
- Z^k représente la variable Z à l'issue de l'itération k de la fonction de compression.

4.1.2 Algorithme d'extension de domaine

Shabal repose sur l'utilisation d'un nouvel algorithme d'extension de domaine, dont la primitive interne est une permutation paramétrée $\mathcal{P}$. Nous désignons cet algorithme par *mode de Shabal* dans la suite de ce manuscrit. Il dispose d'une preuve d'indifférenciabilité de l'oracle aléatoire, décrite dans le chapitre 5. La définition de Shabal requiert uniquement la spécification de cet algorithme, d'une méthode de padding, d'une valeur d'initialisation et de la permutation $\mathcal{P}$, qui est donnée plus bas.

4.1.2.1 Description

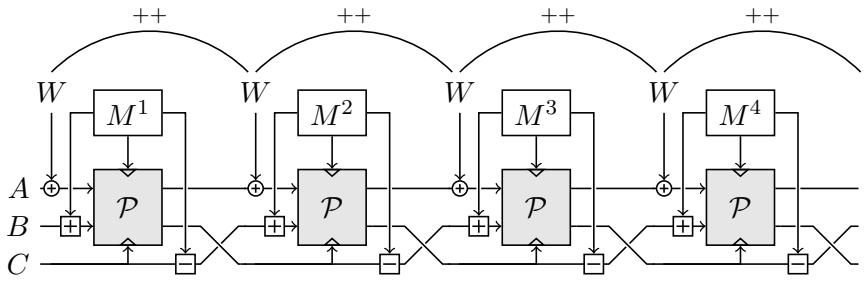


FIGURE 4.1 – Extension de domaine de Shabal : traitement du message

Notre construction agit sur un état interne divisé en trois parties $(A, B, C) \in \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_m} \times \{0,1\}^{\ell_m}$, dont les valeurs initiales sont fixées à une constante (A_0, B_0, C_0). Un registre auxiliaire

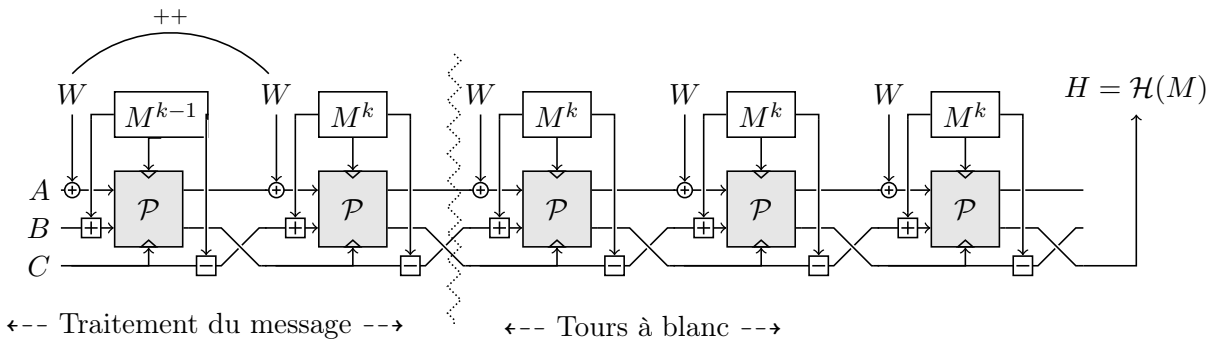


FIGURE 4.2 – Tours à blanc : première représentation

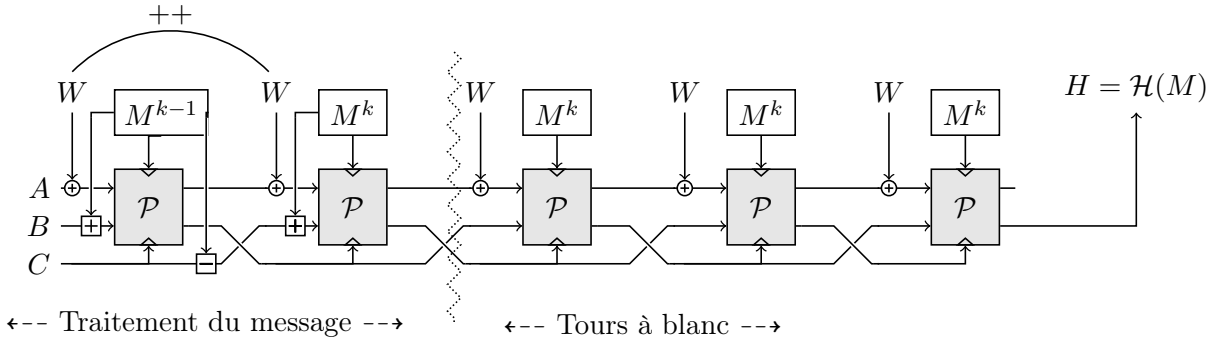


FIGURE 4.3 – Tours à blanc : deuxième représentation

$W \in \{0,1\}^{64}$ est utilisé comme compteur du nombre de blocs de messages traités. Pour pouvoir traiter des messages de longueur arbitraire, le compteur du nombre de blocs déjà traités est pris modulo 2^{64}. Du fait de son rôle particulier, ce registre n'est pas représenté comme une partie de l'état interne. Notre algorithme traite itérativement des blocs de message de taille ℓ_m. Il fait appel à une permutation $\mathcal{P}$ paramétrée telle que

$$\mathcal{P} : \{0,1\}^{\ell_m} \times \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_m} \times \{0,1\}^{\ell_m} \rightarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_m} \quad (4.1)$$

$$(M, A, B, C) \mapsto \mathcal{P}(M, A, B, C) = \mathcal{P}_{C,M}(A, B). \quad (4.2)$$

Lorsqu'on fixe les paramètres M et C de taille ℓ_m, la fonction partielle $(A, B) \mapsto \mathcal{P}_{M,C}(A, B)$ est une permutation.

Dans le cas de Shabal, $\ell_a = 384$, et $\ell_m = 512$. Shabal permet donc de traiter des blocs de 512 bits, et modifie un état interne de 1408 bits. $\mathcal{P}$ est donc une permutation d'une variable de 896 bits, paramétrée par une clé de 1024 bits.

Fonction de compression. Par analogie avec la structure des fonctions de la famille MD-SHA, nous appelons *fonction de compression* de Shabal l'action sur l'état interne du traitement d'un bloc de message, c'est-à-dire

$$(A, B, C) \leftarrow \mathcal{F}(M, A, B, C) \; .$$

Nous excluons l'ajout du compteur et son incrémentation éventuelle du périmètre de la fonction de compression (notée $\mathcal{F}$), pour plusieurs raisons. D'une part, le dernier bloc de message est traité de manière spécifique : la transformation usuelle est appliquée quatre fois au lieu d'une, sans incrémentation du compteur. En excluant le compteur w et ses effets de la définition de $\mathcal{F}$, nous pouvons représenter ce traitement par quatre applications successives de $\mathcal{F}$. D'autre part, les propriétés de sécurité de $\mathcal{F}$ que nous allons décrire n'impliquent pas de valeur particulière du compteur. Ne pas l'intégrer à la fonction de compression permet d'alléger les notations et de rendre les propriétés décrites plus lisibles.

Le calcul de $\mathcal{F}(M, A, B, C)$ est défini par l'Algorithme 4.1.

Algorithme 4.1 Fonction de compression $\mathcal{F}$ de Shabal

Entrées :
$(A, B, C) \in (\{0,1\}^{32})^{12+16+16}$ et $M \in (\{0,1\}^{32})^{16}$
Sorties :
$(A', B', C') \in (\{0,1\}^{32})^{12+16+16}$ tel que $\mathcal{F}(M, A, B, C) = (A', B', C')$
pour $i = 0 \dots 15$ **faire**
$B_i \leftarrow B_i \boxplus M_i$ {Addition sur B}
fin pour
$(A', B') \leftarrow \mathcal{P}_{C,M}(A, B)$
pour $i = 0 \dots 15$ **faire**
$C'_i \leftarrow C_i \boxminus M_i$ {Soustraction sur C}
fin pour
$(B', C') \leftarrow (C', B')$ {Inversion des parties B et C}
renvoyer (A', B', C')

Padding. L'algorithme d'extension de domaine de Shabal permet de traiter des données dont la taille en bits est un multiple de 512. Pour pouvoir hacher des messages de taille quelconque, il est donc nécessaire d'effectuer une première transformation pour générer une donnée dont la taille est multiple de 512 bits. Pour des raisons de sécurité, cette transformation doit être injective : dans le cas contraire, elle donne lieu à des collisions pour la fonction de hachage ainsi construite. Le padding mis en œuvre est le suivant :

$$pad(M) = M||1||0^{511-|M| \bmod 512}. \tag{4.3}$$

En d'autres termes, on concatène au message à hacher un bit valant 1, et le plus petit nombre de bits valant 0 permettant d'obtenir une taille totale multiple de 512 bits.

Encodage. Si Shabal-ℓ_h traite des séquences de bits, les opérations mises en œuvre au cours du calcul de haché impliquent des variables de 32 bits. De même, la sortie de la fonction Shabal-ℓ_h, doit être extraite à partir de variables de 32 bits. La définition de la manière de convertir les données d'un format à l'autre est primordiale à des fins d'implantation, et est calquée sur la fonction MD5 et décrite dans [BCCM+08].

Notons $b_0, \dots, b_{\ell-1}$ la séquence à traiter. Ce mécanisme se décompose en deux étapes :

– Chaque séquence de 8 bits consécutifs $(b_{8i}, \dots, b_{8i+7})$ est transformée en un octet à l'aide de la formule

$$o_i = b_{8i}||b_{8i+1}|| \dots ||b_{8i+7}$$

– Chaque séquence de quatre octets $(o_{4i}, \ldots, o_{4i+3})$ est transformée en un mot

$$M_i = o_{4i+3}||o_{4i+2}||o_{4i+1}||o_{4i}$$

Certaines opérations nécessitent d'interpréter les variables de t bits (octets, mots de 32 bits) comme des entiers modulo 2^t. Nous utilisons alors la convention suivante :

$$b_0||\ldots||b_{t-1} \equiv \sum_{i=0}^{t-1} 2^{t-1-i} b_i \ .$$

Initialisation de l'état interne. La description de Shabal-ℓ_h nécessite également la définition de la valeur initiale de son état interne. Contrairement à d'autres fonctions utilisant un état interne de grande taille comme RadioGatún ou Keccak, les registres de Shabal ne sont pas initialisés à 0. Si une telle valeur était choisie, certaines symétries de l'état interne pourraient être préservées au cours du calcul de hachés, conduisant à une vulnérabilité de la fonction.

Nous avons donc fait le choix d'utiliser des valeurs initiales sans propriétés statistiques particulières. Pour chaque taille de sortie, on utilise des valeurs initiales différentes, afin d'éviter que les hachés obtenus pour des messages identiques mais pour différentes variantes soient des préfixes les uns des autres. Pour y parvenir, l'état interne est d'abord initialisé à 0. On applique alors deux fois la fonction de compression, avec des compteurs valant -1 et 0, à deux blocs de message construits de la manière suivante :

$$\begin{aligned} M^{-1} &= \ell_h||(\ell_h+1)||\ldots||(\ell_h+15) \\ M^{0} &= (\ell_h+16)||(\ell_h+17)||\ldots||(\ell_h+31), \end{aligned}$$

où les entiers $\ell_h, \ldots, \ell_h+31$ sont représentés sur 32 bits. De cette manière, deux options d'implémentation sont possibles. Si suffisamment d'espace mémoire est disponible, on peut stocker l'intégralité de la valeur de l'état interne après hachage de ces deux blocs, c'est-à-dire (A^0, B^0, C^0). Dans le cas contraire, on peut calculer cette valeur à l'aide de peu de données et de la fonction de compression de Shabal.

Description de l'algorithme d'extension de domaine

Initialisation : $(A, B, C, W) \leftarrow (A^0, B^0, C^0, 1)$.

Padding : Concaténer au message un bit valant 1, suivi du plus petit nombre de bits valant 0 tel que la longueur totale soit un multiple de ℓ_m.

Traitement du message : Pour w de 1 à k, faire :

- **addition :** introduction du message.

$$B \leftarrow B \boxplus M^w,$$

où $B \leftarrow B \boxplus M^w$ représente l'addition modulo 2^{32} mot par mot de B et M_w (il n'y a pas de retenue entre les différents mots de 32 bits).

- **compteur :** ajout du compteur sur A_0 et A_1, par un OU EXCLUSIF. En notant $w \bmod 2^{64} = 2^{32}W_1 + W_0$

$$A_0 \leftarrow A_0 \oplus W_0, \quad A_1 \leftarrow A_1 \oplus W_1.$$

- **permutation :** application de la permutation paramétrée $\mathcal{P}$.

$$(A, B) \leftarrow \mathcal{P}_{M^w, C}(A, B).$$

- **soustraction :** le message est soustrait.

$$C \leftarrow C \boxminus M^w,$$

où $C \leftarrow C \boxminus M^w$ représente la soustraction modulo 2^{32} mot par mot de C et M^w.

- **échange :** les valeurs de B et C sont échangées.

$$(B, C) \leftarrow (C, B).$$

Tours à blanc : Après les tours de traitement du message, une série de tours à blanc est appliquée : la fonction de traitement du message est appliquée 3 fois au dernier bloc M^k, le compteur conservant la valeur k.

Sortie : Retourner les mots $C[16 - \ell_h/32]$ à $C[15]$. Les valeurs de A et B sont ignorées.

Une représentation schématique de notre construction est donnée en Figure 4.1. Le choix de réutiliser le dernier bloc de message plusieurs fois est motivé par certaines optimisations (voir les différences entre les Figures 4.2 and 4.3). En particulier, la dernière soustraction de message et le dernier échange sont inutiles, et les effets des additions et soustractions de message entre les tours à blanc s'annulent, de sorte qu'on peut ignorer ces opérations. La première figure montre une représentation de Shabal sous la forme d'une itération de fonctions identiques, alors que la deuxième figure montre une représentation plus efficace, mais qui nécessite plus de code pour les tours à blanc.

4.1.2.2 Représentation algorithmique

Nous donnons ci-dessous une vue algorithmique synthétique de Shabal.

Initialisation : $(A, B, C) \leftarrow (A^0, B^0, C^0)$
Traitement du message : $M = M^1, \ldots, M^k$

Pour w de 1 à k **faire**

1. $B \leftarrow B \boxplus M^w$
2. $A \leftarrow A \oplus w$
3. $(A, B) \leftarrow \mathcal{P}_{M^w, C}(A, B)$
4. $C \leftarrow C \boxminus M^w$
5. $(B, C) \leftarrow (C, B)$

fin faire

Tours à blanc :

Pour i de 0 à 2 **faire**

1. $B \leftarrow B + M^k$
2. $A \leftarrow A \oplus k$
3. $(A, B) \leftarrow \mathcal{P}_{M^k, C}(A, B)$
4. $C \leftarrow C \boxminus M^k$
5. $(B, C) \leftarrow (C, B)$

Fin faire

Sortie : $H = \mathrm{msb}_{\ell_h}(C)$

4.1.2.3 Résultats de sécurité

Les analyses de sécurité de l'algorithme d'extension de domaine de Shabal reposent sur des preuves d'indifférenciabilité de l'oracle aléatoire. Dans le chapitre 5, nous décrivons des résultats d'indifférenciabilité d'une version étendue de notre construction. Ces résultats supposent que la permutation paramétrée $\mathcal{P}$ est remplacée par un algorithme de chiffrement par blocs idéal. Dans le cas de Shabal, on peut montrer qu'aucun attaquant effectuant moins de 2^{448} appels à $\mathcal{P}$ ne peut différencier notre construction d'un oracle aléatoire avec une bonne probabilité.

Dans le cas de Shabal-512, cette borne ne suffit cependant pas à garantir le niveau de sécurité généralement requis pour des attaques en recherche de préimage ou de secondes préimages. Nous avons donc également adapté les preuves en fixant ces objectifs à l'attaquant. Dans le cas où $\mathcal{P}$ est un algorithme de chiffrement par blocs idéal, nous montrons que l'attaque la plus efficace en recherche de (secondes) préimages est la recherche probabiliste et requiert de l'ordre de 2^{ℓ_h} calculs de hachés.

De telles preuves reposent sur l'hypothèse que $\mathcal{P}$ est tirée uniformément sur l'ensemble des fonctions ayant la même taille de paramètres. Dans le cas d'une fonction de hachage, une telle hypothèse est impossible à satisfaire, $\mathcal{P}$ étant fixée et publique. Une fonction fixée et publique ne

peut en aucun cas être indifférenciable d'un oracle aléatoire. Ces preuves sont néanmoins une justification de la robustesse d'un algorithme d'extension de domaine, puiqu'elles garantissent l'absence d'attaques génériques.

De nombreuses attaques dites par *distingueur* sur les fonctions de compression de certains candidats SHA-3 ont été publiées au cours de la compétition, notamment [Pey10, NP10, KNR10, YW10, BC10]. Dans certains cas, ces attaques ne permettent pas de mettre en défaut la sécurité de la fonction de hachage complète. Elles jettent cependant un doute, lorsqu'il existe une preuve d'indifférenciabilité, sur la portée de cette preuve. C'est notamment le cas de Shabal, comme nous allons l'évoquer en section 4.4. Dans le chapitre 6, nous décrivons une nouvelle modélisation des primitives internes des fonctions de hachage, qui permettent d'obtenir des résultats d'indifférenciabilité de l'oracle aléatoire en tenant compte de propriétés statistiques spécifiques de la fonction de compression. Les résultats que nous décrivons ne permettent toutefois pas d'obtenir de borne de sécurité satisfaisante dans le cas particulier de Shabal.

4.1.3 Permutation paramétrée $\mathcal{P}$

Nous décrivons maintenant la permutation paramétrée interne des fonctions de la famille Shabal. Cette permutation est identique pour toutes les variantes de Shabal, et repose sur l'utilisation d'un registre à décalage à rétroaction non linéaire (représenté sur la Figure 4.4).

Permutation paramétrée $\mathcal{P}$ utilisée par Shabal

Entrée : M, A, B, C
Sortie : A, B

Pour i de 0 à 15, faire :
– $B_i \leftarrow B_i \lll 17$
Incrémenter i

Pour j de 0 à 2, faire :
– Pour i de 0 à 15, faire :
  – Calculer

$$\begin{aligned} A_{i+16j \bmod 12} \leftarrow\ & \mathcal{U}\big(A_{i+16j \bmod 12} \oplus \mathcal{V}(A_{i-1+16j \bmod 12} \lll 15) \\ & \qquad \oplus C_{8-i \bmod 16}\big) \\ & \oplus B_{i+13 \bmod 16} \\ & \oplus (B_{i+9 \bmod 16} \wedge \overline{B_{i+6 \bmod 16}}) \\ & \oplus M_i \end{aligned}$$

  – $B_i \leftarrow (B_i \lll 1) \oplus \overline{A_{i+16j \bmod 12}}$
– Incrémenter i
Incrémenter j

Pour j de 0 à 35, faire :
– $A_{j \bmod 12} \leftarrow A_{j \bmod 12} \boxplus C_{j+3 \bmod 16}$
Incrémenter j

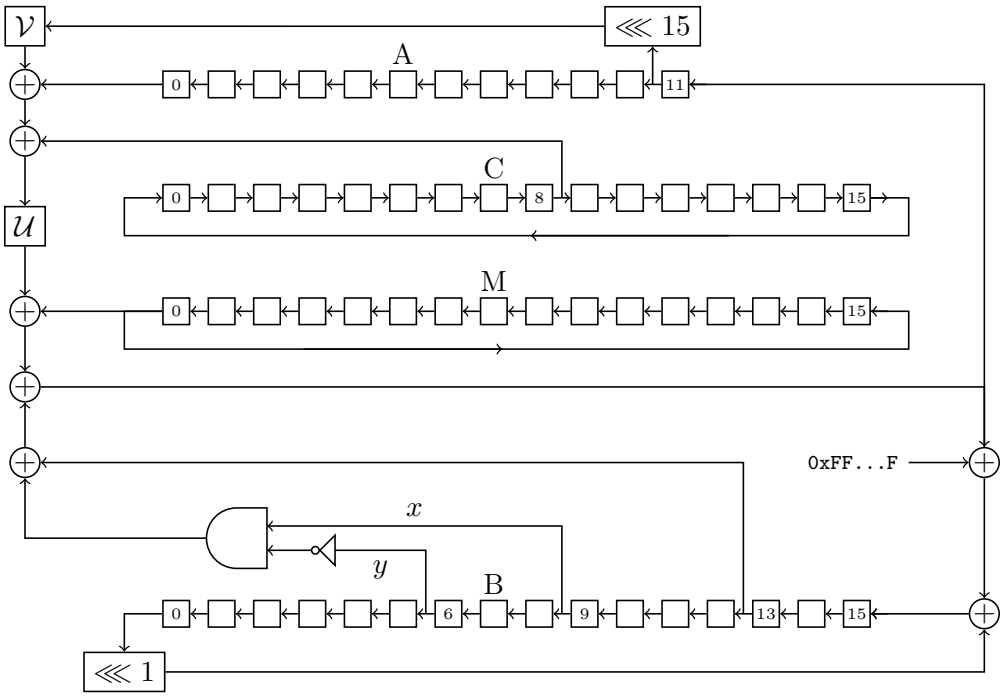


FIGURE 4.4 – Structure de la permutation paramétrée de Shabal.

Dans la description ci-dessus, $\mathcal{U} : x \mapsto 3 \times x \bmod 2^{32}$ et $\mathcal{V} : x \mapsto 5 \times x \bmod 2^{32}$ sont utilisées pour leurs propriétés non linéaires et le haut degré algébrique d'une partie de leurs sorties en fonction de leurs entrées.

La permutation $\mathcal{P}$ peut être décomposée en trois boucles. La non-linéarité de Shabal est obtenue par l'utilisation d'opérations de ET logiques et d'additions modulo 2^{32}. Ces dernières opérations traitent les bits d'entrée de manière dissymétrique. En particulier, le bit de poids faible du résultat est une fonction linéaire de deux bits d'entrée, et les bits de poids fort des entrées n'interviennent que dans des monômes de degré 1 sur la sortie. Pour faire varier les effets des bits d'entrée au cours du calcul, nous commençons par appliquer une rotation de 17 positions vers la gauche à chacune des variables B_i.

La deuxième boucle constitue le cœur de la permutation $\mathcal{P}$. C'est cette étape qui est représentée en Figure 4.4. Elle consiste à itérer deux registres à décalages interagissant l'un avec l'autre. A chaque étape, une variable du registre A et une variable du registre B sont modifiées. La nouvelle valeur de $A_{i+16j \bmod 12}$ dépend de la valeur précédente de cette variable, de la variable de A mise à jour à l'étape précédente, de deux variables de clé M_i et C_{8-i} et de 3 variables de B. B_i est mis à jour par une rotation d'une position vers la gauche, et l'addition bit-à-bit de la variable de A qui vient d'être modifiée.

La troisième boucle consiste à additionner modulo 2^{32} des variables de C à des variables de A, afin de pallier certaines faiblesses de $\mathcal{P}^{-1}$.

Les principes de conception de $\mathcal{P}$ sont résumés en section 4.3.2

4.2 Performances et implémentations

Les critères de sélection retenus par le NIST pour les phases successives de la compétition SHA-3 s'articulent principalement autour de deux axes. Le premier est évidemment celui de la sécurité : la nouvelle norme SHA-3 doit résister aux tentatives de cryptanalyse au cours de la compétition. Mais elle doit également offrir un niveau de résistance laissant à penser qu'elle résistera aux attaques sur le long terme.

L'autre axe de sélection retenu par le NIST est celui des performances. L'objectif de la compétition est de dégager une nouvelle norme, prévue pour remplacer SHA-1, qui reste la fonction de hachage la plus utilisée, dans des applications concrètes. L'objectif pour la norme SHA-3 est d'offrir des performances au moins équivalentes à celles de SHA-2.

La comparaison des performances des différents candidats n'est pas chose aisée. En effet, des objectifs différents peuvent être fixés en fonction du contexte d'emploi. Par exemple, sur un PC, les mécanismes cryptographiques sont implémentés en logiciel, et le principal critère de comparaison est la rapidité du traitement des données. Dans des environnements contraints tels que des composants matériels ou des systèmes embarqués, la fonction de hachage doit partager les ressources disponibles avec d'autres mécanismes. La quantité de ressources consommées par l'implémentation est dans ce cas un critère important. On peut établir un classement des candidats SHA-3 en fonction de leur rapidité, de la taille de leur implémentation ou d'un compromis entre les deux. Le but de cette section n'est pas d'établir un classement détaillé des performances des différentes fonctions, mais de déterminer de manière qualitative les atouts et limitations de Shabal en matière de performances.

Plateforme de référence. La rapidité des implémentations logicielles des fonctions de hachage dépendent en partie de la plateforme sur laquelle elles sont implantées. En effet, des instructions particulières ou le type d'architecture peuvent varier d'une plateforme à l'autre. D'une part, certains candidats SHA-3 tirent parti des jeux d'instructions particuliers. Citons les instructions vectorielles SSE2 (SIMD) présentes sur certains processeurs, ou encore les instructions AES (`Grøstl`, ECHO, SHAvite-3, Fugue) disponibles sur les derniers processeurs Intel. D'autre part, certaines fonctions sont plus performantes sur des architectures 32 bits, d'autres sur des architectures 64 bits.

Afin de pouvoir comparer les performances des candidats, le NIST a donc défini une plateforme de référence : un PC équipé d'un processeur Intel Core2 Duo à 2,4 GHz avec 2Go de RAM, utilisant le système d'exploitation Windows Vista Ultimate (versions 32 et 64 bits). Toutefois, sur une même plateforme, le classement des candidats peut varier en fonction de la longueur des messages à hacher. Par exemple, Shabal impose l'application de 4 itérations de la fonction de compression au dernier bloc, ce qui entraîne un surcoût non négligeable pour le hachage de messages courts.

Implémentations logicielles. Le projet eBASH [ECR], coordonné par Bernstein, a permis la comparaison d'implémentations optimisées des différents candidats sur un large panel de plateformes. Parmi les candidats du deuxième tour, Shabal affiche de très bonnes performances sur les messages longs (moins de 10 cycles par octet). Sur les architectures 64 bits, Shabal est généralement devancé par Blue Midnight Wish et Skein, et est au même niveau que BLAKE. Sur des architectures 32 bits, seul Blue Midnight Wish est significativement plus rapide. Pour des messages courts ou très courts, Shabal se situe un peu en-dessous de la moyenne des candidats du second tour de la compétition.

Systèmes embarqués. Shabal a été conçu pour pouvoir être implanté dans des environnements contraints. De manière générale, les efforts d'implémentation sur ces plateformes ont été moins importants que sur des architectures plus répandues. Les résultats disponibles ne font donc pas état d'implantations très optimisées. Le projet XBX, présenté à CHES 2010 [WBG10], détaille la rapidité et la taille du code en ROM et en RAM des différents candidats du second tour, sur différentes plateformes. Shabal y apparaît comme le candidat le plus rapide, et figure parmi ceux dont le code est le plus compact.

Composants FPGA. Le terme FPGA regroupe plusieurs familles de composants matériels programmables. Gaj, Homsirikamol et Rogawski ont implanté tous les algorithmes du second tour de la compétition SHA-3, et ont comparé leurs performances sur ce type de composants. Leur méthode et certains résultats ont été présentés à CHES 2010 [GHR10]. La structure de registre à décalage utilisée par Shabal limite le débit qu'il est possible d'obtenir avec cette fonction. En revanche, Shabal peut être implémenté de manière très compacte. Les résultats obtenus dans [GHR10] placent Shabal parmi les moins bonnes fonctions en termes de débit, et dans la moyenne pour la surface occupée.

D'autres implantations optimisées de Shabal ont été publiées. À SAC 2010, Detrey, Gaudry et Khalfallah ont montré qu'il est possible d'utiliser certaines caractéristiques des FPGA de la famille Virtex pour obtenir des implémentations très compactes de Shabal [DGK10]. Sur cette famille particulière, leur travail place Shabal à la troisième place pour la surface occupée, et à la première pour le ratio débit/surface. Une autre implantation réalisée par Francq et Thuillet [FT10] obtient des performances légèrement moins bonnes, mais concerne une plus grande variété d'architectures. De manière générale, on peut estimer que les performances matérielles sur FPGA de Shabal sont bonnes, en particulier en termes de consommation de ressources.

Composants ASIC. Guo, Huang, Nazhandali et Schaumont ont effectué des comparaisons d'implémentations sur ASIC des candidats du second tour de la compétition SHA-3 [GHNS10]. Les résultats présentés tendent à montrer que `Luffa` et Keccak sont très rapides et compactes, `Grøstl` et Blue Midnight Wish sont rapides, mais très consommatrices de ressources. echo, SIMD et Fugue semblent nécessiter beaucoup de ressources, sans offrir un bon débit. Les sept autres candidats du second tour offrent des performances comparables à celles de SHA-2, à la fois en termes de débit et de ressources consommées. Shabal figure toutefois en queue de ce peloton.

4.3 Principes de conception de Shabal

Nous décrivons maintenant les principes ayant présidé à la conception de Shabal, ainsi que les critères ayant permis le choix de certains paramètres. Certains de ces principes concernent l'algorithme d'extension de domaine, d'autres concernent la permutation $\mathcal{P}$.

4.3.1 Principes de conception de l'algorithme d'extension de domaine

Idée générale. Shabal a été créée en suivant certains principes de conception de la fonction RadioGatún [BDPA06], notamment l'utilisation d'une variable de chaînage de grande taille et d'une fonction de finalisation complexe. Cette caractéristique visait à rendre difficiles les attaques connues, basées sur les techniques de cryptanalyse différentielle, et permet d'obtenir des débits élevés. La meilleure attaque spécifique à la famille RadioGatún, décrite dans le chapitre 7, est plus coûteuse que les attaques génériques pour des choix naturels de paramètres (registres de 32

bits pour 256 bits de sortie, registres de 64 bits pour des hachés de 512 bits). Des idées similaires sont mises en œuvre par les fonctions utilisant la construction wide pipe (Blue Midnight Wish, ECHO, SIMDou `grøstl`notamment).

Plutôt que de fonder la sécurité de notre candidat sur des preuves de préservation des propriétés de la fonction de compression, nous avons décidé de montrer la sécurité de notre algorithme d'extension de domaine dans le modèle de l'indifférenciabilité de l'oracle aléatoire, introduit par Maurer *et al.* [MRH04]. Une démarche similaire a été adoptée par Coron *et al.* [CDMP05] pour évaluer la sécurité de Chop-MD et par Bertoni *et al.* [BDPA08b] pour justifier la construction de fonctions éponges. Toutefois, nous ne considérons pas cette preuve, décrite dans le chapitre 5, comme une preuve de sécurité de la fonction Shabal. Elle vient seulement certifier que notre algorithme d'extension de domaine résiste aux attaques génériques.

Une des originalités de Shabal vient de l'utilisation d'une primitive $\mathcal{P}$ dont la taille de sortie est inférieure à celle de la variable de chaînage, la sécurité offerte par l'algorithme d'extension de domaine étant supérieure à celle de la primitive interne. L'avantage d'une telle construction est qu'il semble plus facile de construire une bonne fonction si la taille de sortie est plus petite, notamment lorsque la sortie de la fonction est composée de sorties successives d'un registre à décalage.

De manière similaire à RadioGatún, nous avons choisi de surdimensionner la primitive interne que nous utilisons, $\mathcal{P}$, par rapport aux tailles exigées par la preuve de sécurité (seul un bloc de 512 bits serait nécessaire). Par conséquent, le fait d'utiliser une primitive ayant des propriétés statistiques particulières ne conduit pas nécessairement à une attaque.

Algorithme initial. Le mode de Shabal a été obtenu par une légère modification d'un algorithme d'extension de domaine initialement prévu. Ce mode, décrit par la figure 4.5 devait utiliser une primitive $\mathcal{P}$ identique à celle finalement définie pour Shabal.

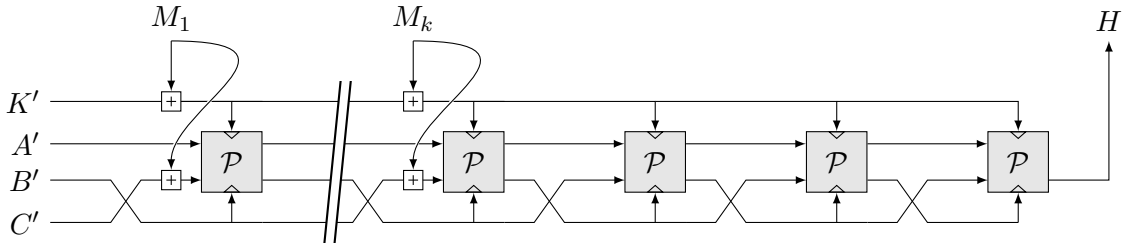


Figure 4.5 – Ancienne version du mode de Shabal

Cette construction permet d'expliquer l'usage de l'expression *tours à blanc* parfois utilisée pour mentionner le traitement du dernier bloc de message par Shabal. En effet, dans notre construction initiale, les 3 itérations supplémentaires de la fonction de compression sont paramétrées par un bloc de message nul, d'où cette expression, qui a ensuite été transposée à notre nouvelle construction.

Notons alors Shabal-$\ell_{h\text{init}}$ la fonction de hachage obtenue en instanciant cette construction avec la permutation $\mathcal{P}$, et avec une valeur initiale (K'^0, A'^0, B'^0, C'^0) telle que

$$\begin{aligned} A'^0 &= A^0 \\ B'^0 \boxminus K'^0 &= B^0 \\ C'^0 &= C^0. \end{aligned}$$

Nous estimons que la résistance des deux fonctions de hachage à la recherche de collisions, de secondes préimages et de préimages est équivalente. En effet, étant donné un message M, il est presque toujours facile de trouver un message M_{init} tel que Shabal-$\ell_{h\text{init}}(M_{init})$ = Shabal-$\ell_h(M)$, et réciproquement. Plus précisément, nous avons les résultats suivants.

Lemme 1 *Soit $\mu \in \{0,1\}^*$ un message, et $M = M^1||\ldots||M^k = pad(\mu)$. Si $M^k \neq M^{k-1}$ (avec la convention $M^0 = K'^0$), il est facile de trouver μ' tel que Shabal-$\ell_{h\,init}(\mu')$ = Shabal-$\ell_h(\mu)$.*

Démonstration : Pour $i \in \{1,\ldots,k\}$, on note $M'^i = M^i \boxminus M^{i-1}$, et $M' = M'^1||\ldots||M'^k$. Comme $M^k \neq M^{k-1}$, $M'^k \neq 0$, et le message $\mu' = pad^{-1}(M')$ est défini.
Nous allons montrer qu'au cours du calcul, on a toujours

$$\begin{aligned} K'^i &= M^i \\ A'^i &= A^i \\ B'^i \boxminus K'^i &= B^i \\ C'^i &= C^i. \end{aligned}$$

Ce résultat s'obtient par récurrence sur i, avec un raisonnement spécifique pour les tours à blanc. Par construction, ces relations sont vraies pour $i = 0$.
Supposons maintenant qu'elles soient vérifiées à l'étape i. On a alors les relations suivantes :

$$\begin{aligned} M^{i+1} &= M^i \boxplus M'^{i+1} \\ &= K'^i \boxplus M'^{i+1} \\ &= K'^{i+1} \\ B^{i+1} &= C^i \boxminus M^{i+1} \\ &= C'^i \boxminus M^{i+1} \\ &= B'^{i+1} \boxminus K'^{i+1} \\ (A^{i+1}, C^{i+1}) &= \mathcal{P}_{C^i, M^{i+1}}(A^i \oplus i, B^i \boxplus M^{i+1}) \\ &= \mathcal{P}_{C'^i, K'^{i+1}}(A'^i \oplus i, B'^i \boxminus M^i \boxplus M^{i+1}) \\ &= \mathcal{P}_{C'^i, K'^{i+1}}(A'^i \oplus i, B'^i \boxminus M'^i) \\ &= (A'^{i+1}, C'^{i+1}) \end{aligned}$$

L'hypothèse de récurrence est vérifiée au rang $i+1$. Pour les tours à blanc, le raisonnement se transpose aisément, les valeurs de M et de K' ne variant pas. Comme Shabal-$\ell_h(\mu) = \text{msb}(C^{k+3})$ et Shabal-$\ell_{h\text{init}}(\mu') = \text{msb}(C'^{k+3})$, on en déduit que Shabal-$\ell_h(\mu)$ = Shabal-$\ell_{h\text{init}}(\mu')$, ce qui achève la démonstration. □

Le résultat inverse s'énonce de la manière suivante.

Lemme 2 *Soit $\mu' \in \{0,1\}^*$ un message, et $M' = M'^1||\ldots||M'^k = pad(\mu')$. Si $\sum_{j=0}^{k} M'^j \neq 0$ (avec la convention $M^0 = K'^0$), il est facile de trouver μ tel que Shabal-$\ell_{h\,init}(\mu')$ = Shabal-$\ell_h(\mu)$.*

Démonstration : Pour $i \in \{0,\ldots,k\}$, on note $M^i = \sum_{j=0}^{i} M'^j$. Par hypothèse, $M^k \neq 0$, donc $\mu = pad^{-1}(M^1||\ldots||M^k)$ est bien défini. On vérifie aisément que pour tout i de $\{1,\ldots,k\}$,

$M^i - M^{i-1} = M'^i$. Par application du Lemme 1, on a Shabal-$\ell_h(\mu) =$ Shabal-$\ell_{h\text{init}}(\mu')$. □

Étudions maintenant les propriétés de sécurité de ces deux fonctions, en commençant par la résistance à la recherche de collisions. N'importe quel couple de message X, Y tel que Shabal-$\ell_h(X) =$ Shabal-$\ell_h(Y)$ peut être transformé aisément en X', Y' tel que Shabal-$\ell_{h\text{init}}(X') =$ Shabal-$\ell_{h\text{init}}(Y')$, sauf si les deux derniers blocs d'un des messages $pad(X), pad(Y)$ sont égaux (ou si $pad(X)$ ou $pad(Y)$ ne contient qu'un bloc égal à K^0).

Si on sait générer une préimage X de h par Shabal-$\ell_h(X) = h$ et si les deux derniers blocs de X sont différents, on sait également générer X' tel que Shabal-$\ell_{h\text{init}}(X') = h$.

La comparaison concernant la résistance à la recherche de secondes préimages est moins immédiate. En effet, l'image de l'élément X de longueur ℓ imposé à l'attaquant est différente dans les scénarios d'attaque contre Shabal-ℓ_h ou Shabal-$\ell_{h\text{init}}$. Dans la définition de l'attaque en seconde préimage donnée dans le chapitre 1, la manière de choisir le message X pour lequel l'attaquant doit trouver une seconde préimage n'est pas définie. Une manière usuelle de procéder consiste à choisir X uniformément parmi les messages de longueur donnée. Or, la transformation de X et X' définie précédemment conserve la longueur en blocs, mais pas la longueur en bits. Cependant, on peut modifier légèrement la définition de cette notion de sécurité, afin de rapprocher les niveaux de sécurité offerts par les deux constructions.

Fixons un paramètre $k > 0$. Un attaquant en recherche de seconde préimage contre une fonction de hachage H cherche un message M' tel que $H(M') = H(M)$, étant donné un message M tiré uniformément sur $\bigcup_{i=512k-511}^{512k} \{0,1\}^i$.

Avec cette nouvelle définition, les messages μ et μ' tels que Shabal-$\ell_h(\mu) =$ Shabal-$\ell_{h\text{init}}(\mu')$ sont tirés avec même probabilité pour une même valeur de paramètre dans les deux jeux. Un attaquant qui trouve une seconde préimage M de Shabal-$\ell_h(\mu)$ peut la transformer aisément en une seconde préimage de Shabal-$\ell_{hinit}(\mu')$, à condition que les deux derniers blocs de $pad(M)$ soient différents.

Ces raisonnements nous ont conduits à considérer que la sécurité des deux constructions était équivalente pour les notions habituelles de résistance à la recherche de collisions, de secondes préimages ou de préimages. Nous avons alors préféré Shabal-ℓ_h à Shabal-$\ell_{h\text{init}}$ pour réduire la taille de l'état interne.

Utilisation d'une permutation. Le choix d'utiliser une permutation paramétrée $\mathcal{P}$ plutôt qu'une fonction non inversible est motivé par le fait que la plupart des constructions classiques de fonctions de compression non inversibles utilisent une permutation et un mécanisme final permettant de mélanger l'entrée et la sortie de la fonction. Cela signifie qu'il est nécessaire de stocker la valeur de l'état initial pendant le calcul de la fonction de compression. L'utilisation d'une permutation paramétrée permet d'économiser de la mémoire.

Traitement du dernier bloc. D'autre part, le mode de Shabal garantit qu'il est impossible qu'une collision sur l'état interne se produise en sortie de $\mathcal{F}$ si on applique $\mathcal{F}$ à des messages différents et à des états internes identiques. En matière de cryptanalyse différentielle, cela permet de développer des arguments de sécurité impliquant le traitement de deux blocs consécutifs. C'est dans cette logique que les tours à blanc ont été imposés : la propagation du dernier bloc de message se fait sur plusieurs itérations consécutives de $\mathcal{P}$. En outre, les tours à blanc permettent d'améliorer les bornes d'indifférenciabilité du mode de Shabal.

Compteur. L'utilisation d'un compteur n'est pas strictement imposée par les preuves d'indifférenciabilité du mode. Il permet cependant de compliquer la tâche d'un attaquant éventuel. En particulier, il garantit l'absence de point fixe sur $\mathcal{F}$ avant les tours à blanc, puisqu'il est incrémenté de façon systématique. Cela empêche l'utilisation éventuelle de points fixes pour trouver facilement des collisions.

D'autre part, l'incrémentation du compteur est arrêtée au cours des tours à blanc. Cette discontinuité dans le traitement du message empêche les attaques par glissement sur les fonctions éponges dues à Gorski, Lucks et Peyrin [GLP08]. En imaginant qu'il n'y ait pas de discontinuité dans $\mathcal{F}$ entre le traitement du message et les tours à blanc, les hachés d'un M' tels que $pad(M') = pad(M)||M^k$ (où M^k est le dernier bloc de $pad(M)$) est obtenu en effectuant l'ensemble des calculs du haché de M, et en appliquant une itération supplémentaire de $\mathcal{F}$. Une telle propriété semble toutefois moins grave dans le cas de Shabal que pour les fonctions éponges, pour lesquelles le haché est obtenu en concaténant des parties de plusieurs sorties consécutives de la fonction de compression.

4.3.2 Principes de conception de $\mathcal{P}$

Les principes de conception de la permutation $\mathcal{P}$ sont détaillés dans le document de soumission de Shabal [BCCM+08]. Nous en rappelons ici quelques éléments, permettant d'expliquer la structure générale de $\mathcal{P}$.

Registres à décalage. La permutation $\mathcal{P}$ repose sur l'interaction de deux registres à décalage, A et B. L'emploi de registres à décalage permet la réutilisation rapide du résultat d'une opération lors des opérations suivantes. La diffusion des variables d'entrée nécessite moins d'opérations que si toutes les mises à jour étaient effectuées en parallèle.

Rôle des registres A et B. Les registres A et B ont deux rôles différents. Lors de la mise à jour de A, la dernière variable mise à jour intervient directement. Les effets des variables de clé M_i et C_{8-i} sont donc rapidement diffusés sur toutes les variables des registres A et B. La variable B_i mise à jour à l'étape i intervient dans la mise à jour de A aux étapes $i+3$, $i+7$ et $i+10$. Informellement, le registre B permet donc d'accumuler l'effet des paramètres C et M, et de les réintroduire dans la mise à jour de A lors d'étapes ultérieures.

Choix des valeurs de décalage 13, 9 et 6. La manière dont nous avons déterminé les positions des variables de B qui interviennent dans la mise à jour de A est expliquée dans [BCCM+08]. Ces positions x, y, z ont été obtenues en linéarisant le calcul de $B_{i+x} \oplus B_{i+y} \wedge \overline{B_{i+z}}$ dans la mise à jour de A, de quatre manières différentes :

1. B_{i+x} ;
2. $B_{i+x} \oplus B_{i+y}$;
3. $B_{i+x} \oplus B_{i+z}$;
4. $B_{i+x} \oplus B_{i+y} \oplus B_{i+z}$.

Le choix de $(x, y, z) = (13, 9, 6)$ permet d'optimiser la diffusion des différences sur le message au cours du calcul de 16 mises à jour des registres A et B.

Source de non linéarité. $\mathcal{P}$ fait appel à quatre opérations non linéaires différentes : la fonction booléenne $x \wedge \overline{y}$, les soustractions modulo 2^{32} de la boucle finale et les multiplications $\mathcal{U}$ et $\mathcal{V}$.

Les multiplications ont été choisies afin de faire grandir rapidement le degré algébrique des variables intermédiaires en fonction des entrées. Les additions et soustractions modulo 2^{32} de $\mathcal{P}$ et de l'algorithme d'extension de domaine ont une propriété similaire.

Utilisation de rotations. Pour une addition ou une soustraction modulo 2^{32}, les bits de poids fort du résultat sont de degré élevé dans les opérandes, alors que les bits de poids faible sont de petit degré. Il en va de même des fonctions $\mathcal{U}$ et $\mathcal{V}$. En outre, le résultat de ces opérations est linéaire en les bits de poids fort des opérandes.

Afin de casser ces structures, des rotations sur les registres de 32 bits ont été ajoutées. Après l'addition du message sur B du mode de Shabal, la première étape de $\mathcal{P}$ est une rotation des mots de B. Au cours de la boucle principale, les registres de A dépendent de la sortie d'une fonction $\mathcal{U}$. A chaque étape, on utilise le mot de A que l'on vient de calculer en entrée de $\mathcal{V}$, après lui avoir appliqué une rotation.

Additions finales. Comme nous le verrons dans la Section 4.4, la diffusion par la permutation inverse $\mathcal{P}^{-1}$ est insuffisante. En particulier, l'inverse de la boucle principale ne permet pas de diffuser rapidement les mots de M et de C sur les valeurs de A et B. La mise à jour finale permet de pallier en partie ce problème. Chaque mot de A et de B dépend alors de tous les mots de C par $\mathcal{P}^{-1}$. Les dépendances dans les mots de M sont également améliorées si on tient compte de l'algorithme d'extension de domaine, qui impose une addition de M sur C juste avant l'application de $\mathcal{P}^{-1}$.

Le nombre d'additions choisies, 36, peut paraître surprenant dans la mesure où chaque mot de C n'est pas utilisé le même nombre de fois (3 fois pour les mots 3 à 6, deux fois seulement pour les autres). Cette mesure permet cependant d'éviter qu'une même somme de mots de C soit soustraite de différents mots de A. En effet, une idée naturelle consisterait à augmenter à 48 le nombre de ces opérations. Cependant, cela reviendrait à effectuer les opérations suivantes :

- Soustraction de $C_3 \boxplus C_7 \boxplus C_{11} \boxplus C_{15}$ sur A_0, A_4 et A_8 ;
- Soustraction de $C_4 \boxplus C_8 \boxplus C_{12} \boxplus C_0$ sur A_1, A_5 et A_9 ;
- Soustraction de $C_5 \boxplus C_9 \boxplus C_{13} \boxplus C_1$ sur A_2, A_6 et A_{10} ;
- Soustraction de $C_6 \boxplus C_{10} \boxplus C_{14} \boxplus C_{12}$ sur A_3, A_7 et A_{11}.

Une telle propriété nous a semblé indésirable. Cette analyse est corroborée par le distingueur dû à Aumasson, Mashatan et Meier [AMM09], qui est plus efficace lorsque le nombre d'additions effectuées est multiple de 24.

Nombre d'étapes de la boucle principale. Malgré l'existence de propriétés statistiques particulières de $\mathcal{P}$ lorsque la boucle principale comporte 2 itérations, aucune attaque n'a pu être identifiée au cours de la phase de conception. Nous avons fixé le nombre d'itérations à 3. A posteriori, un plus grand nombre d'itérations aurait permis d'éviter certains distingueurs liés à la lenteur de la diffusion par $\mathcal{P}^{-1}$. D'autres propriétés statistiques décrites plus bas restent néanmoins vraies indépendamment du nombre d'itérations. Le choix de fixer le nombre d'itérations à 3 nous est apparu comme offrant le meilleur compromis entre le niveau de sécurité apporté et les performances de la fonction.

4.4 Évaluation de la sécurité de $\mathcal{P}$

L'évaluation de la sécurité de $\mathcal{P}$ décrite ici regroupe plusieurs observations, émises par différentes équipes. Certains de ces résultats ont été découverts durant la phase de conception de Shabal, mais nous avons considéré leur impact sur la sécurité de Shabal comme acceptable. Les autres résultats ont été publiés par d'autres équipes, après la soumission du candidat. Tous ces résultats concernent des propriétés particulières de $\mathcal{P}$, qui n'ont toutefois pas été traduites en attaques contre la structure complète de Shabal.

Nous allons maintenant décrire les faiblesses relatives de la permutation paramétrée utilisée dans Shabal qui ont été identifiées à ce jour.

4.4.1 Diffusion imparfaite par $\mathcal{P}^{-1}$

L'un des critères de conception de $\mathcal{P}$ était la diffusion rapide des bits du message, de A et de B dans tout l'état interne. Le choix d'utiliser des structures de type registre à décalage nous a donc conduit à forcer une réutilisation rapide des données qui viennent d'être calculées. En outre, pour des raisons de performances, nous avons limité la complexité de l'étape de mise à jour de l'état interne. En choisissant ces priorités, nous avons privilégié la rapidité de la diffusion par $\mathcal{P}$, au détriment de la diffusion par $\mathcal{P}^{-1}$. Nous allons maintenant montrer les propriétés statistiques qui en découlent.

Permutation inverse. Nous commençons par décrire la permutation $\mathcal{P}^{-1}$, qu'on obtient en inversant les opérations qui constituent $\mathcal{P}$ à l'Algorithme 4.2.

Diffusion des mots de M. Dans la boucle principale, B est mis à jour avant A, et cette mise à jour fait intervenir la valeur de A calculée 12 étapes plus tôt (contre la valeur que l'on vient de calculer dans le sens direct). De plus, la valeur de A calculée à l'étape i n'intervient qu'à l'étape $i+11$ de la mise à jour du registre A (contre $i+1$ dans le sens direct). Enfin, la valeur de B calculée à l'étape i intervient aux étapes $i+6$, $i+9$ et $i+13$ de la mise à jour de A (contre $i+3$, $i+7$ et $i+10$) dans le sens direct.

Du fait de ces propriétés, la diffusion du résultat des différentes opérations est moins bonne par $\mathcal{P}^{-1}$ que par $\mathcal{P}$. Ceci est problématique, notamment parce que le bloc de message M n'intervient que dans la boucle principale. Le document de soumission [BCCM+08] et le mémoire de thèse de Naya-Plasencia [NP09] montrent que des parties de B ne dépendent pas de certains mots de M par $\mathcal{P}^{-1}$. Les résultats présentés dans ces documents mettent l'accent sur les propriétés de $\mathcal{P}$ permettant d'attaquer des versions affaiblies de Shabal. Seules des propriétés exploitables en tenant compte de l'algorithme d'extension de domaine sont décrites. L'impact de M sur la partie A de la sortie n'est en particulier pas exploré. Nous nous attachons ici à établir des propriétés statistiques plus fortes. En remontant les calculs étape par étape, nous pouvons établir quels mots de A et de B en sortie de $\mathcal{P}^{-1}$ sont affectés par les différents mots de M. Ces résultats sont présentés dans le tableau 4.1.

D'après ces résultats, nous pouvons notamment remarquer que le mot M_0 n'affecte pas la valeur de 14 mots de sortie.

Recouvrement de clé à clair connu sur $\mathcal{P}$. De telles propriétés statistiques seraient fatales pour une permutation paramétrée utilisée en tant qu'algorithme de chiffrement par blocs. Dans le cas de $\mathcal{P}$, cela revient à considérer (C, M) comme une clé de 1024 bits, et (A, B) comme un bloc

Algorithme 4.2 Permutation paramétrée inverse $\mathcal{P}^{-1}$ de Shabal

Entrées :
$(A', B', C, M) \in (\{0,1\}^{32})^{12+16+16+16}$
Sorties :
$(A, B) \in (\{0,1\}^{32})^{12+16}$ tel que $\mathcal{P}_{M,C}(A, B) = (A', B')$

pour $i = 35 \ldots 0$ **faire**
$A'_{i \bmod 12} \leftarrow A'_{i \bmod 12} \boxminus C_{i+3 \bmod 16}$ {Inverse de l'addition finale}
fin pour
pour $j = 2 \ldots 0$ **faire**
pour $i = 15 \ldots 0$ **faire**

$$\begin{aligned} B'_i &\leftarrow (B'_i \oplus \overline{A'_{i+16j \bmod 12}}) \ggg 1 \\ A'_{i+16j \bmod 12} &\leftarrow (\mathcal{V}(A'_{i-1+16j \bmod 12}) \lll 15) \\ &\quad \oplus C_{8-i \bmod 16} \\ &\quad \oplus \mathcal{U}^{-1}\left(A'_{i+16j \bmod 12} \oplus B'_{i+13 \bmod 16} \oplus B'_{i+9 \bmod 16} \wedge \overline{B'_{i+6 \bmod 16}} \oplus M_i\right) \end{aligned}$$

{Inverse de la boucle principale}
fin pour
fin pour
pour $i = 15 \ldots 0$ **faire**
$B'_i \leftarrow B'_i \lll 15$ {Inverse de la rotation initiale}
fin pour
renvoyer (A', B')

de 896 bits. La propriété exposée plus haut permet de retrouver la clé dans un scénario d'attaques à clair connu. En d'autres termes, l'attaquant parvient à déterminer une clé de chiffrement à partir de la connaissance de quelques blocs de message clair et des blocs de chiffré correspondant, plus rapidement qu'en cherchant la clé de manière exhaustive. Ce type de scénario est généralement considéré comme peu favorable à l'attaquant en matière d'analyse de chiffrement par blocs.

Supposons qu'un attaquant dispose d'au moins 12 couples clair-chiffré, $(A^{(0)}, B^{(0)}), (A'^{(0)}, B'^{(0)}), \ldots, (A^{(11)}, B^{(11)}), (A'^{(11)}, B'^{(11)})$. Il peut alors retrouver la clé (C^*, M^*) qui a servi au chiffrement de la manière suivante. Pour toutes les clés (C, M) telles que tous les mots de M sauf M_6, M_7, M_{11}, M_{13} sont nuls, l'attaquant teste si le mot à la position B_{15} de $\mathcal{P}^{-1}_{C,M}(A'^{(i)}, B'^{(i)})$ vaut $B^{(i)}_{15}$. Il existe exactement $2^{32(32-12)} = 2^{640}$ telles clés à tester.

Lorsque l'attaquant teste la valeur qui coïncide avec (C^*, M^*) sur toute la partie C et sur M_6, M_7, M_{11}, M_{13}, le test est positif, puisque B_{15} ne dépend pas du reste de la clé par $\mathcal{P}^{-1}$. Lorsque l'attaquant teste une autre valeur de clé, le test est positif avec probabilité $2^{-32\times12} = 2^{-384}$, ce test admet donc en moyenne 2^{256} faux positifs. En outre, pour ces clés, l'attaquant peut arrêter le test dès qu'un échec apparaît au cours de l'un des déchiffrements. Pour une clé donnée, plusieurs déchiffrements sont nécessaires avec probabilité 2^{-32}. On peut donc estimer la complexité de cette phase à 2^{640} opérations de déchiffrement.

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
A_0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_1		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_2	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_3	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_4	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_5	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_6		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A_7	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
A_8	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
A_9	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
A_{10}	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
A_{11}	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓
B_0		✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓
B_1			✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓
B_2	✓			✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓
B_3		✓			✓	✓	✓	✓	✓		✓	✓	✓	✓		✓
B_4	✓		✓			✓	✓	✓	✓	✓		✓	✓	✓	✓	
B_5		✓		✓			✓	✓	✓	✓	✓		✓	✓	✓	✓
B_6			✓		✓			✓	✓	✓	✓	✓		✓	✓	✓
B_7				✓		✓			✓	✓	✓	✓	✓		✓	✓
B_8	✓				✓		✓			✓	✓	✓	✓	✓		✓
B_9	✓	✓				✓		✓			✓	✓	✓	✓	✓	
B_{10}		✓	✓				✓		✓			✓	✓	✓	✓	✓
B_{11}			✓	✓				✓		✓			✓	✓	✓	✓
B_{12}				✓	✓				✓		✓			✓	✓	✓
B_{13}					✓	✓				✓		✓			✓	✓
B_{14}						✓	✓				✓		✓			✓
B_{15}							✓	✓				✓		✓		

TABLE 4.1 – Dépendances en M des mots de la sortie de $\mathcal{P}^{-1}$ pour la permutation paramétrée de Shabal.

Ensuite, pour chacune des clés ayant passé les tests précédents, l'attaquant effectue une recherche exhaustive sur le 12-uplet des $(M_i), i \in \{0, 1, 2, 3, 4, 5, 8, 9, 10, 12, 14, 15\}$, le reste de la clé étant fixé à la valeur trouvée à l'étape précédente. L'attaquant teste alors si la valeur de $\mathcal{P}^{-1}_{C,M}(A'^{(0)}, B'^{(0)}) = (A^{(0)}, B^{(0)})$. Il existe $2^{256} \times 2^{32\times12} = 2^{640}$ telles clés, et chaque valeur erronnée passe le test avec probabilité $2^{-32\times27}$ (la condition étant l'égalité sur tous les mots de A et B, sachant qu'on a égalité sur B_{15}). La probabilité d'existence d'un faux positif pour ce test est donc faible. Cet algorithme permet alors de retrouver la bonne valeur de la clé. La complexité de cette phase est alors 2^{640} opérations de déchiffrement.

Les complexités des deux étapes de l'algorithme décrit sont équivalentes. La complexité totale de la recherche de la clé représente donc environ 2^{641} opérations de déchiffrement. Cette valeur est bien sûr irréaliste lorsqu'on la met en regard des capacités de calcul disponibles, cependant elle est très inférieure à la complexité de la recherche exhaustive 2^{1024} opérations de (dé)chiffrement.

Extension à $\mathcal{F}^{-1}$ et à Shabal. Si les propriétés statistiques décrites plus haut sont fatales pour l'utilisation de $\mathcal{P}$ comme algorithme de chiffrement par blocs, elles ne se traduisent pas par des attaques contre les fonctions de hachage de la famille Shabal. En effet, la manière naturelle d'utiliser une telle propriété consiste à essayer de trouver des (secondes) préimages pour Shabal. Du fait du traitement particulier du dernier bloc de message, il est naturel d'attaquer l'état interne de Shabal avant le dernier bloc, en fixant sa valeur et en calculant les valeurs précédentes de l'état interne.

L'attaquant doit donc exploiter des propriétés de $\mathcal{F}^{-1}$ sur un état interne fixe. $\mathcal{F}^{-1}$ est décrit dans l'algorithme 4.3. L'attaquant ne peut alors exploiter une telle propriété que dans les cas où (A', B', C) est fixé avant d'inverser la fonction de compression. Du fait de la structure de l'algorithme d'extension de domaine, la première étape de l'inversion de la fonction de compression est l'addition

des mots du bloc de message M sur les mots de C'. L'attaquant ne disposant plus de liberté sur C, la première phase de $\mathcal{P}^{-1}$, qui est l'inversion de l'addition finale, introduit des dépendances en M sur les valeurs de A. La propagation du bloc de message est donc plus rapide.

Algorithme 4.3 Inverse $\mathcal{F}^{-1}$ de la fonction de compression de Shabal

Entrées :
$(A', B', C') \in (\{0,1\}^{32})^{12+16+16}$ et $M \in (\{0,1\}^{32})^{16}$
Sorties :
$(A, B, C) \in (\{0,1\}^{32})^{12+16+16}$ tel que $\mathcal{F}_{M,C}(A, B) = (A', B', C')$
$(B', C') \leftarrow (C', B')$ {Inversion des parties B et C}
pour $i = 0 \dots 15$ **faire**
  $C_i \leftarrow C'_i \boxplus M_i$ {Inverse de la soustraction sur C}
fin pour
$(A, B) \leftarrow \mathcal{P}^{-1}_{C,M}(A', B')$ {Inverse de la permutation paramétrée}
pour $i = 0 \dots 15$ **faire**
  $B_i \leftarrow B_i \boxminus M_i$ {Inverse de l'addition sur B}
fin pour
renvoyer (A, B, C)

Dans [NP09], Naya-Plasencia décrit d'autres propriétés statistiques du même type qui tiennent compte de cette contrainte supplémentaire. Si on note $(A, B) = \mathcal{P}^{-1}_{C \boxplus M, M}(A', B')$, ces propriétés sont les suivantes :

- B_{11} peut être exprimé en fonction de A', B', C, M_i pour $i \in \{0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15\}$ et $M_4 \boxplus M_8$
- B_{14} peut être exprimé en fonction de A', B', C, M_i pour $i \in \{1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$ et $M_0 \boxplus M_4$
- B_{15} peut être exprimé en fonction de A', B', C, M_i pour $i \in \{1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15\}$ et $M_0 \boxplus M_4 \boxplus M_{12}$ et $M_4 \boxplus M_8 \boxplus M_{12}$

Chaque mot de B dépend de tous les mots de M, sauf B_{11} et B_{14} qui dépendent de 15 mots sur 16 et B_{15} qui dépend de 14 mots sur 16. Du fait de la marge de sécurité prise en dimensionnant Shabal, ces propriétés ne sont pas exploitables pour obtenir une attaque meilleure que les attaques génériques.

4.4.2 Distingueurs différentiels

Plusieurs travaux de cryptanalyse font état de propriétés différentielles particulières de la permutation paramétrée $\mathcal{P}$. Nous les exposons ci-après.

4.4.2.1 Faiblesse relative aux bits de poids fort

Shabal utilise plusieurs sources de non linéarité : les fonctions $\mathcal{U}$ et $\mathcal{V}$, les additions modulaires, ainsi qu'une opération de OU EXCLUSIF. Les fonctions $\mathcal{U}$ et $\mathcal{V}$ sont basées sur l'addition modulaire, qui a une vulnérabilité connue en cas d'attaque différentielle : une différence sur le bit de poids fort se propage avec probabilité 1. En exploitant cette remarque, Aumasson, Mashatan et Meier [AMM09] ont découvert une propriété de $\mathcal{P}$.

Leur idée consiste à fixer des différences sur les bits de poids fort des mots de C et M, et à éviter que ces différences ne se propagent sur les registres A et B. Considérons les équations de la

boucle principale de Shabal :

$$\begin{aligned} A_{i+16j \bmod 12} &\leftarrow \mathcal{U}\big(A_{i+16j \bmod 12} \oplus \mathcal{V}(A_{i-1+16j \bmod 12} \lll 15) \oplus\ C_{8-i \bmod 16}\big) \\ &\quad \oplus B_{i+13 \bmod 16} \\ &\quad \oplus (B_{i+9 \bmod 16} \wedge \overline{B_{i+6 \bmod 16}}) \\ &\quad \oplus M_i \qquad (4.4) \\ B_i &\leftarrow (B_i \lll 1) \oplus \overline{A_{i+16j \bmod 12}} \qquad (4.5) \end{aligned}$$

Notons $\Delta M = (\delta_{8-i \bmod 16} \times 2^{31})_{i\in\{0\ldots15\}}$, et $M \oplus \Delta M = (M_i \oplus \delta_{8-i \bmod 16} \times 2^{31})_{i\in\{0\ldots15\}}$, où les $\delta_i \in \{0,1\}$. De manière similaire, notons $\Delta C = (\delta_i \times 2^{31})_{i\in\{0\ldots15\}}$, et $C \oplus \Delta C = (C_i \oplus \delta_i \times 2^{31})_{i\in\{0\ldots15\}}$. Nous cherchons à comparer $\mathcal{P}_{C,M}(A,B)$ et $\mathcal{P}_{C\oplus\Delta C, M\oplus\Delta M}(A,B)$.

Les entrées A et B étant identiques dans les deux cas, l'opération de rotation initiale produit des résultats identiques.

Considérons maintenant la boucle principale. A l'étape i, si $\delta_{8-i} = 0$, les deux calculs sont identiques. Dans le cas contaire, il y a une différence sur le bit de poids fort de C_{8-i}. D'après l'Equation (4.4), cette différence se propage sur le bit de poids fort de l'entrée de la fonction $\mathcal{U}$. Or on a $\mathcal{U}(x) = x \boxplus (x \ll 1)$: on a une différence sur le bit de poids fort du premier opérande de cette addition, et pas de différence sur le second opérande. En sortie de $\mathcal{U}$, on a donc une différence sur le bit de poids fort avec probabilité 1. Cette différence est alors annulée par la différence δ_{8-i} portant sur le mot de message M_i. En sortie de l'étape i, les résultats des mises à jour de A et B sont donc identiques.

La dernière étape de $\mathcal{P}$ est la boucle des 36 additions finales. La différence entre les deux calculs porte sur certains des bits de poids fort des mots de C, les mots de A étant identiques. La propagation de différence se fait alors de manière déterministe sur les bits de poids forts des mots de A.

En sortie de $\mathcal{P}$, on a alors une différence $\Delta B' = 0$, et $\Delta A' = ((\delta_{i+3} \oplus \delta_{i+15}\delta_{i+11})2^{31})_{i\in\{0\ldots11\}}$, avec probabilité 1.

Cas particulier. Une remarque de Leurent [Leu09] permet d'identifier un cas particulier intéressant de cette propriété. Si on définit $\delta_0 = \delta_8 = \delta_{12} = 1$ et $\delta_i = 0$ pour les autres valeurs de i, toutes les valeurs de $\Delta A'$ s'annulent, et on a une collision en sortie de $\mathcal{P}$. Dans le cas où $\mathcal{P}$ serait utilisé comme algorithme de chiffrement par blocs, il en résulte que les clés (C,M) et $(C \oplus \Delta C, M \oplus \Delta M)$ sont équivalentes pour toute valeur de M et C et les valeurs de ΔM et ΔC définies au paragraphe précédent. De plus, on a $\Delta M = \Delta C$, donc la soustraction de l'algorithme d'extension de domaine permet d'annuler complètement la différence en sortie de la fonction de compression.

Ce cas particulier peut être généralisé. En effet, l'application linéaire

$$\begin{aligned} \mathcal{L} : \{0,1\}^{16} &\to \{0,1\}^{12} \\ (\delta_i)_{i\in\{0\ldots15\}} &\mapsto (\delta_{i+3} \oplus \delta_{i+15}\delta_{i+11})_{i\in\{0\ldots11\}} \end{aligned}$$

a un noyau de dimension au moins 4. En définissant ΔC à partir d'un vecteur de ce noyau, on obtient une collision en sortie de $\mathcal{P}$ pour l'attaque décrite ci-dessus. Les vecteurs du noyau s'expriment

simplement par les conditions nécessaires et suffisantes suivantes :

$$\begin{aligned}\delta_3 = 0, \delta_7 = \delta_{11} = \delta_{15}\\ \delta_4 = 0, \delta_8 = \delta_{12} = \delta_0\\ \delta_5 = 0, \delta_9 = \delta_{13} = \delta_1\\ \delta_6 = 0, \delta_{10} = \delta_{14} = \delta_2\end{aligned}$$

Pour annuler également la différence sur C à la sortie de la fonction de compression, il faut aussi que $\delta_i = \delta_{8-i}$ pour tout i. Les solutions restantes sont alors dans un espace vectoriel de dimension 2, déterminé par les conditions suivantes :

$$\begin{aligned}\delta_0 = \delta_8 = \delta_{12}\\ \delta_1 = \delta_7 = \delta_9 = \delta_{11} = \delta_{13} = \delta_{15}\\ \delta_2 = \delta_3 = \delta_4 = \delta_5 = \delta_6 = \delta_{10} = \delta_{14} = 0\end{aligned}$$

Pour la fonction de compression $\mathcal{F}$, si on applique les différences δ_i sur les bits de poids forts de B_i, C_i et M_i, on obtient une collision sur l'état interne en sortie de $\mathcal{F}$ avec probabilité 1.

Application à Shabal Du fait de la grande taille de l'état interne, il est cependant plus difficile de trouver deux messages conduisant à des états internes ayant une telle différence que de trouver une collision sur la fonction par application du paradoxe des anniversaires. Cette propriété ne permet donc pas de mettre en danger les fonctions de la famille Shabal.

4.4.2.2 Autres propriétés différentielles

D'autres propriétés différentielles de $\mathcal{P}$ ont également été découvertes, et mettent en lumière le caractère imparfait de la diffusion de cette permutation. Ces propriétés mettent en jeu des différences sur les registres A et B au cours du calcul.

Différentielle sans différence sur (C, M). Tout d'abord, Novotney décrit la propriété suivante dans [Nov10]. On suppose que pour des paramètres M et C identiques, on applique $\mathcal{P}$ à des entrées $A^{(0)}, B^{(0)}$ et $A^{(1)}, B^{(1)}$ identiques, sauf le bit 31 de A_{10} et le bit 13 de B_7. Après les rotations, les différences se retrouvent sur le bit 31 de A_{10} et sur le bit 30 de B_7.

Lorsqu'on suit la propagation de la différence au cours de la boucle principale, la propagation de la différence suit un chemin donné jusqu'à l'étape 26 avec probabilité 2^{-3}. Après l'étape 26, on a une différence de poids faible, la propagation de cette différence au cours des étapes suivantes du calcul est donc imparfaite. La différence sur les premiers mots de B' n'est donc pas uniformément distribuée, et le biais sur ces valeurs permet de distinguer $\mathcal{P}$ d'une permutation aléatoire.

Cette propriété statistique est moins forte que celles décrites plus haut, cependant il s'agit de la seule propriété différentielle connue à paramètre fixé pour $\mathcal{P}$

Diffusion lente par $\mathcal{P}^{-1}$. Dans un article publié sur l'eprint de l'IACR [IS10], Isobe et Shirai exploitent la lenteur de la diffusion par $\mathcal{P}^{-1}$ exposée plus haut pour déterminer des chemins différentiels sur la fonction de compression $\mathcal{F}$.

Comme nous l'avons montré, à l'issue du calcul de $\mathcal{P}^{-1}$, seuls 18 des 32 mots de la sortie dépendent de M_0. Si on ajoute l'étape de soustraction de l'algorithme d'extension de domaine, cette

remarque ne s'applique plus. Néanmoins, cette propriété démontre l'insuffisance de la diffusion de la valeur de M_0 par $\mathcal{P}^{-1}$. Dans leur article, Isobe et Shirai exploitent alors ce raisonnement en étudiant la diffusion de M au niveau du bit.

L'attaquant part alors d'un état (A', B', C') fixe, et lui applique $\mathcal{F}^{-1}$ pour deux blocs de messages dont seuls les bits 31 de M_0 sont différents. Par l'inverse de la soustraction sur C, seul le bit 31 de C_0 porte une différence en entrée de P^{-1}. Dans la boucle d'additions de $\mathcal{P}$, cette différence se propage sur les bits de poids fort de A'_1 et A'_5. Isobe et Shirai ont linéarisé la boucle principale de $\mathcal{P}^{-1}$, ce qui leur a permis de déterminer qu'une valeur particulière de la différence en sortie de $\mathcal{P}^{-1}$ est atteinte avec probabilité 2^{-156}. Cette valeur contient 15 bits différents sur A et 28 bits différents sur B. Par l'inverse de l'addition sur B de M dans $\mathcal{F}^{-1}$, les 28 bits de différence sur B et le bit de différence sur M entraînent 29 bits de différence sur B en sortie de $\mathcal{F}^{-1}$ à des positions précises avec probabilité 2^{-28}.

Finalement, on obtient donc, avec probabilité $2^{-(156+28)} = 2^{-184}$, 15 bits de différence sur A, 29 bits de différence sur B et 1 bit de différence sur C en sortie de $\mathcal{F}^{-1}$ (à des positions précises). Réciproquement, si on part d'états différant sur ces bits et qu'on applique $\mathcal{F}$ avec des blocs de messages différant sur le bit de poids fort de M_0, on obtient une collision sur l'état interne avec probabilité 2^{-184}.

En effet, si on fixe deux blocs de message M et M' différant uniquement sur le bit 31 de M_0, la fonction de compression de Shabal paramétrée par M est une permutation. Notons α, β, γ les valeurs des différences sur A, B, C décrites dans cette attaque : nous avons expliqué que pour A', B', C' tirés uniformément,

$$\Pr\left[\mathcal{F}_M^{-1}(A', B', C') = \mathcal{F}_{M'}^{-1}(A' \oplus \alpha, B' \oplus \beta, C' \oplus \gamma)\right] = 2^{-184} .$$

De manière équivalente, nous avons pour A, B, C tirés uniformément,

$$\Pr\left[\mathcal{F}_M(A, B, C) \oplus \mathcal{F}_{M'}(A, B, C) = (\alpha, \beta, \gamma)\right] = 2^{-184} .$$

Cette probabilité peut être augmentée jusqu'à 2^{-84} si on choisit des entrées spécifiques de manière à satisfaire une partie du chemin différentiel de manière déterministe. D'autre part, du fait de la lenteur de la diffusion par $\mathcal{P}^{-1}$, il existe probablement d'autres propriétés du même type contre $\mathcal{F}^{-1}$, avec des probabilités de succès un peu plus faibles. Néanmoins, elles ne permettent pas d'attaquer les fonctions de la famille Shabal. En effet, pour pouvoir utiliser cette attaque contre la fonction de hachage complète, il est nécessaire de parvenir à des valeurs de l'état interne différant uniquement sur des bits à des positions données, ce qui est théoriquement aussi difficile à réaliser qu'une collision sur l'état interne.

4.4.3 Points fixes

Une note de Knudsen, Matusiewicz et Thomsen [KMT09] établit une classe d'états internes de Shabal pour lesquels la permutation $\mathcal{P}$ a un comportement particulier.

Pour toute valeur de $b \in \{0,1\}^{32}$, on définit $\tilde{b} = b \lll 17$ et $a = \overline{\tilde{b} \oplus (\tilde{b} \lll 1)}$. On choisit alors, pour toute valeur de i :

$$\begin{aligned} A_i &\leftarrow a \\ B_i &\leftarrow b \\ M_i &\leftarrow a \oplus \tilde{b} \oplus \mathcal{U}(a \oplus \mathcal{V}(a \lll 15) \oplus C_{8-i \bmod 16}), \end{aligned}$$

les 16 valeurs de C_i étant aléatoires. Au cours de $\mathcal{P}$, après la première phase de rotations, tous les mots de B valent $\tilde{b}$. En particulier, on a donc $B_{i+9 \bmod 16} \wedge \overline{B_{i+6 \bmod 16}} = 0$. On vérifie alors que A et B sont inchangés au cours de la boucle principale. A est ensuite modifié par les additions finales, mais les mots de B' en sortie de $\mathcal{P}$ valent tous $\tilde{b}$.

Cette propriété peut alors être étendue de la manière suivante. Au lieu de choisir une seule valeur pour b, on en choisit 4, $b_0 \dots b_3$. De même que précédemment, on définit $\tilde{b}_i = b_i \lll 17$, et $a_i = \overline{\tilde{b}_i \oplus (\tilde{b}_i \lll 1)}$, pour $i \in \{0 \dots 3\}$. On choisit, pour $i \in \{0 \dots 15\}$ ($\{0 \dots 12\}$ dans le cas de A_i) :

$$\begin{aligned} A_i &\leftarrow a_{i \bmod 4} \\ B_i &\leftarrow b_{i \bmod 4} \\ M_i &\leftarrow a_{i \bmod 4} \oplus \tilde{b}_{i+1 \bmod 4} \wedge \tilde{b}_{i+2 \bmod 4} \oplus \mathcal{U}(a_{i \bmod 4} \oplus \mathcal{V}(a_{i \bmod 4} \lll 15) \oplus C_{8-i \bmod 16}), \end{aligned}$$

Comme dans le cas précédent, la boucle principale préserve la valeur de l'état interne. B reste donc inchangé après la rotation initiale. Un cas particulier d'application de cette propriété est la découverte de points fixes pour $\mathcal{P}$. En effet, si on choisit des valeurs de b_i invariantes par rotations et des valeurs de C_i identiquement nulles, on a $\mathcal{P}_{C,M}(A, B) = (A, B)$.

4.4.4 Distingueurs rotationnels

La cryptanalyse rotationnelle est une technique d'attaque apparue au cours de la compétition SHA-3. Cette notion a été introduite par Khovratovich et Nikolić, et s'applique à des primitives dites ARX (pour Add-Rot-Xor), c'est-à-dire dont les opérations de base sont des additions modulaires, des rotations et des OU EXCLUSIFS. Elle a notamment été appliquée à Threefish, qui est l'algorithme de chiffrement par blocs sur lequel le candidat SHA-3 Skein est construit [KN10, KNR10].

Une note de Van Assche [Ass10] montre que cette technique est applicable à Shabal. En effet, la plupart des opérations internes de Shabal (opérations booléennes telles que les additions bit-à-bit, rotations) commutent avec les rotations sur leurs opérandes. Seules les additions modulaires et les fonctions $\mathcal{U}$ et $\mathcal{V}$ ne sont pas dans ce cas.

Pour une (des) entrée(s) aléatoire(s) dans $\{0,1\}^{32}$, Van Assche a établi les propriétés suivantes :

$$\Pr\left[(x \lll 1) + (y \lll 1) = (x + y) \lll 1\right] \approx 2^{-1,415} \tag{4.6}$$

$$\Pr\left[\mathcal{U}(x) \lll 1 = \mathcal{U}(x \lll 1)\right] = \frac{2^{32} - 1}{3 \times 2^{32}} \approx 2^{-1,585} \tag{4.7}$$

$$\Pr\left[\mathcal{V}(x) \lll 1 = \mathcal{V}(x \lll 1)\right] = \frac{3 \times 2^{32} - 8}{10 \times 2^{32}} \approx 2^{-1,737} \tag{4.8}$$

$\mathcal{P}$ fait intervenir 48 fois chacune des fonctions $\mathcal{U}$ et $\mathcal{V}$ dans la boucle principale, ainsi que 36 additions modulaires pour les additions finales. En considérant que ces opérations commutent avec la rotation indépendamment les unes des autres, on en déduit que

$$\Pr\left[\mathcal{P}_{C \lll 1, M \lll 1}(A \lll 1, B \lll 1) = \mathcal{P}_{C,M}(A, B) \lll 1\right] \approx 2^{-(48 \times (1,585 + 1,737) + 36 \times 1,415)} \approx 2^{-210} \tag{4.9}$$

Pour une permutation paramétrée aléatoire sur des entrées de même taille, cette propriété est vraie avec probabilité 2^{-896}.

Améliorations possibles. Van Assche propose également deux améliorations possibles de cette attaque. La première consiste à considérer uniquement la partie B' de la sortie. Dans ce cas, l'attaquant n'a pas besoin de garantir que les additions finales préservent la rotation, puisqu'elles n'impactent pas la valeur de B'. La complexité de l'attaque devient alors 2^{-159} environ.

Une autre amélioration possible consiste à fixer A et B en entrée de la permutation, et à fixer les mots de C et M les uns après les autres de manière à garantir que les premières étapes des fonctions $\mathcal{U}$ et $\mathcal{V}$ commutent avec la rotation. On peut alors réécrire les équations de la boucle principale de la manière suivante :

$$\begin{aligned} V &= \mathcal{V}(A_{i-1 \bmod 12}) \\ X &= A_{i \bmod 12} \oplus V \oplus C_{8-i \bmod 16} \oplus B_{i+13 \bmod 16} \oplus B_{i+9 \bmod 16} \wedge \overline{B}_{i+6 \bmod 16} \\ U &= \mathcal{U}(X) \\ A_{i \bmod 12} &= U \oplus M_i \\ B_{i \bmod 16} &= B_{i \bmod 16} \lll 1 \oplus \overline{A_{i \bmod 12}} \end{aligned}$$

On peut choisir une valeur initiale A_{11} pour laquelle la rotation et $\mathcal{V}$ commute lors de la première étape. Ensuite, au cours des étapes 0 à 15, on calcule X (respectivement $A_{i \bmod 12}$) en faisant intervenir un mot de C (respectivement M) non encore utilisé. On peut donc adapter le choix de ces valeurs de manière à ce que $\mathcal{U}$ (respectivement $\mathcal{V}$) et la rotation commutent sur X (respectivement $A_{i \bmod 16}$). De cette manière, on garantit que 16 des occurences de $\mathcal{U}$ et 17 des occurences de $\mathcal{V}$ commutent effectivement avec la rotation. La complexité de l'attaque résultante est alors $2^{-155,5}$.

Application à Shabal. Pour Shabal, la valeur initiale de l'état interne est fixe, et ne contient pas de symétrie par rotation. Avant de pouvoir utiliser cette propriété au cours d'un calcul de haché, il faut d'abord parvenir soit à atteindre un état invariant par rotation, soit deux états différents dont l'un est obtenu en appliquant une rotation à l'autre. Du fait de la grande taille de l'état interne, une telle propriété est difficile à obtenir.

Néanmoins, ce type d'attaque pourrait permettre d'attaquer une version de Shabal pour laquelle il n'y aurait pas de traitement spécifique du dernier bloc, pas de compteur, et une valeur initiale de l'état nulle. L'utilisation d'une valeur initiale de l'état interne générée aléatoirement a été décidée dans le but d'éviter des symétries de ce type.

4.5 Conclusion

Malgré son élimination avant la dernière phase, nous pouvons tirer quelques satisfactions du parcours de Shabal dans la compétition SHA-3. D'une part, malgré les distingueurs contre la permutation $\mathcal{P}$ qui ont été publiés, aucune attaque contre une instance complète de Shabal n'a été découverte. D'autre part, la sélection de Shabal parmi 14 candidats pour le deuxième tour de la compétition montre que ses performances sont bonnes. Les implantations des différents candidats qui ont été réalisées placent Shabal parmi les fonctions les plus performantes.

Toutefois, des incertitudes planent sur la sécurité de Shabal, en partie liées au format de la compétition qui impose une période d'évaluation relativement courte par rapport à la durée d'utilisation prévue pour le vainqueur. Compte tenu de la prépondérance de ce critère, la non sélection de Shabal pour la dernière phase de la compétition paraît logique.

Chapitre 5

Extension de domaine et indifférenciabilité d'un oracle aléatoire

Sommaire

Les résultats exposés dans ce chapitre concernent la sécurité des algorithmes d'extension de domaine. Nous étudions l'indifférenciabilité d'un oracle aléatoire d'une classe d'algorithmes d'extension de domaine. Cette notion est définie dans le chapitre 2.

Ces travaux ont été initiés au cours de la définition de Shabal. Une version préliminaire de ces résultats est donc détaillée dans [BCCM+08]. Ils se sont poursuivis dans le cadre du projet Saphir2, avec la motivation d'apporter des éléments concernant la sécurité de Shabal, et d'améliorer ses chances dans la compétition SHA-3. Étant donné la spécificité de l'algorithme d'extension de domaine de Shabal, nous introduisons une description de haut niveau de ce mode. Cette description englobe d'autres algorithmes d'extension de domaine, notamment Chop-MD, défini dans [CDMP05], et utilisé par d'autres candidats SHA-3 tels que Blue Midnight Wish, SIMD, ECHO ou Grøstl.

L'indifférenciabilité de cette construction a déjà été étudiée par Coron *et al.* [CDMP05] et par Chang et Nandi [CN08]. Les résultats que nous avons obtenus améliorent légèrement les bornes précédemment connues.

Dans la section 5.1, nous introduisons notre nouvelle représentation d'une classe d'algorithmes d'extension de domaine, et nous montrons que le mode de Shabal et Chop-MD en font partie. Les algorithmes de cette classe font appel à une primitive interne, qu'on peut instancier soit avec une permutation paramétrée (cas de Shabal) soit avec une fonction sans propriété particulière. Après avoir expliqué notre stratégie pour montrer l'indifférenciabilité d'un oracle aléatoire de notre construction en section 5.2, nous étudions l'indifférenciabilité de notre construction dans le cas le plus général où la primitive interne est représentée comme une fonction aléatoire en section 5.3. Nous traitons notamment le cas où l'encodage utilisé est sans préfixe. En section 5.4, nous nous intéressons au cas où la primitive interne est une permutation paramétrée. Dans la section 5.5, nous adaptons notre preuve au cas plus spécifique de l'algorithme d'extension de domaine de Shabal.

5.1 Représentation générique d'algorithmes d'extension de domaine

La construction dont nous évaluons la sécurité dans ce chapitre est décrite sur la Figure 5.1. Elle permet de hacher des messages $\mathcal{M} \in \{0,1\}^*$ de longueur arbitraire, en traitant itérativement des blocs de ℓ_m bits. Le traitement d'un bloc de message a pour effet de modifier un état interne [1] de n bits.

Afin d'utiliser notre construction pour hacher des messages arbitraires, il est nécessaire de disposer d'une fonction non ambigüe (c'est-à-dire injective) qui transforme une suite finie de bits en une suite finie de blocs de ℓ_m bits. Nous ne spécifions pas de fonction particulière ici, mais un tel encodage est généralement obtenu en composant un padding injectif dont la sortie est une suite de bits dont la longueur est multiple de ℓ_m et une division de cette chaîne en blocs de ℓ_m bits. Nous noterons pad cette opération, et unpad l'opération inverse. pad étant supposé injectif, unpad renvoie soit $\perp$, soit une valeur bien définie.

L'empreinte de M (de longueur ℓ_h) correspond à une partie de la valeur finale de l'état interne, c'est-à-dire après le traitement de la donnée $\mathsf{pad}(M)$ qui est une suite finie de blocs de ℓ_m bits.

Nous considérons plusieurs cas particuliers de cette construction, définis par des propriétés de la fonction d'encodage, qui sont :

1. *Encodage sans préfixe :* un encodage est dit sans préfixe si et seulement si pour tout couple de messages $M \neq M'$, $\mathsf{pad}(M)$ n'est pas un préfixe de $\mathsf{pad}(M')$.
2. *Encodage avec tours à blanc et compteur :* nous considérons également le cas où le dernier bloc de message est traité $n_f + 1$ fois. Cela revient à considérer que la fonction d'encodage retourne des valeurs dont les $n_f + 1$ derniers blocs sont identiques. Un tel renforcement du dernier bloc n'est pas rare, le cas considéré correpond aux $n_f = 3$ tours à blanc du mode de Shabal. Pour être cohérent avec le mode de Shabal, nous introduisons également un compteur du nombre de blocs déjà traités, qui est incrémenté au cours du traitement du message mais pas pendant les tours à blanc.

1. L'expression utilisée habituellement pour la construction de Merkle-Damgård est *variable de chaînage*, *état interne* est plus utilisée dans le contexte des fonctions éponges

La fonction utilisée pour mettre à jour l'état interne à l'aide d'un bloc de message est décomposée en deux opérations. Premièrement, le bloc de message est inséré à l'aide d'une opération $\mathsf{Insert}[M]$, qui réalise une transformation inversible de l'état interne, paramétrée par le bloc de message. Deuxièmement, une fonction $\mathcal{F}$ est appliquée à l'état interne et au bloc de message $M \in \{0,1\}^{\ell_m}$. C'est cette fonction qui constitue le cœur de notre construction et qui est remplacée par une version idéalisée dans les preuves. La nouvelle valeur de l'état interne est alors obtenue en concaténant la sortie de $\mathcal{F}$ avec une partie de l'état interne en entrée de $\mathcal{F}$. Cette partie de l'état interne sera notée C (par analogie avec la définition de Shabal), et sa taille ℓ_c. L'état interne de la construction se compose de trois parties A, B et C. B, de taille ℓ_h correspond à la partie de l'état interne qui constitue le haché après application de la dernière fonction de compression. A représente le reste de l'état interne, sa taille est notée ℓ_a. La primitive interne $\mathcal{F}$ est notée

$$\begin{aligned} \mathcal{F} : \{0,1\}^{\ell_m} \times \{0,1\}^{n} &\rightarrow \{0,1\}^{n-\ell_c} \\ (M, A, B, C) &\mapsto \mathcal{F}_{M,C}(A, B) = (A', B'). \end{aligned}$$

Le traitement du bloc de message i correspond à la transformation suivante :

$$\mathrm{Round}[M^i](x) = (\mathcal{F}(M^i, \mathsf{Insert}[M^i](x)), C^i)$$

où $x = (A^{i-1}, B^{i-1}, C^{i-1})$ est la valeur de l'état interne avant le traitement du bloc M^i, et C^i est la partie C de la sortie que $\mathcal{F}$ ne modifie pas. La valeur initiale de l'état interne est une constante $x^0 = (A^0, B^0, C^0)$.

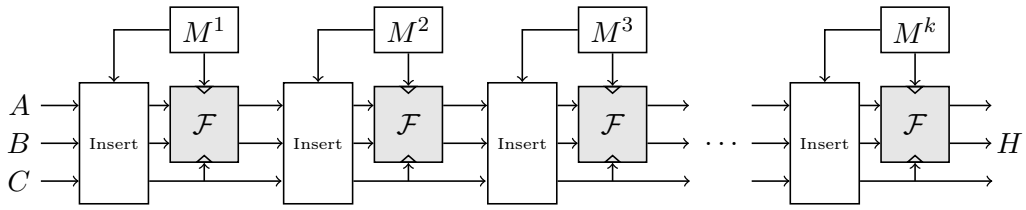


FIGURE 5.1 – Représentation générique des algorithmes d'extension de domaine traités dans le chapitre 5

Notre construction englobe les algorithmes d'extension de domaine de plusieurs fonctions de hachage, notamment les suivants.

Chop-MD. La construction Chop-MD correspond au cas où l'insertion de message est l'identité, c'est-à-dire

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

et où C est vide, soit $\ell_c = 0$. L'état interne coïncide dans ce cas avec la sortie de la fonction de compression. Lorsque la taille de l'état interne est le double de celle de la sortie, c'est-à-dire $n = 2\ell_h$, on obtient la construction définie par Lucks [Luc05], désignée en anglais par le terme *double pipe*.

Shabal. Le mode de Shabal-512 est obtenu pour une insertion de message définie par

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

et lorsque $\ell_h = \ell_c = \ell_m = 512$ et $n = 1408$. En fait, le mode de Shabal tel que représenté sur la Figure 5.2 ne rentre pas directement dans le cadre de la construction de ce chapitre. En effet, les transformations effectuées entre deux applications de $\mathcal{P}$ impliquent deux blocs de message différents. Cette représentation alourdit les techniques de preuves, rendant leur compréhension plus délicate.

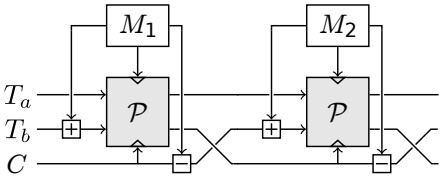


FIGURE 5.2 – Mode de Shabal.

Pour simplifier l'écriture de la preuve, nous proposons de modifier la représentation du mode de Shabal de la manière suivante. Au lieu de soustraire le bloc de message M à C après l'appel à $\mathcal{P}$, on effectue cette soustraction avant l'appel à $\mathcal{P}$. Il est alors nécessaire d'effectuer l'opération inverse, une addition de M sur la variable C, qui rentre comme paramètre de $\mathcal{P}$, $C \to C \boxplus M$, avant le calcul de $\mathcal{P}$. La composition de ces deux fonctions (addition de M sur C et appel à $\mathcal{P}$) est également une permutation paramétrée $\mathcal{Q}$, dont les interfaces sont identiques à celles de $\mathcal{P}$.

D'autre part, l'échange des variables B et C est également fait avant l'appel à $\mathcal{P}$, de manière à être intégré dans la fonction d'insertion de message. Afin de rester cohérent avec la définition de Shabal, les valeurs initiales de A, B, C sous cette représentation sont les constantes (A^0, C^0, B^0) telles que (A^0, B^0, C^0) soit la valeur initiale de l'état interne définie dans la spécification de Shabal [BCCM+08].

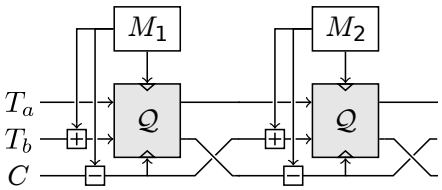


FIGURE 5.3 – Représentation équivalente du mode de Shabal.

Nous pouvons remarquer que les définitions de A et B utilisées dans ce chapitre dépendent essentiellement de la taille du haché, alors que les notations A et B utilisées dans la description de Shabal au chapitre 4 correspondent à des registres à décalage différents. Par conséquent, pour des tailles de haché différentes de 512 bits, ces deux notations ne coincident pas. Sur les Figures 5.2 et 5.3, nous utilisons la notation T_a, T_b pour caractériser les variables notées A, B au chapitre 4.

De plus, le mode de Shabal impose l'utilisation d'un compteur du nombre de blocs traités, codé sur 64 bits et ajouté à 2 mots de A par un OU EXCLUSIF avant chaque application de $\mathcal{P}$. Nous traitons ce cas spécifique à la fin de la section 5.4.

5.2 Techniques de preuve d'indifférenciabilité

Dans cette section nous décrivons les principales techniques qui nous ont permis de réaliser la preuve de l'indifférenciabilité d'un oracle aléatoire de notre construction.

5.2.1 Le modèle d'indifférenciabilité

Le concept d'indifférenciabilité d'un système par rapport à un autre a été introduit par Maurer *et al.* [MRH04] et décrit au chapitre 2. Informellement, il peut être représenté par un scénario d'attaque précis dans lequel un distingueur (ou attaquant) $\mathcal{D}$ interagit avec un système d'oracles Σ. La définition générale du modèle d'indifférenciabilité peut être appliquée à tous les systèmes cryptographiques, nous nous concentrons ici sur son application aux algorithmes d'extension de domaine pour des fonctions de hachage. De manière générale, Σ peut contenir plusieurs composants ; dans le cas que nous étudions, il s'agit d'une fonction de hachage $\mathcal{C}^{\mathcal{F}}$ qui appelle une primitive interne $\mathcal{F}$. Le modèle de l'indifférenciabilité de systèmes cryptographiques permet la comparaison de deux systèmes, dans notre étude le second système est construit à partir d'un oracle aléatoire $\mathcal{H}$.

Informellement, la construction $\mathcal{C}$ est dite indifférenciable (jusqu'à une borne de sécurité donnée) de l'oracle aléatoire si le système $\Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ peut être remplacé par un autre système d'oracles $\Sigma' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$ dont les interfaces sont identiques, sans que $\mathcal{D}$ ne puisse déterminer avec lequel des systèmes il interagit (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 de Shabal, $\mathcal{F}$ est une permutation dont l'inverse peut être calculé facilement. Nous introduisons donc dans le modèle un accès à $\mathcal{F}^{-1}$ pour le distingueur. $\mathcal{S}$ doit donc également pouvoir simuler les réponses à ces requêtes.

Notons d'ores et déjà que dans le modèle de l'indifférenciabilité, $\mathcal{S}^{\mathcal{H}}$ a un accès à l'oracle $\mathcal{H}$ mais ne connaît pas les requêtes de haché émises par $\mathcal{D}$ à $\mathcal{H}$. Ceci constitue une des difficultés principales de l'établissement de la preuve.

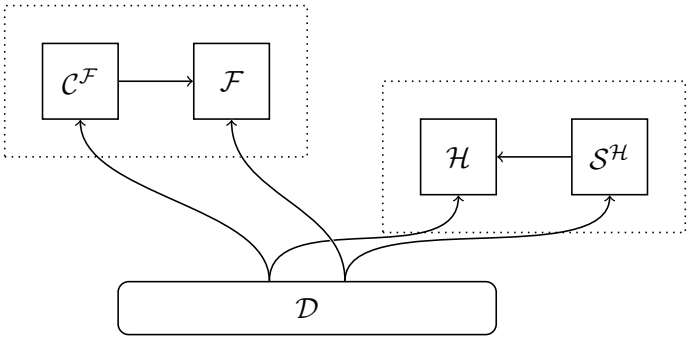


FIGURE 5.4 – La construction $\mathcal{C}^{\mathcal{F}}$ peut questionner l'oracle $\mathcal{F}$. Le simulateur $\mathcal{S}^{\mathcal{H}}$ peut évaluer les sorties de l'oracle aléatoire $\mathcal{H}$. Le distingueur $\mathcal{D}$ interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ ou $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$ et doit différencier les deux scénarios.

Au cours de l'interaction avec Σ ou Σ', le distingueur émet des *requêtes à gauche* à $\mathcal{C}^{\mathcal{F}}$ ou $\mathcal{H}$, et des *requêtes à droite* à $\mathcal{F}$ ou $\mathcal{S}^{\mathcal{H}}$. Nous notons N le poids total des requêtes droites, c'est-à-dire le nombre de requêtes reçues par $\mathcal{F}$ quand $\mathcal{D}$ interagit avec Σ. Ces requêtes peuvent provenir de $\mathcal{D}$ ou de $\mathcal{C}^{\mathcal{F}}$. A la fin de l'interaction, l'attaquant renvoie un bit $\mathcal{D}^{\Sigma}$. Nous définissions l'avantage du distingueur comme

$$\mathsf{Adv}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}^{\Sigma} = 1 \mid \Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})\right] - \Pr\left[\mathcal{D}^{\Sigma} = 1 \mid \Sigma = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})\right]\right|, \tag{5.1}$$

où les probabilités sont évaluées sur les aléas utilisés par toutes les parties du scénario. L'interprétation de l'avantage de $\mathcal{D}$ peut être la suivante. Dans le modèle de l'indifférenciabilité, $\mathcal{D}$ est un algorithme dont le but est de déterminer s'il interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ ou avec $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$, ce qui est

caractérisé par le fait que sa sortie est un bit : l'attaquant cherche à renvoyer 1 s'il interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ et 0 s'il interagit avec $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. L'avantage de l'attaquant caractérise la différence de son comportement selon le système avec lequel il interagit.

$\mathsf{Adv}(\mathcal{D})$ dépend évidemment du nombre de requêtes à gauche et à droite qu'il peut émettre, de manière générale nous donnerons donc un majorant de l'avantage de $\mathcal{D}$ en fonction du poids total N des requêtes. La preuve d'indifférenciabilité de $\mathcal{C}$ consiste alors à construire un simulateur approprié pour $\mathcal{F}$, ou pour $(\mathcal{F}, \mathcal{F}^{-1})$ dans le cas d'une permutation paramétrée, et à majorer l'avantage de tout distingueur interagissant avec les systèmes Σ et Σ'.

5.2.2 Élements de construction du simulateur

Soit $\mathcal{X} = \{0,1\}^n$ l'ensemble de toutes les valeurs possibles de l'état interne. Pour rappel, le traitement du bloc de message M^i consiste en l'exécution d'une sous-routine

$$\mathrm{Round}[M^i](x) = (\mathcal{F}(M^i, \mathsf{Insert}[M^i](x)), C^i)$$

sur la valeur courante de l'état interne $x \in \mathcal{X}$. L'état interne initial est un vecteur d'initialisation fixe $x^0 = (A^0, B^0, C^0)$. Le simulateur de $\mathcal{F}$ (resp. de $(\mathcal{F}, \mathcal{F}^{-1})$) est obtenu en construisant un graphe orienté $\mathcal{G} = (X, E) \subseteq \mathcal{X} \times \mathcal{X}^2$, dans lequel X représente l'ensemble des nœuds et E l'ensemble des arcs labellisés par des blocs de message.

X est l'ensemble de toutes les valeurs d'état interne $\mathsf{Insert}^{-1}[M](A, B, C)$, où (M, A, B, C) est une requête droite reçue par le simulateur, et de toutes les réponses complétées par la partie C de l'état pour obtenir un état interne dans $\mathcal{X}$. Lorsque $\mathcal{F}$ est une permutation, X contient également les requêtes (M, A', B', C) à $\mathcal{F}^{-1}$, ainsi que les états $\mathsf{Insert}^{-1}[M](A, B, C)$, où (A, B) est la réponse correspondante. Chaque arc du graphe représente le fait que la sortie de la fonction d'itération Round appliquée à un état x et un bloc de message M a été déterminée lors d'une requête et vaut y. Un tel arc est alors notée $x \xrightarrow{M} y$.

Un *chemin* de l'état initial x^0 vers un nœud $x \in X$ dans le graphe est une liste (éventuellement vide) de blocs de messages $\mu = \langle M^1, \ldots, M^k \rangle$ tel 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 $x^k = x$.

Un chemin μ dans le graphe est dit *complet* s'il existe M tel que $\mathsf{pad}(M) = \mu$. On note alors $M = \mathsf{unpad}(\mu)$. Une fonction d'encodage quelconque étant supposée injective, l'unicité de M est immédiate. En revanche, son existence systématique n'est pas garantie sans faire plus d'hypothèse sur la fonction d'encodage.

Un chemin μ dans le graphe est dit *utile* s'il existe $M \in \{0,1\}^*$ et $\mu' \in (\{0,1\}^{\ell_m})^*$ tel que $\mu || \mu' = \mathsf{pad}(M)$. Lorsqu'un chemin est inutile, il ne correspond au début d'aucun calcul de haché.

5.2.3 Preuves par séquence de jeux.

La preuve d'indifférenciabilité d'un oracle aléatoire d'un algorithme d'extension de domaine nécessite la définition d'un simulateur et la majoration de l'avantage de tout attaquant cherchant à différencier les systèmes $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ et $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. Afin de pouvoir borner cet avantage, il est habituel de construire le simulateur en appliquant des modifications élémentaires à partir d'un simulateur passif.

Une telle démarche est appelée *preuve par séquence de jeux*. Elle a été formalisée par Shoup dans [Sho04]. Le jeu initial correspond à la situation où $\mathcal{D}$ interagit avec Σ, et le jeu final, à celle où il interagit avec Σ'. Les jeux successifs de la séquence correspondent aux interactions avec les

systèmes Σ_i basés sur les simulateurs successifs. On note alors W_i l'évènement selon lequel $\mathcal{D}$ renvoie 1 lorsqu'il interagit avec le système Σ_i (avec $\Sigma_0 = \Sigma$ et $\Sigma_\ell = \Sigma'$). Par l'inégalité triangulaire, on a

$$\mathsf{Adv}(\mathcal{D}) \leq \sum_{i=0}^{\ell-1} |\Pr[W_{i+1}] - \Pr[W_i]| .$$

En d'autres termes, l'avantage de l'attaquant est borné par la somme des différences de comportement qu'il peut avoir entre deux jeux successifs.

Afin de pouvoir quantifier ces avantages, nous définissons la *vue de l'attaquant* $\mathcal{D}$ comme la succession des distributions de probabilité des réponses à toutes les requêtes qu'il émet.

Les transitions entre deux jeux successifs peuvent être classées selon trois types :

- Dans les transitions passives, les réponses que reçoit l'attaquant sont identiques entre les deux jeux. Les différences peuvent résider dans les interfaces internes du système Σ.
- Dans les transitions transparentes, le simulateur génère ses réponses différemment entre les deux jeux. Toutefois, cette différence de comportement du simulateur laisse inchangée la distribution de probabilités des réponses, et ne modifie donc pas la vue de l'attaquant.
- Dans les transitions basées sur des cas d'échec, le simulateur peut modifier les distributions de probabilités des réponses à certaines requêtes. Ces modifications entraînent alors le marquage d'un échec de la simulation. La différence de comportement de l'attaquant est bornée par la probabilité d'occurrence d'un échec, en utilisant le lemme de différence décrit plus bas. Les évènements d'échec doivent rester marginaux pour permettre de garantir la proximité du comportement de l'attaquant.

5.2.4 Lemme de différence.

Au cours de la preuve par séquence de jeux, la vue de l'attaquant peut être modifiée entre deux jeux consécutifs. Afin de matérialiser et de quantifier cette différence, on a généralement recours à la définition d'évènements d'échec. On utilise alors un résultat que Shoup nomme lemme de différence dans [Sho04]. Nous en donnons ici une version généralisée.

Lemme 3 *Soient A, B, F, G quatre évènements définis selon une certaine distribution de probabilités, et supposons que*

$$\Pr\left[A \wedge \overline{F}\right] = \Pr\left[B \wedge \overline{G}\right] .$$

La différence de probabilités entre les occurrences de A et B vérifie alors

$$|\Pr[B] - \Pr[A]| \leq \max(\Pr[F], \Pr[G])$$

Démonstration : On peut décomposer $\Pr[B] - \Pr[A]$ de la manière suivante :

$$\begin{aligned} \Pr[B] - \Pr[A] &= \Pr[B \wedge G] - \Pr[A \wedge F] + \Pr\left[B \wedge \overline{G}\right] - \Pr\left[A \wedge \overline{F}\right] \\ &= \Pr[B \wedge G] - \Pr[A \wedge F] . \end{aligned}$$

On en déduit l'encadrement :

$$-\Pr[F] \leq \Pr[B] - \Pr[A] \leq \Pr[G] .$$

On en déduit la relation du lemme en passant à la valeur absolue. □

Dans son article, Shoup écrit la version de ce lemme pour $F = G$. Il est en fait possible que les évènements F et G ne soient pas rigoureusement identiques dans la pratique, comme nous allons le voir dans notre preuve.

5.2.5 Simulation d'une fonction aléatoire.

Dans la preuve de la section 5.3, $\mathcal{S}$ doit simuler le comportement d'une fonction tirée uniformément sur l'ensemble des fonctions de $n + \ell_m$ bits vers $n - \ell_c$ bits. Pour ce faire, il n'est pas nécessaire de choisir une fonction aléatoire au début de chaque jeu. La connaissance que $\mathcal{D}$ peut avoir de $\mathcal{F}$ lorsqu'il interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ provient uniquement des réponses aux requêtes à $\mathcal{F}$, émises directement ou par l'intermédiaire de $\mathcal{C}$.

Notons F l'ensemble des fonctions de $n + \ell_m$ bits vers $n - \ell_c$ bits. Pour que les réponses de $\mathcal{S}$ ne puissent pas être différenciées d'une fonction tirée uniformément sur F, il faut qu'elles suivent des distributions de probabilités statistiquement proches, les réponses aux requêtes précédentes étant connues. Dans le cas des preuves de ce chapitre, nous cherchons à simuler parfaitement $\mathcal{F}$. En d'autres termes, à la requête k, étant donné les réponses (A'_i, B'_i) aux requêtes (M_i, A_i, B_i, C_i) précédentes, on cherche à avoir pour toute valeur de (A'_k, B'_k) :

$$\begin{aligned} &\Pr\left[\mathcal{S}(A_k, B_k, C_k, M_k) = (A'_k, B'_k) | \mathcal{F}(A_i, B_i, C_i, M_i) = (A'_i, B'_i) \forall i \in \{1, \ldots, k-1\}\right] \quad = \\ &\Pr_{\mathcal{F} \leftarrow F}\left[\mathcal{F}(A_k, B_k, C_k, M_k) = (A'_k, B'_k) | \mathcal{F}(A_i, B_i, C_i, M_i) = (A'_i, B'_i) \forall i \in \{1, \ldots, k-1\}\right] \end{aligned} \tag{5.2}$$

Lorsque $\mathcal{F}$ est tirée uniformément sur F, on a toujours

$$\Pr_{\mathcal{F} \leftarrow F}\left[\mathcal{F}(A_k, B_k, C_k, M_k) = (A'_k, B'_k) | \mathcal{F}(A_i, B_i, C_i, M_i) = (A'_i, B'_i) \forall i \in \{1, \ldots, k-1\}\right] = 2^{-(\ell_a + \ell_h)}$$

si $(A_i, B_i, C_i, M_i) \neq (A_k, B_k, C_k, M_k)$ pour tout i. Les réponses de $\mathcal{S}$ aux différentes requêtes doivent donc toujours être indépendantes et identiquement distribuées. Lorsque cette propriété est incompatible avec la cohérence avec les réponses de l'oracle aléatoire, nous définissons des évènements d'échec.

5.2.6 Identification d'évènements d'échec

Afin de caractériser les différences de comportement entre les simulateurs successifs de la séquence de jeux, ces derniers maintiennent une variable Drapeau permettant de caractériser les évènements d'échec. Les différents évènements d'échec sont caractérisés par des valeurs Echec prises dans un ensemble Z de cas d'échec. Cette variable est écrite par une sous-routine `NoteEchec`(Echec), décrite en Figure 5.1.

Algorithme 5.1 Sous-routine `NoteEchec` du Simulateur

Entrées : Echec $\in Z$
  si Drapeau = Succès **alors**
    Drapeau $\leftarrow$ Echec
  fin si

5.2.7 Simulation de $\mathcal{F}$ et détection d'évènements d'échec

La stratégie utilisée implique à la fois la simulation d'une fonction aléatoire $\mathcal{F}$ et la formalisation de la détection d'évènements d'échec. Le comportement à adopter pour réaliser ces deux opérations est différent, cependant elles impliquent toutes deux la connaissance de certaines valeurs de $\mathcal{F}$ déjà définies au cours du jeu. De manière artificielle, ces deux opérations sont dévolues au simulateur $\mathcal{S}$. Cependant, dans le système Σ', $\mathcal{S}^{\mathcal{H}}$ n'a pas accès aux requêtes de hachage émises par $\mathcal{D}$, alors que nous allons voir que ces requêtes peuvent provoquer des évènements indésirables.

Dans les jeux intermédiaires, nous introduisons un programme supplémentaire, noté $\mathcal{I}$ et appelé *intercepteur*, dont le rôle est de permettre à $\mathcal{S}$ de connaître les requêtes de haché émises par $\mathcal{D}$. Lorsque l'accès à l'oracle $\mathcal{F}$ est remplacé par une simulation, il est nécessaire de s'assurer que $\mathcal{F}$ n'utilise pas d'information venant de $\mathcal{I}$ pour répondre aux requêtes venant de $\mathcal{D}$.

5.3 Preuve d'indifférenciabilité lorsque $\mathcal{F}$ est une fonction

5.3.1 Borne sur l'avantage du distingueur

Nous considérons d'abord le cas le plus générique de l'algorithme d'extension de domaine de la Figure 5.1, avec une fonction d'encodage quelconque. L'étude de l'indifférenciabilité du mode est alors effectuée en instanciant $\mathcal{F}$ avec une fonction tirée uniformément dans l'ensemble des fonctions de $\ell_m + n$ bits vers $n - \ell_c$ bits. Nous présentons maintenant le simulateur $\mathcal{S}$ (Figure 5.5) utilisé pour montrer l'indifférenciabilité des systèmes Σ et Σ'.

Initialisation de $\mathcal{S}$

Pas d'entrée, pas de sortie

1. Définir $X = \{x^0\}$ et $E = \emptyset$

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C)

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \xrightarrow{M} y \in E$
   (a) renvoyer (A', B') tel que $y = (A', B', C)$
3. si x a un chemin utile μ dans le graphe $\mathcal{G}$ et si $\mu||M$ est un chemin complet
   (a) si μ n'est pas unique, arrêter $\mathcal{S}$
   (b) calculer $\mathcal{M} = \mathsf{unpad}(\mu||M)$
   (c) appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
   (d) définir $B' = h$
   (e) choisir aléatoirement $A' \leftarrow \{0,1\}^{\ell_a}$ et définir $y = (A', B', C)$.
4. sinon
   (a) choisir aléatoirement $B' \leftarrow \{0,1\}^{\ell_h}$
   (b) choisir aléatoirement $A' \leftarrow \{0,1\}^{\ell_a}$ et définir $y = (A', B', C)$.
5. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
6. renvoyer (A', B')

FIGURE 5.5 – Simulateur $\mathcal{S}$ pour l'algorithme d'extension de domaine générique dane le cas où $\mathcal{F}$ est une fonction idéale, pour une fonction d'encodage quelconque.

A l'aide de notre simulateur, nous obtenons des bornes supérieures précises sur l'avantage d'un distingueur dans le scénario d'indifférenciabilité. Notre preuve fonctionne par séquence de jeux, qui permettent de modifier progressivement les réponses de $\mathcal{F}$ pour construire le simulateur $\mathcal{S}$ de

manière cohérente avec les réponses de l'oracle aléatoire $\mathcal{H}$.

La borne d'indifférenciabilité d'un oracle aléatoire de notre construction dépend de la probabilité d'occurrence de t-$\mathsf{Coll}(\ell, m)$, défini de la manière suivante. Soient $x_1, \ldots, x_\ell$ ℓ éléments tirés de manière indépendante et uniforme sur un ensemble E de cardinal m. Nous définissons

$$t\text{-}\mathsf{Coll}(\ell, m) = \text{vrai} \Leftrightarrow \exists I \subset \{1, \ldots, \ell\}, x \in E \text{ tels que } |I| = t \text{ et } \forall i \in I, x_i = x.$$

En d'autre termes, t-Coll caractérise l'existence d'un ensemble d'au moins t valeurs égales lors de ℓ tirages indépendants et uniformes sur un ensemble de m valeurs.

Théorème 2 *Considérons l'algorithme d'extension de domaine décrit en Figure 5.1, avec une fonction d'encodage quelconque fixée, et le simulateur $\mathcal{S}$ défini par la figure 5.5. L'avantage de tout attaquant $\mathcal{D}$ interagissant avec Σ ou Σ' et effectuant des requêtes à gauche et à droite d'un poids total au plus N satisfait l'inégalité*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{N(N+1)}{2} 2^{-(\ell_a+\ell_h)} + 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] .$$

5.3.2 Esquisse de la séquence de jeux

Notre preuve d'indifférenciabilité repose sur une séquence de jeux qui part de la construction $\mathcal{C}$ appliquée à $\mathcal{F}$ et qui consiste à construire progressivement le simulateur $\mathcal{S}$. Nous commençons par donner une brève description des étapes de la construction du simulateur. Certaines étapes correspondent à plusieurs jeux consécutifs, dans ce cas nous esquissons également les étapes de la preuve.

Jeu 0 Nous commençons avec le jeu d'origine. Dans ce jeu le distingueur interagit avec un système $\Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ qui représente notre construction $\mathcal{C}$ appliquée à une fonction aléatoire $\mathcal{F}$ [2].

Jeu 1 Un simulateur $\mathcal{S}_{1,2}$ est introduit dans le jeu à la place de la fonction aléatoire $\mathcal{F}$. Il se contente de transmettre les requêtes à $\mathcal{F}$ et de renvoyer les réponses à $\mathcal{D}$. Du point de vue de $\mathcal{D}$, ce jeu n'a aucune différence avec le précédent.

Jeux 2 à 4 Un programme appelé intercepteur $\mathcal{I}_{2,3,4}$ est ajouté au scénario. Il permet aux simulateurs intermédiaires $\mathcal{S}_i$ d'accéder aux requêtes de haché émises par $\mathcal{D}$. Son rôle est totalement transparent du point de vue de l'attaquant, il permet simplement à $\mathcal{S}_{4,5}$ d'identifier des évènements d'échec. Afin de simplifier l'écriture du simulateur, $\mathcal{I}_2$ décompose le message à hacher en blocs, et transmet ses requêtes à $\mathcal{S}_i$ bloc par bloc.

Jeu 5 Le scénario est identique, et le simulateur utilisé est $\mathcal{S}_{4,5}$, introduit au Jeu 4. Au lieu de transmettre les requêtes de haché à $\mathcal{S}_{4,5}$ en temps réel, l'intercepteur $\mathcal{I}_{5,6}$ stocke les messages à hacher et émet ses requêtes à $\mathcal{S}_{4,5}$ à la fin du jeu, c'est-à-dire après toutes les interactions faisant intervenir $\mathcal{D}$.

Jeu 6 L'oracle $\mathcal{F}$ est remplacé par un oracle de hachage $\mathcal{H}$. La construction $\mathcal{C}$ disparaît du jeu. Lorsque les requêtes reçues par $\mathcal{S}_6$ correspondent au traitement du dernier bloc de message au cours d'un calcul de haché, $\mathcal{S}_6$ génère sa réponse en questionnant l'oracle $\mathcal{H}$. Dans le cas contraire, $\mathcal{S}$ simule sa réponse de manière aléatoire. C'est dans la transition entre ces deux scénarios que réside la différence potentielle sur la vue de l'attaquant. Comme dans le

2. Nous identifions la construction $\mathcal{C}$ et le programme permettant de la calculer

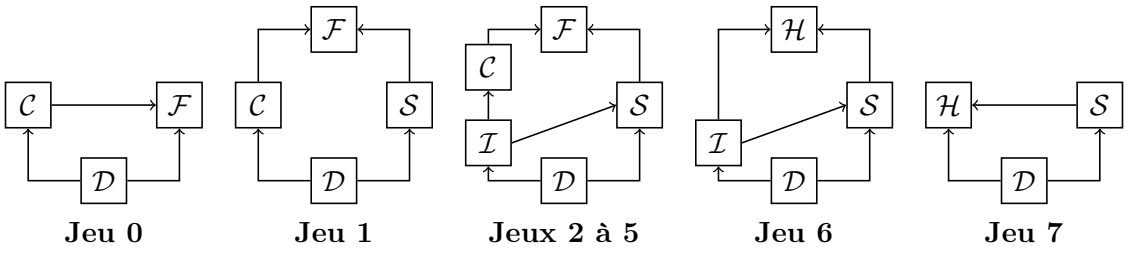


FIGURE 5.6 – Construction du simulateur $\mathcal{S}$.

cas précédent, $\mathcal{S}_6$ reconstruit à la fin du jeu les requêtes de haché émises par $\mathcal{D}$. La vue de l'attaquant n'est donc pas affectée par ces interactions qui se déroulent uniquement entre $\mathcal{S}_6$ et $\mathcal{I}_{5,6}$, l'intercepteur défini au Jeu 5.

Jeu 7 L'intercepteur est retiré du jeu. La vue de l'attaquant n'est pas modifiée par cette opération. Le scénario ainsi défini correspond au jeu final, où $\mathcal{D}$ interagit avec $\Sigma' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

Cette stratégie est résumée sur la Figure 5.6.

Lors de chaque transition, nous majorons l'écart des probabilités que $\mathcal{D}$ renvoie 1 lorsqu'il interagit avec le système avant et après la transition. Par la suite nous notons W_i l'évènement correspondant au fait que $\mathcal{D}$ réponde 1 lors du Jeu i.

Transition entre les Jeux 4 et 5. La distinction entre les Jeux 4 et 5 est motivée par la raison suivante. L'introduction de l'intercepteur $\mathcal{I}$ permet au simulateur d'identifier des évènements indésirables impliquant des requêtes de haché émises par $\mathcal{D}$. Dans le Jeu 4, $\mathcal{S}$ reçoit ces requêtes en temps réel. L'attaquant ne dispose pas d'information dont ne dispose pas $\mathcal{S}$ et qui lui permettrait de provoquer de tels évènements. Leur probabilité d'occurrence peut alors être bornée en connaissant uniquement l'état de $\mathcal{S}$.

Dans le schéma 5 (et dans les suivants), les requêtes de haché sont obtenues à la fin du jeu. Ceci permet de garantir que la vue de l'attaquant est identique à celle du jeu précédent, avec des cas d'échec équivalents. Ce faisant, nous nous assurons également que $\mathcal{S}$ ne dispose d'aucune information venant de $\mathcal{I}$ au moment de répondre à une requête à droite venant de l'attaquant.

5.3.3 Séquence de jeux.

Jeu 0. Ce scénario correspond au jeu d'origine dans lequel $\mathcal{D}$ interagit avec $\mathcal{C}^{\mathcal{F}}$ et la fonction aléatoire $\mathcal{F}$. Par définition du Jeu 0, nous avons :

$$\Pr[W_0] = \Pr\left[\mathcal{D}^{\Sigma} = 1 \mid \Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})\right] .$$

Jeu 1. Un simulateur passif $\mathcal{S}_{1,2}$ est introduit dans le jeu. Ce simulateur reçoit des requêtes pour des valeurs de $\mathcal{F}$, venant de $\mathcal{D}$. Il transmet ces requêtes à l'oracle $\mathcal{F}$ et transmet les réponses correspondantes à $\mathcal{D}$. Le simulateur $\mathcal{S}_{1,2}$ est identique pour les Jeux 1 et 2. Il construit deux graphes

$$\mathcal{G}_{\mathcal{I}} = (X_{\mathcal{I}}, E_{\mathcal{I}}) , \qquad \mathcal{G}_{\mathcal{D}} = (X_{\mathcal{D}}, E_{\mathcal{D}}) ,$$

en mémorisant passivement les entrées et sorties de $\mathcal{F}$ définies par les requêtes émises respectivement par $\mathcal{I}$ et $\mathcal{D}$. Dans le Jeu 1, l'intercepteur n'a pas encore été défini, donc $\mathcal{G}_{\mathcal{I}}$ reste vide. $\mathcal{S}_{1,2}$ est

représenté sur la Figure 5.7. L'action de $\mathcal{S}_{1,2}$ ne modifie pas la vue de $\mathcal{D}$, ce qui implique que

$$\Pr[W_1] = \Pr[W_0]. \tag{5.3}$$

Initialisation de $\mathcal{S}_{1,2}$

1. définir $X_{\mathcal{I}} = \{x^0\}$ et $E_{\mathcal{I}} = \emptyset$
2. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O} =$ soit $\mathcal{I}$ soit $\mathcal{D}$

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. appeler $\mathcal{F}$ pour obtenir $(A', B') = \mathcal{F}(M, A, B, C)$
3. définir $y = (A', B', C)$
4. ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{O}}$
5. renvoyer (A', B') à $\mathcal{O}$

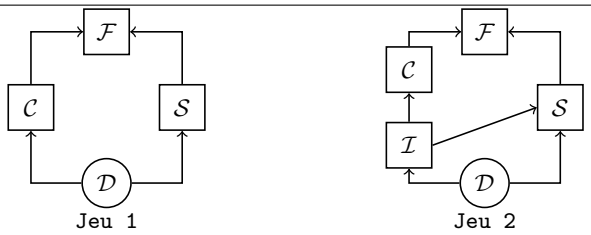


FIGURE 5.7 – Simulateur $\mathcal{S}_{1,2}$ pour $\mathcal{F}$ dans les Jeux 1 et 2.

Jeu 2. Sans modifier le simulateur, nous lui donnons maintenant accès aux requêtes de haché. Cet accès n'est jamais nécessaire au simulateur du point de vue fonctionnel, il lui permet cependant de caractériser des évènements d'échec. Nous modélisons ceci à l'aide d'un intercepteur $\mathcal{I}_{2,3,4}$, qui intercepte les requêtes de haché entre $\mathcal{D}$ et $\mathcal{C}$, les décompose et les transmet à $\mathcal{S}_{1,2}$. L'intercepteur $\mathcal{I}_{2,3,4}$ est représenté en Figure 5.8

Comme dans le jeu précédent, la vue de l'attaquant n'est pas modifiée, et

$$\Pr[W_2] = \Pr[W_1]. \tag{5.4}$$

Jeu 3. Nous effectuons maintenant une modification mineure sur $\mathcal{S}_{1,2}$, qui prépare le remplacement de l'accès à un oracle $\mathcal{F}$ par une simulation. Dans le jeu final, si $\mathcal{S}$ reçoit une requête de $\mathcal{D}$ pour une valeur de $\mathcal{F}_{C,M}(A, B)$ qu'il a déjà précédemment définie, il doit répondre avec cette valeur. Nous modifions maintenant $\mathcal{S}_1$ pour que lorsqu'il connaît déjà la réponse à une requête (M, A, B, C) à $\mathcal{F}$, il renvoie cette valeur sans questionner $\mathcal{F}$. En procédant ainsi, nous ne modifions pas les réponses renvoyées par le simulateur à $\mathcal{I}_{2,3,4}$ ou à $\mathcal{D}$, puisque elles restent cohérentes avec les valeurs définies par l'oracle $\mathcal{F}$.

Traitement d'une requête à $\mathcal{H}$

Entrée : $\mu \in \{0,1\}^*$

Sortie : $h = \mathcal{H}(\mu)$

1. calculer $(M^1 || \dots || M^\ell) = \mathsf{pad}(\mu)$
2. définir $(A^0, B^0, C^0) = x^0$
3. pour i de 1 à ℓ
   (a) définir $(a, b, C^i) = \mathsf{Insert}[M^i](A^{i-1}, B^{i-1}, C^{i-1})$
   (b) appeler $\mathcal{S}$ pour obtenir $(A^i, B^i) = \mathcal{F}_{M^i, C^i}(a, b)$
4. appeler $\mathcal{C}$ pour obtenir $h = \mathcal{H}(\mu)$
5. renvoyer h à $\mathcal{D}$

FIGURE 5.8 – Intercepteur $\mathcal{I}_{2,3,4}$ dans les Jeux 2 à 4.

Initialisation de $\mathcal{S}_3$

1. définir $X_{\mathcal{I}} = \{x^0\}$ et $E_{\mathcal{I}} = \emptyset$
2. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O}$ = soit $\mathcal{I}$ soit $\mathcal{D}$

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \xrightarrow{M} y$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$
   (a) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{O}}$
   (b) renvoyer (A', B') tel que $y = (A', B', C)$ à $\mathcal{O}$
3. sinon
   (a) appeler $\mathcal{F}$ pour obtenir $(A', B') = \mathcal{F}(M, A, B, C)$
   (b) définir $y = (A', B', C)$
   (c) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{O}}$
   (d) renvoyer (A', B') à $\mathcal{O}$

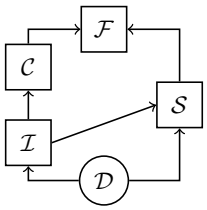


FIGURE 5.9 – Simulateur $\mathcal{S}_3$ pour $\mathcal{F}$ dans le Jeu 3.

Le simulateur $\mathcal{S}_3$ est maintenant décrit sur la Figure 5.9.

Dans le Jeu 3, la vue de l'attaquant n'est pas modifiée. Nous avons donc

$$\Pr[W_3] = \Pr[W_2]. \tag{5.5}$$

Jeu 4. Nous introduisons maintenant dans le simulateur la détection de cas d'échec. Les cas d'échec doivent couvrir les deux évènements suivants sur le graphe $\mathcal{G} = \mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$.

Collision interne : un nœud x du graphe $\mathcal{G}$ a deux chemins utiles distincts μ et μ'.

Prédéfinition d'un haché : $\langle M_1|\dots|M_\ell\rangle = \mathsf{pad}(\mu)$ est un chemin complet, et le graphe $\mathcal{G}$ contient ℓ arcs $x^i \overset{M_{i+1}}{\to} x^{i+1}$, le haché de μ est prédéfini si $x^{\ell-1} \overset{M_\ell}{\to} x^\ell$ n'est pas le dernier de ces arcs à être inséré au graphe.

Nous définissons maintenant les cas d'échec suivants.

Echec_1**.** Cet évènement peut être identifié au moment d'insérer un arc $x \overset{M}{\to} y$ dans $E_{\mathcal{D}}$ tel que x ait un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et tel que $\mu||M$ est un chemin utile. Si y a déjà un chemin utile dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$, l'insertion de $x \overset{M}{\to} y$ crée une collision interne dans le graphe, et Echec_1 est détecté.

Echec_2**.** Un autre évènement indésirable peut être identifié au moment d'insérer un arc $x \overset{M}{\to} y$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ tel que x ait un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$. S'il existe un arc $y \overset{M'}{\to} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ tel que $\mu||M||M'$ soit un chemin utile, l'insertion de $x \overset{M}{\to} y$ peut entraîner une collision interne ou la prédéfinition d'un haché. Dans ce cas, Echec_2 est détecté.

Attaques par extension de longueur. D'après la Figure 5.6 et la description de haut niveau de la séquence de jeux, la prochaine étape consiste à modifier l'ordre dans lequel $\mathcal{S}$ reçoit les requêtes. Il est donc possible qu'une séquence de requêtes provoquant la prédéfinition d'un haché au Jeu 5 ne le fasse pas au Jeu 4. Ce type d'évènement est généralement désigné par le terme d'*attaque par extension de longueur*, et correspond typiquement au scénario suivant. L'attaquant choisit deux messages μ et μ' tels qu'il existe un bloc M tel que $\mathsf{pad}(\mu') = \mathsf{pad}(\mu)||M$. Il émet ensuite une requête gauche pour obtenir une valeur de haché $\mathcal{H}(\mu)$, et reconstruit la partie C de l'état interne après le haché de μ par des requêtes droites. Cela nécessite de calculer la construction $\mathcal{C}^{\mathcal{F}}$ appliquée à $\mathsf{pad}(\mu)$, à l'exception du dernier bloc. Pour toutes les valeurs de A possibles, l'attaquant émet ensuite une requête droite de la forme $(M, \mathsf{Insert}[M](A, \mathcal{H}(\mu), C))$. Enfin, l'attaquant termine de reconstruire le haché de μ' à l'aide de la construction, ce qui provoque la prédéfinition d'une valeur de haché.

Echec_3**.** Afin d'anticiper les attaques par extension de longueur, nous définissons maintenant l'évènement Echec_3. Cet évènement est identifié lorsque $\mathcal{S}$ reçoit une requête (M, A, B, C) provenant de $\mathcal{D}$, telle que $x = \mathsf{Insert}^{-1}[M](A, B, C)$ n'a pas de chemin utile dans $\mathcal{G}_{\mathcal{D}}$ mais que $\mathcal{F}_{C,M}(A, B)$ a déjà été défini dans $E_{\mathcal{I}}$.

Le nouveau simulateur $\mathcal{S}_{4,5}$ est maintenant décrit sur la Figure 5.10.

Dans le Jeu 4, comme dans les jeux précédents, l'intervention de $\mathcal{I}$ et du simulateur est transparente du point de vue de l'attaquant. Nous avons donc

$$\Pr[W_4] = \Pr[W_3]. \tag{5.6}$$

Initialisation de $\mathcal{S}_{4,5}$

1. définir $X_{\mathcal{I}} = \{x^0\}$ et $E_{\mathcal{I}} = \emptyset$
2. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O} =$ soit $\mathcal{I}$ soit $\mathcal{D}$

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$ et s'il existe un arc $x \stackrel{M}{\to} y$ dans $E_{\mathcal{I}}$
   (a) si l'origine $\mathcal{O} = \mathcal{D}$, NoteEchec($\mathsf{Echec}_3$)
3. s'il existe un arc $x \stackrel{M}{\to} y$ dans E
   (a) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \stackrel{M}{\to} y$ à $E_{\mathcal{O}}$
   (b) renvoyer (A', B') tel que $y = (A', B', C)$ à $\mathcal{O}$
4. sinon
   (a) appeler $\mathcal{F}$ pour obtenir $(A', B') = \mathcal{F}(M, A, B, C)$
   (b) définir $y = (A', B', C)$
   (c) si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et $\mu||M$ est un chemin utile
      i. si y a un chemin utile μ' dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$, NoteEchec($\mathsf{Echec}_1$)
      ii. s'il existe un arc $y \stackrel{M'}{\to} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ et si $\mu||M||M'$ est un chemin utile, NoteEchec(Echec_2)
   (d) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \stackrel{M}{\to} y$ à $E_{\mathcal{O}}$
   (e) renvoyer (A', B') à $\mathcal{O}$

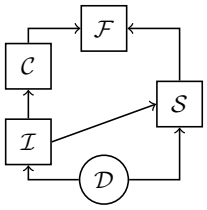


FIGURE 5.10 – Simulateur $\mathcal{S}_{4,5}$ pour $\mathcal{F}$ dans les Jeux 4 et 5.

Nous cherchons maintenant à nous assurer que les évènements d'échec définis ci-dessus permettent d'éviter les collisions internes et la prédéfinition de hachés. Pour cela, nous pouvons maintenant montrer la propriété suivante sur le graphe $\mathcal{G}_\mathcal{I} \cup \mathcal{G}_\mathcal{D}$.

Propriété 1 *Soit $\mathcal{G}_p$ le graphe $\mathcal{G}_\mathcal{I} \cup \mathcal{G}_\mathcal{D}$ construit dans le Jeu 4 par le simulateur $\mathcal{S}_{4,5}$ décrit en Figure 5.10 après la réponse à p requêtes. Soit $x \neq x^0$ un nœud ayant un chemin utile dans le graphe final $\mathcal{G}_f$, et $y \xrightarrow{M} x$ le dernier arc de ce chemin. Supposons que Drapeau = Succès à la fin du jeu.*

Sous ces hypothèses, x a un chemin utile dans $\mathcal{G}_q$, où $q \in \{1, \ldots, \ell\}$ est l'indice de la requête au cours de laquelle $y \xrightarrow{M} x$ est inséré dans $\mathcal{G}_\mathcal{I} \cup \mathcal{G}_\mathcal{D}$.

Démonstration : Soit $x \in X$ un nœud avec un chemin utile $\mu = \langle M^1, \ldots, M^k \rangle$ dans $\mathcal{G}$. Par définition, il existe une suite d'états $x^1, \ldots, x^{k-1}$ tels que

$$x^0 \xrightarrow{M^1} x^1 \ldots x^{k-1} \xrightarrow{M^k} x^k = x.$$

Soit q' le plus petit entier tel que ce chemin soit complètement défini après la $q'^{ème}$ requête. Cela signifie qu'il existe un entier i, $1 \leq i \leq k$, tel que l'arc $x^{i-1} \xrightarrow{M^i} x^i$ est ajouté au graphe au cours du traitement de la $q'^{ème}$ requête, alors que tous les autres arcs du chemin sont dans $E(q')$ (l'ensemble des arcs du graphe avant la requête q'). q' étant minimal, $x^{i-1} \xrightarrow{M^i} x^i$ n'est pas dans $E(q')$.
En raisonnant par l'absurde, nous supposons maintenant que $i < k$. $x^i \xrightarrow{M^{i+1}} x^{i+1}$ est alors dans E avant la requête q'. x^{i-1} a un chemin utile $\langle M^1, \ldots, M^{i-1} \rangle$, et $\langle M^1, \ldots, M^i \rangle$ et $\langle M^1, \ldots, M^{i+1} \rangle$ sont des chemins utiles. Echec_2 est alors noté lors de la $q'^{ème}$ requête, ce qui contredit une des hypothèses de l'énoncé. □

Le graphe du simulateur du Jeu 4 vérifiant la propriété 1, nous pouvons montrer qu'il satisfait également la propriété suivante :

Propriété 2 *Soient (M, A, B, C) la $q^{ème}$ requête reçue par $\mathcal{S}_{4,5}$ dans le Jeu 4 et $x = \mathsf{Insert}^{-1}[M](A, B, C)$. Si Drapeau = Succès à la fin du jeu, l'une des deux propositions suivantes est vraie :*

- *x a un unique chemin utile μ dans le graphe $\mathcal{G}_\mathcal{D} \cup \mathcal{G}_\mathcal{I}$ avant la $q^{ème}$ requête et après tous les appels suivants à $\mathcal{S}$*
- *x n'a pas de chemin utile dans le graphe $\mathcal{G}_\mathcal{D} \cup \mathcal{G}_\mathcal{I}$ avant la $q^{ème}$ requête ni lors d'aucun appel suivant à $\mathcal{S}$.*

Démonstration : Nous traitons d'abord le premier cas en montrant par l'absurde que x ne peut jamais avoir deux chemins utiles différents dans le graphe $\mathcal{G} = \mathcal{G}_\mathcal{I} \cup \mathcal{G}_\mathcal{D}$. Soit x un nœud avec deux chemins différents dans $\mathcal{G}$, que l'on note $\mu_1 = \langle M^0, \ldots, M^k \rangle$ and $\mu_2 = \langle P^0, \ldots, P^\ell \rangle$, avec $\ell \geq k$ Alors il existe deux séquences d'états $u^1, \ldots, u^k$ et $v^1, \ldots, v^\ell$ tels que

$$x^0 \xrightarrow{M^0} u^1 \ldots u^k \xrightarrow{M^k} x$$

et

$$x^0 \xrightarrow{P^0} v^1 \ldots v^\ell \xrightarrow{P^\ell} x.$$

Soit $i \geq 0$ le plus petit entier tel que $u^{k-i} \neq v^{\ell-i}$, s'il existe. Soit $x' = u^{k-i+1} = v^{\ell-i+1}$ lorsque $i > 0$ et $x' = x$ sinon. Alors x' a deux chemins dans le graphe qui terminent par des arcs différents

$$u_{k-i} \xrightarrow{M^{k-i}} x' \qquad \text{et} \qquad v^{\ell-i} \xrightarrow{P^{\ell-i}} x' .$$

Nous déduisons de la propriété 1 que les deux arcs finaux et x' ont été insérés lors du même appel à $\mathcal{S}_{4,5}$, ce qui est impossible puisqu'au plus un arc est inséré par requête.
Si une telle valeur de i n'existe pas, les deux chemins étant distincts, ils ne peuvent être de même longueur. Nous avons donc $v^{\ell-k} = x^0$. Lors de l'insertion de l'arc $v^{\ell-k-1} \overset{P_{\ell-k-1}}{\rightarrow} v^{\ell-k}$, d'après la propriété 1, le chemin partiel $\langle P^0, \ldots, P^{\ell-k} \rangle$ est complètement défini, or comme $v^{\ell-k} = x^0$, Echec_1 se produit.
Le deuxième cas est l'expression de la contraposée de la propriété 1. □

Jeu 5. Nous modifions maintenant le comportement de $\mathcal{I}_{2,3,4}$. Au lieu de transmettre les requêtes de haché à $\mathcal{S}_{4,5}$ dès leur réception, l'intercepteur les stocke, et ne les transmet à $\mathcal{S}_{4,5}$ qu'à la fin du jeu (c'est-à-dire après la réponse de $\mathcal{D}$ sur le scénario dans lequel il se trouve). Le simulateur du Jeu 5 est identique au simulateur $\mathcal{S}_{4,5}$ du Jeu 4. Notons toutefois que l'évènement Echec_3 ne peut pas se produire, les requêtes de haché étant traitées après les requêtes à $\mathcal{F}$. Le nouvel intercepteur $\mathcal{I}_{5,6}$ est décrit en Figure 6.14. Là encore, la vue de l'attaquant n'est pas modifiée, d'où

$$\Pr[W_5] = \Pr[W_4]. \tag{5.7}$$

Initialisation de $\mathcal{I}_{5,6}$

1. définir $\mathcal{L}_{\mathcal{H}} = \emptyset$
2. définir $(A^0, B^0, C^0) = x^0$

Traitement d'une requête à $\mathcal{H}$

Entrée : $\mu \in \{0,1\}^*$

Sortie : $h = \mathcal{H}(\mu)$

1. ajouter μ à $\mathcal{L}_{\mathcal{H}}$
2. appeler $\mathcal{C}$ (Jeu 5) ou $\mathcal{H}$ (Jeu 6) pour obtenir $h = \mathcal{H}(\mu)$
3. renvoyer h à $\mathcal{D}$

Simulation des requêtes à $\mathcal{H}$ (à la fin du jeu)

1. pour tout μ de $\mathcal{L}_{\mathcal{H}}$
   (a) calculer $(M^1 || \ldots || M^\ell) = \mathsf{pad}(\mu)$
   (b) pour i de 1 à ℓ
      i. définir $(a, b, C^i) = \mathsf{Insert}[M^i](A^{i-1}, B^{i-1}, C^{i-1})$
      ii. appeler $\mathcal{S}$ pour obtenir $(A^i, B^i) = \mathcal{F}_{M^i, C^i}(a, b)$

FIGURE 5.11 – Intercepteur $\mathcal{I}_{5,6}$ dans les Jeux 5 et 6.

Nous montrons maintenant que les cas d'échec des Jeux 4 et 5 sont équivalents.

Propriété 3 *Pour un même aléa de $\mathcal{D}$ et une même fonction $\mathcal{F}$, les graphes construits par $\mathcal{S}_{4,5}$ à la fin des Jeux 4 et 5 sont identiques, et $\mathcal{S}_{4,5}$ note un cas d'échec dans le Jeu 4 si et seulement*

si $\mathcal{S}_{4,5}$ note un cas d'échec dans le Jeu 5. En d'autres termes, $\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4] \vee \mathsf{Echec}_3[4] = \mathsf{Echec}_1[5] \vee \mathsf{Echec}_2[5]$.

Démonstration : A la fin du Jeu 4 comme du Jeu 5, le graphe contient les arcs correspondant aux requêtes à $\mathcal{F}$ émises par $\mathcal{D}$, et à la construction des requêtes de haché. Pour une même fonction $\mathcal{F}$, les réponses reçues par l'attaquant sont identiques. Pour un aléa identique de l'attaquant aux Jeux 4 et 5, l'attaquant émet alors la même séquence de requêtes. Les graphes résultant sont donc identiques.

Cas d'échec au cours du Jeu 4. Si Echec_1 se produit au cours du Jeu 4, alors un nœud y a deux chemins utiles différents dans le graphe, ce qui implique la définition d'un cas d'échec au cours du Jeu 5, d'après la propriété 2.
Supposons maintenant que Echec_2 se produit au cours du Jeu 4. L'un de ces cas de figure se produit alors.

- Si Echec_2 se produit au cours du traitement d'une requête émise par $\mathcal{I}$, lorsque cette même requête est traitée au cours du Jeu 5, le graphe $\mathcal{G}$ contient au moins les mêmes arcs qu'au Jeu 4, et Echec_2 se produit également.
- Si Echec_2 se produit au cours du traitement d'une requête émise par $\mathcal{D}$, il existe un arc $y \xrightarrow{M} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$. $E_{\mathcal{I}}$ ne contient que des arcs construits au cours de calculs de hachés, donc si $y \xrightarrow{M} z \in E_{\mathcal{I}}$, y a un chemin utile. y a deux chemins utiles différents dans le graphe, ce qui provoque un cas d'échec dans le Jeu 5. Si $y \xrightarrow{M} z \in E_{\mathcal{D}}$, alors cet arc est également dans $E_{\mathcal{D}}$ au cours du traitement de la même requête dans le Jeu 5, et Echec_2 se produit.

Supposons que Echec_3 se produit au cours du Jeu 4 : $\mathcal{D}$ émet une requête (M, A, B, C) telle que $x = \mathsf{Insert}^{-1}[M](A, B, C)$ a un chemin utile μ dans $E_{\mathcal{I}}$ mais pas dans $E_{\mathcal{D}}$, et $\mu||M$ est un chemin utile. Soit $z \xrightarrow{M^*} x$ le dernier arc de ce chemin. Dans le Jeu 5, cet arc est inséré :

- soit à la fin du jeu suite à une requête de hachage, ce qui provoque Echec_2 ;
- soit à la suite d'une requête à $\mathcal{F}$ alors que z a un chemin dans le graphe, ce qui provoque Echec_2 ;
- soit à la suite d'une requête à $\mathcal{F}$ alors que z n'a pas de chemin dans le graphe. A la fin du jeu, z a un chemin dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ ce qui a provoqué un cas d'échec.

Un cas d'échec au Jeu 4 implique donc un cas d'échec au Jeu 5.

Cas d'échec au cours du Jeu 5. Supposons maintenant que Echec_1 se produit au cours du Jeu 5. Dans ce cas un nœud y a deux chemins utiles différents dans le graphe, ce qui implique qu'un cas d'échec se produit également au cours du Jeu 4.
Supposons que Echec_2 est détecté au Jeu 5 lors de l'insertion d'un arc $x \xrightarrow{M} y$. Les raisonnements permettant de montrer l'occurrence d'un cas d'échec au Jeu 4 sont plus complexes, et nécessitent la définitions de sous cas de figure.

1. Si Echec_2 se produit au moment d'insérer $x \xrightarrow{M} y$ au cours du traitement d'une requête venant de $\mathcal{D}$, il existe un arc $y \xrightarrow{M'} z$ dans $E_{\mathcal{D}}$ ($E_{\mathcal{I}}$ étant vide à ce moment) et x a un chemin utile dans $\mathcal{G}_{\mathcal{D}}$.
   (a) Au cours du traitement de la même requête dans le Jeu 4, si $x \xrightarrow{M} y$ n'est pas dans $E_{\mathcal{I}}$, $y \xrightarrow{M'} z$ est également dans $E_{\mathcal{D}}$ et Echec_2 se produit.
   (b) Si $x \xrightarrow{M} y$ est dans $E_{\mathcal{I}}$ et a y été introduit avant l'introduction de $y \xrightarrow{M} z$ dans $E_{\mathcal{D}}$:

i. Lorsque $y \xrightarrow{M} z$ est introduit dans $E_{\mathcal{D}}$, si y a déjà un chemin dans $\mathcal{G}_{\mathcal{D}}$, y a un chemin différent dans $\mathcal{G}_{\mathcal{I}}$ ($x \xrightarrow{M} y$ n'étant pas dans $\mathcal{G}_{\mathcal{D}}$ à ce moment). Cela implique l'occurrence d'un cas d'échec précédemment dans le Jeu 4.

ii. Si y n'a pas de chemin dans $\mathcal{G}_{\mathcal{D}}$, Echec_3 se produit.

(c) Si $x \xrightarrow{M} y$ est dans $E_{\mathcal{I}}$ et y a été introduit après l'introduction de $y \xrightarrow{M} z$ dans $E_{\mathcal{D}}$, Echec_2 se produit au moment de l'insertion de $x \xrightarrow{M} y$ dans $E_{\mathcal{I}}$ au Jeu 4.

2. Si Echec_2 se produit au cours du traitement d'une requête venant de $\mathcal{I}$, il existe un arc $y \xrightarrow{M} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$.

(a) Si $y \xrightarrow{M} z$ est dans $E_{\mathcal{I}}$, alors y a deux chemins différents dans $\mathcal{G}_{\mathcal{I}}$ et Echec_2 se produit lors du traitement de la même requête au Jeu 4.

(b) Si $y \xrightarrow{M} z$ est dans $E_{\mathcal{D}}$, on se ramène aux cas 1.(b). et 1.(c)..

D'après l'ordre du traitement des requêtes au Jeu 5, Echec_3 ne peut pas se produire. Un cas d'échec au Jeu 5 implique donc un cas d'échec au Jeu 4. Ceci achève la démonstration. □

Jeu 6. Nous remplaçons maintenant l'accès à $\mathcal{F}$ par un accès à l'oracle aléatoire $\mathcal{H}$. De ce fait, la construction $\mathcal{C}$ disparaît du jeu. $\mathcal{S}_6$ doit donc simuler lui-même les sorties de $\mathcal{F}$, en appelant l'oracle aléatoire pour définir les sorties de haché. Le simulateur correspondant est décrit en Figure 5.12. Notons que comme dans le Jeu 5, lorsqu'une requête provient de $\mathcal{D}$, $E_{\mathcal{I}}$ est vide.

Dans le Jeu 5 comme dans le Jeu 6, en l'absence de cas d'échec, les réponses aux requêtes de haché sont indépendantes et uniformément distribuées (si toutes les requêtes concernent des messages deux à deux distincts). Les réponses aux requêtes à $\mathcal{F}$ sont cohérentes avec la construction de hachage, et les parties ne correspondant pas à des valeurs de haché sont tirées de manière indépendante et uniforme. En l'absence de cas d'échec, les vues de l'attaquant au Jeu 5 et au Jeu 6 sont identiques, d'où

$$\Pr\left[W_6 \wedge \overline{\mathsf{Echec}_1} \wedge \overline{\mathsf{Echec}_2}\right] = \Pr\left[W_5 \wedge \overline{\mathsf{Echec}_1} \wedge \overline{\mathsf{Echec}_2}\right]. \tag{5.8}$$

Jeu 7. L'intercepteur $\mathcal{I}$ est maintenant supprimé du Jeu. Son action étant transparente du point de vue de $\mathcal{D}$, sa vue n'est pas modifiée. D'autre part, les identifications de cas d'échec du simulateur sont supprimées. Là encore, cela ne modifie pas la vue de l'attaquant. Nous avons donc :

$$\Pr[W_7] = \Pr[W_6]. \tag{5.9}$$

Le scénario du Jeu 7 correspond au Jeu final, dans lequel $\mathcal{D}$ interagit avec le système $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. Le simulateur correspondant à ce Jeu est décrit en Figure 5.5.

5.3.4 Majoration de l'avantage de l'attaquant.

Nous utilisons maintenant les Équations (5.3) à (5.9) pour borner l'avantage du distingueur. La seule différence dans la vue du distingueur intervient entre les Jeux 5 et 6.

D'après le lemme de différence,

$$|\Pr[W_6] - \Pr[W_5]| \leq \max(\Pr[\mathsf{Echec}_1[6] \vee \mathsf{Echec}_2[6]], \Pr[\mathsf{Echec}_1[5] \vee \mathsf{Echec}_2[5]]).$$

Initialisation de $\mathcal{S}_6$

1. définir $X_{\mathcal{I}} = \{x^0\}$ et $E_{\mathcal{I}} = \emptyset$
2. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O}$ = soit $\mathcal{I}$ soit $\mathcal{D}$

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \overset{M}{\to} y$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$
   (a) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \overset{M}{\to} y$ à $E_{\mathcal{O}}$
   (b) renvoyer (A', B') tel que $y = (A', B', C)$ à $\mathcal{O}$
3. sinon
   (a) si x a un chemin μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et si $\mu || M$ est un chemin complet
      i. si μ n'est pas unique, arrêter $\mathcal{S}$
      ii. calculer $\mathcal{M} = \mathsf{unpad}(\mu || M)$
      iii. appeler $\mathcal{H}$ pour obtenir $B' = \mathcal{H}(\mathcal{M})$
      iv. tirer $A' \leftarrow \{0,1\}^{\ell_a}$
   (b) sinon
      i. tirer $(A', B') \leftarrow \{0,1\}^{\ell_a + \ell_h}$
   (c) définir $y = (A', B', C)$
   (d) si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et $\mu || M$ est un chemin utile
      i. s'il existe un arc $y \overset{M'}{\to} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ et si $\mu || M || M'$ est un chemin utile, $\mathsf{NoteEchec}(\mathsf{Echec}_1)$
      ii. si y a un chemin utile μ' dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$, $\mathsf{NoteEchec}(\mathsf{Echec}_2)$
   (e) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \overset{M}{\to} y$ à $E_{\mathcal{O}}$
   (f) renvoyer (A', B') à $\mathcal{O}$

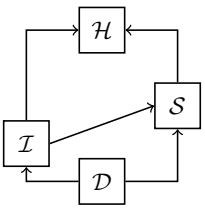


FIGURE 5.12 – Simulateur $\mathcal{S}_6$ pour $\mathcal{F}$ dans le Jeu 6.

Tant qu'aucun cas d'échec n'est identifié au cours des Jeux 5 et 6, les distributions des valeurs insérées au graphe sont identiques. Nous pouvons en déduire que $\Pr[\mathsf{Echec}_1[5] \vee \mathsf{Echec}_2[5]] = \Pr[\mathsf{Echec}_1[6] \vee \mathsf{Echec}_2[6]]$. D'autre part, nous avons montré que

$$\Pr[\mathsf{Echec}_1[5] \vee \mathsf{Echec}_2[5]] = \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]] + \Pr[\mathsf{Echec}_3[4]] \ .$$

Nous pouvons en déduire :

$$|\Pr[W_6] - \Pr[W_5]| \leq \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]] + \Pr[\mathsf{Echec}_3[4]] \ .$$

D'après les Équations (5.3) à (5.8),

$$\Pr[W_5] = \Pr[W_0] \text{ et } \Pr[W_7] = \Pr[W_6] \ .$$

Nous en déduisons :

$$\begin{aligned} \mathsf{Adv}\mathcal{D} &= |\Pr[W_7] - \Pr[W_0]| \\ &= |\Pr[W_6] - \Pr[W_5]| \\ &\leq \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]] + \Pr[\mathsf{Echec}_3[4]] \ . \end{aligned}$$

Nous utilisons ensuite les majorations suivantes, démontrées dans les lemmes 10 et 12 en Annexe A.

$$\begin{aligned} \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]] &\leq \frac{N(N+1)}{2} 2^{-(\ell_a+\ell_h)} \\ \Pr[\mathsf{Echec}_3[4]] &\leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr[t\text{-}\mathsf{Coll}] \end{aligned}$$

De ces relations nous déduisons

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{N(N+1)}{2} 2^{-(\ell_a+\ell_h)} + 2^{-\ell_a} N \sum_{t=1}^{N} \Pr[t\text{-}\mathsf{Coll}] \,,$$

ce qui prouve le théorème 2.

5.3.5 Évaluation de l'avantage de l'attaquant en fonction du nombre de requêtes.

En tenant compte du lemme 13 montré en Annexe A, nous montrons le théorème suivant, concernant l'indifférenciabilité de notre construction lorsque la primitive interne est une fonction aléatoire $\mathcal{F}$.

Théorème 3 *Considérons la construction générale d'extension de domaine représenté en Figure 5.1 et le simulateur $\mathcal{S}$ défini en Figure 5.5. Pour tout attaquant $\mathcal{D}$ interagissant avec Σ ou Σ' et totalisant au plus N requêtes à $\mathcal{F}$:*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(3+2e)N(N+1)}{2} 2^{-(\ell_a+\ell_h)} + \ell_h 2^{-(\ell_a-1)} N \ . \tag{5.10}$$

De plus, lorsque $\ell_h > \ell_a$ *:*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{N(N+1)}{2}2^{-(\ell_a+\ell_h)} + 2^{-\ell_a}N\left\lceil\frac{\ell_h+\ell_a-\log_2(\sqrt{2\pi})}{\ell_h-\ell_a-\log_2 e}\right\rceil . \tag{5.11}$$

L'avantage de $\mathcal{D}$ *est alors majoré par la plus petite des deux bornes* (5.10) *et* (5.11)*.*

Démonstration : Les bornes (5.10) et (5.11) se déduisent immédiatement du lemme 13 permettant d'évaluer $\sum_{t=1}^{N}\Pr[t\text{-}\mathsf{Coll}]$, avec $k=\ell_h$ et $N \leq 2^{\ell_a}$. □

Notons que de manière générale, la borne de sécurité ci-dessus ne dépend pas de la taille ℓ_c du paramètre C. En réalité, la définition d'un tel paramètre apporte à la sécurité de la construction lorsque des tours à blanc sont combinés avec une discontinuité de la fonction de compression. Ce cas de figure est étudié en Section 5.5.

5.3.6 Application à Chop-MD

Considérons maintenant l'algorithme d'extension de domaine Chop-MD défini par Coron *et al.* [CDMP05]. Nous montrons le corollaire suivant au théorème 3.

Corollaire 1 *En instanciant* $\mathcal{F}$ *avec une fonction aléatoire, la construction générique représentée en Figure 5.1 avec* $\ell_c = 0$ *et* $\mathsf{Insert}[M] = Id$*. correspond à la construction Chop-MD avec une variable de chaînage de* n *bits et des empreintes de* ℓ_h *bits. Pour tout distingueur* $\mathcal{D}$ *émettant au plus* N *appels à* $\mathcal{F}$*,*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(3+2e)N(N+1)}{2}2^{-n} + \ell_h 2^{-(n-\ell_h-1)}N . \tag{5.12}$$

De plus, lorsque $\ell_h > n/2$*,*

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-n}\frac{N(N+1)}{2} + 2^{-(n-\ell_h)}N\left\lceil\frac{n-\log_2(\sqrt{2\pi})}{2\ell_h-n-\log_2 e}\right\rceil . \tag{5.13}$$

Une borne supérieure sur $\mathsf{Adv}(\mathcal{D})$ *est alors donnée par la plus petite des bornes des équations* (5.12),(5.13)*.*

Notre résultat peut être comparé à la meilleure borne d'indifférenciabilité précédemment connue pour Chop-MD, due à Chang et Nandi [CN08] et qui établit :

$$\mathsf{Adv}(\mathcal{D}) \leq 2\cdot 2^{-n}N^2 + (3\ell_h+1)2^{-(n-\ell_h)}N + 2^{-(\ell_h-1)}N .$$

Cette borne n'a de sens que pour $\ell_h \geq n/2$, c'est-à-dire lorsqu'au plus la moitié des bits de la variable de chaînage sont tronqués. Notre résultat montre une borne généralisée pour tous les choix de $\ell_h \in [0,n]$, et améliore légèrement le résultat de Chang et Nandi lorsque $\ell_h \geq n/2$.

5.3.7 Borne d'indifférenciabilité lorsque l'encodage est sans préfixe

Nous nous concentrons maintenant sur la variante de notre algorithme d'extension de domaine lorsqu'une fonction d'encodage sans préfixe est utilisée. Cette idée a été proposée par Coron *et al.*. L'utilisation conjointe de Chop-MD et d'une fonction d'encodage sans préfixe a été proposé dans [CDMP05]. Nous adaptons donc ici le calcul des probabilités d'occurrence d'évènements d'échec de la preuve. Cela permet d'améliorer la borne d'indifférenciabilité. Nous exploitons une propriété particulière des encodages sans préfixe. Un chemin complet dans le graphe est défini comme la séquence des blocs composant $\mathsf{pad}(M)$ pour un M donné. Le fait qu'aucun message encodé $\mathsf{pad}(M)$ ne soit le préfixe d'un autre message encodé $\mathsf{pad}(M')$ se traduit en termes de chemins par l'impossibilité pour un chemin complet μ à être le début d'un autre chemin complet μ'.

Nous utilisons cette propriété pour majorer plus précisément la borne de la probabilité de l'évènement Echec_3 au Jeu 4, en montrant le lemme suivant.

Lemme 4 *Lorsque la construction de la Figure 5.1 est utilisée avec une fonction d'encodage sans préfixe, la probabilité d'occurrence de* Echec_3 *au Jeu 4 après* N *requêtes à* $\mathcal{F}$ *vérifie*

$$\Pr\left[\mathsf{Echec}_3\right] \leq \frac{N(N-1)}{2} 2^{-(\ell_a+\ell_h)} .$$

Démonstration :
Contrairement au cas général, nous ne divisons pas le calcul en fonction de l'existence de multicollisions. Nous calculons directement $\Pr\left[\mathsf{Echec}_3(q)\right]$, la probabilité que Echec_3 se produise au cours du traitement de la $q^{ème}$ requête, pour n'importe quel $q \leq N$ correspondant à une requête émise par $\mathcal{D}$.
Soit $u \in \mathcal{X}$ l'état défini par $u = \mathsf{Insert}^{-1}[M](A, B, C)$, où (M, A, B, C) est la $q^{ème}$ requête. Soit x un nœud qui a un chemin dans $\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{I}}$:

$$x^0 \overset{M^1}{\rightarrow} x^1 \ldots x^{k-1} \overset{M^k}{\rightarrow} x.$$

Comme montré dans la preuve du lemme 12, le dernier arc $x^{k-1} \overset{M^k}{\rightarrow} x$ ne peut appartenir à $\mathcal{G}_{\mathcal{D}}$. Supposons maintenant que le chemin $\langle M^1 \mid M^2 \mid \ldots \mid M^k \mid M \rangle$ soit complet. L'encodage utilisé étant sans préfixe, $\langle M^1 \mid M^2 \mid \ldots \mid M^k \rangle$ n'est pas un chemin complet.
x ne peut pas être dans $X_{\mathcal{D}}$. En effet, cela impliquerait l'existence d'un arc $\tilde{z} \overset{\tilde{M}}{\rightarrow} x$ dans $E_{\mathcal{D}}$ avec $\tilde{M} \neq M$. L'insertion d'un des arcs $x^{k-1} \overset{M^k}{\rightarrow} x$, $\tilde{z} \overset{\tilde{M}}{\rightarrow} x$ a alors provoqué l'évènement Echec_1.
Au moment de l'insertion de $x^{k-1} \overset{M^k}{\rightarrow} x$, $x = (\alpha, \beta, \gamma)$ est généré de la manière suivante :
– γ est la partie C de $\mathsf{Insert}[M^k](x^{k-1})$.
– (α, β) est tiré uniformément sur $\{0,1\}^n$ et est indépendant de la vue de l'attaquant.
Pour tout nœud x qui a un chemin dans $\mathcal{G}_{\mathcal{I}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$, on a donc

$$\Pr\left[u = x\right] \leq 2^{-(\ell_a+\ell_h)}$$

Notons $Y(q)$ l'ensemble des nœuds x ayant un chemin dans $\mathcal{G}_{\mathcal{I}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$ après $q-1$ requêtes. Chaque requête introduisant au plus 1 tel nœud dans le graphe, on en déduit $|Y(q)| \leq q-1$, et

$$\begin{aligned} \Pr\left[\mathsf{Echec}_3(q)\right] &\leq \sum_{x \in Y(q)} \Pr\left[u = x\right] \\ &\leq (q-1)2^{-(\ell_a+\ell_h)} \end{aligned}$$

En sommant sur q, on en déduit

$$\Pr\left[\mathsf{Echec}_3\right] \leq \frac{N(N-1)}{2} 2^{-(\ell_a+\ell_h)}$$

. □

Nous obtenons finalement le théorème suivant.

Théorème 4 *Considérons la construction générale décrite sur la Figure 5.1, instanciée avec une fonction aléatoire $\mathcal{F}$ et une fonction d'encodage* pad *sans préfixe, et le scénario d'indifférenciabilité induit par le simulateur défini par la Figure 5.5. Pour tout distingueur $\mathcal{D}$ interagissant avec Σ ou Σ' et dont le poids total des requêtes est d'au plus N, nous avons*

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(\ell_a+\ell_h)} N^2 \ . \tag{5.14}$$

5.4 Preuve d'indifférenciabilité lorsque $\mathcal{F}$ est une permutation

Nous montrons maintenant un résultat d'indifférenciabilité équivalent à celui de la section précédente, en modélisant $\mathcal{F}$ comme une permutation de (A, B) paramétrée par (C, M). Ce cas correspond à la construction utilisée dans Shabal, à ceci près que le compteur qui est ajouté à une partie de l'état interne est supprimé. Nous appliquons cette simplification au mode de Shabal afin de simplifier l'écriture de la preuve. Le résultat que nous montrons ici est adapté au mode de Shabal avec tours à blanc et compteur dans la section 5.5

Théorème 5 *Considérons l'algorithme d'extension de domaine décrit en Figure 5.1 avec une fonction d'encodage quelconque, et le simulateur $\mathcal{S}$ défini par la Figure 5.13, $\mathcal{F}$ étant modélisée comme une permutation paramétrée aléatoire. L'avantage de tout attaquant $\mathcal{D}$ interagissant avec Σ ou Σ' et effectuant au plus N requêtes droites à $\mathcal{F}$,*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{3N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)} + 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] \ ,$$

*où t-*Coll *est défini avant le théorème 2.*

Remarque. Le simulateur $\mathcal{S}$ utilise des ensembles $U_{C,M}$,$V_{C,M}$ et $W_{C,M,B}$. Nous définissons maintenant ces ensembles. Pour tout paramètre (C, M), nous notons les ensembles des valeurs de (A, B) sur lesquelles $\mathcal{F}$ ou $\mathcal{F}^{-1}$ sont déjà connues de la manière suivante.

$$\begin{aligned} U_{C,M} &= \{(A,B) \in \{0,1\}^{\ell_a+\ell_h} | \exists x \xrightarrow{M} y \in E_{\mathcal{I}} \cup E_{\mathcal{D}}, y = (A,B,C)\} \ , \\ V_{C,M} &= \{(A,B) \in \{0,1\}^{\ell_a+\ell_h} | \exists x \xrightarrow{M} y \in E_{\mathcal{I}} \cup E_{\mathcal{D}}, \mathsf{Insert}[M](x) = (A,B,C)\} \ . \end{aligned} \tag{5.15}$$

Nous définissons également

$$W_{C,M,B} = \{A \in \{0,1\}^{\ell_a} | \exists x \xrightarrow{M} y \in E_{\mathcal{I}} \cup E_{\mathcal{D}}, y = (A,B,C)\} \ . \tag{5.16}$$

Initialisation de $\mathcal{S}$

1. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$

Traitement des requêtes à $\mathcal{F}$

Entrée : (M, A, B, C)
Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \xrightarrow{M} y$ dans $E_{\mathcal{D}}$
   (a) renvoyer (A', B') tel que $y = (A', B', C)$ à $\mathcal{D}$
3. sinon
   (a) si x a un chemin μ dans $\mathcal{G}_{\mathcal{D}}$ et si $\mu||M$ est un chemin complet
      i. si μ n'est pas unique, arrêter $\mathcal{S}$
      ii. calculer $\mathcal{M} = \mathsf{unpad}(\mu||M)$
      iii. appeler $\mathcal{H}$ pour obtenir $B' = \mathcal{H}(\mathcal{M})$
      iv. tirer $A' \leftarrow \{0,1\}^{\ell_a} \setminus W_{C,M,B'}$
   (b) sinon
      i. tirer $(A', B') \leftarrow \{0,1\}^{\ell_a+\ell_h} \setminus U_{C,M}$
   (c) définir $y = (A', B', C)$
   (d) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (e) renvoyer (A', B') à $\mathcal{D}$

Traitement des requêtes à $\mathcal{F}^{-1}$

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

1. définir $y = (A', B', C)$
2. s'il existe un arc $x \xrightarrow{M} y \in E_{\mathcal{D}}$
   (a) renvoyer (A, B) tel que $(A, B, C) = \mathsf{Insert}[M](x)$
3. sinon
   (a) tirer $(A, B) \leftarrow \{0,1\}^{\ell_a+\ell_h} \setminus V_{C,M}$
   (b) définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
   (c) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (d) renvoyer (A, B) à $\mathcal{D}$

FIGURE 5.13 – Simulateur $\mathcal{S}$ pour une permutation paramétrée $\mathcal{F}$ et son inverse $\mathcal{F}^{-1}$ pour le théorème 5.

5.4.1 Modélisation d'une permutation paramétrée

Dans cette section, $\mathcal{F}$ est instanciée par une permutation paramétrée $\mathcal{F}(M, A, B, C) = \mathcal{F}_{C,M}(A, B)$ tirée uniformément dans l'ensemble des permutations paramétrées. En cela, nous cherchons à nous rapprocher des fonctionnalités de la permutation interne de Shabal. La permutation paramétrée de Shabal étant facilement inversible, nous donnons à l'attaquant l'accès à un oracle permettant de calculer $(A, B) = \mathcal{F}^{-1}_{C,M}(A', B')$. Ce type de démarche a été introduit par Liskov dans [Lis06], dans le but de caractériser certaines faiblesses des fonctions de compression, vues comme des fonctions aléatoires. Dans [HS08], Hoch et Shamir affinent cette démarche en tenant compte de la distribution du nombre d'antécédents d'un élément par une fonction aléatoire. Dans les travaux présentés ici, $\mathcal{F}$ est tirée dans un ensemble de permutations paramétrées. Ceci implique les propriétés suivantes sur la simulation de $\mathcal{F}$.

Simulation de $\mathcal{F}^{-1}$. Toutes les permutations paramétrées $\mathcal{F} : (M, A, B, C) \mapsto \mathcal{F}_{C,M}(A, B)$ ne sont pas calculables ni inversibles facilement. On pourrait imaginer des instanciations pratiques de notre construction avec des fonctions à sens unique, c'est-à-dire faciles à calculer et difficiles à inverser. Ce n'est pas le cas de la permutation utilisée dans Shabal, pour laquelle nous avons décrit au Chapitre 4 la séquence d'opérations permettant de l'inverser. Afin de prendre en compte cette propriété, nous donnons à $\mathcal{D}$ un accès par un oracle à $\mathcal{F}^{-1}$ définie par $\mathcal{F}^{-1}(M, A, B, C) = \mathcal{F}^{-1}_{C,M}(A, B)$. $\mathcal{F}$ étant une permutation paramétrée, $\mathcal{F}^{-1}(M, A, B, C)$ a une valeur unique.

Distribution des sorties de $\mathcal{F}$. Lorsque $\mathcal{F}$ est tirée uniformément parmi l'ensemble des fonctions de $n+\ell_m$ bits vers $n-\ell_c$ bits, les sorties de $\mathcal{F}$ sont indépendantes et identiquement distribuées. Ce n'est pas le cas lorsque $\mathcal{F}$ est une permutation paramétrée. En effet, lorsque les paramètres (C, M) sont fixés, les valeurs de $\mathcal{F}_{C,M}(A, B)$ et de $\mathcal{F}_{C,M}(A', B')$ ne peuvent pas être égales lorsque $(A, B) \neq (A', B')$. Cela implique qu'une fois $\mathcal{F}_{C,M}(A, B)$ défini, la distribution de $\mathcal{F}_{C,M}(A', B')$ n'est plus uniforme. Cette caractéristique des permutations doit être prise en compte dans la preuve.

5.4.2 Description de la séquence de jeux.

Le déroulement général de la preuve est similaire à celui du cas où $\mathcal{F}$ est instanciée par une fonction aléatoire. Nous décrivons donc ici les modifications à apporter à la séquence de jeux précédente pour tenir compte des spécificités décrites ci-dessus. En particulier, nous modifions les simulateurs pour tenir compte de la simulation de $\mathcal{F}^{-1}$.

Jeu 0. Dans le Jeu 0, le simulateur n'est pas présent. $\mathcal{D}$ peut envoyer des requêtes à $\mathcal{C}^{\mathcal{F}}$, $\mathcal{F}$ et $\mathcal{F}^{-1}$.

Jeu 1. Comme dans le cas précédent, $\mathcal{S}_{1,2}$ est une simple interface qui transmet les requêtes à $\mathcal{F}$ ou $\mathcal{F}^{-1}$ et les réponses à leur destinataire, et construit le graphe $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$. Comme dans la preuve de la section précédente, $\mathcal{G}_{\mathcal{I}}$ reste vide dans ce Jeu. Le traitement des requêtes à $\mathcal{F}^{-1}$ est décrit en Figure 5.14. On a toujours

$$\Pr[W_1] = \Pr[W_0]. \tag{5.17}$$

Simulation de $\mathcal{F}^{-1}$ par $\mathcal{S}_{1,2}$
Entrée : (M, A', B', C), origine $\mathcal{O} = \mathcal{D}$
Sortie : (A, B)
1. définir $y = (A', B', C)$
2. appeler $\mathcal{F}^{-1}$ pour obtenir $(A, B) = \mathcal{F}^{-1}_{C,M}(A', B')$
3. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
4. ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
5. renvoyer (A, B) à $\mathcal{D}$

FIGURE 5.14 – Traitement des requêtes à $\mathcal{F}^{-1}$ par $\mathcal{S}_{1,2}$ dans les Jeux 1 et 2.

Jeu 2. Comme dans la preuve précédente, nous introduisons maintenant l'intercepteur $\mathcal{I}_{2,3,4}$ décrit en Figure 5.8. Cette modification n'affecte pas les réponses envoyées à l'attaquant, d'où

$$\Pr[W_2] = \Pr[W_1]. \tag{5.18}$$

Jeu 3. Comme dans la section 5.3, nous modifions le comportement de $\mathcal{S}_{1,2}$ pour qu'il ne questionne pas $\mathcal{F}$ ou $\mathcal{F}^{-1}$ afin de déterminer des valeurs qu'il connaît déjà. Le simulateur obtenu est décrit en Figure 5.15. Cette modification ne change pas les réponses reçues par l'attaquant, d'où

$$\Pr[W_3] = \Pr[W_2]. \tag{5.19}$$

Simulation de $\mathcal{F}^{-1}$ par $\mathcal{S}_3$
Entrée : (M, A', B', C), origine $\mathcal{O} = \mathcal{D}$
Sortie : (A, B)
1. définir $y = (A', B', C)$
2. s'il existe un arc $x \xrightarrow{M} y \in E_{\mathcal{I}} \cup E_{\mathcal{D}}$
   (a) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (b) renvoyer (A, B) tel que $(A, B, C) = \mathsf{Insert}[M](x)$
3. sinon
   (a) appeler $\mathcal{F}^{-1}$ pour obtenir $(A, B) = \mathcal{F}^{-1}_{C,M}(A', B')$
   (b) définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
   (c) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (d) renvoyer (A, B) à $\mathcal{D}$

FIGURE 5.15 – Traitement des requêtes à $\mathcal{F}^{-1}$ par $\mathcal{S}_3$ dans le Jeu 3.

Jeu 4. De manière similaire à la preuve précédente, nous introduisons maintenant la détection de cas d'échec dans la preuve. Les cas d'échec lors des requêtes à $\mathcal{F}$ sont identiques à ceux de la preuve précédente. Les cas d'échec Echec_4 et Echec_5 caractérisent l'occurrence d'un évènement indésirable lors des requêtes à $\mathcal{F}^{-1}$, et sont décrits en Figure 5.16.

Echec_4. Cet évènement peut être identifié au moment d'insérer un arc $x \xrightarrow{M} y$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ suite à une requête à $\mathcal{F}^{-1}$. Si x a déjà un chemin dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$, $x \xrightarrow{M} y$ peut entraîner la prédéfinition d'un haché, ou des collisions internes.

Echec_5. L'évènement Echec_5 est l'équivalent de Echec_3 lors d'une requête à $\mathcal{F}^{-1}$. Le simulateur reçoit une requête venant de $\mathcal{D}$ pour une valeur de $\mathcal{F}$ qui a déjà été définie dans $\mathcal{G}_{\mathcal{I}}$.

La vue de l'attaquant n'est pas modifiée par ces opérations internes au simulateur, d'où

$$\Pr[W_4] = \Pr[W_3]. \tag{5.20}$$

Simulation de $\mathcal{F}^{-1}$ par $\mathcal{S}_{4,5}$

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

Sortie : (A, B)

1. définir $y = (A', B', C)$
2. si y a un chemin utile $\mu||M$ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$
   (a) NoteEchec(Echec_5)
3. s'il existe un arc $x \xrightarrow{M} y \in E_{\mathcal{I}} \cup E_{\mathcal{D}}$
   (a) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (b) renvoyer (A, B) tel que $(A, B, C) = \mathsf{Insert}[M](x)$
4. sinon
   (a) appeler $\mathcal{F}^{-1}$ pour obtenir $(A, B) = \mathcal{F}^{-1}_{C,M}(A', B')$
   (b) définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
   (c) si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et si $\mu||M$ est un chemin complet
      i. NoteEchec(Echec_4)
   (d) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
   (e) renvoyer (A, B) à $\mathcal{D}$

FIGURE 5.16 – Traitement des requêtes à $\mathcal{F}^{-1}$ par $\mathcal{S}_{4,5}$ dans les Jeux 4 et 5.

Nous étendons maintenant les propriétés définies lors de la preuve précédente au cas où $\mathcal{F}$ est une permutation paramétrée facilement inversible.

Propriété 4 *Soit $\mathcal{G}$ le graphe $\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{I}}$ construit par le simulateur pour $(\mathcal{F}, \mathcal{F}^{-1})$ décrit sur les Figures 5.10 et 5.16 après la réponse à N requêtes. Supposons que Drapeau = Succès, et soit $y \xrightarrow{M} x$ un nœud qui a été inséré dans le graphe lors de la $q^{ème}$ requête, pour un certain $q \leq N$. S'il existe un chemin utile μ entre x^0 et x, il était entièrement déterminé après la $q^{ème}$ requête.*

Démonstration : Soit x un nœud ayant un chemin utile $\mu = \langle M^1, \ldots, M^k \rangle$ dans $\mathcal{G}$. Il existe aussi une séquence d'états $x^1, \ldots, x^{k-1}$ tels que

$$x^0 \xrightarrow{M^1} x^1 \ldots x^{k-1} \xrightarrow{M^k} x^k = x.$$

Soit q' le plus petit entier tel que ce chemin soit complet après la $q'^{ème}$ requête. Cela 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 ajoutée au graphe lors de la $q'^{ème}$ requête

alors que tous les autres arcs du chemin étaient déjà dans $E(q')$. q' étant minimal, $x^{i-1} \xrightarrow{M^i} x^i$ n'est pas dans $E(q')$.
Supposons maintenant que $q' > q$. Dans ce cas, x^{i-1} et x^i sont dans $X(q')$. En effet, comme $x^{i-1} \xrightarrow{M^i} x^i$ est l'unique arc du chemin à ne pas être dans $E(q')$, nous déduisons que si $i \geq 2$, $x^{i-2} \xrightarrow{M^{i-1}} x^{i-1}$ est dans $E(q')$, ce qui implique que $x_{i-1} \in X(q')$ si $i > 1$. De même, si $i \leq k-1$, $x^i \xrightarrow{M^{i+1}} x^{i+1}$ est dans $E(q')$, ce qui implique que $x^i \in X(q')$ si $i < k$. De plus on sait que $x^0 \in X(q')$ car il y est inséré lors de l'initialisation de $\mathcal{S}$ et $x^k = x \in X(q) \subset X(q')$. De ce fait, si $x^{i-1} \xrightarrow{M^i} x^i$ est généré au cours de la simulation de $\mathcal{F}$, Echec_2 se produit, et si $x^{i-1} \xrightarrow{M^i} x^i$ est généré au cours de la simulation de $\mathcal{F}^{-1}$, Echec_4 se produit parce que x^{i-1} a un chemin $\langle M^1, \ldots, M^{i-1} \rangle$ et $\langle M^1, \ldots, M^i \rangle$ est un chemin utile.
Un drapeau d'échec a donc été levé, ce qui contredit les hypothèses de la propriété. Ceci achève la démonstration. □

De la même manière que dans la preuve précédente, nous déduisons la propriété 2 de la propriété 4.

Jeu 5. Comme dans la preuve de la section précédente, le Jeu 5 est obtenu à partir du Jeu 4 en remplaçant $\mathcal{I}_{2,3,4}$ par $\mathcal{I}_{5,6}$. Au lieu de décomposer les requêtes de haché et de les transmettre à $\mathcal{S}_{4,5}$ dès leur réception, le nouvel intercepteur attend la fin du jeu. Le simulateur du Jeu 5 est toujours $\mathcal{S}_{4,5}$, et $\mathcal{I}_{5,6}$ est représenté dans la Figure 6.14. La vue de l'attaquant n'est pas modifiée, donc

$$\Pr[W_5] = \Pr[W_4]. \tag{5.21}$$

De manière analogue à la preuve de la section précédente, nous montrons la propriété suivante.

Propriété 5 *Pour un même aléa de $\mathcal{D}$ et une même fonction $\mathcal{F}$, les graphes construits par $\mathcal{S}_{4,5}$ dans les Jeux 4 et 5 sont identiques, et $\mathcal{S}_{4,5}$ note un cas d'échec dans le Jeu 4 si et seulement si $\mathcal{S}_{4,5}$ note un cas d'échec dans le Jeu 5.*

Démonstration :
La preuve de cette propriété est très similaire à celle qui est réalisée dans la section 5.3.
A la fin du Jeu 4 comme du Jeu 5, le graphe contient les arcs correspondant aux requêtes à $\mathcal{F}$ et à $\mathcal{F}^{-1}$ émises par $\mathcal{D}$, et à la construction des requêtes de haché. Pour une même fonction $\mathcal{F}$, les réponses reçues par l'attaquant sont identiques. Pour un aléa identique de l'attaquant aux Jeux 4 et 5, l'attaquant émet alors la même séquence de requêtes. Les graphes résultant sont donc identiques.
Nous montrons maintenant que les cas d'échec définis dans les Jeux 4 et 5 sont équivalents, en montrant qu'un échec au Jeu 4 implique un échec au Jeu 5, puis qu'un échec au Jeu 5 implique un échec au Jeu 4.

Cas d'échec au cours du Jeu 4. Supposons maintenant que Echec_1 ou Echec_2 se produit au cours du Jeu 4. Le raisonnement est identique à celui de la preuve de la propriété 3.
Supposons que Echec_3 se produit au cours du Jeu 4 : $\mathcal{D}$ émet une requête (M, A, B, C) telle que $x = \mathsf{Insert}^{-1}[M](A, B, C)$ a un chemin utile μ dans $E_{\mathcal{I}}$ mais pas dans $E_{\mathcal{D}}$, et $\mu || M$ est un chemin utile. Soit $z \xrightarrow{M^*} x$ le dernier arc de ce chemin. Dans le Jeu 5, cet arc est inséré :
– soit à la fin du jeu suite à une requête de hachage, ce qui provoque Echec_2 ;

– soit à la suite d'une requête à $\mathcal{F}$ alors que z a un chemin dans le graphe, ce qui provoque Echec_1 ;
– soit à la suite d'une requête à $\mathcal{F}$ alors que z n'a pas de chemin dans le graphe. A la fin du jeu, z a un chemin dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ ce qui a provoqué un cas d'échec ;
– soit à la suite d'une requête à $\mathcal{F}^{-1}$, ce qui provoque également un cas d'échec, z ayant un chemin à la fin du jeu.

Supposons que Echec_4 se produit au cours du Jeu 4 : au moment d'insérer un arc $x \xrightarrow{M} y$ dans $\mathcal{G}_{\mathcal{D}}$, x a un chemin μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ tel que $\mu||M$ soit un chemin utile. Si x a un chemin dans $\mathcal{G}_{\mathcal{D}}$, alors c'est également le cas lorsque $x \xrightarrow{M} y$ doit être inséré au Jeu 5, et Echec_2 se produit. Sinon, x a un chemin dans $\mathcal{G}_{\mathcal{I}}$, notons $z \xrightarrow{M^*} x$ son dernier arc. Au moment du traitement de la requête de haché conduisant à l'insertion de $z \xrightarrow{M} x$ à la fin du Jeu 5, Echec_1 se produit.
Enfin, supposons que Echec_5 se produit au cours du Jeu 4. Sur une requête de la forme (M, y) à $\mathcal{F}^{-1}$, il existe un arc $x \xrightarrow{M} y$ dans $E_{\mathcal{I}}$. Si x a un chemin dans $\mathcal{G}_{\mathcal{D}}$, Echec_4 se produit dans le Jeu 5. Dans le cas contraire, x a un chemin à la fin du jeu, ce qui implique la levée d'un cas d'échec.
Un cas d'échec au Jeu 4 implique donc un cas d'échec au Jeu 5.

Cas d'échec au cours du Jeu 5. Supposons maintenant que Echec_1 ou Echec_2 se produit au cours du Jeu 5, lors de l'insertion d'un arc $x \xrightarrow{M} y$. Le raisonnement de la preuve de la propriété 3 se transpose ici également.
Enfin, supposons que Echec_4 se produit au Jeu 5. Au moment d'insérer $x \xrightarrow{M} y$ dans $E_{\mathcal{D}}$, x a déjà un chemin utile μ dans $\mathcal{G}_{\mathcal{D}}$, et $\mu||M$ est un chemin utile. $\mathcal{G}_{\mathcal{D}}$ étant construit dans le même ordre dans les Jeux 4 et 5, c'est également le cas lorsque $x \xrightarrow{M} y$ doit être ajouté à $E_{\mathcal{D}}$ au Jeu 4. A ce moment, si $x \xrightarrow{M} y$ n'est pas dans $E_{\mathcal{I}}$, Echec_4 se produit. Dans le cas contraire, Echec_5 se produit.
D'après l'ordre du traitement des requêtes au Jeu 5, Echec_3 et Echec_5 ne peuvent pas se produire.
Un cas d'échec au Jeu 5 implique donc un cas d'échec au Jeu 4. Ceci achève la démonstration. □

Jeu 6. Nous remplaçons maintenant l'accès à $\mathcal{F}$ et à $\mathcal{F}^{-1}$ par un accès à l'oracle aléatoire $\mathcal{H}$. $\mathcal{S}_6$ doit donc simuler les réponses aux requêtes à $\mathcal{F}$ et à $\mathcal{F}^{-1}$, de manière cohérente avec l'oracle aléatoire. Cette simulation doit également tenir compte du fait que $\mathcal{F}_{C,M}$ est une permutation. A cette fin, $\mathcal{S}_6$ utilise les ensembles $U_{C,M}$, $V_{C,M}$ et $W_{C,M,B}$ définis par les équations (5.15) et (5.16).

De plus, les cas d'échec 3 et 5 ne pouvant pas se produire, nous supprimons leur détection. Le simulateur du Jeu 6 est représenté en Figure 5.17.

Etant donnée la vue de l'attaquant, les simulations de $\mathcal{F}$ effectuées dans les Jeux 5 et 6 suivent la même distribution de probabilités tant qu'aucun cas d'échec ne s'est produit. Un léger biais peut cependant intervenir sur les distributions des requêtes de haché. En effet, dans le Jeu 6, les réponses sont générées par $\mathcal{H}$, et sont donc indépendantes et uniformément distribuées. Dans le Jeu 5, si des valeurs $(A', B') = \mathcal{F}_{C,M}(A, B)$ telles que la dernière itération de la fonction de compression impose le calcul de $\mathcal{F}_{C,M}(A^*, B^*)$ sont déjà connues, on ne peut pas avoir $\mathcal{F}_{C,M}(A^*, B^*) = (A', B')$, ce qui induit un léger biais sur B' et donc sur la valeur de haché.

Plus concrètement, soit $\mathcal{L}$ la vue de l'attaquant avant la $q^{ème}$ requête de haché M_q, et supposons qu'aucun cas d'échec ne se produit au cours du jeu et que $\mathcal{H}(M_q)$ ne peut pas être déterminé à partir de $\mathcal{L}$. Les Jeux 3 et 4 étant équivalents, nous nous plaçons dans le Jeu 3, dans lequel les requêtes de hachage sont construites en temps réel. Nous notons alors (M, A, B, C) l'entrée de la dernière fonction de compression, et m le dernier bloc de $\mathsf{pad}(M_q)$.

Nous avons alors :

Initialisation de $\mathcal{S}$

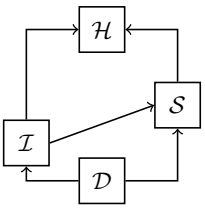
1. définir $X_{\mathcal{I}} = \{x^0\}$ et $E_{\mathcal{I}} = \emptyset$
2. définir $X_{\mathcal{D}} = \{x^0\}$ et $E_{\mathcal{D}} = \emptyset$
3. définir Drapeau = Succès

Traitement des requêtes à $\mathcal{F}$

Entrée : (M, A, B, C), origine $\mathcal{O}$ = soit $\mathcal{I}$ soit $\mathcal{D}$

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \overset{M}{\to} y$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$
   (a) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \overset{M}{\to} y$ à $E_{\mathcal{O}}$
   (b) renvoyer (A', B') tel que $y = (A', B', C)$ à $\mathcal{O}$
3. sinon
   (a) si x a un chemin μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et si $\mu||M$ est un chemin complet
      i. si μ n'est pas unique, arrêter $\mathcal{S}_6$
      ii. calculer $\mathcal{M} = \mathsf{unpad}(\mu||M)$
      iii. appeler $\mathcal{H}$ pour obtenir $B' = \mathcal{H}(\mathcal{M})$
      iv. tirer $A' \leftarrow \{0,1\}^{\ell_a} \setminus W_{C,M,B'}$
   (b) sinon
      i. tirer $(A', B') \leftarrow \{0,1\}^{\ell_a+\ell_h} \setminus U_{C,M}$
   (c) définir $y = (A', B', C)$
   (d) si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ et $\mu||M$ est un chemin utile
      i. si y a un chemin utile μ' dans $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$, NoteEchec($\mathsf{Echec}_1$)
      ii. s'il existe un arc $y \overset{M'}{\to} z$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ et si $\mu||M||M'$ est un chemin utile, NoteEchec(Echec_2)
   (e) ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \overset{M}{\to} y$ à $E_{\mathcal{O}}$
   (f) renvoyer (A', B') à $\mathcal{O}$



Traitement des requêtes à $\mathcal{F}^{-1}$

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

Sortie : (A, B)

1. définir $y = (A', B', C)$
2. s'il existe un arc $x \overset{M}{\to} y \in E_{\mathcal{D}}$
   (a) renvoyer (A, B) tel que $(A, B, C) = \mathsf{Insert}[M](x)$
3. sinon
   (a) tirer $(A, B) \leftarrow \{0,1\}^{\ell_a+\ell_h} \setminus V_{C,M}$
   (b) définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
   (c) si x a un chemin utile μ dans $\mathcal{G}_{\mathcal{D}}$ et si $\mu||M$ est un chemin complet, NoteEchec(Echec_4)
   (d) ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \overset{M}{\to} y$ à $E_{\mathcal{D}}$
   (e) renvoyer (A, B) à $\mathcal{D}$

FIGURE 5.17 – Simulateur $\mathcal{S}_6$ pour une permutation $\mathcal{F}$ et pour $\mathcal{F}^{-1}$ dans le Jeu 6.

$$
\begin{aligned}
& \sum_{b' \in \{0,1\}^{\ell_h}} \left| \Pr\left[\mathcal{H}(M_q) = b' | \mathcal{L}\right][5] - \Pr\left[\mathcal{H}(M_q) = b' | \mathcal{L}\right][3] \right| \\
\leq & \sum_{b' \in \{0,1\}^{\ell_h}} \left| 2^{-\ell_h} - \sum_{a' \in \{0,1\}^{\ell_a}} \Pr\left[\mathcal{F}_{M,C}(A,B) = (a',b') | \mathcal{L}\right][3] \right| \\
\leq & \sum_{(a',b') \in \{0,1\}^{\ell_a+\ell_h}} \left| 2^{-(\ell_a+\ell_h)} - \Pr\left[\mathcal{F}_{M,C}(A,B) = (a',b') | \mathcal{L}\right][3] \right| \\
\leq & \; 2^{-(\ell_a+\ell_h)} u + (2^{-(\ell_a+\ell_h)} - u) \left(\frac{1}{2^{\ell_a+\ell_h}} + \frac{1}{2^{\ell_a+\ell_h} - u} \right) \\
\leq & \; \frac{2r}{2^{\ell_a+\ell_h}} \, ,
\end{aligned}
$$

u étant un majorant de $|U_{m,c}|$ avant la $q^{ème}$ requête de haché. Si le traitement du dernier bloc de M_q correspond à la $r^{ème}$ requête reçue par $\mathcal{S}$ dans le Jeu 4, $u = r$ convient. Le biais sur la vue de l'attaquant est alors majoré par la distance statistique entre les réponses reçues, qui est la moitié de cette quantité. Le biais total au cours du jeu est alors majoré par $\frac{N^2}{2 \times 2^{\ell_a+\ell_h}}$, d'où :

$$
\left| \Pr\left[W_6 \wedge \overline{\mathsf{Echec}_1 \vee \mathsf{Echec}_2}\right] - \Pr\left[W_5 \wedge \overline{\mathsf{Echec}_1 \vee \mathsf{Echec}_2}\right] \right| \leq \frac{N^2}{2 \times 2^{\ell_a+\ell_h}} \tag{5.22}
$$

Jeu 7. En supprimant l'intercepteur $\mathcal{I}_{5,6}$ et la détection de cas d'échecs, nous ne modifions pas la vue de l'attaquant, et nous aboutissons au Jeu final. Le simulateur final est décrit en Figure 5.13. Nous avons alors

$$
\Pr[W_7] = \Pr[W_6] \, . \tag{5.23}
$$

5.4.3 Majoration de l'avantage de l'attaquant.

Le raisonnement de la Section précédente peut être transposé en remplaçant les évènements $\mathsf{Echec}_1 \vee \mathsf{Echec}_2$ et Echec_3 par $\mathsf{Echec}_1 \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4$ et $\mathsf{Echec}_3 \vee \mathsf{Echec}_5$. On en déduit :

$$
\begin{aligned}
|\Pr[W_7] - \Pr[W_0]| \leq & \; \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4] \vee \mathsf{Echec}_4[4]] \\
& + \Pr[\mathsf{Echec}_3[4] \vee \mathsf{Echec}_5[4]] + \frac{N^2}{2} 2^{-(\ell_a+\ell_h)} \, .
\end{aligned}
$$

Nous utilisons ensuite les majorations suivantes, démontrées dans les lemmes 10 et 12 en Annexe A.

$$
\begin{aligned}
\Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4] \vee \mathsf{Echec}_4[4]] & \leq \frac{N^2}{2^{\ell_a+\ell_h} - N} \\
\Pr[\mathsf{Echec}_3[4] \vee \mathsf{Echec}_5[4]] & \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr[t\text{-}\mathsf{Coll}]
\end{aligned}
$$

De ces relations nous déduisons

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{3N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)} + 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right],$$

ce qui prouve le théorème 5.

5.4.4 Évaluation de l'avantage de l'attaquant en fonction du nombre de requêtes.

Nous pouvons appliquer le même raisonnement que dans la section précédente pour majorer $\sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right]$. Nous obtenons le théorème suivant.

Théorème 6 *Considérons la construction générale d'extension de domaine représentée en Figure 5.1 et le simulateur $\mathcal{S}$ défini en Figure 5.13. Pour tout attaquant $\mathcal{D}$ interagissant avec Σ ou Σ' et totalisant au plus N requêtes à $\mathcal{F}$ et $\mathcal{F}^{-1}$:*

$$\mathsf{Adv}(\mathcal{D}) \quad \leq \quad \frac{(5+2e)N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)} + \ell_h 2^{-(\ell_a-1)} N \; . \tag{5.24}$$

De plus, lorsque $\ell_h > \ell_a$:

$$\mathsf{Adv}(\mathcal{D}) \quad \leq \quad \frac{3N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)} + 2^{-\ell_a} N \left\lceil \frac{\ell_h + \ell_a - \log_2(\sqrt{2\pi})}{\ell_h - \ell_a - \log_2 e} \right\rceil \; . \tag{5.25}$$

L'avantage de $\mathcal{D}$ est alors majoré par la plus petite des deux bornes (5.24) *et* (5.25)*.*

Démonstration : Les bornes (5.24) et (5.25) se déduisent immédiatement du lemme 13 permettant d'évaluer $\sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right]$, avec $k = \ell_h$ et $N \leq 2^{\ell_a}$. □

5.5 Borne d'indifférenciabilité avec des tours à blanc et un compteur

L'utilisation d'une fonction d'encodage sans préfixe améliore la sécurité de la construction, cependant elle introduit généralement une baisse des performances de la fonction. Un exemple connu de tel encodage consiste à fixer le premier bit de chaque bloc de message à 0, sauf pour le dernier bloc où ce bit est fixé à 1. Cette mesure est cependant contraignante, les données manipulées n'étant jamais définies au niveau du bit dans la pratique. Pour des instanciations pratiques de cet encodage, il est nécessaire de réserver plus d'un bit dans chaque bloc. Une autre solution est de traiter ce bit différemment des bits du message. C'est ce qui est fait en pratique, et qui peut prendre la forme de l'ajout d'un compteur à certains endroits de l'état interne. On peut également introduire une légère modification en complexifiant légèrement le traitement du dernier bloc de message.

Une alternative à l'utilisation d'un encodage sans préfixe consiste à modifier le traitement du dernier bloc de message, ce qui peut être fait en ajoutant un ou plusieurs *tours à blanc*. Ces tours à blanc consistent à introduire des blocs supplémentaires à la fin de l'encodage. C'est cette approche que nous avons adoptée lors de la définition de Shabal : la fonction de compression est appliquée 4 fois au dernier bloc de message, sans incrémenter le compteur. Nous étudions maintenant l'effet de cette mesure sur l'indifférenciabilité de l'algorithme d'extension de domaine. Les tours à blanc peuvent

être instanciés de différentes manières, pour fixer les idées, nous décrivons la mesure employée par Shabal.

On dit qu'une fonction d'encodage pad impose n_f tours à blanc si pour tout M de $\{0,1\}^*$, $\mathsf{pad}(M)$ est composé d'au moins n_f blocs, et les n_f+1 derniers blocs de $\mathsf{pad}(M)$ sont tous identiques. Shabal utilise conjointement un encodage avec 3 tours à blanc, et un compteur sur $\ell_w = 64$ bits qui est incrémenté pendant le traitement de tous les blocs à l'exception des n_f derniers, qui alimentent les tours à blanc. L'algorithme d'extension de domaine obtenu est représenté sur les Figures 5.18 et 5.19.

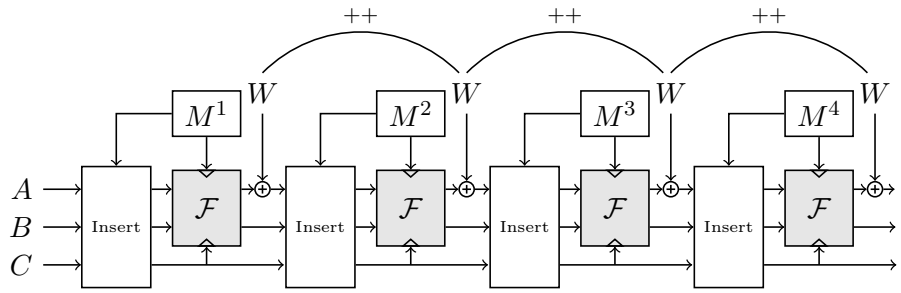


FIGURE 5.18 – Extension de domaine : cas avec ajout d'un compteur du nombre de blocs

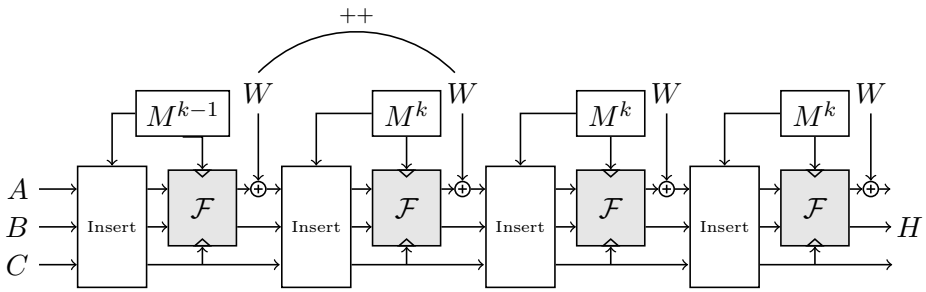


FIGURE 5.19 – Extension de domaine : finalisation avec 2 tours à blanc sans incrémentation de compteur

En considérant le compteur comme une partie du bloc de message et en limitant la longueur après encodage des messages traités à 2^{ℓ_w} blocs, l'utilisation conjointe des tours à blanc et du compteur garantit un encodage sans préfixe. Cependant, au vu du traitement spécifique réservé au compteur, il ne semble pas réaliste de traiter la fonction $\mathcal{F}$ résultante comme une fonction aléatoire. Nous modifions donc maintenant le simulateur de la section 5.4 afin de prendre en compte les spécificités de l'algorithme d'extension de domaine de Shabal. La séquence de jeux est également modifiée.

Construction de chemins et de chemins complets. L'introduction d'un compteur modifie la construction de la fonction de compression à partir de la primitive interne. De ce fait, la notion de chemin précédemment définie doit être adaptée pour être compatible avec cette nouvelle construction. En effet, la notion de chemin caractérise les états x correspondant aux valeurs intermédiaires de l'état interne au cours du calcul d'un haché. Ces chemins doivent maintenant inclure la valeur du compteur. L'application de tours à blanc introduisant une discontinuité dans la gestion du comp-

teur, les valeurs de l'état interne pendant les tours à blanc sont intrinsèquement différentes des valeurs de l'état interne pendant le début du calcul de haché. Afin de caractériser cette différence, nous distinguons maintenant les chemins initiaux et les chemins finaux. Nous modifions donc la définition des arcs, pour prendre en compte la valeur du compteur.

Nous définissons la notion de chemin de manière récursive.

- L'état initial x^0 a un 0-chemin initial $\emptyset$.
- Un état y a un $w+1$-chemin initial $\langle M^1 \mid \ldots \mid M^{w+1}\rangle$ s'il existe $x \in X$ ayant un w-chemin initial $\langle M^1 \mid \ldots \mid M^w\rangle$ et un arc $x \overset{M^{w+1}}{\rightarrow} y \oplus (w+1) \in E$.

La notation $x \oplus w$ représente l'addition du compteur w sur les ℓ_w premiers bits de la partie A de x. Les chemins initiaux caractérisent alors les valeurs de l'état interne au cours du traitement d'un message avant les tours à blanc, juste après l'addition du compteur. Pour modéliser la discontinuité introduite par les tours à blanc, nous introduisons la notion de chemin final définie de la manière suivante.

- Pour tout $w \in \{1, \ldots, 2^{\ell_w}\}$, un état x a un $(w, 0)$-chemin final s'il a un w-chemin initial.
- Pour tout $w \in \{1, \ldots, 2^{\ell_w}\}$ et tout $r \in \{1, \ldots, n_f\}$, un état $y \in X$ a un (w, r)-chemin final $\langle M_1 \mid \ldots \mid M_w\rangle$ s'il existe $x \in X$ ayant un $(w, r-1)$-chemin final $\langle M_1 \mid \ldots \mid M_w\rangle$ et un arc $x \overset{M^w}{\rightarrow} y \oplus w \in E$.

Les chemins finaux caractérisent les valeurs de l'état interne au cours des tours à blanc. Un chemin complet est donc maintenant un (w, n_f)-chemin final.

Modification de la séquence de jeux. Afin de simplifier l'écriture, nous supposons que tout chemin initial est un chemin utile, c'est-à-dire que toute séquence de blocs correspond au préfixe d'un message paddé.

Une première modification de la séquence de jeux concerne la définition de cas d'échecs, elle vient des nouvelles définitions liées à la notion de chemin. Une modification importante induite par ces nouvelles définitions est que l'insertion d'un arc $x \overset{M}{\rightarrow} y$ dans le graphe peut générer deux nouveaux nœuds ayant un chemin. En effet, si x a un w-chemin initial se terminant par M, alors $y \oplus w$ a un $w, 1$ chemin final et $y \oplus (w+1)$ a un $(w+1)$-chemin initial. Dans ce cas, la détection de Echec_1 doit porter sur $y \oplus w$ et $y \oplus (w+1)$ (au lieu de y dans les preuves précédentes). Les majorations relatives au nombre de nœuds du graphe ayant un chemin s'en trouvent modifiées.

D'autre part, les simulateurs sont modifiés de manière à tirer parti de la forme particulière de l'encodage utilisé. En effet, dans le cas général, il n'est pas possible pour le simulateur d'anticiper le calcul de la dernière fonction de compression d'un calcul de haché : le nombre de valeurs possibles du dernier bloc n'étant borné que par 2^{ℓ_m}, la complexité du simulateur exploserait. Dans le cas d'un encodage avec compteur et tours à blanc, si un nœud y a un $w, n_f - 1$ chemin final (pour $n_f > 1$), alors le calcul du haché peut être complété, le dernier bloc de message étant identique au bloc précédent.

Nous modifions ainsi les simulateurs des Jeux 1 à 6. Dans la simulation de $\mathcal{F}$, si l'origine de la requête est $\mathcal{D}$ et si l'insertion de $x \overset{M}{\rightarrow} y$ dans $E_{\mathcal{D}}$ implique que $y \oplus w$ a un $w, n_f - 1$-chemin final dans $\mathcal{G}_{\mathcal{D}}$ dont le dernier bloc est M, le simulateur calcule $(A, B, C) = \mathsf{Insert}[M](y \oplus w)$ et définit $\mathcal{F}_{C,M}(A, B)$ en s'aidant de l'oracle $\mathcal{F}$ (dans les Jeux 1 à 4) ou de l'oracle $\mathcal{H}$ (dans les Jeux 5 et 6). Dans les Jeux 5 et 6, l'oracle $\mathcal{H}$ n'est utilisé que dans ce cas de figure, en effet si $\mathcal{D}$ émet une requête à $\mathcal{F}$ lui permettant de terminer un calcul de haché, cette valeur a été simulée de manière anticipée par $\mathcal{S}$ et est déjà dans $\mathcal{G}_{\mathcal{D}}$.

Résultat d'indifférenciabilité. Nous obtenons alors le théorème suivant, relatif à l'indifférenciabilité de la construction utilisée par Shabal.

Théorème 7 *Considérons l'algorithme d'extension de domaine décrit en Figures 5.18 et 5.19 avec une fonction d'encodage quelconque, et le simulateur $\mathcal{S}$ défini par la Figure 5.20, $\mathcal{F}$ étant modélisée comme une permutation paramétrée aléatoire. L'avantage de tout attaquant $\mathcal{D}$ interagissant avec Σ ou Σ' et effectuant au plus N requêtes droites à $\mathcal{F}$,*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{3N^2}{2^{\ell_a+\ell_h}-2N} + N^2 \times \max\left(\frac{1}{2^{\ell_a+\ell_h}-2N}, 2P_c 2^{-\ell_a}\right) ,$$

où

$$P_c = \max_{M,c,C,U}\left(\Pr\left[\mathsf{Insert}_C[M](a,b,c) = C | (a,b) \leftarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h} \setminus U\right]\right) ,$$

où U parcourt les ensembles de taille au plus $2N$.

Dans le cas de Shabal-512,

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{5N^2}{2^{\ell_a+\ell_h}-2N}.$$

Pour les autres versions de Shabal,

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{4N^2}{2^{\ell_a+\ell_h}-2N}.$$

Démonstration : La séquence de jeux de la Section 5.4 est adaptée en insérant les modifications décrites au paragraphe précédent. Le simulateur du jeu final est maintenant celui de la Figure 5.20. Un raisonnement identique à celui de la Section 5.4 permet d'établir que

$$|\Pr[W_7] - \Pr[W_0]| \leq \Pr[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]] + \Pr[\mathsf{Echec}_3[4] \vee \mathsf{Echec}_4[4]] + \frac{N^2}{2^{\ell_a+\ell_h}}.$$

Le terme $\frac{N^2}{2^{\ell_a+\ell_h}}$, qui correspond au biais sur les valeurs de haché, est le double de celui de la preuve précédente. En effet, le nombre d'arcs dans le graphe avant la $q^{ème}$ requête est maintenant $2q-2$ au lieu de $q-1$.
Les lemmes 11 et 14 montrés en Annexe A permettent de conclure que

$$\begin{aligned}\mathsf{Adv}(\mathcal{D}) &\leq \frac{2N^2}{2^{\ell_a+\ell_h}-2N} + N^2 \times \max\left(\frac{1}{2^{\ell_a+\ell_h}-2N}, 2P_c 2^{-\ell_a}\right) + \frac{N^2}{2^{\ell_a+\ell_h}} \\ &\leq \frac{3N^2}{2^{\ell_a+\ell_h}-2N} + N^2 \times \max\left(\frac{1}{2^{\ell_a+\ell_h}-2N, 2P_c 2^{-\ell_a}}\right).\end{aligned}$$

La fonction d'insertion utilisée par Shabal construit la partie C en prenant ℓ_c bits de la sortie de $\mathcal{F}$ au tour précédent, qui sont donc uniformément distribués, et en soustrayant M par mots de 32 bits. La soustraction de M est une permutation, donc pour tous $M, c, C, U_{M,C}$:

$$\Pr\left[\mathsf{Insert}_C[M](a,b,c) = C | (a,b) \leftarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h} \mathcal{U}_{M,C}\right] \leq \frac{2^{\ell_a+\ell_h-\ell_c}}{2^{\ell_a+\ell_h}-2N} ,$$

la taille de $U_{M,C}$ étant majorée par $2N$ au cours du jeu. Nous en déduisons $P_c = \frac{2^{\ell_a+\ell_h-\ell_c}}{2^{\ell_a+\ell_h}-2N}$. Nous avons alors :

$$2P_c 2^{-\ell_a} = \frac{2 \times 2^{\ell_h-\ell_c}}{2^{\ell_a+\ell_h}-2N} \tag{5.26}$$

Initialisation de $\mathcal{S}$

Pas d'entrée, pas de sortie

1. définir $X = \{x^0\}$ et $E = \emptyset$

Traitement des requêtes à $\mathcal{F}$

Entrée : (M, A, B, C)

Sortie : (A', B')

1. définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
2. s'il existe un arc $x \xrightarrow{M} y \in E$
   (a) renvoyer (A', B') tel que $y = (A', B', C)$
3. sinon
   (a) choisir aléatoirement $(A', B') \leftarrow \{0,1\}^{\ell_a+\ell_h} \setminus U_{C,M}$ et définir $y = (A', B', C)$
   (b) ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
   (c) si $y \oplus w$ a un $w, (n_f - 1)$-chemin final $\mu = M_1 \mid \ldots \mid M$ dans le graphe $\mathcal{G}$ et si $\mu \parallel M$ est un chemin complet
      i. s'il n'existe pas d'arc $y \xrightarrow{M} z$ dans E
         A. calculer $\mathcal{M} = \mathsf{unpad}(\mu \parallel M)$
         B. calculer $(\alpha, \beta, \gamma) = \mathsf{Insert}[M](y \oplus w)$
         C. appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
         D. définir $B'' = h$
         E. choisir aléatoirement $A'' \leftarrow \{0,1\}^{\ell_a} \setminus W_{\gamma,M,B''}$ et définir $z = (A'', B'', \gamma)$
         F. ajouter le nœud z à X et l'arc $(y \oplus w) \xrightarrow{M} z$ à E
   (d) renvoyer (A', B')

Simulation de $\mathcal{F}^{-1}$

Entrée : (M, A', B', C)

Sortie : (A, B)

1. définir $y = (A', B', C)$
2. s'il existe un arc $x \xrightarrow{M} y \in E$
   (a) interpréter $\mathsf{Insert}[M](x)$ comme $\mathsf{Insert}[M](x) = (A, B, C)$
   (b) renvoyer (A, B)
3. choisir aléatoirement $(A, B) \leftarrow \{0,1\}^{\ell_a} + \ell_h \setminus V_{C,M}$ et définir $x = \mathsf{Insert}^{-1}[M](A, B, C)$
4. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
5. renvoyer (A, B)

FIGURE 5.20 – Simulateur $\mathcal{S}$ lorsque $\mathcal{F}$ est une permutation paramétrée et que des tours à blanc et un compteur sont utilisés

Pour Shabal-512, $\ell_c = \ell_h$ et

$$\max\left(\frac{1}{2^{\ell_a+\ell_h} - 2N}, 2P_c 2^{-\ell_a}\right) = \frac{2}{2^{\ell_a+\ell_h} - 2N}$$

Nous avons alors

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{5N^2}{2^{\ell_a+\ell_h} - 2N}.$$

Pour les autres versions de Shabal,

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{4N^2}{2^{\ell_a+\ell_h} - 2N}.$$

□

Utilité du paramètre C Dans le cas de l'indifférenciabilité, nous pouvons remarquer que la borne de sécurité établie dans cette dernière étude est la seule qui bénéficie effectivement de la présence de la variable C. Cette borne correspond à l'attaque suivante.

Pour un grand nombre T de messages $M_1,\ldots,M_t$, et pour tout i :

- $\mathcal{D}$ calcule les hachés de M_i.
- $\mathcal{D}$ questionne $\mathcal{F}$ pour reconstituer la valeur y de l'état interne au cours du hachage de M_i avant la dernière fonction de compression.
- $\mathcal{D}$ calcule $C_i = \mathsf{Insert}_C[\mu](y)$, où μ est le dernier bloc de $\mathsf{pad}(M_i)$.
- $\mathcal{D}$ questionne $\mathcal{F}^{-1}$ sur $A, H(M_i), C_i, \mu$ pour une valeur de A aléatoire.

Si $\mathcal{D}$ est en face de $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, il existe une valeur de A parmi 2^{ℓ_a} pour laquelle

$$(\mathcal{F}^{-1}_{C_i,\mu}(A, H(M_i)), C_i) = \mathsf{Insert}[\mu](y). \tag{5.27}$$

Lorsque A est choisi aléatoirement par l'attaquant, la valeur reçue par $\mathcal{D}$ lors de la dernière requête vérifie cette équation avec probabilité $2^{-\ell_a}$. L'Égalité (5.27) est donc vérifiée pour $T/2^{-\ell_a}$ valeurs de i en moyenne.

Si $\mathcal{D}$ est en face de $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$, lors de la dernière requête, $\mathcal{S}$ n'ayant pas accès aux requêtes directes à $\mathcal{H}$, il simule généralement la réponse à la dernière requête en choisissant une valeur aléatoire. L'égalité (5.27) est donc vraie pour $T/2^{\ell_a+\ell_h}$ valeurs de i en moyenne.

Cette attaque permet donc d'obtenir un distingueur relativement efficace lorsque N est de l'ordre de 2^{ℓ_a}. L'utilisation conjointe du fil C et l'anticipation du dernier tour à blanc permettent de pallier cette attaque.

Chapitre 6

Le modèle d'indifférenciabilité généralisé

Sommaire

Les résultats et le modèle exposés dans le chapitre précédent permettent d'évaluer la sécurité d'un algorithme d'extension de domaine, lorsqu'il est instancié avec une primitive interne choisie aléatoirement. Or dans le monde réel, les fonctions de hachage sont des algorithmes publics, combinant un algorithme d'extension de domaine avec une primitive fixe et publique. De par sa définition, il est impossible de garantir l'indifférenciabilité d'une fonction de hachage par rapport à l'oracle aléatoire.

En revanche, la preuve d'indifférenciabilité permet de garantir que pour attaquer une telle fonction de hachage (avec une meilleure probabilité de succès que celle qui résulte de la preuve de sécurité), il est nécessaire d'exploiter des propriétés particulières de la primitive interne utilisée.

Tant qu'aucune propriété particulière de la fonction de compression n'est connue, la preuve d'indifférenciabilité de l'algorithme d'extension de domaine constitue un argument de la sécurité de la fonction de hachage. En revanche, si une ou plusieurs propriétés particulières sont découvertes, cet argument est considéré comme caduc.

Les résultats que nous présentons dans ce chapitre permettent d'étendre la validité de preuves d'indifférenciabilité d'algorithmes d'extension de domaine à des constructions reposant sur des primitives imparfaites.

Plan de l'étude. Après avoir explicité la problématique à traiter en section 6.1, nous décrivons un modèle permettant de représenter les propriétés statistiques d'une fonction de compression en section 6.2. Dans la section 6.3, nous montrons une borne d'indifférenciabilité d'un oracle aléatoire à usage public pour la construction Chop-MD. Dans la section 6.4, nous donnons un résultat d'indifférenciabilité forte pour une classe de propriétés plus restreinte.

Lien avec Shabal. Les travaux présentés ici ont été initiés dans le cadre de l'étude de la sécurité de Shabal, et notamment lorsque nous avons commencé à étudier l'impact de la découverte de propriétés indésirables de la permutation interne sur la sécurité de la fonction de hachage. Ils n'ont cependant pas permis d'aboutir à un résultat d'indifférenciabilité forte de l'algorithme d'extension de domaine de Shabal, principalement pour deux raisons :

- les techniques que nous avons développées ne permettent pas la prise en compte de l'accès de l'attaquant à la primitive inverse (lorsque celle-ci est une permutation) ;
- dans le cas de l'indifférenciabilité forte, la connaissance préalable par l'attaquant de certaines valeurs de hachés peut biaiser la distribution de la primitive $\mathcal{F}$. La représentation de $\mathcal{F}$ que nous utilisons ne permet pas de majorer ce biais dans le cas général.

6.1 Preuves d'indifférenciabité et attaques par distingueur

Attaques par distingueurs. Le terme d'*attaques par distingueurs* regroupe un certain nombre de techniques permettant d'identifier des propriétés spécifiques à une fonction de compression, voire à une fonction de hachage. Sans constituer un danger direct pour une fonction de hachage, une telle attaque sur la fonction de compression est de nature à jeter un doute sur sa sécurité. La compétition SHA-3 a jusqu'à présent été le cadre de plusieurs attaques de ce type, comme évoqué au chapitre 4, et la publication de telles attaques a souvent été rédhibitoire. En effet, cette compétition se déroule sur une durée contrainte (un peu plus de trois ans entre la publication des candidats et le choix final) et les candidats étaient initialement nombreux. Afin de garantir une analyse suffisante de la norme finale, il a été nécessaire de réduire rapidement le nombre de candidats. Dans cette optique, toutes les propriétés identifiables de ces fonctions sont des sources d'inquiétudes quant à leur robustesse et permettent d'orienter la sélection.

La notion d'attaque par distingueur a été définie par analogie avec la notion d'indistinguabilité utilisée pour évaluer la sécurité d'algorithmes de chiffrement par blocs. Fonctionnellement, un algorithme de chiffrement par blocs $\mathcal{E}$ est une permutation d'un bloc de m bits paramétrée par une clé de k bits. L'indistinguabilité de $\mathcal{E}$ d'une permutation aléatoire correspond à la difficulté pour un attaquant de faire la différence entre une permutation aléatoire et $\mathcal{E}_k$, pour une clé k inconnue fixée. La clé étant inconnue de l'attaquant, cette définition correspond à des cas d'utilisation réels.

Extension de la validité des preuves. Considérons maintenant le problème suivant. Soit $\mathcal{C}^{\mathcal{F}}$ une fonction de hachage construite à partir d'un algorithme d'extension de domaine $\mathcal{C}$ utilisant une primitive interne $\mathcal{F}$. Nous nous intéressons au cas dans lequel $\mathcal{C}$ dispose d'une preuve d'indifférenciabilité d'un oracle aléatoire, qui garantit que $\mathcal{C}^{\mathcal{F}}$ se comporte de manière idéale si tel est le cas

de $\mathcal{F}$. Pour spécifier une fonction de hachage, il est nécessaire de choisir une instance concrète de $\mathcal{F}$ et d'en donner une description fonctionnelle. Supposons maintenant que la primitive choisie ne soit pas idéale, au sens requis par la preuve d'indifférenciabilité de $\mathcal{C}$. A première vue, il semble que le résultat d'indifférenciabilité de $\mathcal{C}$ ne procure plus aucun argument de sécurité concernant $\mathcal{C}^{\mathcal{F}}$, l'hypothèse de base sur laquelle la preuve est construite étant violée. L'objet de ce chapitre est de déterminer dans quelle mesure la sécurité de $\mathcal{C}^{\mathcal{F}}$ est affectée par les caractéristiques spécifiques de $\mathcal{F}$.

D'un point de vue plus général, cette question relève d'un autre paradigme : est-il possible de prouver qu'une construction $\mathcal{C}^{\mathcal{F}}$ se comporte idéalement sachant que $\mathcal{F}$ ne le fait pas ? Nous pouvons aussi nous demander à quel point $\mathcal{C}^{\mathcal{F}}$ diffère d'une fonction de hachage idéale lorsque $\mathcal{F}$ s'éloigne d'une primitive idéale. Cette notion peut être considérée comme une forme de résilience : même dans le cas où une découverte future révèlerait une faiblesse potentielle de $\mathcal{F}$, la fonction $\mathcal{C}^{\mathcal{F}}$ resterait sûre. Notre démarche est donc motivée par des problématiques concrètes. En effet, le remplacement d'un algorithme normalisé largement déployé comme une fonction de hachage est un processus suffisamment délicat pour que sa résistance à de possibles découvertes futures concernant la primitive interne employée soit prise dès sa conception.

Dans ce chapitre nous introduisons une généralisation du modèle d'indifférenciabilité, qui constitue un élément de réponse à ces questions. Notre modèle repose sur l'hypothèse suivante : au lieu d'être modélisée par une primitive idéale sélectionnée uniformément dans l'ensemble FUNC des fonctions ayant les mêmes paramètres, $\mathcal{F}$ est représentée par une primitive affaiblie, tirée dans ce même ensemble selon une distribution biaisée.

6.2 Modélisation de primitives imparfaites

Derrière l'expression d'attaque par distingueur peuvent se cacher différents types de propriétés d'une fonction de compression $\mathcal{F}$, pouvant avoir des impacts divers sur la fonction de hachage reposant sur $\mathcal{F}$. Par exemple, les cryptanalyses différentielles des fonctions des familles MD et SHA permettent d'obtenir des collisions sur ces fonctions en exploitant les propriétés différentielles de leurs fonctions de compression. D'autres attaques par distingueurs semblent en revanche difficiles à traduire en attaques contre la fonction de hachage complète. Les garanties de sécurité apportées par la construction dépendent donc de la forme des attaques sur la fonction de compression que l'on prend en compte. Par conséquent, la modélisation de la primitive $\mathcal{F}$ que nous adoptons dépend des propriétés identifiées de $\mathcal{F}$. $\mathcal{F}$ est alors choisie aléatoirement dans FUNC selon une distribution $\mathcal{V}$. Une façon particulière de modéliser une primitive biaisée est de considérer que $\mathcal{V}$ est la distribution uniforme sur un sous-ensemble $\mathrm{FUNC}' \subset \mathrm{FUNC}$.

Dans la suite de ce chapitre, nous appelons *cas idéal* le cas de preuves pour lesquelles $\mathcal{F}$ est choisi uniformément sur FUNC, comme par exemple dans le chapitre précédent. Nous désignons par *cas biaisé* le cas de preuves pour lesquelles $\mathcal{F}$ est choisi selon une distribution biaisée sur FUNC. Par extension, nous appelons *fonction biaisée* les primitives étudiées dans le cas biaisé.

6.2.1 Adaptation immédiate de la preuve

Une première manière d'évaluer l'impact de la connaissance d'une propriété particulière de $\mathcal{F}$ sur l'indifférenciabilité de la fonction $\mathcal{C}^{\mathcal{F}}$ d'un oracle aléatoire consiste à utiliser le simulateur de la preuve d'indifférenciabilité de $\mathcal{C}$. La preuve par séquence de jeux peut être adaptée au cas biaisé en ajoutant un jeu initial -1 correspondant au cas où $\mathcal{D}$ interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, $\mathcal{F}$ étant tirée sur

FUNC′. En notant $\mathsf{Adv}_{ideal}(\mathcal{D})$ l'avantage de l'attaquant $\mathcal{D}$ dans le cas idéal et $\mathsf{Adv}_{biased}(\mathcal{D})$ son avantage dans le cas biaisé, nous avons alors par l'inégalité triangulaire :

$$\begin{aligned} |\Pr[W_9] - \Pr[W_{-1}]| &\leq |\Pr[W_9] - \Pr[W_0]| + |\Pr[W_0] - \Pr[W_{-1}]| \\ \mathsf{Adv}_{biased}(\mathcal{D}) &\leq \mathsf{Adv}_{ideal}(\mathcal{D}) + |\Pr[W_0] - \Pr[W_{-1}]| . \end{aligned}$$

L'impact de l'attaque par distingueur sur la garantie de sécurité apportée par la preuve est donc résumé par le terme $|\Pr[W_0] - \Pr[W_{-1}]|$. Ce terme peut être interprété comme l'avantage d'un attaquant cherchant à différencier les systèmes $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ avec $\mathcal{F}$ tirée uniformément sur FUNC et $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ où $\mathcal{F}$ est tirée uniformément sur FUNC′.

Afin de majorer l'avantage de l'attaquant, nous utilisons le lemme suivant.

Lemme 5 *Soient D_1 et D_2 deux distributions statistiques définies sur un support commun $\mathcal{Z}$ et soit $\|D_1 - D_2\|$ leur distance statistique, c'est à dire*

$$\|D_1 - D_2\| = \frac{1}{2} \sum_{z \in \mathcal{Z}} |\Pr_{Z \leftarrow D_1}[Z = z] - \Pr_{Z \leftarrow D_2}[Z = z]| .$$

Soit E un évènement quelconque. Nous avons alors

$$|\Pr_{Z \leftarrow D_1}[E] - \Pr_{Z \leftarrow D_2}[E]| \leq \|D_1 - D_2\| .$$

Nous pouvons appliquer ce résultat aux distributions uniformes sur FUNC et FUNC′, cependant une manière plus fine de procéder consiste à l'appliquer aux vues de l'attaquant après N requêtes. Lorsque $\mathcal{F}$ est uniforme sur FUNC, nous avons vu au chapitre précédent que ses sorties sont indépendantes et uniformément distribuées. Dans le cas biaisé, nous définissons la distribution

$$D(u, \mathcal{L}(q)) = \{(A', B') = \mathcal{F}(u), \mathcal{F} \leftarrow \mathrm{FUNC}'[\mathcal{L}(q)]\} ,$$

où $\mathcal{L}(q)$ représente l'ensemble des entrées et sorties de $\mathcal{F}$ définies avant la $q^{ème}$ requête, et $\mathrm{FUNC}'[\mathcal{L}(q)]$ est l'ensemble des fonctions $\mathcal{F}$ de FUNC′ telles que $\mathcal{F}(u) = v$ pour tout $(u, v) \in \mathcal{L}(q)$. Nous en déduisons la Proposition suivante :

Proposition 1 *Considérons une construction $\mathcal{C}^{\mathcal{F}}$. Supposons $\mathcal{F}$ tirée uniformément sur un sous-ensemble* $\mathrm{FUNC}' \subset \mathrm{FUNC}$ *et soit $\mathcal{H}$ un oracle aléatoire. Pour tout distingueur $\mathcal{D}$ qui interagit avec $\mathcal{S}^{\mathcal{H}}_{\mathrm{id}}$ et tel que $\mathcal{F}$ reçoit au plus N appels,*

$$\mathsf{Adv}(\mathcal{D}) \leq \mathsf{Adv}_{ideal}(\mathcal{D}) + N \max ||D - U||$$

où $\mathsf{Adv}_{ideal}(\mathcal{D})$ est l'avantage de $\mathcal{D}$ lorsque $\mathcal{F}$ est une fonction idéale et où $\max ||D - U||$ est défini par

$$\max ||D - U|| = \max_{u, \mathcal{L}, |\mathcal{L}| \leq N} ||D(u, \mathcal{L}) - U|| .$$

Cette borne n'est cependant significative que si la distance statistique maximale $\max ||D - U||$ est petite. En particulier, cette technique ne peut être appliquée lorsque les sorties correspondant à des entrées particulières de $\mathcal{F}$ sont fortement corrélées. Cette situation se produit fréquemment, et il arrive que des propriétés statistiques se produisent avec probabilité 1 sur la primitive interne dans des cas pratiques.

Exemples.

- Knudsen, Matusiewicz et Thomsen [KMT09] ont montré que pour certaines entrées X sur la permutation paramétrée de Shabal, il est possible de déterminer un paramètre K_X tel que $\mathcal{F}_{K_X}(X)$ et X aient la même partie B avec probabilité 1. Des détails concernant cette propriété sont donnés au chapitre 4.
- Dans [GT10], Guo et Thomsen montrent que la fonction de compression de Blue Midnight Wish vérifie certaines propriétés différentielles de manière déterministe. Plus précisément, pour certaines valeurs de différence en entrée de la fonction de compression, la différence sur certains bits de sortie est déterministe. De la même façon, la fonction de compression de Hamsi-256 a des propriétés différentielles vérifiées avec probabilité 1. Par exemple Calik et Turan montrent dans [CT10], que pour toute valeur du bloc de message M, les sorties correspondant à deux valeurs de la variable de chaînage $X, X \oplus e_{71} \oplus e_{199}$ différant seulement sur les bits 71 et 199 ont des bits 228 et 230 identiques. Pour un historique $\mathcal{L} = \{\mathcal{F}_M(X) = Y\}$, la sortie de $\mathcal{F}_M(X \oplus e_{71} \oplus e_{199})$ peut prendre au plus un quart des valeurs possibles.
- L'étude du degré algébrique d'une fonction de compression peut donner lieu à la découverte de distingueurs différentiels d'ordre supérieur ou de distingueurs dits à somme nulle [AM09]. Cela peut notamment se produire lorsque la fonction de compression est construite en itérant une fonction de bas degré. Une somme nulle de taille 2^κ correspond alors à un ensemble de 2^κ entrées $\{X_1, \ldots, X_{2^\kappa}\}$ telles que $\bigoplus_i X_i = 0$ et $\bigoplus_i F(K, X_i) = 0$. Dans ce cas, la connaissance des sorties correspondant à $(2^\kappa - 1)$ des entrées de la somme nulle détermine totalement la sortie correspondant à l'entrée restante. De telles sommes nulles ont été identifiées pour la fonction de compression de Hamsi-256 [AM09], pour la fonction de compression de `Luffa` v2 et pour la permutation interne de Keccak [BC10, BCdC11].
- Dans le cas de la construction de Davies et Meyer, $\mathcal{F}(C, M) = \mathcal{E}_M(C) \oplus C$. Le calcul de $C = \mathcal{E}_M^{-1}(0)$ permet d'obtenir (C, M) tel que $\mathcal{F}(C, M) = C$, ce qui constitue un point fixe de $\mathcal{F}$. La fonction de compression de SIMD utilise une version modifiée de la construction de Davies et Meyer. Cette modification permet d'éviter la découverte facile de points fixes pour la fonction de compression. Malgré cela, comme mentionné dans [BFL10], Gauravaram et Bagheri ont montré la possibilité de trouver des couples d'entrées-sorties de la fonction de compression de SIMD ayant des propriétés similaires.

Il apparaît cependant que la plupart de ces distingueurs ont un faible impact pratique sur la sécurité de la fonction de hachage qu'ils concernent. En effet, ils contraignent les valeurs de sorties pour un petit nombre d'entrées de $\mathcal{F}$. Leur existence ne peut cependant pas être prise en compte par le simulateur du cas idéal, l'attaquant $\mathcal{D}$ pouvant aisément émettre des requêtes successives pour des entrées dont les sorties sont corrélées. Pour prendre en compte de telles propriétés, il est nécessaire de modifier le simulateur dans le but de l'adapter au cas biaisé.

6.2.2 Représentation algorithmique d'une fonction biaisée

Nous considérons maintenant une primitive $\mathcal{F}$ biaisée, c'est-à-dire pour laquelle certaines propriétés statistiques sont connues. Le modèle que nous définissons est applicable à différents algorithmes d'extension de domaine. Nous étudions ici le mode Chop-MD qui est un cas particulier de la construction wide pipe décrite au chapitre 2 et que nous représentons en Figure 6.1, et adaptons les notations concernant $\mathcal{F}$ en conséquence.

$$\begin{aligned} \mathcal{F} : \{0,1\}^{\ell_m} \times \{0,1\}^n &\to \{0,1\}^n \\ (M, A, B) &\mapsto \mathcal{F}_M(A, B) = (A', B') \ . \end{aligned}$$

La variable M représente le bloc de message, la variable B' représente la partie de la sortie de $\mathcal{F}$ qui constitue le haché après la dernière itération de la fonction de compression, et la variable A' représente le reste de la sortie de $\mathcal{F}$. Si $\mathcal{F}_M(x) = (A', B')$, avec $B' \in \{0,1\}^{\ell_h}$ et $A' \in \{0,1\}^{n-\ell_h}$, nous noterons

$$\begin{aligned} \mathcal{F}_M^{(A)}(x) &= A' \\ \mathcal{F}_M^{(B)}(x) &= B' \end{aligned}$$

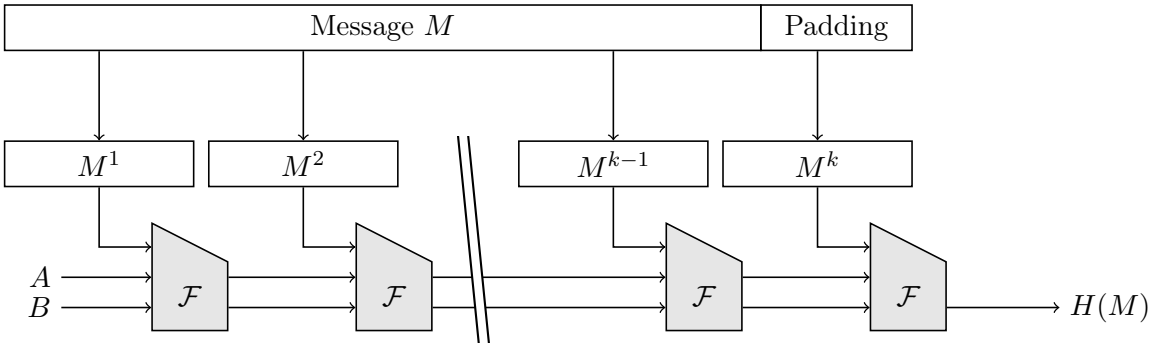


FIGURE 6.1 – Construction Chop-MD

6.2.2.1 Limitation du biais sur $\mathcal{F}$

Les parties A' et B' de la sortie ayant un rôle différent au cours du calcul de haché, il est naturel de considérer que leurs distributions statistiques respectives ont des impacts différents sur la sécurité de la construction. Plus précisément, B' étant la partie de la sortie de $\mathcal{F}$ correspondant au haché, seul un faible biais par rapport à la distribution uniforme peut être toléré. En revanche, les contraintes sur la distribution de A' peuvent être moins restrictives. De ce fait, nous définissons deux grandeurs différentes, ε et τ, pour caractériser le biais sur la primitive interne. Ces idées sont formalisées dans la suite de cette section.

Rôle de ε. Pour certaines requêtes à $\mathcal{F}$, la partie B' de la sortie peut être une valeur de haché. Dans ce cas, même un faible biais par rapport à la distribution uniforme peut permettre de différencier $\mathcal{C}^{\mathcal{F}}$ d'un oracle aléatoire. Nous cherchons donc à garantir que pour toute requête à $\mathcal{F}$ impliquée dans le calcul d'un haché, la distance statistique entre la distribution uniforme sur B' et la distribution renvoyée par $\mathcal{F}$ reste inférieure à ε.

Rôle de τ. Certaines attaques pourraient également tirer parti du biais sur la partie A' de la sortie, un léger biais peut toutefois être toléré sur cette valeur. De manière générale, nous cherchons à garantir qu'au cours du calcul d'un haché,

$$\Pr\left[\mathcal{F}_M(A,B) = (A',B')\right] \leq 2^{-n+\tau}$$

6.2.2.2 Relatifs d'un ensemble d'entrées

Dans la pratique, les attaques par distingueur induisent rarement un petit biais statistique sur toutes les sorties de $\mathcal{F}$. Le plus souvent, ces attaques exploitent des biais plus importants pour certaines entrées particulières, voire des corrélations entre les sorties correspondant à des entrées données. De ce fait, la connaissance des sorties pour certaines entrées entraîne un biais important sur d'autres sorties de $\mathcal{F}$.

C'est notamment le cas des distingueurs différentiels. Supposons que pour des valeurs de δ_M, δ_A, δ_B, $\delta_{A'}$ et $\delta_{B'}$ fixées, on ait pour un choix uniforme de M, A et B :

$$\Pr\left[\mathcal{F}_{M\oplus\delta_M}(A\oplus\delta_A, B\oplus\delta_B)\oplus F_M(A,B) = (\delta_{A'},\delta_{B'})\right] = p$$

Si $p > \varepsilon$, une fois déterminée la valeur de $\mathcal{F}_M(A,B)$, le biais sur $\mathcal{F}_{M\oplus\delta_M}(A\oplus\delta_A, B\oplus\delta_B)$ dépasse ε, et les bornes définies plus haut ne sont plus valables. De même, les distingueurs à somme nulle sont définis comme des corrélations entre les valeurs de sorties correspondant à un ensemble d'entrées. Le cas d'entrées ainsi corrélées doit donc être traité de manière spécifique par le simulateur.

Tout d'abord, nous définissons l'ensemble des entrées d'un historique.

Définition 1 (Ensemble des entrées d'un historique $\mathcal{L}$) *Soit $\mathcal{L}$ un historique, c'est-à-dire un ensemble d'entrées et de sorties de la fonction $\mathcal{F}$. Nous notons $I(\mathcal{L})$ l'ensemble des entrées de $\mathcal{L}$*

$$I(\mathcal{L}) = \{u \in \{0,1\}^{n+\ell_m} \text{ tels que } \exists (u,v) \in \mathcal{L}\}$$

Nous définissons maintenant les *relatifs* d'un historique $\mathcal{L}$, qui sont les valeurs de (M,A,B) dont l'image par une fonction $\mathcal{F}$ tirée selon la distribution $\mathcal{V}$ est affectée par la connaissance de $\mathcal{L}$.

Définition 2 ((ε,τ)-relatifs d'un historique) *Soit ε un réel dans $[0,1]$, et τ un réel de $[0, n-\ell_h]$. Soit $\mathcal{L}$ un ensemble de couples entrée-sortie de $\mathcal{F}$ dans $\{0,1\}^{n+\ell_m}\times\{0,1\}^n$. Nous définissons l'ensemble des (ε,τ)-relatifs de $\mathcal{L}$ pour la distribution $\mathcal{V}$ comme*

$$\begin{aligned}\texttt{REL}(\mathcal{L},\varepsilon,\tau) \quad = \quad & \{u=(M,A,B) \notin I(\mathcal{L}) \text{ tels que : } ||D_B(u,\mathcal{L}) - U|| > \varepsilon \\ & \text{ou } \exists (A',B') \in \{0,1\}^n \text{ tel que } \Pr\left[\mathcal{F}_M(A,B) = (A',B') | \mathcal{L}\right] > 2^{-n+\tau}\}\end{aligned}$$

où U est la distribution uniforme sur $\{0,1\}^{\ell_h}$ et

$$D_B(u,\mathcal{L}) = \Pr\left[\mathcal{F}^{(B)}(u) = B' | \mathcal{L}\right] .$$

Afin d'alléger l'écriture de la suite, nous faisons l'hypothèse que l'ensemble des relatifs de $\mathcal{L}$ ne dépend pas des sorties de $\mathcal{L}$. Cette restriction nous semble raisonnable. Par exemple, elle permet de tenir compte de tous les distingueurs connus sur Shabal présentés au chapitre 4. Nous énonçons maintenant la propriété suivante, sous cette hypothèse.

Propriété 6 ((ε,τ)-relatifs d'un ensemble d'entrées) *Pour toute paire d'historiques $\mathcal{L}$ et $\mathcal{L}'$ tels que $I(\mathcal{L}) = I(\mathcal{L}') = V$,*

$$\texttt{REL}(\mathcal{L},\varepsilon,\tau) = \texttt{REL}(\mathcal{L}',\varepsilon,\tau)\ .$$

Nous notons alors cet ensemble $\texttt{REL}(V,\varepsilon,\tau)$.

Par exemple, dans le cas du distingueur différentiel précédemment mentionné, cette propriété est vérifiée et

$\texttt{REL}(V,\varepsilon,\tau) \supset \{(M\oplus\delta_M, A\oplus\delta_A, B\oplus\delta_B, (M,A,B)\in V\}$ pour tout $\varepsilon < p$.

6.2.2.3 Entrées atypiques

Pour certains types de distingueurs, l'ensemble $\texttt{REL}(\emptyset,\varepsilon,\tau)$ peut être non vide. C'est le cas des distingueurs par points fixes. De telles propriétés de la fonction de compression de Shabal ont été découvertes dans [KMT09] : pour certaines entrées (M,A,B,C) bien définies, la sortie (A',B') vérifie $B' = B$. Dans ce cas de figure, une seule valeur de B' est possible en sortie de $\mathcal{F}$. De telles entrées jouent un rôle particulier dans la preuve. Nous les appelons *entrées atypiques*.

Définition 3 (Entrées (ε,τ)-atypiques.) *Soit ε un réel de $[0,1]$, et τ un réel de $[0, n-\ell_h]$. les* entrées (ε,τ)-atypiques *pour la distribution $\mathcal{V}$ sont les entrées*

$$\texttt{AT}(\varepsilon,\tau) = \texttt{REL}(\emptyset,\varepsilon,\tau)\ .$$

A contrario, nous appelons entrées (ε,τ)-typiques les autres entrées. Il est intéressant de noter que dans le cas particulier où $\varepsilon = 0$, les parties B' des sorties correspondant aux entrées $(0,\tau)$-typiques prennent toutes les valeurs possibles avec la même probabilité. Ce cas est étudié dans la section 6.4.

6.2.2.4 Relatifs d'une entrée

Pour chaque appel au simulateur, nous devons nous assurer qu'il n'y a pas d'incohérence entre les sorties correspondant aux relatifs de l'ensemble des requêtes précédentes. Pour y parvenir, nous définissons maintenant l'ensemble des relatifs d'une requête (ou de manière équivalente, d'une entrée de $\mathcal{F}$). En utilisant les définitions précédentes, nous définissons l'ensemble des ε,τ-relatifs de v relativement à V, pour tout $v \notin V$, de la manière suivante.

Définition 4 ((ε,τ)-relatifs d'une entrée relativement à un ensemble V) *Soit ε un réel de $[0,1]$, et τ un réel de $[0, n-\ell_h]$. Soit V un ensemble d'entrées dans $\{0,1\}^{n+\ell_m}$ et $v \in \{0,1\}^{n+\ell_m}$ une entrée n'appartenant pas à V. Nous définissons l'ensemble des (ε,τ)-relatifs de v relativement à V,*

$$\mathcal{R}_{asym}(v,V,\varepsilon,\tau)$$

comme

$$\mathcal{R}_{asym}(v,V,\varepsilon,\tau) = \texttt{REL}(\{v\}\cup V,\varepsilon,\tau) \setminus \texttt{REL}(V,\varepsilon,\tau).$$

Nous insistons sur le fait que cette définition ne suffit pas à garantir que $v' \in \texttt{REL}(v,V,\varepsilon,\tau)$ définit une relation transitive entre v et v'. Une relation non-transitive est obtenue notamment lorsqu'il existe un distingueur rotationnel sur $\mathcal{F}$, tel que celui décrit au chapitre 4. Soit ε tel que, pour tout $v \in I(\mathcal{L})$, $||D_B(v \lll 1) - U|| > \varepsilon$ et $||D_B(v \lll 2) - U|| \leq \varepsilon$. On en déduit que $(v \lll 2) \in \mathcal{R}_{asym}(v \lll 1, \emptyset, \varepsilon)$ et $(v \lll 1) \in \mathcal{R}_{asym}(v,\emptyset,\varepsilon,\tau)$, alors que $(v \lll 2) \notin \mathcal{R}_{asym}(v,\emptyset,\varepsilon,\tau)$. Cette

définition ne garantit pas non plus que $v' \in \mathcal{R}_{asym}(v, V, \varepsilon, \tau)$ définit une relation symétrique, d'où la notation $\mathcal{R}_{asym}$. Dans les preuves de sécurité des sections 6.3 et 6.4, nous devons nous assurer que l'ensemble des relatifs de v ne varie pas au cours de la simulation. Nous donnons maintenant une définition des relatifs de v qui ne dépend pas de l'historique.

Définition 5 ((ε, τ)-relatifs d'une entrée) *Soit ε un réel de $[0, 1]$, τ un réel de $[0, n]$, et $v \in \{0,1\}^{n+\ell_m}$ une entrée de $\mathcal{F}$. Nous définissons l'ensemble $\mathcal{R}(v, \varepsilon, \tau)$ des (ε, τ)-relatifs de v comme*

$$\begin{aligned} \mathcal{R}_{asym}(v, \varepsilon, \tau) &= \bigcup_{V \in INPUT(N)} \mathcal{R}_{asym}(v, V, \varepsilon, \tau) \\ \mathcal{R}(v, \varepsilon, \tau) &= \mathcal{R}_{asym}(v, \varepsilon, \tau) \cup \{u \text{ tel que } v \in \mathcal{R}_{asym}(u, \varepsilon, \tau)\} . \end{aligned}$$

où N est la taille maximale de l'historique et $INPUT(N) = \bigcup_{q \in \{0,\ldots,N\}} (\{0,1\}^{n+\ell_m})^q$ représente l'union des ensembles d'entrée de taille au plus N.

En particulier, cette définition garantit que tout relatif d'un ensemble d'entrées V est soit une entrée atypique, soit un relatif d'un élément de V.

Lemme 6 *Soit ε un réel de $[0, 1]$, τ un réel de $[0, n]$ et $V \in (\{0,1\}^{n+\ell_m})^q$ un ensemble de q entrées de $\mathcal{F}$, pour un certain $q \leq N$. Dans ce cas $v' \in \mathtt{REL}(V, \varepsilon, \tau)$ et $v' \notin \mathtt{AT}(\varepsilon, \tau)$ impliquent que $\exists v \in V$ tel que $v' \in \mathcal{R}(v, \varepsilon, \tau)$.*

Démonstration : Soit v' une telle entrée. Notons $V = \{v_1, \ldots, v_q\}$, $V_0 = \emptyset$ et $V_r = \{v_1, \ldots, v_r\}$ pour $r \leq q$. Nous considérons maintenant le plus petit entier t tel que $v' \in \mathtt{REL}(V_t, \varepsilon, \tau)$. Comme cette propriété est vraie pour q, t est toujours défini, et comme $v' \notin \mathtt{AT}(\varepsilon, \tau)$, $t > 0$. Nous savons alors que $v' \in \mathtt{REL}(V_{t-1} \cup \{v_t\}, \varepsilon, \tau)$ et $v' \notin \mathtt{REL}(V_{t-1}, \varepsilon, \tau)$, ce qui signifie que $v' \in \mathcal{R}(v_t, V_{t-1}, \varepsilon, \tau)$. On en déduit que $v' \in \mathcal{R}(v_t, \varepsilon, \tau)$. □

Nous pouvons remarquer que cette définition est plus générale que celle qui a été utilisée dans des travaux antérieurs comme [BCCM+08], où nous imposions que $\mathcal{R}_{asym}(v, V, \varepsilon, \tau)$ ne dépende pas de V. Il en résulte que cette nouvelle définition permet de prendre en compte un plus grand nombre d'attaques par distingueur. Dans la suite de ce chapitre, nous montrons que la sécurité de Chop-MD dépend de la taille maximale de $\mathcal{R}(v, \varepsilon, \tau)$. Informellement, si cette taille est trop grande, de nombreuses entrées v' peuvent être des relatifs d'une entrée v, soit au moment de son insertion, soit dans le futur. Notre technique de preuve reste valide, mais la borne qu'on peut en dériver sur la meilleure attaque n'apporte aucune garantie de sécurité. C'est notamment le cas des distingueurs à somme nulle sur les fonctions de compression de bas degré, pour lesquelles la somme des sorties correspondant aux entrées de tout sous-espace vectoriel de petite dimension de $\{0,1\}^{\ell_m+n}$ vaut 0.

6.2.2.5 Constantes relatives au biais sur $\mathcal{F}$.

Après avoir défini les notions de couples d'entrées de $\mathcal{F}$ relatives l'une de l'autre et d'entrées atypiques, nous définissons la valeur de certains paramètres associés au biais sur $\mathcal{F}$ et au choix de ε et τ. Les bornes d'indifférenciabilité que nous montrons dans la suite de ce chapitre dépendent de la valeur de ces paramètres.

Remarque. Par analogie avec le chapitre précédent, nous appelons *nœuds* les valeurs de la variable de chaînage. Cette notation est justifiée par les preuves par séquences de jeux décrites dans les chapitres suivants.

Paramètre $\mathcal{X}_{atyp}$. Le premier de ces paramètres est le nombre $\mathcal{X}_{atyp}$ de nœuds x tels qu'il existe M tel que (M, x) est une entrée atypique de $\mathcal{F}$. Nous avons

$$\mathcal{X}_{atyp} = |\{x \in \{0,1\}^n \text{ tels que } \exists M \in \{0,1\}^{\ell_m}, (M,x) \in \mathtt{AT}(\varepsilon,\tau)\}|$$

Paramètre $\mathcal{X}_{rel}$. D'autres nœuds singuliers sont ceux auxquels on peut adjoindre deux messages différents et obtenir des entrées relatives l'une de l'autre. Nous définissons donc

$$\mathcal{X}_{rel} = |\{x \in \{0,1\}^n \text{ tels que } \exists (M,M') \in (\{0,1\}^{\ell_m})^2, (M,x) \in \mathcal{R}((M',x),\varepsilon,\tau)\}|$$

Paramètre $\mathcal{R}_x$. Le paramètre $\mathcal{R}_x$ est défini comme le maximum sur les entrées (M, x) de $\mathcal{F}$ du cardinal d'un ensemble dépendant de (M, x). Il s'agit de l'ensemble des nœuds x' auxquels on peut adjoindre un bloc M' de manière à obtenir un relatif de (M, x). Nous avons donc

$$\mathcal{R}_x = \max_{(M,x)\in\{0,1\}^{\ell_m+n}} |\{x' \in \{0,1\}^n \text{ tels que } \exists M' \in \{0,1\}^{\ell_m}, (M,x) \in \mathcal{R}((M',x'),\varepsilon,\tau)\}|$$

Paramètre $\mathcal{R}_{indep}$. Le dernier des paramètres que nous définissons concerne les relatifs *deux à deux indépendants*. Deux relatifs (M_1, x_1) et (M_2, x_2) d'une même entrée (M, x) de $\mathcal{F}$ sont dits indépendants s'ils ne sont pas relatifs l'un de l'autre. $\mathcal{R}_{indep}$ est défini comme le cardinal du plus grand ensemble de relatifs deux à deux indépendants d'une même entrée de $\mathcal{F}$. Formellement, nous notons $\mathcal{T}$ l'ensemble des jeux d'entrées défini par

$$\begin{aligned} X \in \mathcal{T} \quad \Leftrightarrow \quad & \exists u^* \in \{0,1\}^{\ell_m+n} | \forall u \in X, u \in \mathcal{R}(u^*,\varepsilon,\tau) \\ & \text{et } \forall (u,v) \in X^2, u \notin \mathcal{R}(v,\varepsilon,\tau) \end{aligned}$$

Nous définissons

$$\mathcal{R}_{indep} = \max_{T\in\mathcal{T}} |T|.$$

6.2.2.6 Sous-routines du simulateur

Nous définissons maintenant certaines notions permettant de construire le simulateur.

Dans le cas biaisé, la fonction $\mathcal{F}$ est tirée aléatoirement selon une distribution $\mathcal{V}$. Cette distribution dépend des propriétés statistiques identifiées de la primitive. Ces propriétés ne sont pas explicitées dans la preuve. Il en résulte que la distribution $\mathcal{V}$ reste implicite. C'est donc également le cas des distributions des sorties de $\mathcal{F}$ lors de chaque requête. De ce fait, la simulation des sorties de $\mathcal{F}$, ainsi que certains tests, ne peuvent être spécifiés explicitement dans la preuve. Nous admettrons donc l'existence de sous-routines efficacement calculables effectuant les tests et les échantillonnages suivants.

- Un algorithme d'échantillonnage `Simule` qui tire une sortie $(A', B') = \mathcal{F}_M(A, B)$ pour toute entrée (M, A, B) et tout historique $\mathcal{L}$. Pour tout historique $\mathcal{L}$ tel que $\Pr_{\mathcal{F}\leftarrow\mathcal{V}}[\mathcal{L}] > 0$ et pour toute sortie (A', B'), nous imposons que

$$\Pr\left[\mathtt{Simule}(M,A,B,\mathcal{L}) = (A',B')\right] = \Pr_{\mathcal{F}\leftarrow\mathcal{V}}[\mathcal{F}_M(A,B) = (A',B')|\mathcal{L}]$$

– Un algorithme d'échantillonnage `A-Simule` permettant de calculer la partie A' d'une sortie de F pour un historique $\mathcal{L}$ donné, la partie B' ayant été fixée par l'oracle aléatoire. Pour toute entrée de $\mathcal{F}$ (M, A, B), toute sortie (A', B') et tout historique $\mathcal{L}$ tel que $\Pr_{\mathcal{F}\leftarrow\mathcal{V}}[\mathcal{L} \wedge \mathcal{F}_M^{(B)}(A, B) = B'] > 0$, nous imposons

$$\Pr\left[\texttt{A-Simule}(M, A, B, B', \mathcal{L}) = (A')\right] = \Pr_{\mathcal{F}\leftarrow\mathcal{V}}[\mathcal{F}_M^{(A)}(A, B) = A' | \mathcal{L} \wedge \mathcal{F}_M^{(B)}(A, B) = B']$$

– Au cours de la simulation, les entrées atypiques sont traitées de manière spécifique : comme la distribution statistique de la partie B des sorties correspondantes peut beaucoup différer de la distribution uniforme, elles peuvent être différenciées des sorties d'un oracle aléatoire. De ce fait, nous devons garantir que dans le graphe construit par le simulateur, il n'y a pas de nœud ayant un chemin dans le graphe qui puisse conduire à une entrée atypique par ajout d'un bloc de message M bien choisi. Nous admettons l'existence d'un algorithme (ε, τ)-`TestAtypique` qui permet de déterminer s'il existe M tel que (M, A, B) est une entrée (ε, τ)-atypique et $\mu||M$ est un préfixe de messages après application de la fonction d'encodage, étant données une valeur (A, B) de la variable de chaînage et une chaîne de blocs μ.

– L'idée de la preuve consiste à traiter chaque requête au simulateur en ajoutant non seulement l'image demandée au graphe, mais aussi les images des relatifs de l'entrée lorsque ceux-ci ont un chemin dans le graphe. Nous procédons ainsi car les valeurs de ces images peuvent être à la fois contraintes par la connaissance de la réponse à la requête et de l'historique, et par la nécessité de cohérence avec l'oracle aléatoire. Certains problèmes de cohérence par rapport à l'oracle aléatoire peuvent survenir au cours de la simulation. C'est notamment le cas lorsque deux nœuds (pas forcément distincts) x et x' ayant un chemin depuis x^0 vérifient que l'ajout de blocs de message bien choisis conduit à des entrées relatives l'une de l'autre. Nous traitons ce problème à l'aide de deux algorithmes. Nous supposons l'existence d'un algorithme efficace (ε, τ)-$\texttt{TestRelatifs}_1$ permettant de déterminer si pour deux variables de chaînage x, x' données, et deux suites de blocs μ et μ', il existe m, m' tels que $(m, x) \in \mathcal{R}((m', x'), \varepsilon, \tau)$ et $\mu||m$ et $\mu'||m'$ sont des préfixes de messages après application de la fonction d'encodage. Nous supposons aussi l'existence d'un second test assez similaire (ε, τ)-$\texttt{TestRelatifs}_2$, qui étant donnés deux variables de chaînage x et x', un bloc de message m' et une suite de blocs μ, détermine l'existence de m tel que $(m, x) \in \mathcal{R}((m', x'), \varepsilon, \tau)$ et $\mu||m$ est un préfixe d'un message après application de la fonction d'encodage.

6.3 Indifférenciabilité d'un oracle aléatoire public de la construction Chop-MD avec une fonction de compression biaisée

Dans cette section nous modifions la preuve d'indifférenciabilité de Chop-MD, afin de prendre en compte les distingueurs sur $\mathcal{F}$, qui est modélisée comme étant choisie aléatoirement sur FUNC selon une distribution $\mathcal{V}$. Nous obtenons un résultat d'indifférenciabilité d'un oracle aléatoire à usage public. Ce résultat est un cas particulier du travail effectué dans le cadre du projet Saphir2, dont l'objet est l'étude de la construction 5.1.

Pour simuler les réponses aux requêtes à $\mathcal{F}$ du distingueur, nous utilisons l'accès à l'oracle aléatoire, ainsi qu'à des algorithmes d'échantillonnage définis de manière implicite. Dans des modèles plus traditionnels, la distribution de $\mathcal{F}$ est bien définie, et les réponses aux requêtes sont construites de manière explicite. En particulier, il est possible que la réponse à la $q^{ème}$ requête ait déjà été définie au cours des interactions précédentes. $\mathcal{S}$ doit alors le détecter et renvoyer cette réponse. Dans

notre étude, les algorithmes d'échantillonnage agissent en fonction de l'historique du simulateur. Ils peuvent donc détecter si la réponse à une requête est déterminée par les échanges précédents, et cette éventualité ne constitue pas un évènement particulier à traiter à part.

Dans la suite, ε et τ sont des paramètres fixes du simulateur. Nous les omettons afin d'alléger l'écriture.

6.3.1 Principes de conception du simulateur

Les principes de conception du simulateur sont proches de ceux énoncés dans le chapitre précédent. Le simulateur garde en mémoire l'ensemble E des arcs $(A, B) \xrightarrow{M} (A', B')$ qu'il a précédemment simulés. Dans le cas de Chop-MD, la fonction d'insertion est l'identité, de ce fait l'existence d'un arc $(A, B) \xrightarrow{M} (A', B')$ correspond à la définition de $(A', B') = \mathcal{F}_M(A, B)$. Nous notons X l'ensemble des nœuds (A, B) d'où part un arc de E et Y l'ensemble des nœuds où arrive un arc de E. L'ensemble E des arcs forme alors un graphe $\mathcal{G}$ permettant de déterminer quelles valeurs de l'état interne peuvent être atteintes depuis l'origine x^0 en traitant une suite appropriée de blocs de message.

Les définitions de chemin, de chemin complet et de chemin utile données au chapitre précédent sont toujours valables. Nous ajoutons également au simulateur l'ensemble P, qui contient des couples (μ, x) où μ est un chemin utile et x est un nœud tel que μ est un chemin de x dans $\mathcal{G}$.

Contrairement au chapitre précédent, le simulateur reçoit à la fois les requêtes provenant de l'attaquant $\mathcal{D}$ et de la construction $\mathcal{C}$. Les simulateurs intermédiaires n'ont donc pas besoin de mémoriser l'origine de la requête. Dans le cas de la preuve d'indifférenciabilité forte donnée en section 6.4, cette distinction réapparaît.

6.3.2 Description du simulateur

Nous considérons maintenant le simulateur composé par les algorithmes $\mathcal{I}$ et $\mathcal{S}$ pour $\mathcal{F}$ décrits en Figures 6.2 et 6.3. $\mathcal{I}$ reçoit des valeurs de messages émises par $\mathcal{H}^{reveal}$ et envoie des requêtes à $\mathcal{S}$. $\mathcal{S}$ se charge de maintenir la cohérence avec l'oracle aléatoire et d'utiliser les sous routines `A-Simule` et `Simule`. Pour ce simulateur, une variante du théorème 2 peut être prouvée.

Théorème 8 *Soit $\mathcal{F}$ une fonction choisie dans* FUNC *selon une distribution $\mathcal{V}$, et soit $\mathcal{H}$ un oracle aléatoire à usage public. Supposons que l'état initial x^0 soit tel que* `TestAtypique`(x^0) *et* `TestRelatifs`$_1(x^0, \emptyset, x^0, \emptyset)$ *renvoient FAUX. Dans ce cas, le simulateur $(\mathcal{I}, \mathcal{S})$ défini en Figures 6.2 et 6.3 est tel que pour tout attaquant $\mathcal{D}$ émettant des requêtes d'un poids total d'au plus N,*

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep})(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 + \mathcal{R}_{indep}((\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} + 2\varepsilon) N$$

si $\mathcal{R}_{indep} \geq 1$, et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)} \frac{N(N+1)}{2} + (2\varepsilon + \mathcal{X}_{atyp} 2^{-(n-\tau)}) N$$

si $\mathcal{R}_{indep} = 0$.

6.3.3 Esquisse de la preuve par séquence de jeux

Le déroulement de la séquence de jeux permettant de montrer le théorème 8 repose sur une stratégie proche du début de la séquence de jeux que nous utilisons au chapitre 5. Le système $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ avec lequel $\mathcal{D}$ interagit est modifié progressivement afin d'aboutir à $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. Cette évolution est faite de la manière suivante.

Initialisation de $\mathcal{S}$

1. choisir $\varepsilon \in [0,1]$ et $\tau \in [0, n-\ell_h]$
2. définir $X = \{x^0\}$ and $E = \emptyset$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrée : (M, A, B)

Sortie : (A', B')

1. Pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \{(M, A, B)\} \cup \mathcal{R}(M, A, B, \varepsilon, \tau)$
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) si $\tilde{x}$ a un chemin $\tilde{\mu}$ dans $\mathcal{G}$ et si $\tilde{\mu}||\tilde{M}$ est un chemin utile
      i. si $\tilde{\mu}$ n'est pas unique, arrêter $\mathcal{S}$
      ii. calculer $\mathcal{M} = \mathsf{unpad}(\tilde{\mu}||\tilde{M})$
      iii. appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
      iv. définir $\tilde{B}' = h$
      v. lancer A-Simule$(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{B}', E)$ pour obtenir $\tilde{A}'$
   (c) sinon
      i. lancer Simule$(\tilde{M}, \tilde{A}, \tilde{B}, E)$ pour obtenir $(\tilde{A}', \tilde{B}')$
      ii. définir $\tilde{y} = (\tilde{A}', \tilde{B}')$
   (d) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}$ à E
2. lancer Simule(M, A, B, E) pour obtenir (A', B')
3. définir $y = (A', B')$
4. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
5. renvoyer (A', B')

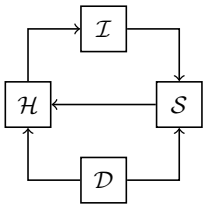


FIGURE 6.2 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ pour la preuve d'indifférenciabilité d'un oracle aléatoire à usage public de Chop-MD dans le cas biaisé.

Traitement d'une information venant de $\mathcal{H}^{reveal}$

Entrée : $\mu \in \{0,1\}^*$

Pas de sortie

1. calculer $(M^1||\ldots||M^\ell) = \mathsf{pad}(\mu)$
2. définir $(A^0, B^0) = x^0$
3. pour i de 1 à ℓ
   (a) appeler $\mathcal{S}$ pour obtenir $(A^i, B^i) = \mathcal{F}_{M^i}(A^{i-1}, B^{i-1})$

FIGURE 6.3 – Intercepteur $\mathcal{I}$ pour la preuve d'indifférenciabilité à l'oracle aléatoire public de Chop-MD.

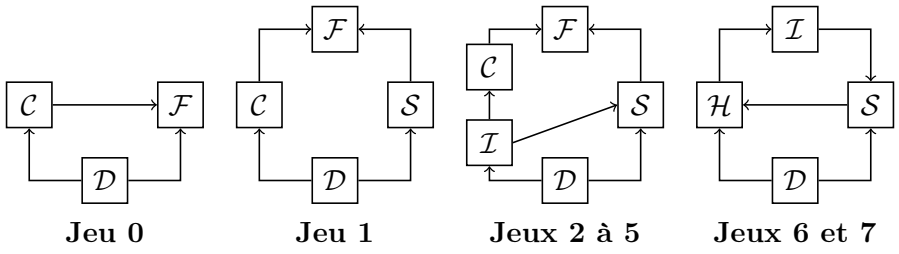


FIGURE 6.4 – Construction du simulateur $\mathcal{S}$.

Jeu 0 Nous commençons avec le jeu d'origine. Dans ce jeu le distingueur interagit avec un système $\Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ qui représente notre construction $\mathcal{C}$ appliquée à une fonction aléatoire $\mathcal{F}$.

Jeu 1 Un simulateur $\mathcal{S}_{1,2}$ est introduit dans le jeu à la place de la fonction aléatoire $\mathcal{F}$. Il se contente de transmettre les requêtes à $\mathcal{F}$ et de renvoyer les réponses à $\mathcal{D}$. Du point de vue de $\mathcal{D}$, ce jeu n'a aucune différence avec le précédent.

Jeux 2 à 5 Un programme appelé intercepteur $\mathcal{I}$ est ajouté au scénario. Il permet aux simulateurs intermédiaires $\mathcal{S}_i$ d'accéder aux requêtes de haché émises par $\mathcal{D}$. Contrairement au chapitre précédent, les simulateurs intermédiaires doivent prendre en compte l'existence de biais sur la fonction de compression $\mathcal{F}$. Pour cela, lors de la réception d'une requête de la forme (M, A, B), $\mathcal{S}$ émet une requête à $\mathcal{F}$ pour obtenir $\mathcal{F}_M(A, B)$ et demande certaines valeurs de $\mathcal{F}_{M^*}(A^*, B^*)$ où (M^*, A^*, B^*) est un relatif de (M, A, B). Ceci permet au simulateur de détecter des cas d'échec. Cette manière d'anticiper la simulation de valeurs de $\mathcal{F}$ dont la distribution pourrait être biaisée est la principale nouveauté de cette séquence de jeux par rapport à celles du chapitre précédent.

Jeu 6 L'accès à la fonction de compression $\mathcal{F}$ est remplacé par un accès à un oracle aléatoire à usage public $\mathcal{H}$. $\mathcal{S}$ simule maintenant les réponses aux requêtes à la fonction de compression. Lorsque $\mathcal{H}$ reçoit une requête venant de $\mathcal{D}$ pour $\mathcal{H}(M)$, il envoie M au simulateur. L'intercepteur $\mathcal{I}$ est utilisé pour traduire M en une succession d'appels à la fonction de compression.

Jeu 7 L'intercepteur est intégré au simulateur, et la détection de cas d'échec est supprimée. Dans ce scénario, $\mathcal{D}$ interagit avec $\Sigma' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

Cette stratégie est résumée sur la Figure 6.4.

6.3.4 Construction de la séquence de jeux

Afin de démontrer le théorème 8, nous adaptons maintenant au cas biaisé la preuve par séquence de jeux présentée au chapitre précédent.

Jeu 0. Il s'agit du jeu d'origine, dans lequel $\mathcal{D}$ interagit avec $\mathcal{C}^{\mathcal{F}}$ et la fonction $\mathcal{F}$, choisie dans FUNC selon la distribution $\mathcal{V}$. Par définition du Jeu 0, on a

$$\Pr[W_0] = \Pr\left[\mathcal{D}^{\Sigma} = 1 \mid \Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})\right] .$$

Jeu 1. Un simulateur passif $\mathcal{S}_1$ est inséré dans le Jeu 1. $\mathcal{S}_1$ transmet les requêtes qu'il reçoit à $\mathcal{F}$ et renvoie les réponses de $\mathcal{F}$ à $\mathcal{D}$. Au cours du jeu, $\mathcal{S}_1$ construit le graphe $\mathcal{G}$ résultant des requêtes

reçues. $\mathcal{S}_1$ est décrit en Figure 6.5. La vue de l'attaquant dans ce jeu est identique à celle du jeu précédent, d'où

$$\Pr[W_1] = \Pr[W_0] \tag{6.1}$$

Initialisation de $\mathcal{S}_1$

1. définir $X = \{x^0\}$ et $E = \emptyset$

Simulation de $\mathcal{F}$

Entrée : (M, A, B)

Sortie : (A', B')

1. définir $x = (M, A, B)$
2. appeler $\mathcal{F}$ pour obtenir $(A', B') = \mathcal{F}(M, A, B)$
3. définir $y = (A', B')$
4. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
5. renvoyer (A', B')

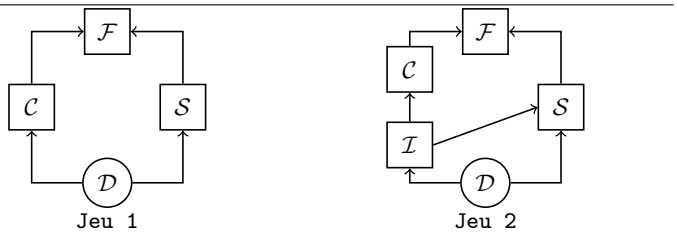


FIGURE 6.5 – Simulateur $\mathcal{S}_{1,2}$ dans les Jeux 1 et 2.

Jeu 2. Le Jeu 2 correspond à l'insertion de l'intercepteur $\mathcal{I}$ dans la séquence de jeux. Lorsque $\mathcal{D}$ émet une requête de haché, $\mathcal{I}$ intercepte cette requête avant la construction $\mathcal{C}$ et exécute la construction $\mathcal{C}^{\mathcal{F}}(M)$ à l'aide d'appels à $\mathcal{S}_{1,2}$. Le simulateur $\mathcal{S}_{1,2}$ utilisé au Jeu 2 est le même que celui du Jeu 1. L'intercepteur est décrit en Figure 6.6.

L'insertion de l'intercepteur ne modifie pas la vue de $\mathcal{D}$, d'où

$$\Pr[W_2] = \Pr[W_1] \ . \tag{6.2}$$

Jeu 3. Nous introduisons maintenant une modification significative par rapport à la preuve précédente. Par rapport au Jeu 2, cette modification est la suivante. Avant d'insérer l'arc $x \xrightarrow{M} y$ avec $x = (A, B)$, nous insérons également dans le graphe tous les arcs définissant les images par $\mathcal{F}$ des (ε, τ)-relatifs $(\tilde{M}, \tilde{A}, \tilde{B})$ de la requête (M, A, B) vérifiant que $(\tilde{A}, \tilde{B})$ a un chemin dans le graphe $\mathcal{G}$.

Nous introduisons également un ensemble $\mathcal{P}$ de nœuds ayant un chemin utile dans $\mathcal{G}$. $\mathcal{S}_3$ maintient une liste $\mathcal{P}$ de nœuds ayant un chemin utile dans $\mathcal{G}$. Pour ce faire, $\mathcal{S}_3$ procède de la manière suivante. $\mathcal{P}$ est initialisé à $\{(x^0, \emptyset)\}$ où $\emptyset$ représente le chemin vide. Au moment d'insérer un arc $x \xrightarrow{M} y$ dans E, pour tout c tel que $(x, c) \in \mathcal{P}$, si $c||M$ est un chemin utile, $\mathcal{S}_3$ ajoute $(y, c||M)$ à $\mathcal{P}$. Le simulateur correspondant est décrit en Figure 6.7.

Traitement d'une requête à $\mathcal{H}$

Entrée : $\mu \in \{0,1\}^*$

Sortie : $h = \mathcal{H}(\mu)$

1. calculer $(M^1||\ldots||M^\ell) = \mathsf{pad}(\mu)$
2. définir $(A^0, B^0) = x^0$
3. pour i de 1 à ℓ
   (a) appeler $\mathcal{S}$ pour obtenir $(A^i, B^i) = \mathcal{F}_{M^i}(A^{i-1}, B^{i-1})$
4. dans les Jeux 2 à 5, appeler $\mathcal{C}$ pour obtenir $h = \mathcal{H}(\mu)$
5. dans les Jeux 2 à 5, renvoyer h à $\mathcal{D}$

FIGURE 6.6 – Intercepteur $\mathcal{I}_{2-7}$ pour la preuve d'indifférenciabilité à l'oracle aléatoire public de Chop-MD.

La raison d'être de ce Jeu est de préparer l'introduction de l'oracle aléatoire $\mathcal{H}$, en anticipant les biais éventuels pouvant intervenir lorsque des entrées relatives de $\mathcal{F}$ doivent être traitées successivement par $\mathcal{S}$.

Par rapport à $\mathcal{S}_{1,2}$, $\mathcal{S}_3$ fait donc des requêtes supplémentaires à $\mathcal{F}$ pour traiter chaque requête provenant de $\mathcal{I}$ ou $\mathcal{D}$. Les réponses qu'il renvoie à $\mathcal{D}$ ou à $\mathcal{I}$ sont néanmoins celles qu'il a reçues de $\mathcal{F}$. La vue de l'attaquant n'est donc pas modifiée par rapport au Jeu 2, d'où

$$\Pr[W_3] = \Pr[W_2] \ . \tag{6.3}$$

Jeu 4. Comme dans les preuves du chapitre 5, nous introduisons maintenant des cas d'échec permettant d'identifier les collisions internes, c'est-à-dire le fait qu'un nœud x ait plusieurs chemins utiles distincts dans $\mathcal{G}$, et la prédéfinition de hachés, c'est-à-dire le fait qu'une valeur de haché soit entièrement déterminée par un chemin dans $\mathcal{G}$ et que le dernier arc de ce chemin ne soit pas le dernier à avoir été inséré dans $\mathcal{G}$. Les cas d'échec définis sont identiques aux évènements Echec_1 et Echec_2 de la preuve dans le cas idéal. Nous les notons ici Echec_{a1} et Echec_{a2}. Le simulateur ainsi obtenu est décrit sur la Figure 6.8. Les étapes 3 et 4 ne sont utiles que si x n'a pas de chemin dans $\mathcal{G}$: dans le cas contraire, la valeur de $\mathcal{F}_M(x)$ est générée à l'étape 2. Il n'est donc pas nécessaire de tester les cas d'échec à cet endroit.

A l'exception de la prise en compte d'évènements d'échec, le Jeu 4 est identique au Jeu 3. Nous avons donc

$$\Pr[W_4] = \Pr[W_3] \ . \tag{6.4}$$

Nous montrons maintenant les propriétés suivantes pour le simulateur du Jeu 4.

Propriété 7 *Soit $\mathcal{G}_p$ le graphe construit dans le Jeu 4 par le simulateur $\mathcal{S}_4$ décrit en Figure 6.8 après avoir effectué p requêtes à $\mathcal{F}$. Soit $x \neq x^0$ un nœud ayant un chemin utile dans le graphe final $\mathcal{G}_f$, et $y \xrightarrow{M} x$ le dernier arc de ce chemin. Supposons que Drapeau = Succès à la fin du Jeu.*

Sous ces hypothèses, x a un chemin utile dans $\mathcal{G}_q$, où $q \in \{1,\ldots,\ell\}$ est l'indice de la requête au cours de laquelle $y \xrightarrow{M} x$ est inséré dans $\mathcal{G}$.

Initialisation de $\mathcal{S}_3$

1. définir $X = \{x^0\}$, $E = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$

Simulation de $\mathcal{F}$

Entrées : (M, A, B)

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, \varepsilon, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}$ tel que $\mu||\tilde{M}$ est un chemin utile
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) appeler $\mathcal{F}$ pour obtenir $\tilde{y} = \mathcal{F}_{\tilde{M}}(\tilde{A}, \tilde{B})$
   (c) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \overset{\tilde{M}}{\to} \tilde{y}$ à E
   (d) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c||\tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c||\tilde{M})$ à $\mathcal{P}$
3. appeler $\mathcal{F}$ pour obtenir $y = \mathcal{F}_M(A, B)$
4. ajouter les nœuds x et y à X et l'arc $x \overset{M}{\to} y$ à E
5. renvoyer $(A', B') = y$.

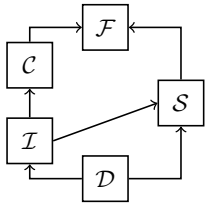


FIGURE 6.7 – Indifférenciabilité : Simulateur $\mathcal{S}_3$ pour $\mathcal{F}$ au Jeu 3 dans le cas biaisé.

Propriété 8 *Soient (M, A, B) l'entrée de la $q^{ème}$ requête à $\mathcal{F}$ par $\mathcal{S}$, et $x = (A, B)$. Si Drapeau = Succès à la fin du Jeu, l'une des deux propositions suivantes est vraie :*

- *x a un unique chemin utile μ dans le graphe $\mathcal{G}$ avant la $q^{ème}$ simulation et avant toutes les simulations effectuées par $\mathcal{S}$ dans la suite du Jeu 4.*
- *x n'a pas de chemin utile dans le graphe $\mathcal{G}$ avant la $q^{ème}$ simulation, et n'en acquiert pas dans la suite du Jeu 4.*

Démonstration : La preuve des propriétés 7 et 8 est identique à celle des propriétés 1 et 2 montrées au chapitre 5. □

Propriété 9 *Soit $\mathcal{G}_f$ le graphe construit par $\mathcal{S}_4$ à la fin du Jeu 4. Supposons que Drapeau = Succès. Si x a un chemin utile μ_x dans $\mathcal{G}$, (x, μ_x) est dans $\mathcal{P}$.*

Démonstration : La propriété 7 permet de garantir que si Drapeau = Succès et si un nœud x a un chemin utile μ_x dans le graphe, le dernier arc $y \overset{M}{\to} x$ de ce chemin est le dernier à être inséré dans E. Soit μ_y tel quel $\mu_x = \mu_y||M$. Si (y, μ_y) est dans $\mathcal{P}$, alors (x, μ_x) est ajouté à $\mathcal{P}$. Un raisonnement par récurrence sur la longueur du chemin μ_x permet de conclure que (x, μ_x) est dans $\mathcal{P}$. □

Jeu 5. Nous introduisons maintenant dans le simulateur des évènements d'échec supplémentaires, qui permettent de détecter qu'une valeur de haché non encore calculée est biaisée, du fait des valeurs

Initialisation de $\mathcal{S}_4$

1. choisir $\varepsilon \in [0,1]$ et $\tau \in [0, n - \ell_h]$
2. définir $X = \{x^0\}$, $E = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrées : (M, A, B)

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, \varepsilon, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}$ tel que $\mu||\tilde{M}$ est un chemin utile
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) appeler $\mathcal{F}$ pour obtenir $\tilde{y} = \mathcal{F}_{\tilde{M}}(\tilde{A}, \tilde{B})$
   (c) si $\tilde{x} \overset{\tilde{M}}{\to} \tilde{y} \notin E$
      i. si $\tilde{y}$ a un chemin utile $\tilde{\mu}'$ dans $\mathcal{G}$, NoteEchec(Echec_{a1})
      ii. s'il existe un arc $\tilde{y} \overset{\tilde{M}'}{\to} \tilde{z}$ dans E tel que $\mu||\tilde{M}||\tilde{M}'$ est un chemin utile, NoteEchec(Echec_{a2})
   (d) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \overset{\tilde{M}}{\to} \tilde{y}$ à E
   (e) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c||\tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c||\tilde{M})$ à $\mathcal{P}$
3. appeler $\mathcal{F}$ pour obtenir $y = \mathcal{F}_M(A, B)$
4. ajouter les nœuds x et y à X et l'arc $x \overset{M}{\to} y$ à E
5. renvoyer $(A', B') = y$.

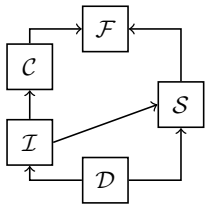


FIGURE 6.8 – Indifférenciabilité : Simulateur $\mathcal{S}_4$ pour $\mathcal{F}$ au Jeu 4 dans le cas biaisé.

de $\mathcal{F}$ déjà connues. $\mathcal{F}$ étant tirée selon une distribution $\mathcal{V}$ non uniforme, ses valeurs peuvent être corrélées et une valeur de haché peut être biaisée sans pour autant être complètement définie dans le graphe.

Echec_b. Supposons qu'au moment d'ajouter (x, μ_x) dans $\mathcal{P}$, il existe une entrée (ε, τ)-atypique de $\mathcal{F}$ de la forme (x, M) tel que $\mu_x||M$ soit un chemin utile. La valeur de B' telle que $(A', B') = \mathcal{F}_M(x)$ est alors biaisée. Or, cette valeur peut être une valeur de haché. Si un tel évènement se produit, $\mathcal{S}_5$ note Echec_b. Il le détecte grâce au test $\texttt{TestAtypique}(x, \mu_x)$.

Echec_{c1}. De manière similaire, supposons que deux nœuds x et y du graphe aient des chemins utiles μ_x et μ_y, et qu'il existe M_x et M_y tels que $\mu_x||M_x$ et $\mu_y||M_y$ soient des chemins utiles et $(x, M_x) \in \mathcal{R}(y, M_y)$. Les deux sorties de $\mathcal{F}$ correspondantes sont corrélées, or elles peuvent contenir des valeurs de haché. Afin d'éviter cette situation, $\mathcal{S}_5$ note un évènement Echec_{c1} lorsque le deuxième des couples (x, μ_x) et (y, μ_y) est inséré au graphe. Cet évènement est détecté grâce à $\texttt{TestRelatifs}_1(x, \mu_x, y, \mu_y)$.

Echec_{c2}. Un cas assez similaire à Echec_{c1} se produit si au moment d'insérer (x, μ_x) dans $\mathcal{P}$, il existe un bloc M_x tel que $\mu_x||M_x$ soit un chemin utile et un arc $y \overset{M_y}{\rightarrow} z$ dans E tel que $(x, M_x) \in \mathcal{R}(y, M_y)$. Dans ce cas, la valeur de sortie de $\mathcal{F}_{M_x}(x)$ est déjà contrainte. $\mathcal{S}_5$ note donc un évènement Echec_{c2}, grâce au test $\texttt{TestRelatifs}_2(x, \mu_x, y, M_y)$.

Echec_d. Un cas particulier de Echec_{c1} se produit lorsque $x = y$. Ce cas implique deux relatifs de la forme (x, M) et (x, M'). Il est traité à part par la levée d'un évènement Echec_d à l'aide du test $\texttt{TestRelatifs}_1(x, \mu_x, x, \mu_x)$.

Le simulateur ainsi obtenu est décrit en Figure 6.9. Le Jeu 5 rajoute des cas d'échec par rapport au Jeu 4, mais la vue de l'attaquant n'est pas modifiée, d'où

$$\Pr[W_5] = \Pr[W_4] \ . \tag{6.5}$$

Nous montrons maintenant les propriétés suivantes sur le graphe construit au Jeu 5.

Propriété 10 *Soit $\mathcal{G}$ le graphe construit par le simulateur décrit en Figure 6.9 après N requêtes. Soit (M, A, B) un élément de $\{0,1\}^{\ell_m+n}$ tel que (A, B) ait un chemin dans $\mathcal{G}$. Si (ε, τ)-$\texttt{TestRelatifs}_1(x^0, \emptyset, x^0, \emptyset)$ est faux, il n'y a pas d'élément $(M', A', B') \in \mathcal{R}((M, A, B), \varepsilon, \tau)$ tel que (A', B') ait un chemin dans $\mathcal{G}$.*

Démonstration : Soient (A, B) et (A', B') deux nœuds ayant des chemins μ, μ' dans le graphe $\mathcal{G}$ tels qu'il existe M, M' tels que $(M', A', B') \in \mathcal{R}((M, A, B), \varepsilon, \tau)$. Supposons que cette situation se produise pour la première fois après la $q^{ème}$ simulation de $\mathcal{F}$. Les ensembles de relatifs ne changeant pas au cours de la simulation, au moins l'un des nœuds (A, B), (A', B') est inséré dans le graphe au cours de la $q^{ème}$ simulation. Sans perte de généralité, supposons qu'il s'agit de (A, B).
Nous commençons par traiter le cas où $(A, B) \neq (A', B')$. De la propriété 8, nous déduisons que le chemin de (A, B) dans $\mathcal{G}$ a été complété par l'insertion de (A, B), au cours de la $q^{ème}$ simulation. A ce moment, (A', B') a déjà un chemin dans le graphe (d'après la propriété 8). Au moment de l'insertion de (A, B), $(\varepsilon, \tau) - \texttt{TestRelatifs}_1((A, B), \mu, (A', B'), \mu')$ est vrai, donc l'évènement Echec_{c_1} se produit.

Initialisation de $\mathcal{S}_5$

1. choisir $\varepsilon \in [0,1]$ et $\tau \in [0, n-\ell_h]$
2. définir $X = \{x^0\}$, $E = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrées : (M, A, B)

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, \varepsilon, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}$ tel que $\mu||\tilde{M}$ est un chemin utile
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) appeler $\mathcal{F}$ pour obtenir $\tilde{y} = \mathcal{F}_{\tilde{M}}(\tilde{A}, \tilde{B})$
   (c) si $\tilde{x} \overset{\tilde{M}}{\to} \tilde{y} \notin E$
      i. si $\tilde{y}$ a un chemin utile $\tilde{\mu}'$ dans $\mathcal{G}$, NoteEchec(Echec_{a1})
      ii. s'il existe un arc $\tilde{y} \overset{\tilde{M}'}{\to} \tilde{z}$ dans E tel que $\mu||\tilde{M}||\tilde{M}'$ est un chemin utile, NoteEchec(Echec_{a2})
      iii. si $(\varepsilon, \tau) -$ TestAtypique$(\tilde{y}, \mu||\tilde{M})$, NoteEchec($\mathsf{Echec}_b$)
      iv. s'il existe (z, μ_z) dans $\mathcal{P}$ tel que $(\varepsilon, \tau) -$ TestRelatifs$_1(\tilde{y}, \mu||\tilde{M}, z, \mu_z)$, NoteEchec($\mathsf{Echec}_{c1}$)
      v. s'il existe $z \overset{M_z}{\to} t$ dans E tel que $(\varepsilon, \tau) -$ TestRelatifs$_2(\tilde{y}, \mu||\tilde{M}, z, M_z)$, NoteEchec($\mathsf{Echec}_{c2}$)
      vi. si $(\varepsilon, \tau) -$ TestRelatifs$_1(\tilde{y}, \mu||\tilde{M}, \tilde{y}, \mu||\tilde{M})$,NoteEchec($\mathsf{Echec}_d$)
   (d) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \overset{\tilde{M}}{\to} \tilde{y}$ à E
   (e) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c||\tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c||\tilde{M})$ à $\mathcal{P}$
3. appeler $\mathcal{F}$ pour obtenir $y = \mathcal{F}_M(A, B)$
4. ajouter les nœuds x et y à X et l'arc $x \overset{M}{\to} y$ à E
5. renvoyer $(A', B') = y$.

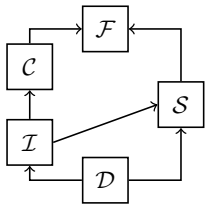


FIGURE 6.9 – Indifférenciabilité : Simulateur $\mathcal{S}_5$ pour $\mathcal{F}$ au Jeu 5 dans le cas biaisé.

Si au contraire $(A, B) = (A', B')$, $(A, B) \neq x^0$ car $(\varepsilon, \tau) - \texttt{TestRelatifs}_1(x^0, \emptyset, x^0, \emptyset)$ est faux. (A, B) est donc inséré dans le graphe au cours d'une simulation. (A, B) a un chemin dans le graphe au moment de son insertion, et $(\varepsilon, \tau) - \texttt{TestRelatifs}_1((A, B), \mu, (A, B), \mu)$ est vrai et Echec_d se produit.
De tels nœuds (A, B) et (A', B') ne peuvent donc pas exister, ce qui achève la démonstration.

□

Propriété 11 *Soit $\mathcal{G}$ le graphe construit par le simulateur décrit en Figure 6.9 après N requêtes. Si ε, τ-$\texttt{TestAtypique}(x^0, \emptyset)$ est faux, alors pour tout $(M, A, B) \in \{0,1\}^{\ell_m+n}$ tel que (A, B) a un chemin utile μ dans $\mathcal{G}$, (ε, τ)-$\texttt{TestAtypique}((A, B), \mu)$ est faux.*

Démonstration : Cette propriété découle de la propriété 8. Soit (A, B) un nœud ayant un chemin utile μ dans $\mathcal{G}$, différent de x^0. Au moment de l'insertion de (A, B), μ était déjà complètement défini. Si (ε, τ)-$\texttt{TestAtypique}((A, B), \mu)$ est vrai, alors Echec_b se produit et le simulateur s'arrête. □

Nous montrons enfin la propriété suivante, qui borne le nombre d'arcs du graphe $\mathcal{G}$ au Jeu 5 lorsqu'aucun évènement d'échec ne s'est produit.

Propriété 12 *Soit $\mathcal{S}_5$ le simulateur défini par la Figure 6.9,et $\mathcal{G}(q) = (X(q), E(q))$ le graphe construit avant la $q^{ème}$ requête reçue par $\mathcal{S}_5$. Si Drapeau = Succès avant la $q^{ème}$ requête, le nombre d'arcs dans le graphe vérifie*

$$|E(q)| \leq (q-1)(\mathcal{R}_{indep} + 1) \ .$$

Le nombre de nœuds ayant un chemin utile dans $\mathcal{G}$ est $|\mathcal{P}(q)|$ et vérifie

$$|\mathcal{P}(q)| \leq (q-1)\max(\mathcal{R}_{indep}, 1) + 1 \ .$$

Démonstration : Soit (M, x) une requête reçue par $\mathcal{S}_5$. Si x a un chemin utile μ dans $\mathcal{G}(q)$ et si $\mu||M$ est un chemin utile, la propriété 10 montre qu'aucun ε, τ-relatif de x ne peut avoir de chemin utile dans $\mathcal{G}$. $\mathcal{S}$ émet la seule requête (M, x) à $\mathcal{F}$ et ajoute l'arc $x \xrightarrow{M} \mathcal{F}_M(x)$ à E.
Dans le cas contraire, nous notons $\mathcal{T}$ l'ensemble des entrées $(\tilde{x}, \tilde{M})$ telles que $\tilde{x}$ a un chemin utile $\tilde{\mu}$ dans $\mathcal{G}$ et $\tilde{\mu}||\tilde{M}$ est un chemin utile et que $\mathcal{S}_5$ soumet à $\mathcal{F}$. Ces entrées ont toutes (M, x) comme relatif commun, et la propriété 10 garantit qu'aucune paire d'éléments de $\mathcal{T}$ n'est composée d'entrées relatives l'une de l'autre. Par définition de $\mathcal{R}_{indep}$, nous avons donc $|\mathcal{T}| \leq \mathcal{R}_{indep}$. En ajoutant la requête pour $\mathcal{F}_M(x)$, nous avons $E(q+1) \leq E(q) + (\mathcal{R}_{indep} + 1)$.
Comme avant la première requête, $E(1) = 0$, nous en déduisons

$$|E(q)| \leq (q-1)(\mathcal{R}_{indep} + 1) \ .$$

La propriété 9 garantit que si x a un chemin utile μ dans $\mathcal{G}$, alors (μ, x) est dans $\mathcal{P}$. Par construction de $\mathcal{P}$, la réciproque est vraie. D'après la propriété 8, un tel chemin est unique. Le nombre de nœuds ayant un chemin dans $\mathcal{G}$ avant la $q^{ème}$ requête est donc bien $|\mathcal{P}(q)|$.
Chaque requête entraîne l'insertion de $\max(\mathcal{T}, 1)$ éléments dans $\mathcal{P}$, qui en contient initialement 1. Nous avons donc

$$|\mathcal{P}(q)| \leq (q-1)\max(\mathcal{R}_{indep}, 1) + 1 \ .$$

□

Jeu 6. Dans le Jeu 6, l'oracle $\mathcal{F}$ est remplacé par un accès à l'oracle aléatoire $\mathcal{H}$. L'intercepteur est placé entre l'interface $\mathcal{H}^{reveal}$ de $\mathcal{H}$ et l'entrée du simulateur $\mathcal{S}_6$. $\mathcal{S}_6$ doit maintenant simuler lui-même les valeurs de $\mathcal{F}$, en appelant $\mathcal{H}$ pour assurer la cohérence des valeurs de haché. Contrairement aux preuves dans le cas idéal, les images par $\mathcal{F}$ ne sont pas uniformes et indépendantes les unes des autres. La représentation que nous avons adoptée vise à restreindre le moins possible la forme de cette distribution. Nous supposons donc pouvoir la simuler à l'aide d'algorithmes d'échantillonage, `Simule` et `A-Simule`. Le simulateur ainsi défini est décrit en Figure 6.10.

Dans le Jeu 6, les valeurs de $\mathcal{F}$ peuvent être générées en partie par un oracle aléatoire $\mathcal{H}$, et ont par conséquent une distribution de probabilités différente de celle du Jeu 5. Nous cherchons maintenant à borner le biais statistique introduit sur la vue de l'attaquant lors de chaque simulation de $\mathcal{F}$. Nous utilisons pour cela le paramètre $\mathcal{R}_{indep}$ défini en section 6.2.

Considérons une étape intermédiaire des Jeux 5 et 6 pour une requête identique, et supposons que les valeurs de $\mathcal{F}$ définies précédemment sont identiques. Supposons également qu'aucun cas d'échec n'ait été identifié. Nous cherchons à majorer le biais introduit sur la vue de l'attaquant par la simulation d'une valeur de $\mathcal{F}$. Le seul cas où la distribution de cette valeur diffère entre les Jeux 5 et 6 est celui où $\mathcal{S}_6$ appelle l'oracle $\mathcal{H}$ (pour une valeur de haché non encore définie). Par définition de `A-Simule`, nous avons :

$$
\begin{aligned}
& \sum_{(A',B')} \left|\Pr_5\left[\mathcal{F}_M(A,B)=(A',B')|E\right] - \Pr_6\left[\mathcal{F}_M(A,B)=(A',B')|E\right]\right| \\
\leq\ & \sum_{(A',B')} \left|\Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M(A,B)=(A',B')|E\right] - \Pr\left[\mathcal{H}(\mathcal{M})=B'\right]\Pr\left[\texttt{A-Simule}(M,A,B,B',E)=A'\right]\right| \\
\leq\ & \sum_{(A',B')} \left|\Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M(A,B)=(A',B')|E\right] - 2^{-\ell_h}\Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M^{(A)}(A,B)=A'|E,\mathcal{F}_M^{(B)}(A,B)=B'\right]\right| \\
\leq\ & \sum_{B'} \left(\left|\Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M^{(B)}(A,B)=B'|E\right] - 2^{-\ell_h}\right| \sum_{A'} \Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M^{(A)}(A,B)=A'|E,\mathcal{F}_M^{(B)}(A,B)=B'\right]\right) \\
\leq\ & \sum_{B'} \left|\Pr_{\mathcal{F}\leftarrow\mathcal{V}}\left[\mathcal{F}_M^{(B)}(A,B)=B'|E\right] - 2^{-\ell_h}\right|
\end{aligned}
$$

Supposons maintenant qu'une valeur de $\mathcal{F}_{M^*}(A^*,B^*)$ soit dans E pour (M^*,A^*,B^*) relatif de (M,A,B).

Comme Drapeau = Succès, il n'y avait pas de relatif de (M,A,B) dans le graphe au moment de l'introduction de $(A,B),\mu_{A,B}$ dans $\mathcal{P}$. Si un relatif (M^*,A^*,B^*) de (M,A,B) avait été introduit dans le graphe après, il ne peut l'avoir été que suite à une requête de $\mathcal{D}$ ou de $\mathcal{I}$ pour $\mathcal{F}_{M^*}(A^*,B^*)$. En effet, le cas d'échec Echec_{c1} empêche qu'il soit introduit à l'étape 2 du simulateur (le fait que (A^*,B^*) ait un chemin utile aurait dû entraîner un cas d'échec). Dans ce cas, la valeur de $\mathcal{F}_M(A,B)$ a déjà été définie à l'étape 2 du simulateur lors de la requête pour $\mathcal{F}_{M^*}(A^*,B^*)$, d'où une contradiction.

D'après la définition de ε, nous avons donc :

$$\frac{1}{2}\sum_{(A',B')} \left|\Pr_5\left[\mathcal{F}_M(A,B)=(A',B')|E\right] - \Pr_6\left[\mathcal{F}_M(A,B)=(A',B')|E\right]\right| \leq \varepsilon\ .$$

Pour chaque requête reçue par $\mathcal{S}$, au plus $\mathcal{R}_{indep}$ valeurs de $\mathcal{F}$ sont simulées. Nous en déduisons la relation suivante :

Initialisation de $\mathcal{S}_6$

1. définir $X = \{x^0\}$, $E = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$
2. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrées : (M, A, B)

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, \varepsilon, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}$ tel que $\mu||\tilde{M}$ est un chemin utile
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) si $\tilde{\mu}||\tilde{M}$ est un chemin complet
      i. calculer $\mathcal{M} = \mathsf{unpad}(\tilde{\mu}||\tilde{M})$
      ii. appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
      iii. définir $\tilde{B}' = h$
      iv. lancer A-Simule$(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{B}', E)$ pour obtenir $\tilde{A}'$
   (c) sinon
      i. lancer Simule$(\tilde{M}, \tilde{A}, \tilde{B}, E)$ pour obtenir $(\tilde{A}', \tilde{B}')$
   (d) définir $\tilde{y} = (\tilde{A}', \tilde{B}')$
   (e) si $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \notin E$
      i. si $\tilde{y}$ a un chemin utile $\tilde{\mu}'$ dans $\mathcal{G}$, NoteEchec(Echec_{a1})
      ii. s'il existe un arc $\tilde{y} \xrightarrow{\tilde{M}'} \tilde{z}$ dans E tel que $\mu||\tilde{M}||\tilde{M}'$ est un chemin utile, NoteEchec(Echec_{a2})
      iii. si $(\varepsilon, \tau) - \mathsf{TestAtypique}(\tilde{y}, \mu||\tilde{M})$, NoteEchec$(\mathsf{Echec}_{b})$
      iv. s'il existe (z, μ_z) dans $\mathcal{P}$ tel que $(\varepsilon, \tau) - \mathsf{TestRelatifs}_1(\tilde{y}, \mu||\tilde{M}, z, \mu_z)$, NoteEchec$(\mathsf{Echec}_{c1})$
      v. s'il existe $z \xrightarrow{M_z} t$ dans E tel que $(\varepsilon, \tau) - \mathsf{TestRelatifs}_2(\tilde{y}, \mu||\tilde{M}, z, M_z)$, NoteEchec$(\mathsf{Echec}_{c2})$
      vi. si $(\varepsilon, \tau) - \mathsf{TestRelatifs}_1(\tilde{y}, \mu||\tilde{M}, \tilde{y}, \mu||\tilde{M})$,NoteEchec$(\mathsf{Echec}_{d})$
   (f) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}$ à E
   (g) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c||\tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c||\tilde{M})$ à $\mathcal{P}$
3. lancer Simule(M, A, B, E) pour obtenir $y = (A', B')$
4. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
5. renvoyer (A', B').

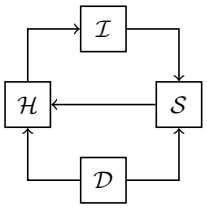


FIGURE 6.10 – Indifférenciabilité : Simulateur $\mathcal{S}_6$ pour $\mathcal{F}$ au Jeu 6 dans le cas biaisé.

$$\left|\Pr\left[W_6 \wedge \overline{\mathsf{Echec}[6]}\right] - \Pr\left[W_5 \wedge \overline{\mathsf{Echec}[5]}\right]\right| \leq \max(1, \mathcal{R}_{indep})N\varepsilon \ . \tag{6.6}$$

Comme les cas d'échec sont définis de manière déterministe à partir de graphes dans lesquels Drapeau = Succès, nous avons également

$$|\Pr\left[\mathsf{Echec}[6]\right] - \Pr\left[\mathsf{Echec}[5]\right]| \leq \max(1, \mathcal{R}_{indep})N\varepsilon \ . \tag{6.7}$$

Jeu 7. Nous aboutissons maintenant au Jeu final, pour où $\mathcal{D}$ interagit avec $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. L'intercepteur est intégré au simulateur, et les cas d'échec sont supprimés. Nous avons

$$\Pr\left[W_7\right] = \Pr\left[W_6\right] \ . \tag{6.8}$$

6.3.5 Majoration de l'avantage de l'attaquant

Au cours de la séquence de jeux, la seule différence sur la vue de l'attaquant intervient entre les Jeux 5 et 6. D'après l'équation (6.6), nous avons

$$\begin{aligned} |\Pr\left[W_6\right] - \Pr\left[W_5\right]| &\leq \max(1, \mathcal{R}_{indep})N\varepsilon + |\Pr\left[W_6 \wedge \mathsf{Echec}[6]\right] - \Pr\left[W_5 \wedge \mathsf{Echec}[5]\right]| \\ &\leq \max(1, \mathcal{R}_{indep})N\varepsilon + \max(\Pr\left[\mathsf{Echec}[5]\right], \Pr\left[\mathsf{Echec}[6]\right]) \ . \end{aligned}$$

D'après l'équation (6.7), nous avons

$$|\Pr\left[W_6\right] - \Pr\left[W_5\right]| \leq 2\max(1, \mathcal{R}_{indep})N\varepsilon + \Pr\left[\mathsf{Echec}[5]\right] \ .$$

D'après les équations (6.1) à (6.8),

$$\Pr\left[W_5\right] = \Pr\left[W_0\right] \text{ et } \Pr\left[W_7\right] = \Pr\left[W_6\right] \ ,$$

d'où

$$\mathsf{Adv}(\mathcal{D}) = |\Pr\left[W_7\right] - \Pr\left[W_0\right]| \leq 2\max(1, \mathcal{R}_{indep})N\varepsilon + \Pr\left[\mathsf{Echec}[5]\right] \ .$$

Enfin, le lemme 7 montré en section 6.5 nous permet de conclure que

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep})(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)}N^2 + \mathcal{R}_{indep}((\mathcal{X}_{rel} + \mathcal{X}_{atyp})2^{-(n-\tau)} + 2\varepsilon)N$$

si $\mathcal{R}_{indep} \geq 1$, et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)}\frac{N(N+1)}{2} + (\mathcal{X}_{atyp}2^{-(n-\tau)} + 2\varepsilon)N \ .$$

Nous retrouvons donc bien les bornes annoncées dans le théorème 8.

6.4 Indifférenciabilité forte de Chop-MD dans le cas biaisé

Intéressons-nous maintenant à l'indifférenciabilité forte de Chop-MD, lorsque la primitive $\mathcal{F}$ est représentée par une fonction tirée selon une distribution $\mathcal{V}$ dans FUNC. La méthode employée dans la section précédente, en caractérisant le biais introduit par l'existence d'une propriété particulière de $\mathcal{F}$ par les deux grandeurs ε et τ, ne suffit pas à garantir l'indifférenciabilité forte de cette construction.

Après avoir montré brièvement les raisons pour lesquelles la méthode définie ci-dessus ne s'applique pas, nous montrerons une borne d'indifférenciabilité forte de Chop-MD valable pour une classe réduite d'attaques par distingueur.

6.4.1 Insuffisance de la caractérisation du biais par ε et τ

Pour étudier l'indifférenciabilité forte de Chop-MD dans le cas biaisé, une idée naturelle pourrait être de transposer les techniques de preuves utilisées dans le cas idéal étudié au chapitre 5. Ces preuves reposent sur le fait que la connaissance de valeurs de haché ne peut donner à l'attaquant de l'information que sur les sorties de $\mathcal{F}$ correspondant à une faible partie des entrées. $\mathcal{S}$ peut alors définir un cas d'échec à chaque fois que l'attaquant effectue une requête sur l'une de ces entrées.

Dans le cas biaisé, nous avons défini des bornes ε et τ dans le but d'autoriser des biais faibles sur toutes les sorties, ou des corrélations faibles sur les sorties correspondant à plusieurs entrées. Nous avons vu plus haut que la définition de ces bornes suffisait pour dériver des garanties concernant l'indifférenciabilité d'un oracle aléatoire public de la construction Chop-MD.

Dans le cas de l'indifférenciabilité forte, cette caractérisation du biais sur $\mathcal{F}$ ne suffit pas. La connaissance par l'attaquant de valeurs de haché que le simulateur ignore introduit un biais potentiel sur toutes les sorties de $\mathcal{F}$, que $\mathcal{S}$ ne peut pas prendre en compte.

Illustration. Nous montrons maintenant un exemple permettant informellement d'illustrer ce problème. Considérons la construction Chop-MD avec $n = 2\ell_h$, et $\mathcal{F}$ tirée uniformément parmi l'ensemble $\mathcal{E}$ des fonctions vérifiant :

$$\forall m, \mathcal{F}_m(x^0) = (A_m, B_m) \Rightarrow lsb(\mathcal{F}_m^{(A)}(0, \overline{B_m})) = 0$$

En d'autres termes, si (A_m, B_m) est l'image par $\mathcal{F}_m$ de l'IV de la fonction de hachage, alors le bit de poids faible de la partie A de $\mathcal{F}_m(0, \overline{B_m})$ vaut 0.

Considérons maintenant l'attaquant $\mathcal{D}$ suivant :

1. $\mathcal{D}$ choisit μ tel que $m = \mathsf{pad}(\mu)$ ne contient qu'un bloc, et envoie une requête gauche pour obtenir $h = H(\mu)$.
2. $\mathcal{D}$ envoie une requête droite pour obtenir $(a, b) = \mathcal{F}_m(0, \overline{h})$
3. $\mathcal{D}$ renvoie $lsb(a)$.

Dans le cas où $\mathcal{D}$ interagit avec $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, par définition de $\mathcal{F}$, $\mathcal{D}$ renvoie systématiquement 0. Si $\mathcal{D}$ interagit avec $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$, $(m, 0, \overline{h})$ est la première requête reçue par $\mathcal{S}$. Si $\mathcal{S}$ ne simule pas d'autres valeurs de $\mathcal{F}$ au moment où il reçoit cette requête, il renvoie alors (a, b) tel que $\mathcal{F}$ soit tirée uniformément sur $\mathcal{E}$.

Nous avons alors

$$\begin{aligned}
\Pr\left[lsb(a) = 0\right] &= \Pr\left[lsb(a) = 0 \wedge \mathcal{F}_m^{(B)}(x^0) = h\right] + \Pr\left[lsb(a) = 0 \wedge \mathcal{F}_m^{(B)}(x^0) \neq h\right] \\
&= 2^{-\ell_h} + \frac{1}{2}(1 - 2^{-\ell_h}) \\
&= \frac{1}{2}(1 + 2^{-\ell_h}) \,.
\end{aligned}$$

La probabilité pour que $\mathcal{D}$ renvoie 0 est alors $1/2(1 + 2^{-\ell_h})$, ce qui donne un avantage proche de $1/2$.

Pour que $\mathcal{D}$ n'ait pas un avantage conséquent avec seulement deux requêtes, il faut donc que $\mathcal{S}$ commence par simuler $\mathcal{F}_m(x^0)$.

Or, si on fixe $\tau > 1$, $(m, 0, \overline{h})$ n'est pas un relatif de (m, x^0), seul un bit de la partie A de $\mathcal{F}_m(0, \overline{h})$ étant impacté par la relation. D'autre part,

$$\begin{aligned}
\Pr\left[\mathcal{F}_m^{(B)}(x^0) = h^* | lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = 0\right] &= \frac{\Pr\left[F_m^{(B)}(x^0) = h^* \wedge lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = 0\right]}{\Pr\left[lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = 0\right]} \\
&= \frac{2 \times 2^{-\ell_h}}{1 + 2^{-\ell_h}} \text{ et} \\
\Pr\left[\mathcal{F}_m^{(B)}(x^0) = h^* | lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = 1\right] &= 0 \,.
\end{aligned}$$

Par symétrie, les autres valeurs de sortie de $\mathcal{F}_m^{(B)}(x^0)$ sont équiprobables. Le biais statistique sur $\mathcal{F}_m^{(B)}(x^0)$ vérifie donc la relation suivante :

$$\begin{aligned}
& \frac{1}{2} \sum_{h \in \{0,1\}^{\ell_h}} \left| \Pr\left[\mathcal{F}_m^{(B)}(x^0) = h | lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = b\right] - 2^{-\ell_h} \right| \\
\leq & \left| \Pr\left[\mathcal{F}_m^{(B)}(x^0) = h^* | lsb(\mathcal{F}_m^{(A)}(0,\overline{h^*})) = b\right] - 2^{-\ell_h} \right| \\
\leq & \; 2^{-\ell_h} \,.
\end{aligned}$$

Pour un choix de $\varepsilon > 2^{-\ell_h}$, (m, x^0) n'est donc pas un relatif de $(m, 0, \overline{h})$. Dans ce cas, un simulateur $\mathcal{S}$ construit sur cette définition de relatifs n'a pas de raison de simuler $\mathcal{F}_m(x^0)$ avant $\mathcal{F}_m(0,\overline{h})$, et $\mathcal{D}$ a un avantage proche de 1/2.

Cet exemple ne montre pas que la preuve d'indifférenciabilité de la construction ainsi définie est impossible à obtenir, mais elle montre que le modèle de biais basé sur la définition de (ε, τ)-relatifs ne suffit pas à construire un simulateur résistant à tous les attaquants $\mathcal{D}$ pour tous les types de biais sur $\mathcal{F}$.

6.4.2 Limitation sur le biais de $\mathcal{F}$

Afin de contourner le problème identifié au paragraphe précédent, nous nous intéressons maintenant à l'indifférenciabilité forte de la construction Chop-MD, lorsqu'une limitation supplémentaire s'applique au biais sur $\mathcal{F}$.

Faisons donc l'hypothèse suivante. Pour tous historiques $\mathcal{L}$ et $\mathcal{L}'$ tels que $I(\mathcal{L}) \cap I(\mathcal{L}') = \emptyset$, toute entrée $(M, A, B) \notin I(\mathcal{L}) \cup I(\mathcal{L}') \cup \texttt{REL}(\mathcal{L})$, toute sortie (A', B'), nous avons

$$\begin{aligned}
\Pr\left[F_M^{(B)}(A,B) = B' | \mathcal{L}\right] &= 2^{-\ell_h} \\
\Pr\left[F_M(A,B) = (A',B') | \mathcal{L} \cup \mathcal{L}'\right] &= \Pr\left[F_M(A,B) = (A',B') | \mathcal{L}'\right] \,.
\end{aligned}$$

La première condition correspond à la restriction $\varepsilon = 0$, la seconde traduit que la distribution des sorties de $\mathcal{F}_M(A,B)$ peut être biaisée, du moment qu'elle n'est pas impactée par la connaissance des sorties de $F'_M(A',B')$ lorsque (M',A',B') n'est pas un relatif de $\mathcal{F}_M(A,B)$.

6.4.3 Indifférenciabilité forte pour un biais limité

En tenant compte des hypothèses précédentes, nous montrons maintenant le théorème suivant.

Théorème 9 *Soit $\mathcal{F}$ une fonction choisie dans* FUNC *selon une distribution $\mathcal{V}$, et soit $\mathcal{H}$ un oracle aléatoire. Supposons que l'état initial x^0 soit tel que $(0,\tau) - \texttt{TestAtypique}(x^0, \emptyset)$ et $(0,\tau) - \texttt{TestRelatifs}_1(x^0, \emptyset, x^0, \emptyset)$ renvoient FAUX. Dans ce cas, le simulateur $\mathcal{S}$ défini en Figure 6.11 est tel que pour tout attaquant $\mathcal{D}$ émettant des requêtes d'un poids total d'au plus N,*

$$\begin{aligned}\mathsf{Adv}(\mathcal{D}) \quad \leq \quad & \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep})(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 \\ & + \mathcal{R}_{indep}(\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} N + (\mathcal{R}_x + 1) 2^{-(n-\ell_h-\tau)} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] N\end{aligned}$$

si $\mathcal{R}_{indep} \geq 1$, et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)} \frac{N(N+1)}{2} + \mathcal{X}_{atyp} 2^{-(n-\tau)} N + 2^{-(n-\ell_h-\tau)} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] N$$

si $\mathcal{R}_{indep} = 0$. Dans le cas d'un encodage sans préfixe, ces bornes deviennent

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep} + 1)(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 + + \mathcal{R}_{indep}(\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} N$$

si $\mathcal{R}_{indep} \geq 1$ et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)} N^2 + \mathcal{X}_{atyp} 2^{-(n-\tau)} N$$

si $\mathcal{R}_{indep} = 0$.

6.4.4 Preuve par séquence de jeux.

Afin de montrer le théorème 9, nous adaptons la preuve par séquence de jeux de la section 5.3. Le déroulement est similaire à celui de cette preuve, il est résumé par la Figure 6.12. Nous ajoutons à la séquence de jeux de la section 5.3 le traitement des relatifs des valeurs simulées décrit à la section précédente.

Jeu 0. Il s'agit du Jeu initial, dans lequel $\mathcal{D}$ interagit avec le système $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$.

Jeux 1 à 5. Les Jeux 1 à 5 sont identiques à ceux de la preuve d'indifférenciabilité d'un oracle aléatoire à usage public. Les propriétés montrées dans la section précédente sont donc également vérifiées ici, et d'après les équations (6.1) à (6.5),

$$\Pr\left[W_5\right] = \Pr\left[W_0\right] \ . \tag{6.9}$$

Jeu 6. Dans le Jeu 6, nous traitons différemment les requêtes venant de $\mathcal{I}$ et de $\mathcal{D}$. Comme dans le chapitre précédent, $\mathcal{S}_6$ définit deux graphes $\mathcal{G}_{\mathcal{I}}$ et $\mathcal{G}_{\mathcal{D}}$. Nous introduisant un cas d'échec Echec_e, similaire au cas Echec_3 défini dans le chapitre précédent, et permettant de traiter les attaques par extension de longueur. Le simulateur $\mathcal{S}_6$ est décrit en Figure 6.13.

Ce nouveau cas d'échec n'introduit pas de modification dans la vue de l'attaquant, d'où

$$\Pr\left[W_6\right] = \Pr\left[W_5\right] \ . \tag{6.10}$$

Initialisation de $\mathcal{S}$

1. choisir $\tau \in [0, n - \ell_h]$
2. définir $X_{\mathcal{D}} = \{x^0\}$ and $E_{\mathcal{D}} = \emptyset$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

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

Sortie : (A', B')

1. Pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \{(M, A, B)\} \cup \mathcal{R}(M, A, B, 0, \tau)$
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) si $\tilde{x}$ a un chemin $\tilde{\mu}$ dans $\mathcal{G}_{\mathcal{D}}$ et si $\mathcal{M} = \mathsf{unpad}(\tilde{\mu}||\tilde{M})$ est un chemin utile
      i. calculer $\mathcal{M} = \mathsf{unpad}(\tilde{\mu}||\tilde{M})$
      ii. appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
      iii. définir $\tilde{B}' = h$
      iv. lancer A-Simule$(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{B}', E_{\mathcal{D}})$ pour obtenir $\tilde{A}'$
   (c) sinon
      i. lancer Simule$(\tilde{M}, \tilde{A}, \tilde{B}, E_{\mathcal{D}})$ pour obtenir $(\tilde{A}', \tilde{B}')$
      ii. définir $\tilde{y} = (\tilde{A}', \tilde{B}')$
   (d) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à $X_{\mathcal{D}}$ et l'arc $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}$ à $E_{\mathcal{D}}$
2. lancer Simule$(M, A, B, E_{\mathcal{D}})$ pour obtenir (A', B')
3. définir $y = (A', B')$
4. ajouter les nœuds x et y à $X_{\mathcal{D}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{D}}$
5. renvoyer (A', B')

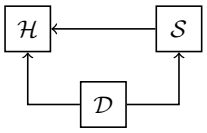


FIGURE 6.11 – Simulateur $\mathcal{S}$ de $\mathcal{F}$ pour la preuve d'indifférenciabilité forte de Chop-MD dans le cas biaisé.

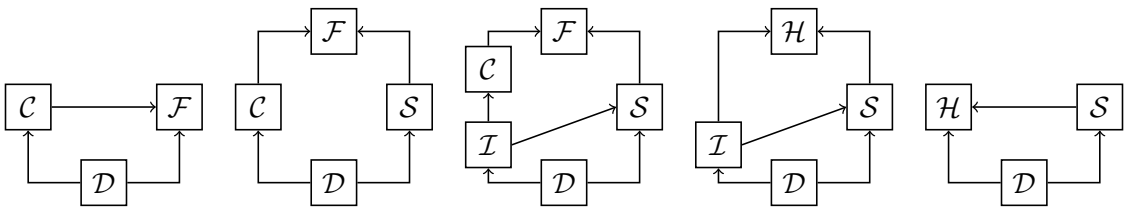


FIGURE 6.12 – Construction du simulateur $\mathcal{S}$.

Initialisation de $\mathcal{S}_{6,7}$

1. choisir $\tau \in [0, n - \ell_h]$
2. définir $X_{\mathcal{I}} = \{x^0\}$, $E_{\mathcal{I}} = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$
3. définir $X_{\mathcal{D}} = \{x^0\}$, $E_{\mathcal{D}} = \emptyset$
4. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrées : (M, A, B), origine $\mathcal{O}$ = soit $\mathcal{I}$ soit $\mathcal{D}$

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, 0, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ tel que $\mu || \tilde{M}$ est un chemin utile
   (a) si μ n'est pas dans $\mathcal{G}_{\mathcal{D}}$ et si $\mathcal{O} = \mathcal{D}$, NoteEchec($\mathsf{Echec}_e$)
   (b) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (c) appeler $\mathcal{F}$ pour obtenir $\tilde{y} = \mathcal{F}_{\tilde{M}}(\tilde{A}, \tilde{B})$
   (d) si $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \notin E_{\mathcal{C}} \cup E_{\mathcal{D}}$
      i. si $\tilde{y}$ a un chemin utile $\tilde{\mu}'$ dans $\mathcal{G}_{\mathcal{C}} \cup \mathcal{G}_{\mathcal{D}}$, NoteEchec($\mathsf{Echec}_{a1}$)
      ii. s'il existe un arc $\tilde{y} \xrightarrow{\tilde{M}'} \tilde{z}$ dans $E_{\mathcal{C}} \cup E_{\mathcal{D}}$ tel que $\mu || \tilde{M} || \tilde{M}'$ est un chemin utile, NoteEchec(Echec_{a2})
      iii. si $(0, \tau) - \mathtt{TestAtypique}(\tilde{y}, \mu || \tilde{M})$, NoteEchec($\mathsf{Echec}_b$)
      iv. s'il existe (z, μ_z) dans $\mathcal{P}$ tel que $(0, \tau) - \mathtt{TestRelatifs}_1(\tilde{y}, \mu || \tilde{M}, z, \mu_z)$, NoteEchec($\mathsf{Echec}_{c1}$)
      v. s'il existe $z \xrightarrow{M_z} t$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ tel que $(0, \tau) - \mathtt{TestRelatifs}_2(\tilde{y}, \mu || \tilde{M}, z, M_z)$, NoteEchec($\mathsf{Echec}_{c2}$)
      vi. si $(0, \tau) - \mathtt{TestRelatifs}_1(\tilde{y}, \mu || \tilde{M}, \tilde{y}, \mu || \tilde{M})$, NoteEchec($\mathsf{Echec}_d$)
   (e) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à $X_{\mathcal{O}}$ et l'arc $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}$ à $E_{\mathcal{O}}$
   (f) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c || \tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c || \tilde{M})$ à $\mathcal{P}$
3. appeler $\mathcal{F}$ pour obtenir $y = \mathcal{F}_M(A, B)$
4. ajouter les nœuds x et y à $X_{\mathcal{O}}$ et l'arc $x \xrightarrow{M} y$ à $E_{\mathcal{O}}$
5. renvoyer $(A', B') = y$

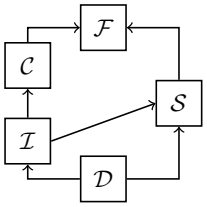


FIGURE 6.13 – Simulateur $\mathcal{S}_{6,7}$ pour $\mathcal{F}$ aux Jeux 6 et 7 pour la preuve d'indifférenciabilité forte de Chop-MD.

Jeu 7. Comme dans le chapitre 5, l'intercepteur est modifié : lorsqu'il voit passer une requête de haché, il la stocke et ne la décompose en appels à $\mathcal{F}$ qu'à la fin du Jeu. Le simulateur est identique à celui du Jeu 6, l'intercepteur $\mathcal{I}_{retard}$ est décrit en Figure 6.14. Le cas d'échec Echec_e ne peut pas se produire, le graphe $\mathcal{G}_{\mathcal{I}}$ étant vide au moment du traitement des requêtes provenant de $\mathcal{D}$.

Initialisation de $\mathcal{I}_{retard}$

1. définir $\mathcal{L}_{\mathcal{H}} = \emptyset$
2. définir $(A^0, B^0) = x^0$

Traitement d'une requête à $\mathcal{H}$

Entrée : $\mu \in \{0,1\}^*$

Sortie : $h = \mathcal{H}(\mu)$

1. ajouter μ à $\mathcal{L}_{\mathcal{H}}$
2. appeler $\mathcal{C}$ (Jeu 7) ou $\mathcal{H}$ (Jeu 8) pour obtenir $h = \mathcal{H}(\mu)$
3. renvoyer h à $\mathcal{D}$

Simulation des requêtes à $\mathcal{H}$ (à la fin du Jeu)

1. pour tout μ de $\mathcal{L}_{\mathcal{H}}$
   (a) calculer $(M^1||\dots||M^\ell) = \mathsf{pad}(\mu)$
   (b) pour i de 1 à ℓ, appeler $\mathcal{S}$ pour obtenir $(A^i, B^i) = \mathcal{F}_{M^i}(A^{i-1}, B^{i-1})$

FIGURE 6.14 – Intercepteur $\mathcal{I}_{retard}$ dans les Jeux 7 et 8.

Cette modification du simulateur n'introduit pas de biais dans la vue de l'attaquant, d'où

$$\Pr[W_7] = \Pr[W_6] \ . \tag{6.11}$$

Contrairement aux preuves du chapitre précédent, les cas d'échec définis dans les Jeux 6 et 7 ne sont pas tout à fait équivalents. Nous montrons maintenant la propriété suivante, qui est plus faible que la propriété 3, mais qui suffit dans le cadre de notre preuve.

Propriété 13 *Soit $\mathcal{F}$ tirée selon la distribution $\mathcal{V}$. Pour un aléa de $\mathcal{D}$ identique aux Jeux 6 et 7, les propositions suivantes sont vérifiées.*

Si à la fin du Jeu 6, Drapeau = Succès, alors Drapeau = Succès à la fin du Jeu 7. Dans ce cas, le graphe $\mathcal{G}_6$ construit par $\mathcal{S}_{6,7}$ à la fin du Jeu 6 contient tous les arcs du graphe $\mathcal{G}_7 = \mathcal{G}_{\mathcal{I}} \cup \mathcal{G}_{\mathcal{D}}$ construit par $\mathcal{S}_{6,7}$ à la fin du Jeu 7.

Démonstration :
Nous montrons simultanément les deux propositions, par récurrence sur le nombre d'arcs de $\mathcal{G}_7$.
Supposons que les requêtes émises par $\mathcal{D}$ et les valeurs renvoyées par l'oracle $\mathcal{F}$ ne provoquent pas de cas d'échec de $\mathcal{S}_{6,7}$ au cours du Jeu 6.
Si $\mathcal{G}_7$ ne contient aucun arc, la propriété de l'énoncé est évidemment vérifiée, les évènements d'échec ne pouvant se produire que si $\mathcal{G}$ contient au moins un arc.
Soit alors $x_i \overset{M_i}{\rightarrow} y_i$ le $i^{ème}$ arc inséré par $\mathcal{S}_{6,7}$ dans $\mathcal{G}_7$. Par hypothèse de récurrence, aucun évènement d'échec ne s'est produit avant son insertion et tous les arcs $x_1 \overset{M_1}{\rightarrow} y_1 \dots x_{i-1} \overset{M_{i-1}}{\rightarrow} y_{i-1}$ sont dans $\mathcal{G}_6$.
Nous considérons maintenant les trois cas suivants.

1. Si $x_i \overset{M_i}{\to} y_i$ est inséré au cours du traitement d'une requête venant de $\mathcal{I}$, alors cette requête concerne $\mathcal{F}_{M_i}(x_i)$. En effet, si un relatif $(\tilde{M}_i, \tilde{x_i})$ de (M_i, x_i) a un chemin $\tilde{\mu}$ tel que $\tilde{\mu}||\tilde{M}_i$ est un chemin utile, Echec_{c1} aurait dû se produire. Au cours du Jeu 6, cette même requête est émise par $\mathcal{I}$ et le même arc $x_i \overset{M_i}{\to} y_i$ est inséré à $\mathcal{G}_6$.
2. Si $x_i \overset{M_i}{\to} y_i$ est inséré au cours du traitement d'une requête (M_i, x_i) venant de $\mathcal{D}$, il est également inséré au Jeu 6 au cours du traitement de la même requête.
3. Si $x_i \overset{M_i}{\to} y_i$ est inséré au cours du traitement d'une requête (M^*, x^*) telle que $(M_i, x_i) \in \mathcal{R}((M^*, x^*), \varepsilon, \tau)$, x_i a un chemin μ dans $\mathcal{G}_\mathcal{D}$ et $\mu||M_i$ est un chemin utile. Lorsque la même requête est traitée au Jeu 6, le graphe $\mathcal{G}_6$ contient au moins les mêmes arcs, et donc le chemin μ est dans $\mathcal{G}_6$. $x_i \overset{M_i}{\to} y_i$ est donc également inséré au graphe au cours du Jeu 6.

$x_i \overset{M_i}{\to} y_i$ est donc également dans $\mathcal{G}_6$. Nous montrons maintenant que si son insertion provoque un évènement d'échec, alors un évènement d'échec s'est également produit au Jeu 6.

Echec_{a1}. Si Echec_{a1} se produit, alors y_i a deux chemins utiles μ, μ' différents dans $\mathcal{G}_7$ au moment de son insertion. Ces deux chemins sont également entièrement définis dans $\mathcal{G}_6$ ce qui a provoqué un cas d'échec d'après la propriété 7.

Echec_{a2}. Si Echec_{a2} se produit, alors il existe $y_i \overset{M'}{\to} z$ dans $E_\mathcal{I} \cup E_\mathcal{D}$.

1. Si $x_i \overset{M_i}{\to} y_i$ est inséré au cours d'une requête provenant de $\mathcal{D}$, alors $y_i \overset{M'}{\to} z$ est dans $E_\mathcal{D}$ et y est également dans le Jeu 6. Echec_{a2} doit donc se produire.
2. Si $x_i \overset{M_i}{\to} y_i$ est inséré au cours d'une requête provenant de $\mathcal{I}$, $y_i \overset{M'}{\to} z$ est dans $E_\mathcal{I} \cup E_\mathcal{D}$ à la fin du Jeu 6. S'il y est avant l'insertion de $x_i \overset{M_i}{\to} y_i$, Echec_{a2} se produit.
3. Dans le cas contraire, $y_i \overset{M'}{\to} z$ est dans $E_\mathcal{D}$, les requêtes venant de $\mathcal{I}$ étant traitées dans le même ordre aux Jeux 6 et 7. Dans le Jeu 6, au moment de questionner $\mathcal{F}$ pour $\mathcal{F}_{M'}(y_i)$, y_i ne peut pas avoir de chemin μ dans $\mathcal{G}_\mathcal{D}$ (tel que $\mu||M'$ soit un chemin utile, sinon Echec_{a1} se produit). Dans ce cas, Echec_e se produit avant que $\mathcal{S}_{6,7}$ n'envoie cette requête à $\mathcal{F}$.

Echec_b. Si Echec_b se produit, alors y_i a un chemin utile μ dans $\mathcal{G}_7$ et il existe M tel que (M, y) soit une entrée atypique de $\mathcal{F}$ et $\mu||M$ est un chemin utile. Dans le graphe $\mathcal{G}_6$, μ est un chemin utile pour y_i entièrement déterminé et d'après la propriété 11, un cas d'échec s'est produit au Jeu 6.

Echec_{c1}. Supposons que Echec_{c1} se produit. Dans ce cas, il existe M, M', y' tel que $(M, y_i) \in \mathcal{R}(M', y', 0, \tau)$, y' a un chemin μ' dans $\mathcal{G}_7$ et $\mu||M$, $\mu'||M'$ sont des chemins utiles. D'après la propriété 10, un cas d'échec s'est produit au Jeu 6.

Echec_{c2}. Si Echec_{c2} se produit, alors il existe $y' \overset{M'}{\to} z'$ dans $E_\mathcal{I} \cup E_\mathcal{D}$ et un bloc de message M tel que $\mu||M$ est un chemin utile, et $(M, y_i) \in \mathcal{R}(M', y', 0, \tau)$. Si y' a un chemin μ' tel que $\mu'||M'$ est un chemin utile, Echec_{c1} s'est déjà produit. Dans le cas contraire, $y' \overset{M'}{\to} z$ a nécessairement été inséré au cours du traitement de la requête pour $F_{M'}(y')$ venant de $\mathcal{D}$.

1. Au cours du Jeu 6, si cette requête intervient après l'insertion de $x_i \overset{M}{\to} y_i$, l'ordre de l'insertion de ces arcs est modifié, ce qui ne peut intervenir que si $x_i \overset{M_i}{\to} y_i \in E_\mathcal{I}$. Dans ce cas, à la réception de la requête (M', y'), $\mathcal{S}_{6,7}$ doit obtenir la valeur de $\mathcal{F}_M(y_i)$ ce qui provoque Echec_e.

2. Dans le cas contraire, au cours du Jeu 6, l'ordre d'insertion de ces deux arcs est le même, et Echec_{c2} se produit.

Echec_d. Enfin, si Echec_d se produit dans le Jeu 7, ce même évènement se produit lors du traitement de la même requête dans le Jeu 6.

Conclusion. Par récurrence, les arcs de $\mathcal{G}_7$ sont donc dans $\mathcal{G}_6$, et un cas d'échec au Jeu 7 implique un cas d'échec au Jeu 6. Par contraposée, si Drapeau = Succès après le Jeu 6, Drapeau = Succès après le Jeu 7.

□

En notant $\mathsf{Echec}[i]$ la disjonction des évènements d'échec au Jeu i, nous déduisons de cette propriété que

$$\Pr\left[\mathsf{Echec}[7]\right] \leq \Pr\left[\mathsf{Echec}[6]\right] \tag{6.12}$$

Jeu 8. L'accès à l'oracle $\mathcal{F}$ est maintenant remplacé par un accès à l'oracle aléatoire $\mathcal{H}$. $\mathcal{S}_8$ doit maintenant simuler les valeurs de $\mathcal{F}$, à l'aide d'un accès à $\mathcal{H}$. Le cas d'échec Echec_e peut être retiré du simulateur, ainsi que la distinction entre les graphes $\mathcal{G}_\mathcal{C}$ et $\mathcal{G}_\mathcal{D}$. Nous obtenons maintenant le simulateur $\mathcal{S}_8$ décrit en Figure 6.15.

Tant qu'aucun cas d'échec ne se produit, la vue de l'attaquant reste inchangée, car $\varepsilon = 0$. Nous en déduisons que

$$\Pr\left[W_8 \wedge \overline{\mathsf{Echec}[8]}\right] = \Pr\left[W_7 \wedge \overline{\mathsf{Echec}[7]}\right] \ . \tag{6.13}$$

Jeu 9. Nous aboutissons au scénario final dans lequel $\mathcal{D}$ interagit avec $(\mathcal{H}, \mathcal{S}^\mathcal{H})$. Le simulateur final est celui qui est décrit dans la Figure 6.2 de la section 6.3. Il s'agit donc du même simulateur que pour la preuve d'indifférenciabilité de la section 6.3, sans le sous-programme $\mathcal{I}$ permettant de traduire l'information venant de l'interface $\mathcal{H}^{reveal}$ de l'oracle aléatoire à usage public.

$$\Pr\left[W_9\right] = \Pr\left[W_8\right] \ . \tag{6.14}$$

6.4.5 Majoration de l'avantage de l'attaquant

Au cours de la séquence de jeux, la seule différence sur la vue de l'attaquant intervient entre les Jeux 7 et 8. D'après l'équation (6.6) et le lemme de différence montré au chapitre 5, nous avons

$$\left|\Pr\left[W_8\right] - \Pr\left[W_7\right]\right| \leq \max(\Pr\left[\mathsf{Echec}[7]\right], \Pr\left[\mathsf{Echec}[8]\right]) \ .$$

Tant qu'aucun cas d'échec ne s'est produit, ces deux jeux sont équivalents, d'où

$$\Pr\left[\mathsf{Echec}[7]\right] = \Pr\left[\mathsf{Echec}[8]\right]$$

D'après la propriété 13, un évènement d'échec au Jeu 7 implique un évènement d'échec au Jeu 6. Nous avons donc

Initialisation de $\mathcal{S}_8$

1. choisir $\tau \in [0, n - \ell_h]$
2. définir $X = \{x^0\}$, $E = \emptyset$ et $\mathcal{P} = \{(x^0, \emptyset)\}$
3. définir Drapeau = Succès

Simulation de $\mathcal{F}$

Entrées : (M, A, B)

Sorties : (A', B')

1. définir $x = (A, B)$
2. pour tout $(\tilde{M}, \tilde{A}, \tilde{B}) \in \mathcal{R}(M, A, B, 0, \tau) \cup \{(M, A, B)\}$ tel que $(\tilde{A}, \tilde{B})$ a un chemin μ dans le graphe $\mathcal{G}$ tel que $\mu||\tilde{M}$ est un chemin utile
   (a) définir $\tilde{x} = (\tilde{A}, \tilde{B})$
   (b) si $\tilde{\mu}||\tilde{M}$ est un chemin complet
      i. calculer $\mathcal{M} = \mathsf{unpad}(\tilde{\mu}||\tilde{M})$
      ii. appeler $\mathcal{H}$ pour obtenir $h = \mathcal{H}(\mathcal{M})$
      iii. définir $\tilde{B}' = h$
      iv. lancer A-Simule$(\tilde{M}, \tilde{A}, \tilde{B}, \tilde{B}', E)$ pour obtenir $\tilde{A}'$
   (c) sinon
      i. lancer Simule$(\tilde{M}, \tilde{A}, \tilde{B}, E)$ pour obtenir $(\tilde{A}', \tilde{B}')$
   (d) définir $\tilde{y} = (\tilde{A}', \tilde{B}')$
   (e) si $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y} \notin E$
      i. si $\tilde{y}$ a un chemin utile $\tilde{\mu}'$ dans $\mathcal{G}$, NoteEchec(Echec_{a1})
      ii. s'il existe un arc $\tilde{y} \xrightarrow{\tilde{M}'} \tilde{z}$ dans E tel que $\mu||\tilde{M}||\tilde{M}'$ est un chemin utile, NoteEchec(Echec_{a2})
      iii. si $(0, \tau) - \mathsf{TestAtypique}(\tilde{y}, \mu||\tilde{M})$, NoteEchec$(\mathsf{Echec}_b)$
      iv. s'il existe (z, μ_z) dans $\mathcal{P}$ tel que $(0, \tau) - \mathsf{TestRelatifs}_1(\tilde{y}, \mu||\tilde{M}, z, \mu_z)$, NoteEchec$(\mathsf{Echec}_{c1})$
      v. s'il existe $z \xrightarrow{M_z} t$ dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ tel que $(0, \tau) - \mathsf{TestRelatifs}_2(\tilde{y}, \mu||\tilde{M}, z, M_z)$, NoteEchec$(\mathsf{Echec}_{c2})$
      vi. si $(0, \tau) - \mathsf{TestRelatifs}_1(\tilde{y}, \mu||\tilde{M}, \tilde{y}, \mu||\tilde{M})$,NoteEchec$(\mathsf{Echec}_d)$
   (f) ajouter les nœuds $\tilde{x}$ et $\tilde{y}$ à X et l'arc $\tilde{x} \xrightarrow{\tilde{M}} \tilde{y}$ à E
   (g) pour tout $(\tilde{x}, c) \in \mathcal{P}$ tel que $c||\tilde{M}$ est un chemin utile, ajouter $(\tilde{y}, c||\tilde{M})$ à $\mathcal{P}$
3. lancer Simule(M, A, B, E) pour obtenir $y = (A', B')$
4. ajouter les nœuds x et y à X et l'arc $x \xrightarrow{M} y$ à E
5. renvoyer (A', B').

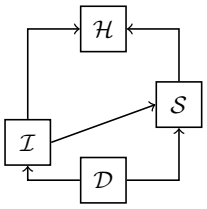


FIGURE 6.15 – Simulateur $\mathcal{S}_8$ pour $\mathcal{F}$ au Jeu 8 pour la preuve d'indifférenciabilité forte de Chop-MD.

$$\Pr\left[\mathsf{Echec}[7]\right] \leq \Pr\left[\mathsf{Echec}[6]\right] \ ,$$

d'où

$$|\Pr\left[W_8\right] - \Pr\left[W_7\right]| \leq \Pr\left[\mathsf{Echec}[6]\right] \ .$$

D'après l'équation (6.15) montrée dans la section 6.5,

$$|\Pr\left[W_8\right] - \Pr\left[W_7\right]| \leq \Pr\left[\mathsf{Echec}_e[6]\right] + \Pr\left[\mathsf{Echec}[5]\right] \ .$$

D'après les équations (6.9) à (6.14),

$$\Pr\left[W_7\right] = \Pr\left[W_0\right] \text{ et } \Pr\left[W_9\right] = \Pr\left[W_8\right] \ ,$$

d'où

$$\mathsf{Adv}(\mathcal{D}) = |\Pr\left[W_9\right] - \Pr\left[W_0\right]| \leq \Pr\left[\mathsf{Echec}_e[6]\right] + \Pr\left[\mathsf{Echec}[5]\right] \ .$$

Enfin, les lemmes 7 et 8 montrés en section 6.5 nous permettent de conclure que

$$\begin{aligned}\mathsf{Adv}(\mathcal{D}) \ \leq \ & \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep})(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 \\ & + \mathcal{R}_{indep}(\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} N + (\mathcal{R}_x + 1) 2^{-(n-\ell_h-\tau)} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] N\end{aligned}$$

si $\mathcal{R}_{indep} \geq 1$, et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)} \frac{N(N+1)}{2} + \mathcal{X}_{atyp} 2^{-(n-\tau)} N + 2^{-(n-\ell_h-\tau)} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] N$$

si $\mathcal{R}_{indep} = 0$. Dans le cas d'un encodage sans préfixe, ces bornes deviennent

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep} + 1)(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 + \mathcal{R}_{indep}(\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} N$$

si $\mathcal{R}_{indep} \geq 1$, et

$$\mathsf{Adv}(\mathcal{D}) \leq 2^{-(n-\tau)} N^2 + \mathcal{X}_{atyp} 2^{-(n-\tau)} N$$

si $\mathcal{R}_{indep} = 0$.

Nous retrouvons donc bien les bornes annoncées dans le théorème 9.

6.4.6 Application

Le théorème 9 permet de dériver une borne d'indifférenciabilité de l'oracle aléatoire pour la fonction SIMD, en tenant compte des distingueurs connus. La sécurité de cette fonction a été étudiée par Bouillaguet, Fouque et Leurent dans [BFL10]. Pour un état interne de taille n, les propriétés connues de la fonction de compression sont les suivantes :

1. il existe $2^{n/2+16}$ couples d'entrées-sorties avec des propriétés particulières, proches de celles de points fixes ;
2. il existe des paires d'entrées (M, A, B) et (M', A', B') pour lesquelles les sorties de $\mathcal{F}$ sont corrélées. Ces entrées vérifient $(A, B) \neq (A', B')$ et pour tout (A, B), il existe au plus un nœud (A', B') tel qu'il existe (M, M') tels que $\mathcal{F}_M(A, B)$ et $\mathcal{F}_{M'}(A', B')$ sont corrélés.

La propriété 1 se traduit par $\mathcal{X}_{atyp} = 2^{n/2+16}$, et la propriété 2 se traduit par $\mathcal{R}_{indep} = 1$, $\mathcal{X}_{rel} = 0$ et $\mathcal{R}_x = 1$. Dans tous les cas non évoqués dans ces propriétés, la sortie de $\mathcal{F}$ est choisie uniformément, d'où $\varepsilon = \tau = 0$. Nous en déduisons que pour tout distingueur $\mathcal{D}_{\mathsf{SIMD}}$:

$$\mathsf{Adv}(\mathcal{D}_{\mathsf{SIMD}}) \leq 3 \times 2^{-n} N^2 + 2^{-n/2+16} N \ .$$

Cette borne est à comparer avec celle donnée dans [BFL10] :

$$\mathsf{Adv}(\mathcal{D}_{\mathsf{SIMD}}) \leq 16 \times 2^{-n} N^2 + 4 \times 2^{-n/2+16} N + 4 \times 2^{n+32} \frac{N^2}{(2^n - N)^2} \ .$$

Nous trouvons donc une borne d'indifférenciabilité proche de celle donnée dans [BFL10], avec un gain d'un petit facteur constant.

6.5 Majoration des probabilités d'échec des simulateurs

Nous donnons ici une expression des probabilités d'occurrence des évènements d'échec dans les preuves du chapitre 6. Les bornes sont calculées dans les Jeux pour lesquels le simulateur reçoit les requêtes de l'attaquant en temps réel : le Jeu 5 dans la preuve d'indifférenciabilité d'un oracle aléatoire public de la section 6.3, et le Jeu 6 dans la preuve d'indifférenciabilité générale de la section 6.4.

Nous séparons les cas d'échec en deux catégories. Les évènements Echec_{a1}, Echec_{a2}, Echec_b, Echec_{c1}, Echec_{c2} et Echec_d sont similaires : une réponse renvoyée par $\mathcal{F}$ a une valeur qui provoque un évènement indésirable. Ensuite, nous nous intéressons au cas de Echec_e dans la preuve d'indifférenciabilité d'un oracle aléatoire au sens habituel.

6.5.1 Probabilité d'occurrence de Echec[5]

Pour la preuve d'indifférenciabilité de la section 6.4, le Jeu 6 est défini à partir du Jeu 5 en rajoutant la prise en compte de l'évènement Echec_e par le simulateur. Si un autre évènement d'échec Echec_i est noté au Jeu 6, aucun autre évènement n'est noté plus tôt dans le jeu. Le même évènement est noté au Jeu 5. Nous en déduisons que

$$\Pr\left[\mathsf{Echec}[6]\right] \leq \Pr\left[\mathsf{Echec}[5]\right] + \Pr\left[\mathsf{Echec}_e[6]\right] \ . \tag{6.15}$$

Nous montrons maintenant le lemme suivant.

Lemme 7 *Soit $\mathcal{S}_5$ le simulateur défini par la Figure 6.9, et $\mathcal{F} \in \mathrm{FUNC}$ une fonction tirée selon la distribution $\mathcal{V}$. Si $\mathcal{R}_{indep} \geq 1$, la probabilité d'échec de $\mathcal{S}_5$ vérifie*

$$\Pr\left[\mathsf{Echec}[5]\right] \leq \frac{(\mathcal{R}_{indep}^2 + \mathcal{R}_{indep})(\mathcal{R}_x + 1)}{2} 2^{-(n-\tau)} N^2 + \mathcal{R}_{indep}(\mathcal{X}_{rel} + \mathcal{X}_{atyp}) 2^{-(n-\tau)} N \ .$$

Si $\mathcal{R}_{indep} = 0$, nous avons

$$\Pr\left[\mathsf{Echec}[5]\right] \leq 2^{-(n-\tau)} \frac{N(N+1)}{2} + \mathcal{X}_{atyp} 2^{-(n-\tau)} N \ .$$

Les valeurs de $\mathcal{R}_{indep}$, $\mathcal{R}_x$, $\mathcal{X}_{rel}$ et $\mathcal{X}_{atyp}$ sont définies en section 6.2.

Démonstration : Nous montrons ce résultat en majorant la probabilité d'occurrence d'un évènement d'échec lors de chaque requête à $\mathcal{F}$, notée $\Pr[\mathsf{Echec}[5](q)]$ pour la $q^{ème}$ requête émise par $\mathcal{S}_5$ à $\mathcal{F}$. Pour une telle requête, nous notons $f(q)$ le nombre de requêtes reçues par $\mathcal{F}$ déjà traitées. Réciproquement, nous notons $g(s)$ le nombre total de requêtes à $\mathcal{F}$ émises par $\mathcal{S}_5$ après le traitement de la $s^{ème}$ requête émise par $\mathcal{I}$ ou $\mathcal{D}$. Nous ne comptabilisons dans $g(s)$ que les requêtes concernant des entrées (M, x) telles que x a un chemin μ dans le graphe qui vérifie que $\mu||M$ est un chemin complet (ce qui correspond à l'étape 2 du simulateur).
Dans le Jeu 5, $\mathcal{D}$ n'a pas de connaissance supplémentaire de $\mathcal{F}$ par rapport à $\mathcal{S}_5$. Le graphe $\mathcal{G}$ contient donc toutes les valeurs de $\mathcal{F}$ qui ont déjà été révélées.
Un tel évènement peut se produire suite à une requête à $\mathcal{F}$ pour une entrée (M, x) telle que x a un chemin μ dans $\mathcal{G}$ tel que $\mu||M$ est un chemin utile. La propriété 10 garantit qu'aucune valeur de $\mathcal{F}$ pour une entrée $(M', x') \in \mathcal{R}((M, x), 0, \tau)$ n'a encore été demandée à l'oracle. Nous en déduisons qu'étant donné l'historique $E(q)$ de $\mathcal{S}_5$ avant la $q^{ème}$ requête, pour tout $y \in \{0,1\}^n$,

$$\Pr[F_M(x) = y|E] \leq 2^{-(n-\tau)} .$$

Considérons maintenant la $q^{ème}$ requête émise par $\mathcal{S}$, et supposons qu'elle soit susceptible d'entraîner un évènement d'échec. Nous cherchons maintenant à majorer le nombre de valeurs de y qui conduisent à un cas d'échec. Pour l'évènement d'échec Echec_i, nous notons N_i ce nombre. Nous commençons par traiter le cas où $\mathcal{R}_{indep}(0, \tau) \geq 1$.
Un cas d'échec n'étant noté que si aucun cas d'échec n'a été identifié auparavant, les bornes de la propriété 12 s'appliquent. Chacune des $g(f(q))$ requêtes à $\mathcal{F}$ émises par $\mathcal{S}$ depuis la réception de la $f(q)^{ème}$ requête de $\mathcal{D}$ ou $\mathcal{I}$ entraîne la création d'un chemin pour au plus 1 nœud.
Si Echec_{a1} intervient, alors y a un chemin dans le graphe. Nous avons donc

$$\begin{aligned} N_{a1} &\leq |\mathcal{P}(f(q)+1)| + q - g(f(q)) \\ &\leq 1 + (f(q))\mathcal{R}_{indep} + q - g(f(q)) . \end{aligned}$$

Si Echec_{a2} se produit, y est le point de départ d'un arc de E. Si y a un chemin dans $\mathcal{G}$, y est déjà comptabilisé dans Echec_{a1}. Les autres valeurs de y susceptibles d'entraîner Echec_{a2} sont les variables de chaînage de requêtes émises par $\mathcal{D}$. Nous en déduisons

$$N_{a2} \leq f(q) .$$

Echec_b ne peut se produire que s'il existe M_y tel que (M_y, y) soit une entrée atypique de $\mathcal{F}$. Nous avons donc

$$N_b \leq \mathcal{X}_{atyp} .$$

Si Echec_{c1} se produit, alors il existe y' ayant un chemin utile dans $\mathcal{G}$ et M_y, M'_y tels que $(M_y, y) \in \mathcal{R}((M'_y, y'), 0, \tau)$. Nous en déduisons

$$\begin{aligned} N_{c1} &\leq (|\mathcal{P}(f(q)+1)| + q - g(f(q)))\,\mathcal{R}_x \\ &\leq (1 + f(q)\mathcal{R}_{indep} + q - g(f(q)))\,\mathcal{R}_x . \end{aligned}$$

Si Echec_{c2} est identifié, alors il existe un arc $M_{y'} \overset{M'}{\rightarrow} z$ dans E et un bloc M_y tels que $(M_y, y) \in \mathcal{R}((M'_y, y'), 0, \tau)$. Le cas où y' a un chemin dans $\mathcal{G}$ est déjà comptabilisé dans N_{a1}, donc

$$N_{c2} \leq f(q)\mathcal{R}_x .$$

L'occurrence de Echec_d implique l'existence de M_y, M'_y tels que $(M_y, y) \in \mathcal{R}((M'_y, y), 0, \tau)$. Par définition, nous avons donc

$$N_d \leq \mathcal{X}_{rel} \ .$$

Nous en déduisons la borne suivante pour $\Pr[\mathsf{Echec}[5](q)]$:

$$\begin{aligned}\Pr[\mathsf{Echec}[5](q)] &\leq (N_{a1} + N_{a2} + N_b + N_{c1} + N_{c2} + N_d)2^{-(n-\tau)} \\ &\leq ((1+\mathcal{R}_x)(q - g(f(q)) + 1 + (\mathcal{R}_{indep}+1)f(q)) + \mathcal{X}_{atyp} + \mathcal{X}_{rel})\, 2^{-(n-\tau)} \ .\end{aligned}$$

En notant u le nombre total de requêtes de $\mathcal{S}_5$ à $\mathcal{F}$ à l'étape 2 et v le nombre total de requêtes reçues par $\mathcal{S}_5$, nous avons la borne suivante sur $\Pr[\mathsf{Echec}[5]]$

$$\begin{aligned}\Pr[\mathsf{Echec}[5]] &\leq \sum_{q=0}^{u} \Pr[\mathsf{Echec}[5](q)] \\ &\leq \sum_{q=1}^{u} ((1+\mathcal{R}_x)(q - g(f(q)) + 1 + (\mathcal{R}_{indep}+1)f(q)) + \mathcal{X}_{atyp} + \mathcal{X}_{rel})\, 2^{-(n-\tau)} \ .\end{aligned}$$

Nous divisons cette somme suivant les valeurs de $f(q)$:

$$\begin{aligned}\Pr[\mathsf{Echec}[5]] &\leq \sum_{r=0}^{v-1} \sum_{q=g(r)}^{g(r+1)-1} \Pr[\mathsf{Echec}[5](q)] \\ &\leq \sum_{r=0}^{v-1} \sum_{q=0}^{g(r+1)-g(r)-1} \Pr[\mathsf{Echec}[5](g(r)+q)] \\ &\leq 2^{-(n-\tau)} \left((\mathcal{X}_{atyp} + \mathcal{X}_{rel})u + (1+\mathcal{R}_x) \sum_{r=0}^{v-1} \sum_{q=1}^{g(r+1)-g(r)} (\mathcal{R}_{indep}+1)r + q \right)\end{aligned}$$

Les cas d'échec n'étant identifiés que si Drapeau = Succès au moment du test, nous pouvons borner le nombre $g(r+1) - g(r)$ de requêtes effectuées à l'étape 2 du simulateur par $\mathcal{R}_{indep}$, et le nombre total de ces requêtes u par $\mathcal{R}_{indep}N$, N étant le poids maximal des requêtes émises par $\mathcal{D}$. Nous avons donc

$$\begin{aligned}\Pr[\mathsf{Echec}[5]] &\leq 2^{-(n-\tau)} \left((\mathcal{X}_{atyp} + \mathcal{X}_{rel})\mathcal{R}_{indep}N + (1+\mathcal{R}_x) \sum_{r=0}^{N-1} \sum_{q=1}^{\mathcal{R}_{indep}} (\mathcal{R}_{indep}+1)r + q \right) \\ &\leq 2^{-(n-\tau)} \left((\mathcal{X}_{atyp} + \mathcal{X}_{rel})\mathcal{R}_{indep}N + (1+\mathcal{R}_x)(\sum_{q=1}^{(\mathcal{R}_{indep}+1)N} q - \sum_{r=0}^{N-1} (\mathcal{R}_{indep}+1)(r+1)) \right)\end{aligned}$$

En effet, la somme de sommes dans la premières équation consiste à sommer tous les q de 0 à $(\mathcal{R}_{indep}+1)N$, à l'exception des multiples de $\mathcal{R}_{indep}+1$. Nous en déduisons

$$\begin{aligned} & \sum_{q=1}^{(\mathcal{R}_{indep}+1)N} q - \sum_{r=0}^{N-1}(\mathcal{R}_{indep}+1)(r+1) \\ \leq & \frac{(R_{indep}+1)N((R_{indep}+1)N+1)}{2} - \frac{(R_{indep}+1)N(N+1)}{2} \\ \leq & \frac{(R_{indep}+1)}{2}N^2 \ , \end{aligned}$$

d'où nous déduisons

$$\Pr\left[\mathsf{Echec}[5]\right] \leq 2^{-(n-\tau)}(\mathcal{X}_{atyp}+\mathcal{X}_{rel})\mathcal{R}_{indep}N + 2^{-(n-\tau)}(1+\mathcal{R}_x)\frac{(R_{indep}+1)}{2}N^2 \ ,$$

ce qui constitue la borne de l'énoncé.
Dans le cas où $\mathcal{R}_{indep} = 0$, les ensembles de relatifs sont tous vides. Nous avons donc également $\mathcal{R}_x = 0$ et $\mathcal{X}_{rel} = 0$. Le biais éventuel sur $\mathcal{F}$ réside dans la présence d'entrées atypiques et dans la borne τ. Les cas d'échec ne peuvent se produire que si $\mathcal{D}$ émet une requête pour une entrée (M, x) telle que x a un chemin μ dans $\mathcal{G}$. Pour la $q^{ème}$ requête reçue par $\mathcal{S}$, le même raisonnement que celui exposé en annexe A permet de borner la probabilité d'occurrence de $\mathsf{Echec}_{a1} \vee \mathsf{Echec}_{a2}$ par

$$\Pr\left[\mathsf{Echec}_{a1} \vee \mathsf{Echec}_{a2}\right] \leq \frac{N(N+1)}{2}2^{-(n-\tau)}$$

Le nombre de sorties y provoquant Echec_b est

$$N_b = \mathcal{X}_{atyp} \ ,$$

ce qui implique que

$$\Pr\left[\mathsf{Echec}_b\right] \leq N2^{-(n-\tau)}\mathcal{X}_{atyp} \ .$$

Echec_{c1}, Echec_{c2} et Echec_d ne peuvent pas se produire. Nous en déduisons le résultat de l'énoncé.
□

6.5.2 Probabilité d'occurrence de $\mathsf{Echec}_e[6]$.

La majoration de $\Pr\left[\mathsf{Echec}_e\right]$ au Jeu 6 est assez similaire à celle qui s'applique au cas Echec_3 du chapitre précédent. Elle est donnée par le lemme suivant.

Lemme 8 *Soit $\mathcal{S}_6$ le simulateur défini par la Figure 6.9, et $\mathcal{F} \in$ FUNC une fonction tirée selon la distribution $\mathcal{V}$. La probabilité d'occurrence de Echec_e vérifie*

$$\Pr\left[\mathsf{Echec}_e[6]\right] \leq (\mathcal{R}_x+1)2^{-(n-\ell_h-\tau)}\sum_{t=1}^{N}\Pr\left[t\text{-}\mathsf{Coll}\right]N \ .$$

Si l'encodage utilisé est sans préfixe, nous avons

$$\Pr\left[\mathsf{Echec}_e[6]\right] \leq (\mathcal{R}_x+1)2^{-(n-\tau)}\frac{N(N-1)}{2} \ .$$

La définition de $\mathcal{R}_x$ est donnée en section 6.2.

Démonstration : L'évènement Echec_e se produit à la condition suivante. Lorsque $\mathcal{S}_6$ reçoit une requête (M, x) provenant de $\mathcal{D}$, il existe x' dans $X_{\mathcal{I}}$ tel que x' ait un chemin μ' dans $\mathcal{G}_{\mathcal{I}}$ qui n'est pas dans $\mathcal{G}_{\mathcal{D}}$ et M' tel que $(M', x') \in \mathcal{R}((M, x), 0, \tau)$, et $\mu'||M'$ est un chemin utile. Les requêtes provenant de $\mathcal{I}$ concernent toutes des nœuds ayant un chemin dans $\mathcal{G}$, chacune d'entre elles ne peut entraîner que l'insertion d'un arc et d'un nouveau nœud dans $\mathcal{G}$. A la réception de la $q^{ème}$ requête par $\mathcal{S}$, $\mathcal{G}$ contient au maximum q tels nœuds.
Pour un encodage générique, nous pouvons supposer que $\mathcal{D}$ connaît la partie B de tous ces nœuds (qu'il a obtenue par des requêtes de haché). Lorsque $\mathcal{D}$ choisit une entrée (M, x) au hasard, la probabilité de Echec_e est majorée par la probabilité qu'il existe un x^* tel qu'il existe un relatif (M^*, x^*) de (M, x) qui coincide avec un nœud x' de $X_{\mathcal{I}}$ n'ayant pas de chemin dans $\mathcal{G}_{\mathcal{D}}$, ou que x soit lui-même dans $X_{\mathcal{I}}$ Il existe au plus $\mathcal{R}_x + 1$ tels nœuds.
Pour chacune de ces valeurs x_j, dans le pire des cas la partie B maximise la probabilité d'échec de $\mathcal{S}_6$. En notant β la taille de la plus grosse multicollision sur B parmi les nœuds $x' \in X_{\mathcal{C}} \setminus \{x^0\}$, il y a au plus β choix de A tel que $x_j = (A, B)$ soit dans $X_{\mathcal{C}} \setminus \{x^0\}$. Comme $\varepsilon = 0$, aucun biais n'est toléré sur la partie B des sorties de $\mathcal{F}$. Du point de vue de $\mathcal{D}$, la probabilité de choisir une partie A qui convient est donc majorée par

$$\frac{\beta 2^{-(n-\tau)}}{2^{-\ell_h}} .$$

En sommant sur l'ensemble des valeurs de x^*, nous en déduisons que

$$\Pr\left[\mathsf{Echec}_e(q)|\beta_q\right] \leq \frac{(1+\mathcal{R}_x)\beta_q 2^{-(n-\ell_h-\tau)}}{2}$$

En sommant sur q nous en déduisons

$$\Pr\left[\mathsf{Echec}_e|\beta_N\right] \leq \frac{(1+\mathcal{R}_x)\beta_N 2^{-(n-\ell_h-\tau)} N}{2}$$

Un raisonnement similaire basé sur la taille de la plus grosse multicollision sur la partie B de la sortie de $\mathcal{F}$ est utilisé dans la preuve du lemme 12 en annexe A. Comme $\varepsilon = 0$, les parties B des éléments de $X_{\mathcal{I}}$ sont tirées uniformément et indépendamment les unes des autres. En appliquant le même raisonnement nous en déduisons

$$\Pr\left[\mathsf{Echec}_e\right] \leq \frac{(1+\mathcal{R}_x)\sum_{t=1}^{N}\Pr\left[t\text{-}\mathsf{Coll}\right] 2^{-(n-\ell_h-\tau)} N}{2} .$$

Dans le cas d'un encodage de message sans préfixe, nous ne pouvons pas considérer que $\mathcal{D}$ connaît la partie B des éléments x' provoquant un cas d'arrêt. En effet, dans ce cas x' a un chemin μ' et est la valeur de l'état interne qui contient la valeur d'un haché, et si on concatène un bloc de message M' à μ', $\mu'||M'$ n'est pas un chemin utile, et ne peut pas provoquer Echec_e. L'attaquant ne connaît donc pas les éléments de $X_{\mathcal{I}}$ qui n'ont pas de chemin dans $\mathcal{G}_{\mathcal{D}}$. Pour le nœud x et chaque élément x^* qui vérifie l'existence de M^* tel que (M^*, x^*) soit un relatif de (M, x), la probabilité de tomber sur un nœud de $X_{\mathcal{D}} \setminus \{x^0\}$ est donc d'au plus $q2^{-(n-\tau)}$.
Nous pouvons en déduire

$$\Pr\left[\mathsf{Echec}_e(q)\right] \leq (1+\mathcal{R}_x)2^{-(n-\tau)}q .$$

En sommant sur q nous obtenons bien

$$\Pr\left[\mathsf{Echec}_e\right] \leq (1+\mathcal{R}_x)2^{-(n-\tau)}\frac{N(N-1)}{2} .$$

□

Troisième partie

Cryptanalyse de nouvelles fonctions de hachage

Chapitre 7

Cryptanalyse différentielle symétrique de RadioGatún

Sommaire

7.1 Introduction

Dans ce chapitre nous décrivons une attaque en recherche de collisions sur la famille de fonctions de hachage RadioGatún, réalisée avec Thomas Peyrin et publiée à FSE 2009 [FP09]. Cette famille a été proposée en 2006 par Bertoni, Daemen, Peeters et Van Assche [BDPA06]. Les principes de conception sur lesquels elle repose sont très différents de ceux de la plupart des fonctions existantes à l'époque. En effet, les fonctions présentées dans le chapitre 3 utilisent toutes une fonction de compression fondée sur l'utilisation d'un algorithme de chiffrement par blocs dédié, combiné avec une construction comme celle de Davies et Meyer de manière à rendre le résultat difficile à inverser. Ces fonctions de compression sont alors utilisées avec l'algorithme d'extension de domaine de Merkle-Damgård, avec un état interne étroit. Ces fonctions sont donc l'objet des attaques décrites dans le chapitre 2. De plus, la proximité de leurs principes de conception rendent la plupart d'entre elles vulnérables aux techniques de cryptanalyse différentielle décrites au chapitre 3.

RadioGatún s'inscrit donc dans le processus de définition de nouveaux paradigmes de conception en matière de hachage. La construction de cette famille ne repose pas sur l'utilisation d'une permutation paramétrée, mais sur l'utilisation d'une permutation fixe. À cet égard, les principes de conception de RadioGatún se rapprochent de ceux des algorithmes de chiffrement à flot. Fonctionnellement, RadioGatún est d'ailleurs très proche de la définition des fonctions éponges, définies au chapitre 2, qui peuvent être utilisées comme fonctions de hachage [BDPA07] ou comme algorithmes de chiffrement à flot [BDPA10]. En revanche, la sécurité de RadioGatún n'est pas garantie par la preuve d'indifférentiabilité de la construction des fonctions éponges publiée à Eurocrypt 2008 [BDPA08b], pour plusieurs raisons :

- RadioGatún utilise des tours à blanc entre la phase d'absorption et la phase d'essorage,
- l'insertion du message et la sortie du haché ne se font pas sur les mêmes parties de l'état interne,
- la permutation interne de RadioGatún étant très simple, sa modélisation par une permutation aléatoire est discutable.

RadioGatún est une famille de fonctions dont les caractéristiques dépendent principalement de deux paramètres : la taille w des registres utilisés et la longueur ℓ_h des hachés produits. L'attaque que nous présentons dans ce chapitre s'applique à la structure interne de la fonction de hachage avant la sortie du haché. Elle s'applique à tous les membres de la famille, avec une complexité indépendante de la taille du haché et équivalente à 2^{11w+2} évaluations de la fonction de compression. Il s'agit à ce jour de la meilleure attaque connue contre RadioGatún. Pour les versions recommandées par les auteurs, elle reste cependant moins efficace que l'attaque générique basée sur le paradoxe des anniversaires. En l'absence de preuve de sécurité sur RadioGatún, le résultat présenté ici contribue à l'évaluation de la sécurité offerte par cette famille de fonctions.

Contexte de l'étude. Les concepteurs de RadioGatún estiment que les fonctions de cette famille sont résistantes à la recherche de collisions tant que la taille du haché n'excède pas $19w$, pour des fonctions manipulant des registres de w bits. Pour invalider cette estimation, il faudrait donc trouver une méthode permettant d'exhiber des collisions en un temps de calcul inférieur à $2^{9,5w}$ calculs de hachés, ce qui n'est pas le cas de notre attaque. La meilleure attaque en recherche de collisions sur l'état interne découverte par les concepteurs présentée dans les spécifications de la famille nécessite le calcul de 2^{46w} hachés.

Un résultat de Bouillaguet et Fouque [BF08] permet de faire baisser la complexité de la recherche de collisions à $2^{24,5w}$. Pour y parvenir, ils génèrent des collisions sur l'état interne de la version 1 bit de RadioGatún en utilisant des techniques génériques, et en dérivent des chemins différentiels. En utilisant des techniques algébriques, ils parviennent ensuite à améliorer la complexité de la recherche de collisions. Enfin, Khovratovitch a découvert une attaque permettant de trouver des collisions à valeur initiale choisie et nécessitant 2^{18w} calculs de hachés. Cette attaque nécessite également l'utilisation d'une mémoire de taille proportionnelle à 2^{18w}. Elle est fondée sur la génération de *structures*, qui sont des ensembles spécifiques d'entrées-sorties de la permutation. Une technique similaire a été appliquée dans les attaques à clés corrélées sur l'AES [BKN09].

Résultats obtenus. L'attaque présentée dans ce chapitre améliore donc significativement la recherche de collisions internes sur RadioGatún. Elle se base sur l'utilisation d'une version de la cryptanalyse différentielle, la *cryptanalyse différentielle symétrique*. Nous montrons en annexe B un chemin différentiel trouvé à l'aide d'un algorithme de recherche automatisé, ainsi qu'une collision

sur l'état interne pour le cas $w = 2$ à titre de validation des résultats. Dans la section 7.2, nous commençons par décrire la famille de fonctions de hachage RadioGatún. Ensuite, dans la section 7.3, nous définissons les concepts de *différences symétriques* et de *mots de contrôle*, qui sont les outils principaux utilisés dans notre attaque. Dans la section 7.4, nous montrons comment générer des chemins différentiels pour RadioGatún et dans la section 7.5, nous montrons comment les utiliser pour trouver des collisions. Enfin, nous discutons de l'implication de cette attaque, et des améliorations possibles.

7.2 Description de RadioGatún

RadioGatún est une famille de fonctions de hachage dont les principes de conception se rapprochent de ceux `Panama` [DC98], `StepRightUp` [Dae95] ou `Subterranean` [CDGP93, Dae95], tout en corrigeant les problèmes de sécurité de ces fonctions.

7.2.1 Structure générale

La fonction RadioGatún[w] repose sur l'utilisation d'un état interne de 58 registres de w bits. Cet état est divisé en deux parties et initialisé en mettant tous les registres à 0. La première partie de l'état, le *mill*, contient 19 registres de w bits, et la seconde partie, le *belt*, est généralement représentée sous la forme d'une matrice de 3 lignes et 13 colonnes, dont les cases sont des registres de w bits.

Dans ce chapitre, nous notons M_i^k le i-ème registre du mill avant l'application de la k-ème itération de la fonction de compression (avec $0 \leq i \leq 18$), et $B_{i,j}^k$ représente le registre de la colonne i et de la ligne j du belt avant l'application de la k-ème itération de la fonction de compression (avec $0 \leq i \leq 12$ et $0 \leq j \leq 2$).

La première étape est l'application d'un padding injectif au message à hacher. Ce padding consiste à concaténer un bit valant 1 au message, puis à concaténer au résultat le plus petit nombre de bits valant 0 permettant d'obtenir une donnée dont la longueur totale est un multiple de $3w$ bits, qui est la taille du bloc. La donnée obtenue est ensuite divisée en blocs de 3 registres de w bits, qui sont utilisés de manière itérative pour mettre à jour l'état interne. Au cours de ce chapitre, nous notons m_i^k le i-ème registre du bloc de message m^k (avec $0 \leq i \leq 2$). La fonction de compression consiste alors en deux opérations successives :

- l'incorporation d'un nouveau bloc de message ;
- une mise à jour par application d'une permutation fixe P.

Après avoir incorporé tous les blocs de message, l'état interne est mis à jour par l'application de N_{br} applications de la permutation P appelées *tours à blanc*. Enfin, l'empreinte du message m est extraite par concaténation des valeurs de M_1^k et M_2^k obtenues après des applications successives de P. Cette structure, proche de celle des fonctions éponges, est décrite sur la figure 7.1.

7.2.2 La fonction de compression

Incorporation du message. L'incorporation du message avant la mise à jour k est définie par :

$$\begin{array}{lllllllll} B_{0,0}^k & = & B_{0,0}^k \oplus m_0^k & B_{0,1}^k & = & B_{0,1}^k \oplus m_1^k & B_{0,2}^k & = & B_{0,2}^k \oplus m_2^k \\ M_{16}^k & = & M_{16}^k \oplus m_0^k & M_{17}^k & = & M_{17}^k \oplus m_1^k & M_{18}^k & = & M_{18}^k \oplus m_2^k \end{array}$$

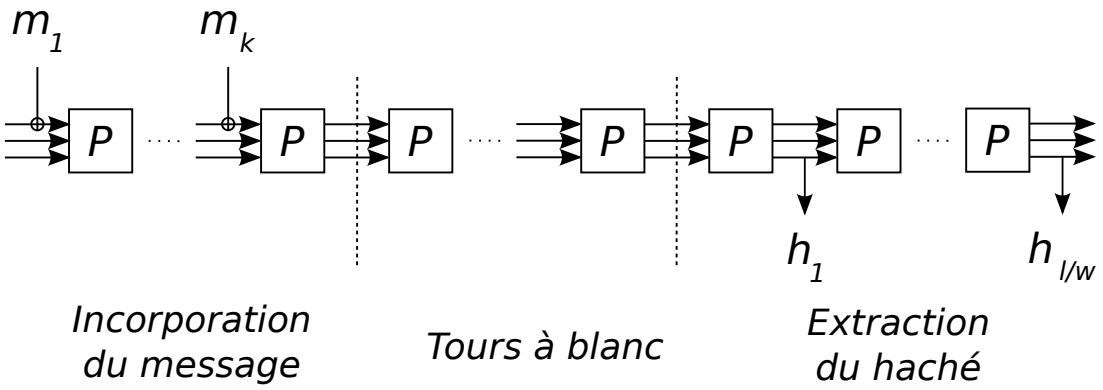


FIGURE 7.1 – Algorithme d'extension de domaine de RadioGatún.

La permutation P. La permutation P se compose de quatre opérations appliquées en parallèle. Le belt et le mill sont mis à jour par applications respectives des fonctions *Belt* et *Mill*. Les interactions entre les deux parties de l'état sont obtenues par application des fonctions *MillToBelt* et *BeltToMill* aux valeurs du belt et du mill en entrée de P. Cette structure est représentée sur la figure 7.2.

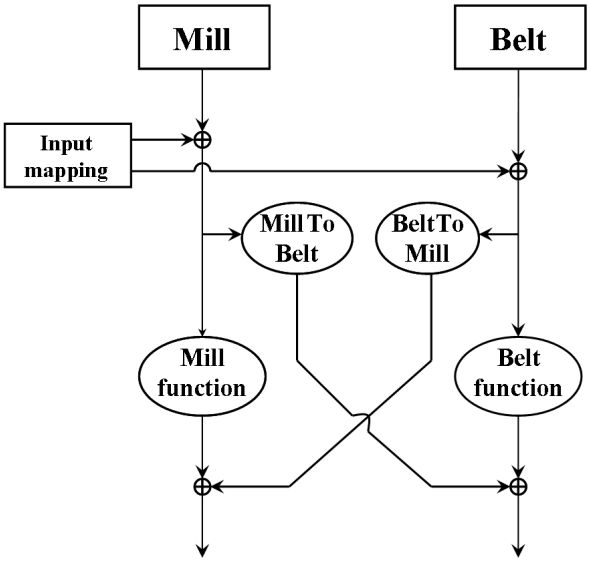


FIGURE 7.2 – Schéma général de la permutation P de RadioGatún.

La fonction *Belt* consiste en une simple rotation des lignes du belt. On a donc, pour $0 \leq i \leq 12$ and $0 \leq j \leq 2$:

$$Belt(B)_{i,j} = B_{i+1 \bmod 13,j}.$$

La fonction *Mill* est l'opération la plus complexe de la permutation interne. C'est également la seule sous-fonction de RadioGatún qui apporte de la non linéarité dans $GF(2)$. La première étape de la fonction *Mill* est une transformation non linéaire γ sur tous les registres. Pour $0 \leq i \leq 18$:

$$M_{i,\gamma} = M_i \oplus \overline{\overline{M_{i+1}} \wedge M_{i+2}},$$

Ensuite, on applique une opération de dispersion, notée π. Cette opération agit à l'intérieur des registres à travers une rotation sur chacun d'entre eux, et entre les registres en permutant leurs positions. Pour $0 \leq i \leq 18$:

$$M_{i,\pi} = M_{7\times i,\gamma} \ggg (i \times (i+1)/2).$$

Une couche de diffusion linéaire, notée θ est ensuite appliquée. Pour $0 \leq i \leq 18$:

$$M_{i,\theta} = M_{i,\pi} \oplus M_{i+1,\pi} \oplus M_{i+4,\pi}.$$

Enfin, la complémentation d'un bit de l'état casse les propriétés d'invariance par rotation de chaque registre de l'état interne. Cette opération est notée ι. Nous avons

$$\begin{aligned} M_{0,\iota} &= M_{0,\theta} \oplus 1 \\ \forall 1 \leq i \leq 18, M_{i,\iota} &= M_{i,\theta}\ . \end{aligned}$$

La sortie de la fonction *Mill* est $Mill(M) = (M_{i,\iota})_{0\leq i\leq 18}$.

La fonction *MillToBelt* permet au mill d'entrée d'influencer la valeur du belt de sortie. On a donc, pour $0 \leq i \leq 11$:

$$MillToBelt(M)_{i+1,i \bmod 3} = M_{i+1}.$$

Pour les indices (i, j) qui ne sont pas de la forme $\{(u+1), u \bmod 3\}$ avec $0 \leq u \leq 11$, on a $MillToBelt(M)_{i,j} = 0$. De manière similaire, la fonction *BeltToMill* permet au belt d'entrée d'influencer la valeur du mill de sortie. Pour $0 \leq i \leq 2$, on a :

$$BeltToMill(B)_{i+13} = B_{12,i}.$$

Les autres registres de $BeltToMill(B)$ sont tous nuls.

La sortie de la permutation interne de RadioGatún est donnée par

$$M' = Mill(M) \oplus BeltToMill(B) \tag{7.1}$$
$$B' = Belt(B) \oplus MillToBelt(M). \tag{7.2}$$

Le schéma général de ces opérations est donné sur la figure 7.3.

7.2.3 Inversibilité de la fonction de compression

Comme mentionné dans le document décrivant RadioGatún, la fonction de mise à jour de l'état interne est une permutation et son inverse peut être calculé efficacement. En effet, de manière triviale, *Belt* est inversible. $Belt^{-1}$ consiste à appliquer une rotation inverse sur les lignes du belt. La fonction *Mill* est également inversible. En effet, les couches de dispersion et d'addition de constante sont trivialement inversibles. On peut vérifier facilement que la couche de diffusion linéaire est également bijective, son inverse est donc une bijection linéaire. L'opération non linéaire γ agit indépendamment sur les w ensembles de 19 bits aux mêmes positions de tous les registres du mill.

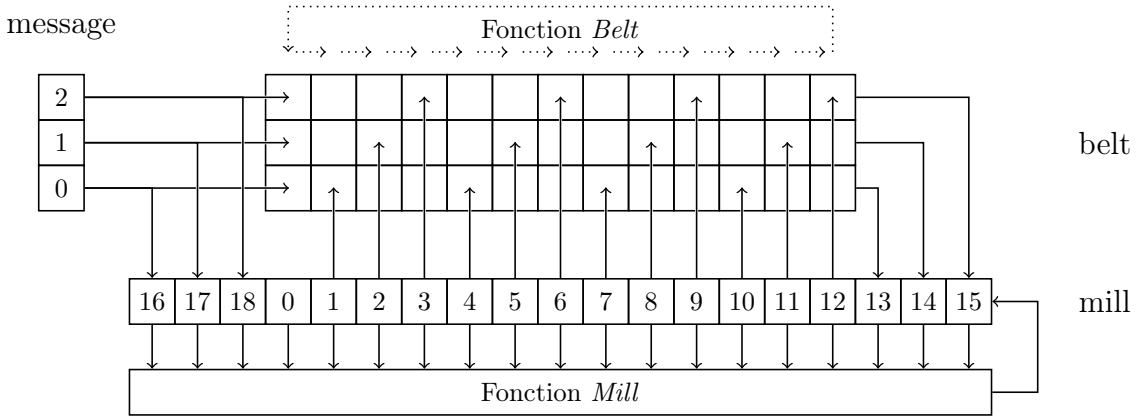


FIGURE 7.3 – Dépendances par la permutation P de RadioGatún.

Indépendamment de la valeur de w, cette opération est inversible si sa version pour $w = 1$ l'est, ce qu'on peut vérifier facilement, cette opération ayant 2^{19} entrées possibles. *Mill* est donc également inversible.

D'autre part, on peut noter que pour tout M,

$$BeltToMill(Belt^{-1}(MillToBelt(M))) = 0 \tag{7.3}$$

En effet, la colonne 0 de $MillToBelt(a)$ est nulle. Après application de l'inverse du belt, cette valeur se retrouve sur la colonne 12. L'application de *BeltToMill* au résultat renvoie alors une valeur nulle du mill.

En utilisant la linéarité de *Belt*, on réécrit alors les équations (7.1) de la manière suivante :

$$\begin{aligned} M &= Mill^{-1}(M' \oplus BeltToMill(B)) \\ B &= Belt^{-1}(B') \oplus Belt^{-1}(MillToBelt(M)). \end{aligned}$$

On déduit alors de la linéarité de *BeltToMill* :

$$\begin{aligned} M &= Mill^{-1}(M' \oplus BeltToMill(Belt^{-1}(B')) \oplus BeltToMill(Belt^{-1}(MillToBelt(M)))) \\ B &= Belt^{-1}(B') \oplus Belt^{-1}(MillToBelt(M)). \end{aligned}$$

D'après 7.3 on a donc :

$$\begin{aligned} M &= Mill^{-1}(M' \oplus BeltToMill(Belt^{-1}(B'))) \\ B &= Belt^{-1}(B') \oplus Belt^{-1}(MillToBelt(M)). \end{aligned}$$

Ces relations permettent d'inverser P. A partir de M' et B', on calcule la valeur M du mill d'entrée, puis la valeur B du belt d'entrée.

7.2.4 Fonctions de hachage utilisant des principes similaires.

RadioGatún constitue un des premiers exemples de construction de hachage utilisant un algorithme de chiffrement par flot et résistant relativement bien aux tentatives de cryptanalyse. Certains

de ses principes de conception ont été repris par la suite. Les concepteurs de RadioGatún ont formalisé certains de ces principes pour définir le concept de fonctions éponges, et l'ont repris pour définir le candidat SHA-3 Keccak, qui est sélectionné pour le dernier tour de la compétition. Keccak est un successeur naturel de RadioGatún, elle utilise une structure générale similaire et le même type d'opérations élémentaires. D'autres candidats SHA-3 tels que `Luffa` ou `cubehash` reposent sur l'utilisation de fonctions éponges ou de structures proches. Dans RadioGatún apparaît également l'idée de séparer un état interne de grande taille en deux parties, la plus petite étant mise à jour à chaque étape et la plus grande servant d'accumulateur d'entropie. On retrouve des idées similaires dans la conception de Shabal.

7.3 Deux outils de cryptanalyse

Dans cette section nous introduisons les deux principaux concepts utilisés par notre attaque. La première est la notion de différence symétrique, qui permet de trouver des chemins différentiels ayant une forme spécifique. Nous étudions ensuite la manière d'utiliser les degrés de liberté sur le message dans le but de trouver une paire de messages satisfaisant un chemin différentiel donné. Pour ce faire nous définissons le concept de mots de contrôle.

7.3.1 Différences symétriques

Nous commençons par définir le concept de *cryptanalyse différentielle symétrique*. Il s'agit d'une application particulière de la cryptanalyse différentielle, que nous avons définie au chapitre 3. Cette technique a été découverte par Rijmen *et al.* et introduite dans [RRPV01]. Elle a été mentionnée comme une menace potentielle pour la sécurité de RadioGatún dans l'article décrivant la famille [BDPA06].

La technique de cryptanalyse différentielle symétrique repose sur la restriction de la forme des différences considérées à un des sous-espaces de l'ensemble des différences qui sont stables par les opérations linéaires mises en œuvre dans la fonction. Dans le cas de RadioGatún, les opérations élémentaires utilisées sont définies sur $GF(2)$. La notion naturelle de différence qui en découle est le ou exclusif. La cryptanalyse différentielle symétrique revient alors à chercher des chemins différentiels pour lesquels la différence visée sur chaque registre de taille w est dans un sous-espace vectoriel de $GF(2)^w$.

Plus précisément, dans le cadre de notre attaque, nous nous intéressons uniquement aux valeurs de différences 0^w et 1^w. En d'autres termes, lors de la recherche de collisions, la différence sur chaque registre est soit nulle, soit porte sur tous les bits du registre.

L'intérêt de la restriction de la forme des différences à ce sous-espace vectoriel est double. D'une part, on restreint considérablement la forme des chemins différentiels considérés, ce qui augmente les chances d'en trouver un qui aboutisse à une différence nulle. D'autre part, ce sous-espace de $GF(2)^w$ est stable par la plupart des opérations impliquées dans l'exécution de RadioGatún. Les rotations à l'intérieur des registres utilisées par la fonction *Mill* laissent les différences 0^w et 1^w inchangées. De plus, le résultat d'une addition entre deux registres portant une différence symétrique porte également une différence symétrique :

$$0^w \oplus 0^w = 0^w \qquad 0^w \oplus 1^w = 1^w \qquad 1^w \oplus 0^w = 1^w \qquad 1^w \oplus 1^w = 0^w$$

Seules les opérations non linéaires de la fonction *Mill* sont susceptibles de conduire à des valeurs de différences n'appartenant pas à ce sous-espace. Il est donc nécessaire d'étudier plus précisément

la propagation de telles différences par ces opérations. Nous commençons par réécrire les opérations non linéaires en utilisant la formule suivante :

$$\overline{\overline{a} \wedge b} = a \vee \overline{b}.$$

Si a et b portent une différence symétrique, le résultat de cette opération porte également une différence symétrique avec une certaine probabilité, ou, de manière équivalente, sous certaines conditions sur a et b. Pour désigner un tel évènement, nous utilisons le terme de transition différentielle. Nous résumons ces résultats dans le Tableau 7.1. Δ_a et Δ_b représentent les différences respectives sur les opérandes a et b, et $\Delta_{a\vee\overline{b}}$ représente la différence attendue sur le résultat de l'opération $a \vee \overline{b}$. La colonne "probabilité" contient la probabilité qu'une telle différence affecte le résultat de l'opération lorsque a et b sont tirés uniformément, et la colonne "condition" donne les conditions nécessaires et suffisantes sur les valeurs de a et b pour obtenir la différence voulue sur le résultat de l'opération $a \vee \overline{b}$.

Δ_a	Δ_b	$\Delta_{a\vee\overline{b}}$	Probabilité	Condition
0^w	0^w	0^w	1	
0^w	1^w	0^w	2^{-w}	$a = 1^w$
0^w	1^w	1^w	2^{-w}	$a = 0^w$
1^w	0^w	0^w	2^{-w}	$b = 0^w$
1^w	0^w	1^w	2^{-w}	$b = 1^w$
1^w	1^w	0^w	2^{-w}	$a = b$
1^w	1^w	1^w	2^{-w}	$a \neq b$

Table 7.1 – Propagation des différences symétriques à travers les opérations non linéaires de la fonction *Mill* de RadioGatún. Δ_a et Δ_b représentent les différences respectives sur les opérandes a et b, et $\Delta_{a\vee\overline{b}}$ représente la différence attendue sur le résultat de l'opération $a \vee \overline{b}$.

L'utilisation de différences symétriques nous permet maintenant de simplifier l'analyse de la famille de fonctions RadioGatún. En effet, nous cherchons uniquement des différences qui affectent les w bits de chaque registre de manière identique. La recherche de chemins différentiels se fait donc de manière indépendante de w, et revient donc à analyser la fonction RadioGatún[1]. Cette fonction utilise des registres de 1 bit, donc les opérations de rotation disparaissent. Cette opération n'a donc pas besoin d'être analysée. Cependant, la probabilité de chaque transition différentielle sur une opération non linéaire du *Mill* dépend de w, comme le montre la table 7.1. De manière équivalente, cet évènement nécessite la satisfaction de w conditions indépendantes sur les entrées. On peut noter que ces probabilités reflètent l'unique source d'incertitude concernant la propagation de différences dans RadioGatún, et le produit de ces probabilités représente par conséquent le cœur du calcul de la complexité de l'attaque.

D'autre part, on peut noter dès à présent que toutes les conditions à remplir sur l'entrée d'une fonction *Mill* ne sont pas nécessairement indépendantes. Par exemple, la présence simultanée de différences sur M_0 et M_2 implique deux conditions sur la valeur de chaque bit de M_1. Ce phénomène peut alors avoir deux effets : soit les conditions induites sont incompatibles, soit elles sont équivalentes, ce qui a pour effet d'augmenter la probabilité qu'elles soient simultanément satisfaites. En

effet, selon les transitions différentielles prévues, il est possible d'avoir deux jeux de conditions de la forme $M_1 = 0^w$ et $M_1 = 1^w$ dans l'exemple précédent. Il est alors impossible de trouver deux messages vérifiant un chemin différentiel pour lequel un tel cas de figure se produit. Dans le deuxième cas, les conditions induites par les deux transitions différentielles impliquent la même valeur de M_1. Elles sont donc redondantes, et remplies simultanément avec probabilité 2^{-w}. Ces aspects doivent être pris en compte lors de la phase de recherche de chemins différentiels, et non au cours de la recherche de messages suivant un chemin déjà trouvé.

7.3.2 Mots de contrôle

L'application de la cryptanalyse différentielle à une fonction de hachage requiert principalement la spécification de deux phases. La première est la sélection d'un chemin différentiel, ou éventuellement un algorithme permettant d'en trouver un. La seconde est l'utilisation des degrés de liberté restant sur le message. Dans la section suivante, nous décrirons une méthode permettant de trouver des chemins différentiels symétriques intéressants. Si la probabilité associée au chemin différentiel est P, la complexité d'une attaque naïve en recherche de collisions est de l'ordre de $2/P$ calculs de hachés : l'attaquant choisit des paires de messages dont la différence est donnée par le chemin différentiel, de manière aléatoire et non adaptative, et calcule leurs valeurs de hachés. Le nombre moyen de paires à tester est alors $1/P$. Cependant, l'attaquant peut généralement augmenter sa probabilité de succès en tirant parti des degrés de liberté restants. En effet, pour chaque paire de messages $\{m, m'\}$, l'attaquant peut choisir entièrement la valeur de m, celle de m' lui est alors imposée par le chemin différentiel. L'attaquant peut également choisir les valeurs de m adaptativement, c'est à dire en fonction des résultats obtenus lors des tests de paires de messages précédentes.

Dans le cas de RadioGatún, 3 mots de w bits sont incorporés à l'état interne lors de chaque itération. Les valeurs de ces mots sont ensuite diffusées dans tout l'état interne, cependant cette diffusion n'est pas immédiate. Comme nous cherchons à remplir certaines conditions sur les valeurs d'entrée de la fonction *Mill*, nous étudions plus spécifiquement le nombre d'itérations nécessaires pour qu'un registre du mill dépende des mots de messages incorporés.

La table 7.2 résume les dépendances entre les mots incorporés lors de l'itération k, et les 19 mots du mill en entrée de la permutation aux itérations k, $k+1$ et $k+2$. Le premier tableau (respectivement le deuxième et le troisième) représente à l'aide du symbole ✓ les dépendances en les mots m_0^k (respectivement m_1^k et m_2^k) des registres d'entrée de la fonction *Mill* aux itérations k, $k+1$ et $k+2$.

De manière informelle, on peut alors calculer certaines parties du mill avec une ou deux itérations d'avance, et adapter le choix des blocs de messages en conséquence. Du fait des opérations non linéaires, il n'est pas toujours possible de choisir la valeur d'un registre de manière à satisfaire un jeu de conditions. Ces phénomènes doivent être pris en compte lors de la recherche automatisée de chemins différentiels et lors de l'estimation effective de la complexité de l'attaque en recherche de collisions induite.

7.4 Un algorithme de recherche de chemins différentiels

Nous décrivons maintenant la manière de représenter des chemins différentiels, puis les caractéristiques des chemins que nous cherchons, et enfin un algorithme permettant d'en trouver.

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																	✓		
k+1		✓	✓		✓	✓				✓			✓	✓				✓	
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																		✓	
k+1		✓			✓	✓				✓			✓	✓		✓	✓		
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																			✓
k+1		✓			✓	✓		✓	✓				✓			✓	✓		
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE 7.2 – Dépendances entre les mots de messages incorporés lors de l'itération k et les 19 registres d'entrée de la fonction *Mill* de RadioGatún aux itérations k, $k+1$ et $k+2$.

7.4.1 Représentation des chemins différentiels

Nous cherchons maintenant à construire un chemin différentiel impliquant uniquement des différences symétriques. La différence sur chaque registre du bloc de message ou de l'état interne peut alors être représentée par un bit, valant 0 si le registre ne porte pas de différence ou 1 s'il porte une différence 1^w. L'objectif de notre attaque est la recherche de collisions internes sur RadioGatún, c'est à dire des paires de messages M, M' dont le traitement conduit à des valeurs identiques de l'état interne avant les tours à blanc. Nous cherchons donc des chemins différentiels de la forme suivante :

- L'état interne ne porte pas de différence au départ du chemin.
- Les itérations successives de P induisent uniquement des transitions différentielles symétriques.
- Après la dernière insertion de message, l'état interne ne porte aucune différence.

Nous avons vu dans la section 7.2 que les différentes opérations effectuées sur l'état interne au cours du calcul de P sont inversibles. Pour toutes les opérations linéaires, on peut retrouver la valeur de la différence sur les entrées à partir de la valeur de la différence sur les sorties. Par conséquent, les différences sur les valeurs de l'état interne en entrée et en sortie de P contiennent suffisamment d'information pour retrouver les valeurs des différences en entrée et en sortie de la couche non linéaire. La spécification de la séquence des différences en entrée et en sortie de P suffit donc à spécifier le chemin différentiel.

7.4.2 Principe de la recherche de chemins différentiels

Notre algorithme de recherche de chemins différentiel repose sur la technique de rencontre au milieu. Plus précisément, nous construisons un chemin différentiel DP en connectant deux chemins générés indépendamment, DP_f et DP_b. Le but de l'algorithme décrit ici est bien la recherche d'un chemin différentiel, la recherche effective de collisions constitue une étape ultérieure de l'attaque. DP_f est construit dans le sens direct, en partant d'une différence nulle sur les valeurs initiales de l'état interne, qui reflète le fait que les états internes sont identiques après la phase d'initialisation. DP_b est quant à lui construit dans le sens inverse, en partant d'une différence nulle sur l'état interne puisque l'on cherche à obtenir une collision sur l'état interne à la fin du chemin DP.

En partant d'un état interne ne comportant pas de différence, la recherche du chemin DP_f

consiste en un parcours d'un arbre, dont les nœuds internes correspondent aux différences sur l'état interne avant incorporation du bloc de message. Pour générer les fils d'un noeud n, l'algorithme parcourt l'ensemble des $2^3 = 8$ différences symétriques possibles sur le bloc de message, et nous étudions les effets de son incorporation sur l'état interne. Etant donné la différence sur le mill en entrée de P, l'algorithme génère les fils de n en parcourant l'ensemble des transitions différentielles symétriques possibles (en se basant sur le Tableau 7.1, puis en appliquant la fin de la fonction *Mill* et les autres opérations de P (qui ont des effets déterministes sur la propagation de différences). Pour le chemin DP_f, la profondeur d'un nœud correspond alors au nombre d'itérations de la fonction de compression pour parvenir à une différence donnée sur l'état.

De manière similaire, le chemin DP_b est calculé en sens inverse en parcourant un arbre, dont les nœuds sont des valeurs de différences sur l'état interne avant désinsertion d'un message. Les fils d'un nœud sont obtenus en parcourant l'ensemble des différences possibles sur le message à désinsérer, puis en parcourant les propagations différentielles symétriques par P^{-1}, ce qui implique le parcours des transitions différentielles symétriques inverses par l'étape non linéaire de la fonction *Mill*, qui est bijective. La transition directe $\delta \rightarrow_{\gamma} \delta'$ et la transition inverse $\delta' \rightarrow_{\gamma^{-1}} \delta$ se produisent avec des probabilités identiques.

La technique de rencontre au milieu consiste à identifier deux chemins DP_f et DP_b qui composent un chemin différentiel complet.

7.4.3 Entropie

Les principes évoqués jusqu'ici permettent de parcourir l'ensemble des chemins différentiels symétriques possibles. Cependant, le chemin que nous cherchons à générer doit ensuite être utilisé pour trouver une collision. La complexité de la recherche de collisions dépend en grande partie du nombre de conditions suffisantes qui devront être remplies au cours du calcul des empreintes. L'estimation exacte de la complexité associée à un chemin différentiel donné est relativement complexe. Pendant la phase de recherche, cette estimation est rendue encore plus difficile par le fait que l'algorithme ne dispose à chaque instant que de chemins partiels. En effet, comme nous l'avons vu dans la section 7.3, certains registres du mill ne dépendent du bloc de message qu'après une ou deux itérations de P.

Afin d'accélérer la recherche et d'éviter de considérer des chemins dont la complexité est trop élevée, nous introduisons le concept d'*entropie*, qui permet d'estimer rapidement mais peu précisément la complexité associée à un chemin partiel. Nous notons H^k l'entropie d'un chemin à la profondeur k et c^k le nombre de conditions sur la valeur de registres du mill qui doivent être remplies avant l'itération k de P. Nous définissons l'entropie par récurrence à partir de la fin d'un chemin différentiel de longueur n à l'aide de la formule suivante :

$$\begin{cases} H^n = 0 \\ \forall k < n, H^k = \max(H^{k+1} + c^k - 3, 0) \end{cases} \tag{7.4}$$

L'entropie est définie dans le but d'évaluer le nombre de paires de messages de longueur k qu'il est nécessaire de générer pour obtenir une collision à l'étape n. Un raisonnement heuristique simple montre que la quantité 2^{wH_k} est une bonne estimation de ce nombre. Supposons que les messages de longueur k générés remplissent tous les conditions suffisantes du chemin à l'itération k. Le nombre de blocs de messages qu'il est possible d'insérer est 2^{3w}, et la probabilité que chacun d'entre eux remplisse les conditions de l'itération $k+1$ est $2^{-wc^{k+1}}$. On peut donc estimer à $max(r2^{w(c^{k+1}-3)}, 1)$

le nombre de messages de longueur k respectant le chemin différentiel qui sont nécessaires pour générer r messages de longueur $k+1$. Par induction, on peut alors estimer à 2^{wH^k} le nombre de paires de messages de longueur k qui est nécessaire pour aboutir à $2^0 = 1$ collision.

Lorsque w est relativement grand, la complexité de la recherche de collisions peut alors être rapidement estimée à $2^{wH_{max}}N_{max}$ évaluations de la fonction de compression, où H_{max} est la valeur maximale de l'entropie au cours du chemin et N_{max} le nombre d'itérations atteignant H_{max}. Cette estimation suppose d'une part que le choix de blocs de messages à l'itération k permettant de remplir les conditions à l'itération $k+1$ est toujours possible, et d'autre part que l'attaquant est capable de les identifier immédiatement. Ce n'est pas le cas dans la réalité, et la section 7.5 contient une évaluation plus précise de la complexité de la recherche de collisions associée à un chemin différentiel donné.

Entropie relative. Au cours de la génération de chemins différentiels dans le sens direct, on ne connaît pas a priori le nombre de paires que l'on va devoir générer à chaque étape. En revanche, nous savons estimer le nombre N^{k+1} de paires disponibles à l'étape $k+1$ en fonction du nombre N^k de paires disponibles à l'étape k à l'aide de la formule

$$N^{k+1} = 2^{w(3-c^k)}N^k.$$

Cette grandeur est relative, puisqu'elle dépend de la valeur N^0 au départ du chemin. En multipliant N^0 par une constante, on multiplie toutes les valeurs de N^k par la même constante, à condition qu'elles restent supérieures ou égales à 1. En passant au logarithme en base 2^w, nous pouvons définir une version relative du concept d'entropie :

$$HR^k = log_{2w}(N^k)$$

Les différentes valeurs de l'entropie aux différentes étapes d'un chemin différentiel soit reliées par les relations suivantes :

$$HR^{k+1} = HR^k + 3 - c^k. \tag{7.5}$$

De même que le nombre de paires disponibles N^k n'est défini qu'à une constante multiplicative près, on peut ajouter ou soustraire une même constante à toutes les valeurs de HR^k le long d'un chemin, tant qu'elles restent positives ou nulles.

7.4.4 Algorithme de recherche de chemins différentiels

Comme mentionné plus haut, nous cherchons maintenant deux chemins partiels DP_f et DP_b dont la composition est un chemin DP permettant de générer des collisions internes. Notre algorithme utilise le concept d'entropie pour limiter la complexité des chemins différentiels générés, à la fois dans le sens direct et dans le sens inverse.

Pendant la génération de DP_b, nous savons précisément que l'objectif de l'attaque est de générer une collision. Nous pouvons donc fixer la valeur $H^n = 0$. Lors du parcours de l'arbre, la formule définissant l'entropie permet alors de calculer l'entropie d'un nœud à partir de celle de son père. Lors de cette phase de la recherche, nous nous limitons à des chemins pour lesquels l'entropie H_{bmax} n'excède pas 8. Pour cette phase nous utilisons une recherche en profondeur d'abord.

La première phase de la recherche de chemins consiste à trouver un chemin DP_f. Lors de la génération de ce chemin, nous ne connaissons pas précisément le nombre de messages qu'il sera nécessaire de générer. Nous utilisons par conséquent la version relative de l'entropie, en fixant $HR^0 = 8$. Ce choix de HR^0 permet de garantir que pour un chemin d'entropie maximale 8, l'entropie relative HR reste positive ou nulle sur tout le chemin DP_f. La stratégie de recherche de chemins différentiels est résumée sur la Figure 7.4.

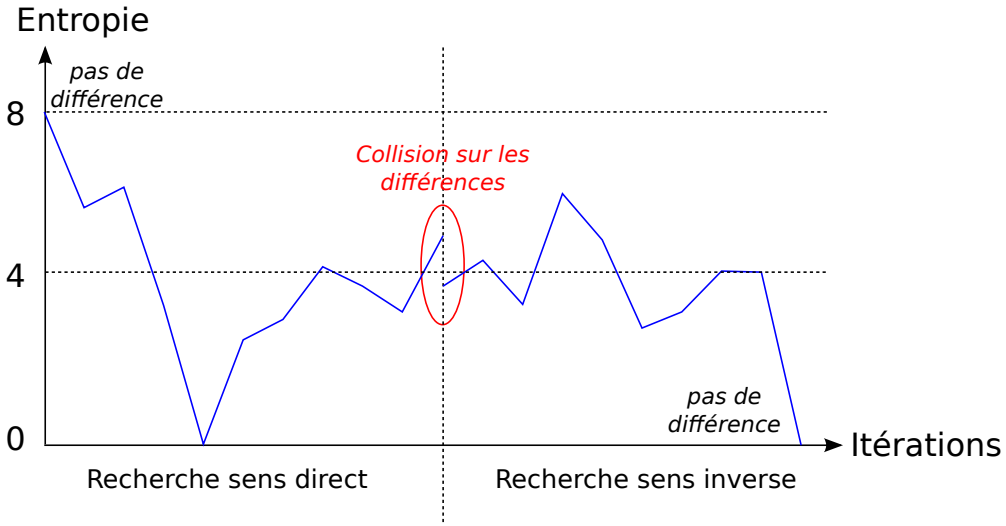


FIGURE 7.4 – Recherche d'un chemin différentiel pour RADIOGATÚN. La recherche se fait en deux phases, une recherche dans le sens direct pour un le début du chemin DP_f, partant d'une différence nulle, et une recherche dans le sens inverse pour la fin du chemin DP_b, qui doit permettre d'aboutir à une collision. On cherche une collision entre la différence à la fin de DP_f et le début de DP_b. Les restrictions sur les valeurs de l'entropie des chemins partiels permettent de garantir que l'entropie du chemin complet n'excède pas 8. L'entropie à la fin de DP_f doit être supérieure ou égale à l'entropie au début de DP_b.

Dans la phase de recherche de DP_f, nous nous limitons aux chemins pour lesquels l'entropie maximale HR_{fmax} excède d'au plus 8 l'entropie minimale HR_{fmin}. En effet, dans le cas contraire, si $HR^k - HR^{k+i} = n > 8$, le nombre estimé de paires à générer à la profondeur k pour en avoir au moins une à l'étape $k+i$ est $2^{nw} > 2^{8w}$. Pour appliquer la technique de rencontre au milieu, nous devons garder une partie de ces chemins en mémoire. Nous gardons uniquement les chemins pour lesquels l'entropie à la dernière étape est d'au moins $HR^{\ell_f} \geq HR_{fmax} - 4$. Dans le cas contraire, en utilisant l'interprétation heuristique de l'entropie, si on limite le nombre de paires de messages à générer jusqu'à la profondeur k à $2^{wHR_{fmax}}$, le nombre estimé de paires disponibles à la fin du chemin DP_f est inférieur à $2^{w(HR_{fmax}-4)}$. Nous imposons cette limite, afin d'éviter de considérer des chemins DP_f permettant d'aboutir à un nombre trop faible de paires de messages en entrée de DP_b.

Lors de cette phase nous utilisons une recherche en largeur d'abord.

La composition de DP_f et DP_b se fait alors de la manière suivante. DP_f est un chemin différentiel symétrique partiel, permettant d'aboutir à une différence δ sur l'état interne juste après une application de P. DP_b est également un chemin différentiel symétrique partiel qui permet, à partir de couples d'états avant application de P dont la différence est δ', d'aboutir à une collision interne.

S'il est possible d'introduire des blocs de message avec une différence δ_m permettant de transformer une différence δ en une différence δ', alors on peut chercher une paire de message m_f, m'_f suivant le chemin différentiel DP_f, puis définir des blocs m_ℓ et $m'_\ell = m_\ell \oplus \delta_m$. On obtient alors une différence δ' en entrée de P, et on cherche deux messages m_b, m'_b de manière à suivre le chemin différentiel DP_b.

Il y a 2^{58} valeurs possibles de δ et δ', et 2^3 valeurs possibles de δ_m. Etant donnés δ, δ', la probabilité qu'un tel δ_m existe est 2^{-55}. Le test de l'existence d'un tel δ_m est immédiat. Notre approche a alors consisté à générer 2^{27} chemins DP_f. Le nombre estimé de valeurs de DP_b à tester est 2^{28}, cependant, les différences sur les états internes en entrée de DP_b et en sortie de DP_f n'étant pas uniformément distribués, cette estimation reste approximative. Nous imposons de plus la condition

$$HR_{fmax} - HR_f + H_b \leq 8,$$

où H_b est l'entropie au départ du chemin DP_b, et HR_f est l'entropie relative à la fin du chemin DP_f.

Nous justifions maintenant les restrictions imposées sur les valeurs de l'entropie. Supposons que nous disposons de chemins DP_f et DP_b de longueurs respectives ℓ_f et ℓ_b itérations, et remplissant toutes ces conditions. Il est alors nécessaire de générer de l'ordre de 2^{wH_b} paires de messages vérifiant tout le chemin DP_f. A partir de la fin du chemin DP_f, nous calculons l'entropie à chaque itération k du chemin DP à l'aide de la formule 7.4. Deux cas de figure se présentent.

- Il existe i entre k et ℓ_f tel que $H^i = 0$. Dans ce cas, le mécanisme de génération du chemin DP_f garantit que $H^k - H^i \leq 8$, donc $H^k \leq 8$.
- Dans le cas contraire, pour i entre ℓ_f et k, l'entropie est toujours calculée par la formule $H^i = H^{i+1} + c^i - 3$, qui est la relation entre les valeurs de l'entropie calculées au cours de la génération de DP_f. On sait alors que $H^k = HR^k - HR_f + H_b$, et donc $H^k \leq HR_{fmax} - HR_f + H_b \leq 8$.

On montre ainsi que l'entropie maximale au cours de DP n'excède pas 8.

7.5 La recherche de collisions

Nous décrivons maintenant l'étape finale de l'attaque, qui consiste à exploiter le chemin différentiel pour trouver une collision. Cette étape repose sur des techniques classiques : une fois le chemin différentiel trouvé, nous utilisons les mots de contrôle de manière à améliorer la probabilité de succès de l'attaquant.

7.5.1 Description de l'algorithme

Pour y parvenir, nous utilisons un algorithme basé sur la technique de retour sur trace (le terme anglais *backtracking* est plus souvent rencontré). En d'autres termes, nous fixons bloc par bloc la valeur d'un message permettant de satisfaire toutes les conditions suffisantes imposées par le chemin différentiel. Dès lors qu'aucune valeur ne peut être choisie pour le bloc à l'itération k, nous modifions la valeur du bloc à l'itération $k-1$. Cette technique était évoquée par les concepteurs de la fonction [BDPA06], et une technique similaire a été employée par Peyrin dans l'attaque de `Grindahl` [Pey07].

L'attaquant cherche le message vérifiant le jeu de conditions suffisantes associé au chemin différentiel à l'aide d'un parcours sur un arbre. Les nœuds de cet arbre sont les messages respectant un préfixe du chemin différentiel. L'attaquant élimine alors les différents candidats aussitôt que

possible, c'est-à-dire dès qu'ils sortent du chemin différentiel. De manière équivalente, les nœuds de l'arbre sont étiquetés par des paires de messages ou par le premier message d'une paire, le second message étant dérivé du premier en utilisant le chemin différentiel. Les fils du nœud m sont alors les nœuds $m||b$, où b est une valeur de bloc de message, et toutes les conditions du chemin différentiel sur la valeur de l'état interne au cours du hachage de m^* qui peuvent être testées en connaissant seulement $m||b$ sont remplies (m^* étant un message quelconque admettant $m||b$ comme préfixe).

L'attaquant utilise une recherche en profondeur d'abord pour trouver un nœud à la profondeur n, n étant le nombre d'itérations de la fonction de compression impliquées dans le chemin différentiel. Afin de limiter la complexité de l'algorithme, l'attaquant teste les conditions suffisantes du chemin différentiel dès qu'il peut calculer les parties de l'état interne qu'elles impliquent. En particulier, certains registres du mill avant l'itération $k+2$ peuvent être calculées dès l'incorporation du bloc de message k. D'éventuelles conditions sur ces valeurs du mill à l'itération $k+2$ peuvent alors être vérifiées lors du parcours des nœuds de profondeur k.

D'après la Table 7.2, nous savons que le k-ème bloc incorporé affecte certains registres du mill avant l'itération k, certains avant l'itération $k+1$ et certains avant l'itération $k+2$. Réciproquement, certaines parties du mill avant l'itération k de P ne dépendent pas des k-ème et/ou $k-1$-ème blocs de messages, et peuvent donc être calculées dès l'incorporation du bloc $k-2$ ou du bloc $k-1$. Plus précisément, après l'incorporation du bloc m^k, on peut calculer $M_{16}^k, M_{17}^k, M_{18}^k$, mais également M_j^{k+1} pour $j \in \{1,2,4,5,7,8,9,12,13,15\}$ et M_j^{k+2} pour $j \in \{0,3,6,10,11,14\}$.

D'autres conditions imposent la valeur de la différence entre les valeurs de deux registres consécutifs du mill, $M_j^k \oplus M_{j+1}^k$. Nous pouvons exprimer ces valeurs en fonction des blocs de message incorporés avant les itérations k et $k-1$, et de l'état interne avant l'incorporation du message $k-1$. De ces expressions nous déduisons qu'après l'incorporation du k-ème bloc de message, il est possible de calculer $M_j^k \oplus M_{j+1}^k$, pour $j \in \{15,16,17,18\}$, $M_j^{k+2} \oplus M_{j+1}^{k+2}$ pour $j \in \{7,10\}$, et $M_j^{k+1} \oplus M_{j+1}^{k+1}$ pour toutes les autres valeurs de j. De ce fait, l'attaquant peut avoir à tester des conditions sur trois valeurs consécutives du mill après chaque incorporation de message.

On qualifie les conditions portant sur ces valeurs de registres du mill de *conditions vérifiables à l'étape k*.

L'idée la plus simple serait de choisir m^k aléatoirement et d'effectuer tous les calculs permettant de tester toutes les conditions possibles. Cependant les remarques suivantes permettent de faire décroître le nombre effectif de valeurs de m^k à tester.

- les conditions sur M_{16}^k, M_{17}^k et M_{18}^k ainsi que sur les valeurs de $M_{15}^k \oplus M_{16}^k$, $M_{16}^k \oplus M_{17}^k$, $M_{17}^k \oplus M_{18}^k$ et $M_{18}^k \oplus M_0^k$ à l'itération k peuvent être satisfaites en choisissant des valeurs de m^k adéquates ;
- toutes les opérations effectuées étant linéaires à l'exception de la première étape de la fonction *Mill*, l'attaquant peut réécrire les valeurs M_j^{k+1} comme une combinaison linéaire des registres $\hat{M}_j^k$ du mill après l'opération non linéaire, avec $j \in \{0,\ldots,18\}$. Les mots $\hat{M}_0^k$ à $\hat{M}_{13}^k$ ne dépendent pas du dernier bloc de message incorporé, et peuvent donc être calculés avant l'insertion du message ;
- un système d'équations dont les variables sont $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$ reste à résoudre. Ces équations proviennent de conditions avant l'itération $k+1$, en inversant la partie linéaire de la fonction *Mill*. Plus précisément, ces équations définissent les valeurs de ces variables, ou de la différence de deux d'entre elles, l'une d'elles ayant préalablement subi une rotation.

Notons I la fonction d'incorporation du bloc de message, et $\omega(m,(M,B),DP,k)$ l'évènement suivant : "étant donné un chemin différentiel symétrique DP de longueur $n \geq k$, $I(m,(M,B))$

remplit toutes les conditions vérifiables à l'étape k de DP".

L'algorithme 7.1 décrit alors la recherche de collisions, en notant I la fonction d'incorporation du bloc de message.

L'énumération des blocs de message vérifiant $\omega(m^i, (M^{i-1}, B^{i-1}), DP, i)$ est faite de la manière suivante.

1. L'attaquant vérifie la compatibilité des équations impliquant $\hat{M}^k_{14}, \ldots, \hat{M}^k_{18}$. Si elles ne sont pas compatibles, la recherche s'arrête. Cette probabilité dépend de la dimension du noyau du système et peut être calculée dès l'analyse du chemin différentiel.
2. L'attaquant énumère les valeurs jointes de $\hat{M}^k_{14}, \ldots, \hat{M}^k_{18}$, M^k_{16}, M^k_{17} et M^k_{18} satisfaisant les conditions portant sur ces valeurs. Ces valeurs peuvent être énumérées par groupes de bits aux même positions, qui sont indépendants les uns des autres. La complexité de cette phase est linéaire en w. Le nombre moyen de fils d'un nœud dépend du nombre de ces conditions.
3. Pour chaque message bloc restant, l'attaquant teste les valeurs des conditions restantes.

7.5.2 Calcul de la complexité

Nous expliquons maintenant comment calculer la complexité de l'algorithme de recherche de collisions. La partie la plus complexe de cet algorithme consiste à chercher les fils des nœuds de l'arbre. La complexité de l'algorithme est alors calculée pour tous les nœuds d'une profondeur donnée. La complexité à la profondeur k est le produit du nombre de nœuds de profondeur k qu'il est nécessaire de parcourir par la complexité du parcours d'un nœud à la profondeur k. Tous ces paramètres croissent exponentiellement avec w, pour des grandes valeurs de w, on peut donc approximer la complexité de la recherche de collisions par la complexité du parcours aux profondeurs les plus coûteuses.

Pour estimer la complexité de la recherche, nous faisons l'hypothèse que pour tous les mesages considérés, les valeurs des registres de l'état interne ne portant pas de conditions sont indépendantes et identiquement distribuées. Cette hypothèse est très souvent utilisée pour estimer la complexité de techniques de cryptanalyse. Elle peut être vérifiée aisément à l'étape 1, à condition que l'attaquant initialise l'état interne par l'incorporation d'une séquence suffisamment longue de blocs de message. Les distributions des valeurs des registres de l'état peuvent alors être vérifiées récursivement, mais leur indépendance relative reste à l'état d'hypothèse.

Soit A^k l'estimation du nombre de nœuds à parcourir à la profondeur k, et C^k la complexité moyenne de la recherche des fils d'un de ces noeuds. Notons également P^k la probabilité que le fils d'un nœud de profondeur k pris aléatoirement respecte le chemin différentiel, et Q^k la probabilité qu'un nœud donné à la profondeur k admette au moins un fils valide. On déduit de ces données le nombre moyen de nœuds à explorer à la profondeur k de la même variable prise à la valeur $k+1$. Notons que lorsque peu de nœuds sont nécessaires à l'étape $k+1$, l'étude du cas moyen ne suffit pas et il devient nécessaire d'évaluer la quantité de calculs nécessaire pour trouver au moins un nœud valide à l'étape $k+1$.

On obtient les relations suivantes, pour $k \in \{0, \ldots, n-1\}$:

$$\begin{cases} A^n = 1 \\ A^k = \max(\dfrac{A^{k+1}}{2^{3w} P^k}, \dfrac{1}{Q^k}) \end{cases}$$

Soit alors K^k la dimension du noyau du système d'équations linéaire à résoudre à la profondeur k, et $\hat{P}^k$ la probabilité que le système d'équations polynomiales sur les valeurs du mill avant et après la fonction non linéaire admette des solutions. $\hat{P}^k$ peut être calculé de manière exhaustive *a priori* pour chaque valeur de k. Un nœud à la profondeur k a au moins un fils valide si les deux conditions suivantes sont remplies :

– les conditions sur les valeurs des bits aux itérations k et $k+1$ peuvent être remplies ;
– les degrés de liberté restants permettent de remplir les conditions restantes.

Pour étudier la première condition il est nécessaire de tenir compte du fait que certaines parties du mill avant les itérations k et $k+1$ dépendent d'une partie seulement des mots de m^k. De ce fait, nous avons :

$$Q^k = \min(2^{-K^k}\hat{P}^k, 2^{3w-N^k_{COND}}),$$

où N^k_{COND} est le nombre total de conditions à remplir lors de la k-ème incorporation de message. Nous avons également $P^k = 2^{-N^k_{COND}}$, chaque condition étant supposée remplie avec probabilité $1/2$ dans le cas moyen, ce qui est vrai à condition que les mots du mill libres (c'est-à-dire dont la valeur n'est pas fixée par des conditions suffisantes et qui ne sont pas reliés à un autre mot du mill) soient indépendants et identiquement distribués.

Le parcours d'un nœud fonctionne alors de la manière suivante : l'attaquant résoud le système d'équations sur les valeurs de $M_{16}, M_{17}, M_{18}, \hat{M}_{14}, \ldots, \hat{M}_{18}$. L'ensemble des valeurs du blocs de message qui vérifient ce système de relations doit alors être parcouru de manière exhaustive pour remplir les conditions restantes, ce qui permet de générer les nœuds à la profondeur $k+1$. C^k représente la complexité de cette recherche exhaustive, et peut être estimé par le nombre moyen de blocs de messages vérifiant le système de relations. Nous avons donc $C^k = 2^{3w}\hat{P}^k$.

Pour chaque nœud de profondeur k, l'attaquant peut commencer par vérifier la compatibilité des conditions sur les registres du mill aux étapes k et $k+1$, ce qui lui permet de ne pas avoir à parcourir des nœuds inutilement. La complexité générique est alors donnée par

$$T = O(\max_k(\frac{C^k A^k}{2^{K^k}}))$$

Le meilleur chemin que nous avons identifié implique une complexité d'environ 4×2^{11w}, ce qui situe la complexité de notre attaque au-delà de la borne de sécurité estimée par les concepteurs de RadioGatún [BDPA06]. En effet, l'attaque en recherche de collisions qui en résulte est plus efficace que l'attaque générique fondée sur le paradoxe des anniversaires uniquement si la longueur du haché est d'au moins $22w+4$, alors que la longueur maximale préconisée par les concepteurs de la fonction est $19w$. Il est décrit en Annexe B. À titre de validation, nous donnons également une paire de messages construite à l'aide de ce chemin pour la version $w = 2$ de RadioGatún. Nous pouvons vérifier que les complexités observées approchent les estimations données.

7.5.3 Améliorations possibles

La recherche d'un chemin différentiel symétrique permettant une attaque sur RadioGatún avec une complexité de 2^{11w} a impliqué une phase de recherche automatisée nécessitant plusieurs heures de calcul sur une machine relativement puissante. La mémoire et le temps de calcul nécessaires sont fonction des paramètres concernant l'entropie imposés à l'algorithme. Les tests que nous avons menés tendent à montrer l'existence de chemins différentiels améliorant la complexité de la recherche

de collisions par rapport à l'attaque générique, pour des valeurs des paramètres considérées comme sûres par les concepteurs de la famille.

Une amélioration théorique possible de l'attaque consiste de plus à augmenter l'espace des différences symétriques considérées, par exemple en admettant les différences 0^w, 1^w, $01^{w/2}$ et $10^{w/2}$ qui constituent un espace stable par les opérations linéaires lorsque w est pair. Les besoins en mémoire et en temps de calcul de l'algorithme de recherche de chemins différentiels excèderaient alors de beaucoup les capacités des machines actuelles, tout en restant en-deçà de la complexité du paradoxe des anniversaires pour des tailles de haché suffisamment grandes. Dans ce cas, la résistance de RadioGatún à la recherche de collisions reposerait non pas sur l'absence de chemin différentiel exploitable, mais sur la difficulté d'en trouver avec les moyens de calcul actuellement disponibles.

Algorithme 7.1 Recherche de collisions pour RadioGatún

Entrées : Un chemin différentiel DP de longueur n blocs, le jeu de conditions sur les valeurs du mill associée

Sorties : $m^{-4}, \ldots, m^n$ dont le calcul de l'empreinte vérifie toutes les conditions suffisantes de DP

boucle
  {Boucle sur la valeur du préfixe}
  $m \leftarrow NULL$
  $(M^{-5}, B^{-5}) \leftarrow 0^{58w}$
  pour $i = -4 \ldots 0$ **faire**
    $m^i \leftarrow \{0,1\}^m$
    $m \leftarrow m||m^i$
    $(M^i, B^i) \leftarrow P(I(m^i, (M^{i-1}, B^{i-1})))$ {Initialisation aléatoire de l'état interne}
  fin pour
  $i \leftarrow 1$
  $j_i \leftarrow 1$
  $Avance \leftarrow VRAI$
  tant que $i > 0$ **faire**
    si $i = n + 1$ **alors**
      renvoyer $(m^{-4}, \ldots, m^n)$ {Sortie du premier message}
    fin si
    si $Avance$ **alors**
      $n_i \leftarrow |\{m^i | w(m^i, (M^{i-1}, B^{i-1}), DP, i)\}|$ {Calcul des fils du nœud courant}
    fin si
    si $j_i > n_i$ **alors**
      $i \leftarrow i - 1$
      $j_i \leftarrow j_i + 1$
      $Avance \leftarrow FAUX$ {Retour sur trace}
    sinon
      $\{m^i_j\}_{1 \leq j \leq n_i} \leftarrow \{m^i | \omega(m^i, (M^{i-1}, B^{i-1}), DP, i)\}$
      $j_i \leftarrow j_i + 1$
      $m^i \leftarrow m^i_{j_i}$
      $(M^i, B^i) \leftarrow P(I(m^i, (M^{i-1}, B^{i-1})))$ {Avance dans le chemin différentiel}
      $i \leftarrow i + 1$
      $j_i \leftarrow 1$ {Initialisation de l'indice du bloc de message à la profondeur suivante}
      $Avance \leftarrow VRAI$
    fin si
  fin tant que
fin boucle

Chapitre 8

Cryptanalyse conditionnelle algébrique de Hamsi-256

Sommaire

Hamsi est une famille de fonctions de hachage conçue par Küçük [Küç09], qui l'a soumise à la compétition SHA-3. Quatre fonctions de cette famille ont été définies, différenciées par la taille des empreintes qu'elles produisent : 224, 256, 384 et 512 bits. Dans ce chapitre nous nous intéressons à la version 256 bits, Hamsi-256. Le résultat principal que nous exposons est une attaque en recherche de secondes préimages. Les propriétés que nous décrivons s'appliquent également à Hamsi-224, toutefois elles ne conduisent pas à des attaques plus rapides que les attaques génériques contre cette fonction.

L'attaque exposée dans ce chapitre a été publiée lors de la conférence Asiacrypt 2010 [Fuh10]. Elle repose sur une étude de certaines propriétés algébriques de la fonction de compression d'Hamsi-256. Nous montrons que si l'on restreint la fonction de compression à un sous-ensemble d'entrées bien choisi, il est possible de trouver des relations affines (dans $\mathbb{F}_2$) entre certains de ses bits de sortie et certains de ses bits d'entrée. Cette propriété peut ensuite être exploitée pour trouver

des secondes préimages pour la fonction de hachage complète avec une complexité équivalente à $2^{251,16}$ évaluations de la fonction de compression. Ce résultat, a constitué la première attaque contre une des notions de sécurité traditionnelles d'un des algorithmes retenus pour le second tour de la compétition SHA-3.

Une autre conséquence est la possibilité de générer des collisions sur la fonction de compression $\mathcal{F}$ de Hamsi-256, c'est-à-dire des couples variable de chaînage - bloc de message (C, M) et (C', M') distincts tels que $\mathcal{F}(C, M) = \mathcal{F}(C', M')$ avec une complexité équivalente à $2^{123,2}$ évaluations de la fonction de compression.

Résultats présentés. Les résultats présentés dans ce chapitre s'articulent de la façon suivante. Dans la section 8.1, nous introduisons le formalisme de la cryptanalyse conditionnelle algébrique. Informellement, cette technique permet de trouver des préimages d'une fonction en exprimant un certain nombre de ses bits de sortie comme une fonction polynômiale d'un petit nombre de variables d'entrée, le reste des entrées étant fixé. Dans la suite du chapitre, nous appliquons cette technique à Hamsi-256. Après avoir décrit la fonction Hamsi-256 dans la section 8.2, nous montrons dans la section 8.3 comment exprimer deux tours de la fonction de compression comme une fonction linéaire de 16 variables. Cette propriété permet d'attaquer une version réduite à deux tours de Hamsi-256, elle sera également utilisée dans la suite pour optimiser l'attaque sur la version complète. Dans la section 8.4, nous exposons une manière d'étendre la propriété de la section 8.3 aux trois tours de la fonction de compression complète. Certains bits de sortie de la fonction de compression complète sont des fonctions affines de variables d'entrée bien choisies. Dans la section 8.5, nous expliquons comment utiliser ces relations affines pour mener une attaque en recherche de préimages sur la fonction de compression, en minimisant le nombre de calculs nécessaires. Enfin, dans la section 8.6, nous transformons l'attaque sur la fonction de compression en une attaque en recherche de secondes préimages sur la fonction de hachage complète, en utilisant des techniques de rencontre au milieu modifiées.

Notations. Dans ce chapitre nous utilisons les notations suivantes :
- X_i représente le $i^{ème}$ mot de 32 bits de la variable X,
- $Y^{(j)}$ représente le $j^{ème}$ bit de la variable Y.

8.1 La cryptanalyse conditionnelle algébrique

Notre attaque repose sur une technique nouvelle que nous avons développée pour exploiter certaines propriétés spécifiques de Hamsi : la cryptanalyse conditionnelle algébrique. Cette technique pourrait s'appliquer à d'autres fonctions de compression de bas degré. En ce sens elle se rapproche des méthodes décrites dans [DS09, Vie07], tout en permettant de prendre en compte des propriétés spécifiques de l'algorithme analysé, non exploitées par ces dernières..

Considérons une fonction de compression

$$\begin{array}{rcl} \mathcal{F} : \{0,1\}^n \times \{0,1\}^m & \longrightarrow & \{0,1\}^n \\ C, M & \mapsto & \mathcal{F}(C, M) = H. \end{array}$$

Chacun des bits de sortie $H^{(i)}$ peut alors être exprimé comme une fonction polynômiale P_i des bits de C et de M, en se plaçant dans l'anneau de polynômes $GF(2)[X]$. La cryptanalyse conditionnelle algébrique repose principalement sur deux idées :

- L'étude de certaines fonctions partielles de $\mathcal{F}$, en réduisant son domaine de définition à certains de ses sous-espaces affines de petite dimension. De cette manière, nous réduisons le degré algébrique des fonctions P_i.
- Un choix judicieux des sous-espaces affines considérés. L'étude des composants de base de $\mathcal{F}$ permet de déterminer des conditions permettant de limiter le degré algébrique des polynômes considérés.

Nous nous restreignons à l'étude de $\mathcal{F}$ sur un ensemble de la forme $\{y \in \{0,1\}^{n+m} | y = L(x)+A, x \in \{0,1\}^r\}$, L étant une application linéaire donnée et A un point constant. Le choix de L permet de sélectionner les parties variables des entrées de $\mathcal{F}$. Le choix de A revient à imposer la partie fixe des entrées de F. L'attaque publiée à Asiacrypt 2010 [Fuh10], tout comme l'attaque de Shamir et Dinur [SD11], sont basées sur ce principe. Nous introduisons maintenant un formalisme permettant de représenter ces deux attaques. Dans [KMNP10], Knellwolf *et al.* utilisent aussi un choix judicieux de certaines parties des entrées d'une fonction, mais pour étudier les propriétés différentielles d'une fonction de compression ou de chiffrement.

Formellement, nous utiliserons donc la représentation suivante. Considérons une application affine $\mathcal{V} : \{0,1\}^r \to \{0,1\}^n \times \{0,1\}^m$, de rang r. Au lieu d'étudier directement les fonctions P_i, nous étudierons les propriétés algébriques de $Q_{i,\mathcal{V}} = P_i \circ \mathcal{V}$, afin d'en déduire certaines propriétés de P_i. Cela revient à appliquer P_i uniquement sur un sous-espace affine de son domaine de définition. Pour certaines fonctions de compression, certains choix judicieux de $\mathcal{V}$ permettent d'identifier des propriétés algébriques particulières. En particulier, dans le cas de `Hamsi`-256, il est possible d'inverser la fonction de compression plus rapidement qu'avec les attaques génériques.

Dans la suite du chapitre, de telles applications affines sont appelées *restrictions de domaine*. En effet, elles correspondent à des restrictions judicieuses du domaine de définition de la fonction de compression, une partie de ses entrées étant fixée à une valeur constante.

8.1.1 Préimages pour la fonction de compression

Dans la suite, nous désignons par J une classe d'applications affines $\mathcal{V}$ et par I l'ensemble des bits de sortie de $\mathcal{F}$ pour lesquels nous calculons $Q_{i,\mathcal{V}}$

Il est alors possible de générer des préimages pour $\mathcal{F}$ de la manière suivante. Soit $H = \{H^{(i)}\}_{i \in \{0,1\}^n}$ la valeur de sortie que l'attaquant cherche à inverser. Il peut choisir des valeurs $\mathcal{V}$ et calculer les polynômes $Q_{i,\mathcal{V}}$ correspondants, puis résoudre le système d'équations $\{Q_{i,\mathcal{V}}(X) = H^{(i)}\}_{i \in I}$. Dans le cas général, nous n'imposons pas ce système soit linéaire, cependant il l'est dans le cas de notre attaque sur `Hamsi`-256. Si $|I| = N_{eq}$, les solutions de ce système vérifient alors $\mathcal{F} \circ \mathcal{V}(X) = H$ avec probabilité $2^{N_{eq}-n}$. $\mathcal{V}(X)$ est alors une préimage de H par $\mathcal{F}$ avec cette même probabilité.

Supposons les éléments $\mathcal{V}_j$ de J indexés par des entiers. On peut alors retrouver une préimage de H en utilisant l'algorithme suivant :

Étude de la complexité. Supposons alors qu'il est possible pour tout $\mathcal{V} \in J$ de retrouver les coefficients de tous les polynômes $\{Q_{i,\mathcal{V}}\}_{i \in I}$ avec une complexité de T_{build} opérations et que la résolution du système d'équations induit ait une complexité T_{solve}. Le nombre moyen de solutions par système est alors de $2^{r-N_{eq}}$, et chacune d'entre elles permet de trouver une préimage de H avec probabilité $2^{N_{eq}-n}$. Il est donc nécessaire de générer en moyenne 2^{n-r} systèmes avant de trouver une solution. En notant $T_{\mathcal{F}}$ la complexité d'une évaluation de la fonction de compression, la complexité de cet algorithme est donc la suivante :

Algorithme 8.1 Calcul de préimages pour la fonction de compression $\mathcal{F}$

Entrées : $H \in \{0,1\}^n$
Sorties : $(C, M) \in \{0,1\}^{n+m}$ tel que $\mathcal{F}(C, M) = H$
  $(C, M) \leftarrow 0^{n+m}$
  $F \leftarrow \mathcal{F}(C, M)$
  $j \leftarrow 0$
  boucle
    $\mathcal{V} \leftarrow \mathcal{V}_j$
    calculer tous les $Q_{i,\mathcal{V}}, i \in I$ {complexité T_{build}}
    $\Sigma \leftarrow \{X \in \{0,1\}^r | \forall i \in I, Q_{i,\mathcal{V}}(X) = H^{(i)}\}$ {complexité T_{solve}}
    pour $X \in \Sigma$ **faire**
      $(C, M) \leftarrow \mathcal{V}(X)$
      $F = \mathcal{F}(C, M)$ {complexité $T_{\mathcal{F}}$}
      si $F = H$ **alors**
        renvoyer (C, M)
      fin si
    fin pour
    $j \leftarrow j + 1$
  fin boucle

$$T_{preimage} = 2^{n-r}(T_{build} + T_{solve}) + 2^{n-N_{eq}} T_{\mathcal{F}}. \tag{8.1}$$

8.1.2 Collisions pour la fonction de compression

L'algorithme précédent requiert le calcul de $\mathcal{F}$ sur des entrées telles que N_{eq} bits de sortie aient une valeur fixée. Par le paradoxe des anniversaires, on obtient une collision sur $\mathcal{F}$ après le calcul de $2^{(n-N_{eq})/2}$ telles sorties. L'algorithme précédent permet donc, en modifiant sa condition d'arrêt et en stockant les résultats obtenus, de calculer des collisions sur la fonction de compression.

Étude de la complexité. Dans ce cas, il est nécessaire de générer $2^{N_{eq}-r+(n-N_{eq})/2} = 2^{(n+N_{eq})/2-r}$ systèmes, d'où la complexité suivante :

$$T_{collision} = 2^{(n+N_{eq})/2-r}(T_{build} + T_{solve}) + 2^{(n-N_{eq})/2} T_{\mathcal{F}}. \tag{8.2}$$

Pour cette attaque on peut remarquer que le nombre de systèmes à générer croît exponentiellement avec le nombre d'équations. Pour un nombre de variables données, il est donc probable que la complexité soit minimale si on ne considère pas toutes les équations dont on dispose.

Remarque. L'algorithme 8.2 peut nécessiter l'utilisation d'une grande mémoire. Cependant, cet algorithme repose sur le paradoxe des anniversaires, et on peut par conséquent lui appliquer les modifications classiques permettant de s'affranchir des besoins en mémoire.

Algorithme 8.2 Calcul de collisions pour la fonction de compression $\mathcal{F}$

Sorties : $(C, M), (C', M') \in (\{0,1\}^{n+m})^2$ tel que $\mathcal{F}(C, M) = \mathcal{F}(C', M')$
  $L \leftarrow \emptyset$
  $j \leftarrow 0$
  boucle
    $\mathcal{V} \leftarrow \mathcal{V}_j$
    pour $\{h^{(i)}\}_{i \in I} \in \{0,1\}^{|I|}$ **faire**
      calculer tous les $Q_{i,\mathcal{V}}, i \in I$ {complexité T_{build}}
      $\Sigma \leftarrow \{X \in \{0,1\}^r | \forall i \in I, Q_{i,\mathcal{V}}(X) = H^{(i)}\}$ {complexité T_{solve}}
      pour $X \in \Sigma$ **faire**
        $(C, M) \leftarrow \mathcal{V}(X)$
        $F = \mathcal{F}(C, M)$ {complexité $T_{\mathcal{F}}$}
        si $\exists (C', M', F') \in L | F = F'$ **alors**
          renvoyer $(C, M), (C', M')$
        sinon
          $L \leftarrow L \cup (C, M, F)$
        fin si
      fin pour
    fin pour
    $j \leftarrow j + 1$
  fin boucle

8.2 Description de la fonction Hamsi-256

Les fonctions de la famille Hamsi reposent sur un choix traditionnel d'algorithme d'extension de domaine : celui de Merkle-Damgård. Leurs fonctions de compression découlent en revanche de principes de conception novateurs. Elles reposent sur l'utilisation d'une permutation fixe et sur une expansion de message linéaire assurant une bonne protection contre la cryptanalyse différentielle. Nous nous intéressons ici uniquement à la version 256 bits de cette famille.

8.2.1 Schéma général de la fonction

Extension de domaine. Les fonctions de compression de la famille Hamsi permettent de traiter des blocs de message courts (32 bits dans le cas de Hamsi-256). La fonction de compression $\mathcal{F}$ de Hamsi-256 permet donc de calculer une valeur de chaînage de 256 bits à partir de la valeur de chaînage précédente de 256 bits et d'un bloc de message de 32 bits. L'algorithme d'extension de domaine utilisé est celui de Merkle-Damgård, avec un padding similaire à celui des standards de la famille MD-SHA. On concatène au message un bit valant "1", puis des bits "0" jusqu'à obtenir une longueur multiple de 32 bits. On concatène ensuite la longueur du message avant padding, codée sur 64 bits. La donnée ainsi obtenue est divisée en blocs de 32 bits. Les deux derniers blocs traités par la fonction de compression contiennent donc la longueur du message. Pour le traitement du dernier bloc, une version renforcée de la fonction de compression est appliquée.

Structures des données considérées. La fonction de compression de Hamsi-256 repose sur l'utilisation d'opérations élémentaires dans $GF(2)$. Dans ce chapitre, les notions d'addition, de fonction linéaire ou affine, ou encore de degré algébrique sont donc induites par les opérations dans

ce corps, ou dans des espaces vectoriels construits sur ce corps. En particulier, l'addition de deux variables de v bits correspond à leur OU EXCLUSIF.

Fonction de compression. La fonction de compression de Hamsi-256 repose sur l'utilisation d'une permutation fixe sur des variables de 512 bits. Pour obtenir un état interne S de 512 bits, on applique une expansion linéaire de 32 bits vers 256 bits au bloc de message M, qui permet de calculer $\mathcal{E}(m) = (m_0, ..., m_7)$.

On concatène ensuite la variable de chaînage d'entrée $C = (c_0, \ldots, c_7)$ au bloc de message après expansion, en utilisant l'opération suivante :

$$\mathcal{C} : (\mathcal{E}(M), C) \mapsto \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} = \begin{pmatrix} m_0 & m_1 & c_0 & c_1 \\ c_2 & c_3 & m_2 & m_3 \\ m_4 & m_5 & c_4 & c_5 \\ c_6 & c_7 & m_6 & m_7 \end{pmatrix}$$

L'état interne de 512 bits est représenté comme un tableau de 4×4 mots de 32 bits.

On applique ensuite à l'état interne une permutation $\mathcal{P}$, décrite plus bas. Pour garantir la non inversibilité de la fonction de compression, la sortie de la permutation $\mathcal{P}$ est ensuite tronquée en ne conservant que les première et troisième lignes de l'état. La sortie de la fonction de compression est alors obtenue en appliquant une addition du résultat à la variable de chaînage d'entrée. On compose donc $\mathcal{P}$ avec les transformations suivantes :

$$\begin{aligned} \mathcal{T} : S &\mapsto \Sigma = (s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}) \\ \mathcal{X} : \Sigma &\mapsto C^* = C \oplus \Sigma. \end{aligned}$$

8.2.2 La permutation $\mathcal{P}$.

La permutation de Hamsi-256 est obtenue en itérant 3 fois une fonction de tour $\mathcal{R}$. Pour le dernier bloc de message, la fonction de compression est renforcée : la fonction $\mathcal{R}$ est appliquée 8 fois. $\mathcal{R}$ est composée des opérations suivantes :

$$\mathcal{R} : S \mapsto \mathcal{L}(\mathcal{S}(\mathcal{A}(S))),$$

où $\mathcal{A}$ correspond à l'addition d'une valeur constante et d'un compteur du nombre de tours à l'état interne, $\mathcal{S}$ est une couche de substitution construite sur l'utilisation de la deuxième boîte S de 4 bits vers 4 bits de Serpent, et $\mathcal{L}$ est une couche de diffusion linéaire qui opère en parallèle sur quatre sous-espaces de l'état de 4×32 bits.

$\mathcal{S}$ consiste à appliquer 128 fois la même boîte S en parallèle sur les colonnes de l'état interne. De manière plus formelle, pour $i \in \{0 \ldots 3\}$ and $j \in \{0 \ldots 31\}$, on applique la boîte S de la Table 8.1 aux bits $s_i^{(j)}, s_{i+4}^{(j)}, s_{i+8}^{(j)}, s_{i+12}^{(j)}$.

Enfin, la couche de diffusion linéaire $\mathcal{L}$ agit sur les diagonales de l'état interne, et fonctionne de la manière suivante. Elle prend en entrée quatre variables de 32 bits $(a, b, c, d) = (s_0, s_5, s_{10}, s_{15})$ (resp. $(s_1, s_6, s_{11}, s_{12})$, (s_2, s_7, s_8, s_{13}), (s_3, s_4, s_9, s_{14})) et consiste en la séquence d'opérations suivante :

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	8	6	7	9	3	C	A	F	D	1	E	4	0	B	5	2

TABLE 8.1 – Boîte S utilisée par `Hamsi`. Entrées et sorties en hexadécimal, les bits de poids faibles de x sont pris dans les variables $s_0, \ldots, s_3$

$$
\begin{aligned}
a &:= a \lll 13 \\
c &:= c \lll 3 \\
b &:= (b \oplus a \oplus c) \lll 1 \\
d &:= (d \oplus c \oplus (a \ll 3)) \lll 7 \\
a &:= (a \oplus b \oplus d) \lll 5 \\
c &:= (c \oplus d \oplus (b \ll 7) \lll 22
\end{aligned}
$$

8.2.3 Cryptanalyse des fonctions de la famille Hamsi : état de l'art

D'autres analyses de sécurité des fonctions de la famille `Hamsi` ont été publiées à ce jour. Ces analyses s'articulent principalement autour de deux axes : l'application de techniques de cryptanalyse différentielle à la fonction de compression de `Hamsi` et l'étude des propriétés algébriques de cette famille de fonctions de hachage. Notre attaque de `Hamsi`-256 présentée dans la suite de ce chapitre relève de ce second axe.

Cryptanalyse différentielle. Les fonctions de la famille `Hamsi` ont la particularité de traiter des blocs de message courts. Leur fonction de compression est donc rapide. Leur résistance à la cryptanalyse différentielle repose sur la distance minimale de l'expansion de message, qui garantit que pour deux blocs de message différents, au moins 70 boîtes S seront actives au premier tour.

Les études différentielles réalisées sur `Hamsi` consistent donc à ne pas introduire de différences sur le bloc de message, mais uniquement sur la variable de chaînage. Elles permettent donc d'obtenir des presque-collisions sur la fonction de compression. Nikolić a montré comment obtenir des presque-collisions sur 223 bits avec une complexité de l'ordre de 2^{21} évaluations de la fonction de compression [Nik09], et Wang *et al.* sont parvenus à obtenir des pseudo-collisions jusqu'à 5 tours de la fonction de la fonction de compression, avec une complexité équivalente à 2^{125} calculs de la fonction de compression. Lors de la conférence ACISP 2010, Aumasson *et al.* ont montré à l'aide de techniques différentielles que 6 tours de la fonction de compression n'ont pas un comportement idéal [AKK+10].

Cryptanalyse algébrique. Du fait des types d'opérations utilisées, le degré algébrique de la permutation de `Hamsi` reste bas, ce qui permet de la distinguer d'une permutation idéale. Une première remarque d'Aumasson [Aum09] montre que 5 tours de la fonction de compression ont un degré maximal de 243. Une exploitation de cette propriété a été la découverte de distingueurs à somme nulle [AM09, BC10].

Par la suite, Calik et Turan ont montré que certains bits de sortie de la fonction de compression ne dépendent pas de certains bits d'entrée, ce qui permet de trouver des secondes préimages sur la fonction de compression avec une complexité d'environ $2^{254.25}$ [CT10].

Enfin, dans un travail qui réutilise certains résultats de notre attaque, Shamir et Dinur [SD11] parviennent à générer des secondes préimages pour la fonction de hachage complète avec une complexité réduite d'un facteur jusqu'à 512 par rapport aux meilleures attaques génériques.

8.3 Une attaque sur deux tours de la fonction de compression

Dans cette section nous étudions une version simplifiée de la fonction de compression de Hamsi-256, pour laquelle la permutation interne est réduite à deux tours. Les résultats que nous décrivons seront utilisés par la suite dans l'attaque de la version complète de Hamsi-256. Nous décrivons ici comment trouver des préimages pour la version réduite de la fonction de compression.

8.3.1 Étude de la boîte S de Hamsi

Notre attaque repose sur certaines propriétés spécifiques de la boîte S de Hamsi. En considérant des couples d'entrée-sortie particulières, nous pouvons mettre en évidence des propriétés algébriques. De $S[9] = 1$, $S[C] = 0$, $S[B] = 4$, et $S[E] = 5$, nous déduisons que

$$\forall (x, b) \in \{0,1\}^2, S[(x, b, \bar{x}, 1)] = (x + b, 0, b, 0). \tag{8.3}$$

De même nous avons $S[3] = 9$ et $S[9] = 1$, ce qui conduit à

$$\forall x \in \{0,1\}, S[(1, x, 0, \bar{x})] = (1, 0, 0, x). \tag{8.4}$$

Ces propriétés seront utilisées lors du premier tour de $\mathcal{P}$. Elles répondent aux critères suivants. Seuls les bits 0 et 2 (resp. 1 et 3) de l'entrée dépendent de la variable x. De ce fait, pour les boîtes S du premier tour appliquées aux colonnes 2 et 3 (resp. 0 et 1), seuls les bits provenant de la variable de chaînage dépendent de la variable x, les bits provenant du message sont constants. De plus, dans les deux cas, seul un bit de la sortie dépend de la variable d'entrée.

8.3.2 Une classe de restrictions de domaine

Considérons maintenant une valeur quelconque M du bloc de message. Sans connaître la variable de chaînage d'entrée, nous pouvons calculer $s_0, s_1, s_6, s_7, s_8, s_9, s_{14}, s_{15}$ après l'addition de la constante du premier tour. Supposons alors que le bit j de s_{14} soit $s_{14}^{(j)} = 1$. Indépendamment de la valeur de $s_6^{(j)}$, si $s_2^{(j)} = x^{(i)}$ et $s_{10}^{(j)} = \overline{x^{(i)}}$, seul le premier bit de la sortie de la j-ième boîte S de la troisième colonne dépend de $x^{(i)}$, d'après la propriété (8.3).

Nous définissons maitenant un ensemble des restrictions de domaine J permettant d'exploiter ces propriétés. Notons A la constante ajoutée au premier tour de $\mathcal{P}$. Pour 16 positions distinctes $\{t_i\}_{i=0\ldots15}$ dans la troisième colonne $(s_2, s_6, s_{10}, s_{14})^t$ de l'état interne, nous identifions 16 variables $x^{(i)}$ aux bits t_i de s_2 .Nous imposons également les conditions $s_{14}^{(t_i)} = 1$ et $s_{10}^{t_i} = \overline{x^{(i)}}$. Les autres bits de l'état interne sont fixés à des valeurs constantes quelconques. Cela se traduit de la manière suivante sur les entrées de la fonction de compression. J est l'ensemble des applications affines $\mathcal{V} : X = (x^{(0)}, \ldots, x^{(15)}) \longrightarrow (C, M)$ vérifiant les propriétés suivantes.

- M est constant
- Il existe 16 valeurs distinctes $t_0, \ldots, t_{15}$ entre 0 et 31 telles que :

$$\begin{aligned} \forall i \in \{0, \ldots, 15\}, \mathcal{E}(M)^{(192+t_i)} &= 1 \oplus A^{(448+t_i)} \\ C^{(t_i)} &= x^{(i)} \oplus A^{(64+t_i)} \\ C^{(128+t_i)} &= \overline{x^{(i)}} \oplus A^{(320+t_i)}. \end{aligned} \tag{8.5}$$

– Les autres bits de C sont constants.

Considérons maintenant une valeur quelconque M du bloc de message. Sans connaître la variable de chaînage d'entrée, nous pouvons calculer $s_0, s_1, s_6, s_7, s_8, s_9, s_{14}, s_{15}$ après l'addition de la constante du premier tour. Supposons alors que le bit j de s_{14} soit $s_{14}^{(j)} = 1$. Indépendamment de la valeur de $s_6^{(j)}$, si $s_2^{(j)} = x^{(i)}$ et $s_{10}^{(j)} = \overline{x^{(i)}}$, seul le premier bit de la sortie de la j-ième boîte S de la troisième colonne dépend de $x^{(i)}$, d'après la propriété (8.3).

Les équations (8.5) se traduisent par $s_{14}^{(t_i)} = 1$, $s_2^{(t_i)} = x^{(i)}$ et $s_{10}^{(t_i)} = \overline{x^{(i)}}$. De ce fait, après la première couche de boîtes S, seul $s_2^{(t_i)}$ dépend de la variable $x^{(i)}$. Après la première couche de diffusion linéaire $\mathcal{L}$ et la deuxième addition de constante, s_2, s_7, s_8, s_{13} dépendent des variables X de manière affine, le reste de l'état interne étant constant. En entrée de la deuxième couche de boîtes S, seul 1 bit d'entrée de chaque boîte S dépend effectivement des variables, et cette dépendance est de degré au plus 1. De ce fait, les sorties de cette couche de boîtes S sont également des fonctions affines des variables. La deuxième couche de diffusion linéaire, la troncation et le rebouclage sont des applications linéaires. On en déduit que $\mathcal{F} \circ \mathcal{V}$ est une application affine.

Nous avons conduit ce raisonnement en appliquant la propriété (8.3) aux boîtes S de la colonne 2 lors du premier tour. Le même raisonnement s'applique à partir de la colonne 3. Un raisonnement similaire s'applique également en partant des colonnes 0 ou 1 en utilisant la propriété (8.4), mais le nombre attendu de variables utilisables est plus petit du fait des deux conditions portant sur les entrées du message pour chaque boîte S.

8.3.3 Inversion de la fonction de compression

Nous pouvons maintenant chercher à inverser la fonction de compression réduite, c'est-à-dire chercher (C, M) tels que $\mathcal{F}(C, M) = H$ pour une valeur H donnée. Le principe de l'attaque consiste à exprimer H comme une fonction affine de 16 variables, en utilisant le raisonnement précédent. On utilise alors l'algorithme 8.1. Les coefficients de cette fonction peuvent être retrouvés par interpolation, en appliquant $\mathcal{F} \circ \mathcal{V}$ à tous les vecteurs de poids au plus 1. La complexité de $\mathcal{V}$ est négligeable, on a donc $T_{build} = 17T_{\mathcal{F}}$. Connaissant H, inverser $\mathcal{F} \circ \mathcal{V}$ revient à résoudre un système de $N_{eq} = 256$ équations linéaires en 16 variables. On a $T_{solve} \leq T_{\mathcal{F}}$. Comme tout l'état est linéaire, la phase de test des solutions devient inutile. D'après l'équation (8.1), on a donc :

$$T_{preimage-2tours} \leq 2^{256-16}(17T_{\mathcal{F}} + T_{\mathcal{F}}) \approx 2^{244,2}T_{\mathcal{F}}$$

Modifications et améliorations possibles. Nous verrons dans le cas de la fonction de compression complète des optimisations possibles de l'algorithme de recherche de préimages. Ces améliorations peuvent également être transposées au cas de la fonction réduite ; cependant, nous ne les décrivons pas ici pour ne pas alourdir la description de cette première attaque.

8.4 Relations affines sur trois tours de la fonction de compression

Dans cette section, nous montrons comment modifier cette technique pour trouver des équations affines pour la fonction de compression complète de `Hamsi-256`. Si on choisit les mêmes classes de restrictions de domaine que dans la section précédente, l'état interne de la permutation avant la troisième couche de boîtes S est une fonction affine des variables, et l'application de $\mathcal{S}$ introduit alors de la non-linéarité.

Cependant, la couche de diffusion est basée sur un schéma de Feistel très simple, et n'assure donc pas une diffusion parfaite. De ce fait, il est possible de trouver des classes plus petites de restrictions de domaine avec moins de variables, de manière à ce que certains bits de sortie soient des fonctions linéaires des variables.

Il n'est pas possible d'obtenir une telle propriété en contraignant les variables sur une seule colonne lors de la première couche $\mathcal{S}$. On y parvient cependant en choisissant les variables dans les colonnes 0 et 1, et en étudiant les propriétés algébriques de la boîte S de Hamsi de manière plus détaillée.

On utilise alors la propriété (8.4), de la manière suivante. Si le bloc de message M est tel qu'avant la première couche de substitution, $s_0^{(j)} = 1$ et $s_8^{(j)} = 0$, et $\mathcal{V}$ est telle que $s_4^{(t_i)} = x^{(i)}$ et $s_{12}^{(t_i)} = \overline{x^{(i)}}$, seul le bit $s_{12}^{(t_i)}$ dépend de $x^{(i)}$ après la première couche de substitution. La même remarque s'applique à s_1, s_5, s_9, s_{13}. Par rapport à la propriété (8.3), on utilise plus de degrés de liberté : pour chaque équation, on en utilise 2 pour le bloc de message et 1 pour la variable de chaînage, au lieu de 2 en tout. Cependant, s_{12} et s_{13} correspondent à l'entrée d de la couche $\mathcal{L}$, qui se propage moins bien que l'entrée a.

8.4.1 Propriétés algébriques de la boîte S de Hamsi

Les couches de boîtes S sont les seuls éléments de la fonction de compression de Hamsi qui introduisent de la non-linéarité. Comme nous cherchons des équations affines, nous commençons par étudier les propriétés algébriques de la boîte S. Notons $L : \{0,1\}^r \to \{0,1\}^4$ une application quelconque. Nous cherchons à établir à quelles conditions $S_i \circ L$ est une application affine, où S_i est la restriction de la boîte S au i-ième bit de sortie.

Tout d'abord, tous les bits de sortie de S dépendent de tous ses bits d'entrée. Si L n'est pas affine, $S_i \circ L$ peut alors être affine uniquement si on impose les valeurs de certains bits d'entrée, ce qu'on ne sait pas faire facilement en entrée des couches de boîtes S 2 et 3 pour Hamsi-256. On se restreint donc au cas où L est une application affine.

Notons alors $L(X) = (L_0(X), L_1(X), L_2(X), L_3(X))$, et $L_i(X) = \bigoplus_{j=1}^{r} \ell_{i,j} x_j \oplus \ell_i$. On définit une relation $R(i,j,k)$ par la Table 8.2. Cette relation est définie de la manière suivante : $R(i,j,k) = 1$ si et seulement si l'expression du bit k de la sortie de la boîte S de Hamsi contient un monôme de la forme $x_i x_j m$ (où m est un monôme quelconque de degré 0 ou 1).

$\{i,j\}$	$\{0,1\}$	$\{0,2\}$	$\{0,3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$
$R(i,j,0)$	0	1	0	0	0	0
$R(i,j,1)$	1	1	1	1	1	1
$R(i,j,2)$	1	1	1	1	1	1
$R(i,j,3)$	1	1	0	1	1	0

TABLE 8.2 – Valeurs de $R(i,j,k)$

Nous montrons maintenant le lemme suivant :

Lemme 9 *Si $S_k \circ L$ n'est pas affine, alors il existe u, v distincts et i, j distincts tels que $\ell_{i,u} = \ell_{j,v} = 1$ et $R(i,j,k) = 1$.*

Démonstration : Commençons par écrire les équations algébriques de S.

$$\begin{aligned} S_0(e) &= e_0e_2 + e_1 + e_2 + e_3 \\ S_1(e) &= e_0e_1e_2 + e_0e_1e_3 + e_0e_2e_3 + e_1e_2 + e_0x_3 + e_2e_3 + e_0 + e_1 + e_2 \\ S_2(e) &= e_0e_1e_3 + e_0e_2e_3 + e_1e_2 + e_1e_3 + e_2e_3 + e_0 + e_1 + e_3 \\ S_3(e) &= e_0e_1e_3 + e_1e_3 + e_0 + e_1 + e_2 + 1 \end{aligned} \tag{8.6}$$

Supposons alors que $S_k \circ L$ n'est pas affine. Le coefficient d'au moins un de ses monômes de degré 2 ou plus vaut 1. L étant affine, cela ne peut se produire que si le coefficient d'au moins un monôme de degré 2 ou 3 de S_k, appliqué à $L(X)$, vaut 1. Notons-le y. Si $y(e) = e_ie_j$ est de degré 2, alors

$$y \circ L(X) = \bigoplus_{u \neq v} \ell_{i,u}\ell_{j,v}x_ux_v \oplus \lambda(X),$$

où λ est affine. Nécessairement, l'un des $\ell_{i,u}\ell_{j,v}$ vaut 1. On vérifie facilement que l'existence d'un tel e_ie_j implique $R(i,j,k) = 1$. Si $y = e_he_ie_j$ est de degré 3, alors

$$y \circ L(X) = \bigoplus_{u \neq v} (\ell_{h,u}\ell_{i,v}\ell_j \oplus \ell_{i,u}\ell_{j,v}\ell_h \oplus \ell_{j,u}\ell_{h,v}\ell_i)x_ux_v \oplus \bigoplus_{u \neq v \neq w \neq u} \ell_{h,u}\ell_{i,v}\ell_{j,w}x_ux_vx_w \oplus \lambda(X).$$

On en déduit également que l'un des $\ell_{h,u}\ell_{i,v}$, $\ell_{i,u}\ell_{j,v}$, $\ell_{j,u}\ell_{h,v}$ vaut 1. Sans perte de généralité, supposons qu'il s'agit de $\ell_{i,u}\ell_{j,v}$. Par définition de R, $R(i,j,k) = 1$, ce qui achève la démonstration. □

Par contraposée du lemme 9, on voit facilement que si dans le cas de `Hamsi-256`, on parvient à garantir que les entrées d'une boîte S du tour 2 ou 3 sont des fonctions affines des variables, et qu'elles ne vérifient pas les hypothèses du lemme pour un bit de sortie k, alors on est certain que ce bit de sortie s'exprime également comme une fonction affine des variables. Cela se produit en particulier dans les deux cas suivants :

- Toutes les entrées d'une boîte S dépendent seulement d'une même variable ;
- Seul un des bits d'entrée d'une boîte S n'est pas constant.

Nous allons maintenant décrire comment exploiter ce résultat pour conduire une recherche automatique, permettant de déterminer des classes de restrictions de domaine qui garantissent que certains des bits de sortie $Q_{i,\mathcal{V}}$ s'expriment comme des fonctions affines des variables.

8.4.2 Recherche de relations affines

Caractérisation des restrictions de domaine. Comme mentionné en introduction de la section, nous cherchons des relations affines sur trois tours en sélectionnant des variables à des positions précises dans les mots s_{12} et s_{13} après la première couche de substitution. La classe de restrictions de domaine correspondante est donc définie relativement à un ensemble d'indices $S = t_1, \ldots, t_r \subset \{0, \ldots, 63\}$ de la manière suivante.

- M est constant
- Les sorties vérifient :

$$\begin{aligned} \forall i \in \{1, \ldots, r\}, \mathcal{E}(M)^{(t_i)} &= 1 \oplus A^{(t_i)} \\ C^{(64+t_i)} &= x^{(i)} \oplus A^{(128+t_i)} \\ \mathcal{E}(M)^{(128+t_i)} &= A^{(256+t_i)} \\ C^{(192+t_i)} &= \overline{x^{(i)}} \oplus A^{(384+t_i)} \end{aligned} \tag{8.7}$$

– Les autres bits de C sont constants.

On cherche alors à déterminer, pour chaque valeur de r, quel ensemble d'indices permet de garantir un maximum de polynômes $Q_{i,\mathcal{V}}$ de degré 1. Comme expliqué dans la section 8.1, deux phases de l'attaque en recherche de préimages nécessitent beaucoup de temps de calcul : l'interpolation des coefficients des relations affines et le test des solutions. La complexité de la première phase dépend essentiellement du nombre r de variables considérées, alors que celle de la deuxième phase dépend uniquement du nombre N_{eq} de relations obtenues. Or plus on ajoute de variables, plus le nombre de relations affines décroît. La recherche des plus grands jeux de relations à r fixé permettra donc d'établir un compromis entre r et N_{eq}. Une évaluation précise de la complexité de l'algorithme sera donnée dans la section 8.5.

De plus, nous verrons que les systèmes de relations trouvées permettent également de rechercher un antécédent d'une image parmi un ensemble d'images. Dans ce cas, le compromis obtenu peut être différent.

Enfin, pour une attaque en recherche de collisions sur la fonction de compression, une augmentation du nombre d'équations fait augmenter la complexité de la phase d'interpolation. De ce fait, nous verrons que le meilleur compromis consiste à réduire le nombre de relations effectivement exploitées.

Algorithme de recherche. Pour un sous-ensemble d'indices fixé, pour tout $\mathcal{V} \in J$, la situation après la première couche $\mathcal{S}$ est la suivante :

– Pour tout $i \in \{1, \ldots, r\}$, $s^{(384+t_i)} = x^{(i)}$,

– Tous les autres bits de s sont constants.

On peut maintenant calculer les coefficients des termes linéaires après la première couche $\mathcal{L}$ et la deuxième addition $\mathcal{A}$. En appliquant la contraposée du lemme 9, on peut alors déterminer quels bits de l'état sont des fonctions affines des variables après $\mathcal{S}$, et de quelles variables ils dépendent. On établit ensuite la propagation de ces dépendances par la couche $\mathcal{L}$ et l'addition $\mathcal{A}$, et on applique le même raisonnement pour la troisième couche de substitution $\mathcal{S}$. Enfin, on détermine la propagation des dépendances par la dernière couche $\mathcal{L}$, la troncation et le rebouclage. On peut alors compter le nombre de bits de sortie dont on peut garantir qu'ils s'expriment comme une fonction affine des variables.

On parcourt l'ensemble des jeux de variables possibles. Pour éviter d'avoir à rechercher les relations affines pour les 2^{64} jeux possibles, on évite les jeux de variables dont un sous-ensemble a déjà été testé et n'a permis d'aboutir à aucune relation affine. On maintient à jour le nombre maximal de relations affines obtenues pour chaque valeur de r.

En utilisant cette méthode, nous parvenons aux résultats du tableau 8.3

Nombre de variables	2	3	4	5	6	7	8	9	10	11	12	13
Nombre d'équations	224	144	75	50	32	21	16	11	9	8	7	6
Nombre de variables	14	15	16	17	18	19	20	21	22	23	24	25
Nombre d'équations	5	4	3	3	3	3	3	2	2	2	2	2

TABLE 8.3 – Nombre maximal N_{eq} de relations linéaires obtenues, en fonction du nombre r de variables

Notons que pour plus de 16 variables, le bloc de message ne contient pas suffisamment de degrés de liberté pour garantir l'existence d'une transformation initiale valide. En effet, la définition de chaque variable impose de fixer deux bits de $\mathcal{E}(\mathcal{M})$. L'étude de ces jeux de variables permet d'établir des propriétés algébriques indésirables de la permutation interne de Hamsi-256.

En particulier, nous avons déterminé les propriétés suivantes pour 8 et 9 variables.

- Pour un ensemble de 8 indices $\{t_i\} = \{2, 9, 27, 37, 38, 45, 62, 63\}$, les 16 polynômes $\{Q_{j,v}\}$ pour $j \in \{10, 38, 43, 44, 73, 80, 129, 136, 154, 188, 220, 230, 231, 247, 253, 254\}$ sont de degré au plus 1. Le nombre de conditions induites sur les entrées de $\mathcal{F}$ est $N_{cond} = 24$.
- Pour un ensemble de 9 indices $\{t_i\} = \{2, 20, 27, 28, 37, 45, 57, 62, 63\}$, les 11 polynômes $\{Q_{j,v}\}$ pour $j \in \{37, 38, 44, 154, 217, 219, 229, 246, 247, 253, 254\}$ sont de degré au plus 1. Le nombre de conditions induites sur les entrées de $\mathcal{F}$ est $N_{cond} = 27$.

8.5 Attaques sur la fonction de compression de Hamsi-256

Nous décrivons comment appliquer les algorithmes 8.1 et 8.2 à Hamsi-256 pour trouver des préimages et des collisions sur la fonction de compression. En particulier, nous montrons comment retrouver les coefficients des systèmes d'équations linéaires de manière efficace, et nous évaluons la complexité des attaques induites.

8.5.1 Recherche de préimages

Une première idée pour retrouver ces coefficients est de les interpoler en appliquant $\mathcal{F} \circ \mathcal{V}$ à tous les vecteurs de poids de Hamming au plus 1, ce qui représente $r + 1$ évaluations de la fonction de compression. Cependant, la diffusion des bits de la variable de chaînage n'est pas immédiate. En particulier, recalculer toute la fonction de compression en modifiant uniquement un bit après la première couche de boîtes S implique de refaire des opérations déjà effectuées. Nous tirons donc parti des idées suivantes.

1. Il n'est pas nécessaire d'effectuer l'intégralité des opérations de la fonction de compression pour calculer une petite partie des bits de sortie. On peut se contenter de calculer les bits des variables intermédiaires dont dépendent effectivement les bits de sortie qui nous intéressent.
2. Il est possible de garder en mémoire les résultats des opérations intermédiaires effectuées pour le calcul des coefficients constants. Une grande partie de ces résultats peuvent alors être réutilisés pour le calcul des coefficients de degré 1, qui impliquent uniquement l'inversion d'un bit en sortie de la première couche de boîtes S.
3. Pour générer deux systèmes d'équations consécutifs, il est possible de choisir les bits libres de C de manière à mutualiser une partie du calcul des coefficients constants.

Les optimisations trouvées pour générer les systèmes fonctionnent selon deux axes : premièrement, le calcul des coefficients constants, et deuxièmement, le calcul des coefficients linéaires, une fois le calcul des coefficients constants effectué.

Calcul des coefficients de degré 1. Le coefficient du monôme x_j, $j \in \{1, \ldots, r\}$ est donné par $Q_{i,\mathcal{V}}(2^{j-1}) \oplus Q_{i,\mathcal{V}}(0)$. $Q_{i,\mathcal{V}}(0)$ a déjà été calculé, et les calculs intermédiaires effectués sont stockés en mémoire. Par construction, on sait que l'état interne de $Q_{i,\mathcal{V}}(2^{j-1})$ après la première couche de boîtes S peut être obtenu en inversant le bit $384 + t_j$ de l'état interne à partir du calcul de $Q_{i,\mathcal{V}}(0)$. Les opérations $\mathcal{L}$ et $\mathcal{A}$ sont déterministes vis-à-vis de la propagation de différences, donc

on retrouve l'état interne du calcul de $Q_{i,\mathcal{V}}(2^{j-1})$ par le calcul de quelques boîtes S du deuxième tour. On détermine ensuite la propagation des différences éventuelles par rapport au calcul du coefficient constant à travers $\mathcal{L}$ et $\mathcal{A}$, puis on recalcule la partie de la troisième couche de boîtes S qui dépend de x_j. Enfin, on applique les transformations linéaires qui permettent de retrouver le bit i de la sortie de la fonction de compression.

Utilisation de variables auxiliaires. Nous cherchons maintenant à accélerer le calcul du coefficient constant. La méthode la plus simple pour calculer ces coefficients consiste à fixer les variables à 0 et à calculer la fonction de compression. Le coefficient constant de $Q_{i,\mathcal{V}}$ est $Q_{i,\mathcal{V}}(0)$, c'est-à-dire le bit i de $\mathcal{F} \circ \mathcal{V}(0)$. Il est cependant possible de calculer ces coefficients plus rapidement. D'après la forme des restrictions de domaine utilisées dans notre attaque, les coefficients constants des systèmes d'équations sont caractérisés par

- M est constant
- Les sorties vérifient :

$$\begin{aligned} \forall i \in \{1,\ldots,r\}, \mathcal{E}(M)^{(t_i)} &= 1 \oplus A^{(t_i)} \\ C^{(64+t_i)} &= A^{(128+t_i)} \\ \mathcal{E}(M)^{(128+t_i)} &= A^{(256+t_i)} \\ C^{(192+t_i)} &= \overline{A^{(384+t_i)}} \end{aligned} \tag{8.8}$$

- Les autres bits de C sont constants.

Nous pouvons remarquer que les conditions portant sur les entrées de $\mathcal{F}$ fixent toutes des valeurs constantes. Nous généralisons maintenant la forme des restrictions de domaine utilisées dans la section 8.3.

- M est constant
- Il existe r' valeurs distinctes $s_1, \ldots, s_{r'}$ entre 0 et 31 telles que :

$$\begin{aligned} \forall i \in \{1,\ldots,n\}, \mathcal{E}(M)^{(192+s_i)} &= 1 \oplus A^{(448+s_i)} \\ C^{(t_i)} &= x^{(i)} \oplus A^{(64+s_i)} \\ C^{(128+s_i)} &= \overline{x^{(i)}} \oplus A^{(320+s_i)}. \end{aligned} \tag{8.9}$$

- Les autres bits de C sont constants.

Nous appelons *restriction de domaine auxiliaire* une restriction de domaine qui remplit ces conditions. Parmi "les autres bits de C" traités au troisième point, figurent les valeurs de $C^{(64+t_i)}$ et $C^{(192+t_i)}$ sur lesquels des conditions sont imposées pour les restrictions de domaine sur 3 tours. Nous pouvons donc choisir les restrictions de domaine auxiliaires de manière à satisfaire toutes ces conditions. Les conditions portant sur $\mathcal{E}(M)$ peuvent également être satisfaites par un choix approprié du bloc de message.

Pour une telle restriction de domaine auxiliaire, tous les choix de $x^{(i)}$ conduisent par conséquent à des valeurs de (C, M) correspondant à des coefficients constants pour une restriction de domaine sur 3 tours. Pour accélerer le calcul des coefficients constants, nous parcourons la classe de restrictions de domaine auxiliaires $\{V'_j\}$ ainsi définis. Pour chaque valeur de l'indice j et du vecteur $y \in \{0,1\}^{r'}$, $\mathcal{V}'_j(y)$ correspond au coefficient constant $\mathcal{V}_{j,y}(0)$ d'un système $\mathcal{V}_{j,y}$. Nous choisissons cette indexation pour les restrictions de domaine sur 3 tours.

Les coefficients constants des $Q_{i,\mathcal{V}_{j,y}}$ sont donc les $Q_{i,\mathcal{V}'_j}(y)$, dont le calcul est linéaire en y jusqu'au début du troisième tour. Les calculs intermédiaires jusqu'au troisième tour peuvent alors

être retrouvés par application de la fonction de compression pour $Q_{i,\mathcal{V}'_j}(0)$, puis par interpolation des deux premiers tours, en utilisant la technique décrite plus haut. Pour chaque vecteur y, il reste alors à calculer la partie du troisième tour de la fonction de compression qui intervient dans le calcul du bit i de la sortie.

L'algorithme 8.3 décrit une version détaillée de la recherche de secondes préimages.

Algorithme 8.3 Calcul de préimages pour la fonction de compression de Hamsi-256

Entrées : $H \in \{0,1\}^n$
Sorties : $(C, M) \in \{0,1\}^{n+m}$ tel que $\mathcal{F}(C, M) = H$

```
j ← 0
V' ← V'_j
calculer les deux premiers tours des Q_{i,V'}(0) et garder les calculs intermédiaires en mémoire
pour k ∈ {1, ..., r} faire
  calculer le coefficient linéaire des deux premiers tours des Q_{i,V'}(e_k)
fin pour
boucle
  pour y ∈ {0,1}^{r'} faire
    V ← V_{j,y}
    retrouver les résultats des opérations intermédiaires du calcul des Q_{i,V}(0) des deux premiers
    tours de F
    calculer tous les Q_{i,V}(0) en appliquant le troisième tour de F
    interpoler les coefficients de degré 1 des Q_{i,V}
    Σ ← {Y ∈ {0,1}^r | ∀i ∈ I, Q_{i,V}(Y) = H^(i)}
    pour Y ∈ Σ faire
      (C, M) ← V(Y)
      F = F(C, M)
      si F = H alors
        renvoyer (C, M)
      fin si
    fin pour
  fin pour
  j ← j + 1
fin boucle
```

8.5.2 Calcul de la complexité

Nous cherchons maintenant à évaluer précisément les complexités T_{build} et T_{solve}. Comme on entre dans les détails de la fonction de compression, ces complexités ne s'expriment pas naturellement en nombre d'appels à la fonction de compression. Nous commençons donc par les exprimer en nombre d'opérations élémentaires sur des bits (OU LOGIQUE, ET LOGIQUE, OU EXCLUSIF). Ces quantités sont ensuite converties en nombre d'appels à la fonction de compression équivalents en utilisant le décompte effectué par Dinur et Shamir [SD11], qui estiment le nombre d'opérations élémentaires impliquées dans le calcul de la fonction de compression à 10500. Pour les couches linéaires, le décompte des opérations est assez simple, chaque bit de sortie étant le XOR de certains bits d'entrée. Pour les boîtes S, le nombre d'opérations nécessaires dépend fortement de l'implantation. Cette boîte S étant tirée de la définition de l'algorithme de chiffrement par blocs Serpent, la

complexité de son calcul a déjà été étudiée. Pour notre attaque, nous nous référons à l'implantation d'Osvik [Osv00]. La complexité du calcul de la sortie d'une boîte est 14 opérations (hors copie de bits). De plus, ce nombre diminue lorsque seulement une partie des bits de sortie doivent être calculés.

D'autre part, la complexité de l'algorithme dépend fortement de la classe de restriction de domaines utilisée. Les applications données ici concernent donc les systèmes de 16 équations en 8 variables et de 11 équations en 9 variables, qui représentent les meilleurs choix.

Les étapes d'initialisation ont une complexité négligeable. De plus, le calcul de $\mathcal{V}$ ne nécessite que quelques opérations de OU EXCLUSIF et peut également être négligé. Le choix de la restriction de domaine auxiliaire peut être mutualisé pour de nombreux systèmes (par exemple, tous ceux pour lesquels $\mathcal{V}_{j,0}(0)$ parcourent toutes les valeurs de C_1 et C_5, soit 2^{64} valeurs possibles de j, le reste de l'état étant constant). Sa complexité est donc également négligée.

Calcul du coefficient constant pour la restriction de domaine auxiliaire. La complexité du calcul des deux premiers tours de $\mathcal{F}$ dépend de X. On peut donc calculer le coût global de cette étape pour les 2^r systèmes obtenus lorsque X prend toutes les valeurs possibles. La première étape consiste alors à calculer la partie deux premiers tours de $\mathcal{F} \circ \mathcal{V}'(0)$ qui intervient dans le calcul des $Q_{i,\mathcal{V}_{j,0}}$. La complexité de cette étape peut être évaluée à 6450 opérations élémentaires, pour chaque valeur de j.

Calcul des coefficients linéaires pour la restriction de domaine auxiliaire. L'étape suivante consiste à calculer les coefficients linéaires de $F \circ \mathcal{V}'$ jusqu'à l'entrée de la troisième couche de substitution. Une fois les variables auxiliaires choisies, leur propagation est déterministe jusqu'à l'entrée de la deuxième couche de boîtes S. Chacune des variables auxiliaires n'impacte qu'un bit en sortie de la première couche de substitution, et impacte donc au maximum 7 bits en entrée de la deuxième couche de substitution. En effet, une différence sur un bit en entrée de la couche de diffusion linéaire se propage sur 7 bits de sortie au maximum. On calcule alors les coefficients linéaires en sortie de $\mathcal{S}_2$ de la manière suivante.

Pour chaque variable et chaque boîte S dont la sortie est susceptible d'être modifiée, on ajoute le coefficient linéaire au coefficient constant, on évalue la boîte S, et on calcule la différence entre la sortie obtenue et la sortie obtenue pour le coefficient constant. Cela nécessite au plus $7\times(1+14+4) = 133$ opérations par variable auxiliaire.

En entrée de la deuxième couche de diffusion $\mathcal{L}_2$, les sorties d'au plus 7 boîtes S dépendent de chaque variable. Pour chaque boîte S, le calcul de la propagation du coefficient linéaire par $\mathcal{L}_2$ nécessite au plus 20 additions (7 pour les entrées a et c, 3 pour les entrées b et d).

Au total, la complexité de cette étape est donc au plus 287 opérations par variable.

Calcul des coefficients constants pour la restriction de domaine principale. On parcourt ensuite toutes les affectations possibles des variables auxiliaires selon un code de Gray. Chacune de ces affectations permet alors la définition d'un système d'équations linéaires dont les inconnues sont les variables principales. Le calcul des deux premiers tours de $\mathcal{F} \circ \mathcal{V}_{j,X}(0)$ est retrouvé en ajoutant les effets linéaires d'une variable auxiliaire en entrée et en sortie des deux premières couches $\mathcal{S}$ et en entrée de la troisième couche S. D'après l'analyse précédente, cela représente au maximum $2 + 1 + 7 + 28 + 140 = 178$ additions.

Les coefficients constants du système sont alors obtenus en calculant partiellement le troisième tour et le rebouclage. La complexité de cette étape est évaluée à 363 opérations pour les systèmes de 11 équations et 9 inconnues.

Calcul des coefficients linéaires du système d'équations. Lors de cette étape on connaît a priori les parties de l'état interne que chacune des variables peut impacter au cours du calcul. Cette étape étant la plus coûteuse de la génération du système, on exploite cette information pour limiter sa complexité. Pour les calculs de boîtes S, on utilise le raisonnement suivant. Si l'entrée d'une boîte S pour le terme constant est x et le masque des coefficients linéaires pour une variable est δ, alors les coefficients linéaires sur la sortie sont donnés par $\Delta_x(S)(\delta) = S(x \oplus \delta) \oplus S(x)$. On distingue alors les deux cas suivants :

- Si δ peut a priori prendre plusieurs valeurs non nulles, on calcule $x \oplus \delta$, puis $S(x \oplus \delta)$ et enfin $\Delta_x(S)(\delta)$.
- Si δ ne peut prendre qu'une valeur non nulle, on calcule directement la différentielle $\Delta_x(S)(\delta)$ comme une fonction booléenne de x.

La dépendance en chaque variable est connue de manière déterministe en entrée de $\mathcal{S}_2$. Il reste donc à évaluer cette dépendance en sortie de $\mathcal{S}_2$, $\mathcal{L}_2$, $\mathcal{S}_3$, $\mathcal{L}_3$ et du rebouclage. Pour les systèmes de 11 équations et 9 inconnues, cela représente au total 479 opérations.

Complexité T_{build}. La complexité du calcul des coefficients du système d'équations linéaires dépend de r. Or, pour chaque choix de M et de C_2, C_6, le nombre R de variables auxiliaires que l'on peut définir varie. Notons alors T_r la complexité moyenne pour construire un système d'équations lorsque r variables auxiliaires peuvent être définies, P_r la probabilité que r variables auxiliaires soient définies et ω_i l'évènement suivant : pour un choix de M aléatoire, il existe exactement i valeurs de j telles que $\mathcal{E}(M)^{(192+j)} \oplus A^{(448+j)} = 1, 0 \leq j \leq 31$

On a alors :

$$\begin{aligned} P_r &= \Sigma_{i=r}^{32} \Pr[R = r|\omega_i] \Pr[\omega_i] \\ &= \Sigma_{i=r}^{32} \frac{\binom{i}{r}}{2^i} \frac{\binom{32}{i}}{2^{32}} \\ T_r &\leq \frac{u + 465r}{2^r} + v \end{aligned}$$

u et v sont des constantes qui dépendent des variables et des bits de sortie choisis. Pour les systèmes de 11 équations et 9 inconnues, $u = 6450$ opérations et $v = 1020$ opérations. On a alors :

$$T_{build} = \Sigma_{i=0}^{32} P_r T_r$$

On en déduit pour les systèmes d'équations à 9 inconnues :

$$T_{build}^{(9)} \approx 2^{-3,2} T_{\mathcal{F}} \tag{8.10}$$

Résolution du système d'équations. L'étape suivante de l'algorithme consiste à résoudre le système d'équations linéaires trouvé. Pour cela, nous utilisons l'algorithme de Gauss, dont la complexité en nombre d'opérations peut être estimée de la manière suivante. Pour chacune des r variables et pour chacune des N_{eq} équations, on calcule au plus $(N_{var} + 1)$ additions, et le

nombre moyen d'additions calculées est $N_{var}/2$. La complexité de cet algorithme est donc environ $N_{var}(N_{var}+1)N_{eq}/2$ opérations par système. Pour les systèmes de 9 équations et 11 inconnues, cela se traduit par :

$$T^{(9)}_{solve} \approx 495 \approx 2^{-4,4} T_{\mathcal{F}} \tag{8.11}$$

Parallélisation. La construction des systèmes peut facilement être parallélisée. En effet, pour des choix identiques des variables auxiliaires, les opérations élémentaires effectuées pour retrouver les coefficients sont les mêmes, et peuvent être implémentées en parallèle. Chaque registre de taille r contient alors r bits qui jouent le même rôle pour deux choix différents des parties constantes de l'état interne.

En revanche, l'algorithme utilisé pour la résolution des systèmes ne contient pas ce parallélisme intrinsèque. La parallélisation de la résolution du système d'équations imposerait une augmentation du nombre d'opérations.

Complexité de la recherche de préimages. Des équations (8.10),(8.11) et (8.1) nous tirons l'estimation suivante. En utilisant l'algorithme 8.3 avec des systèmes de 11 équations et 9 inconnues, il est possible de calculer des préimages sur la fonction de compression de Hamsi-256 avec une complexité

$$T^{(9)}_{preimage} \approx (2^{247}(2^{-3,2} + 2^{-4,4}) + 2^{245}) T_{\mathcal{F}} \approx 2^{245,7} T_{\mathcal{F}}. \tag{8.12}$$

Estimation du nombre de préimages trouvables. Dans l'attaque de la section suivante, il est nécessaire de trouver plusieurs préimages par la fonction de compression d'une valeur de la variable de chaînage. Nous allons donc estimer le nombre de préimages d'un élément H que nous pouvons trouver par cette méthode. Le domaine de définition de $\mathcal{F}$ contient $2^{256+32} = 2^{288}$ éléments. Le nombre moyen de préimages de H est donc de $2^{288-256} = 2^{32}$. Pour qu'une telle préimage puisse être trouvée en utilisant l'algorithme 8.3, il suffit que $(C, M) = \mathcal{V}(x)$, pour une certaine restriction de domaine $\mathcal{V}$ et une certaine affectation des variables x. Cela se produit sous les conditions 8.7. Cela représente $4r$ conditions pour chaque affectation de x_i. Comme il y a 2^r choix possibles pour les valeurs de x_i, la probabilité qu'une préimage (C, M) de H puisse être trouvée par l'algorithme 8.3 est 2^{-3r}. Cet algorithme permet donc en moyenne de trouver 2^{32-3r} préimages, soit 32 en utilisant 11 relations impliquant 9 variables.

8.5.3 Collisions à valeurs initiales choisies sur la fonction de hachage

D'après le raisonnement effectué dans la section 8.1, le calcul de préimages sur la fonction de compression implique le calcul de collisions sur $\mathcal{F}$. La complexité de l'algorithme permettant de trouver ces collisions est de

$$T^{(9)}_{collision} \approx (2^{124,5}(2^{-3,2} + 2^{-4,4}) + 2^{122,5}) T_{\mathcal{F}} \approx 2^{123,2} T_{\mathcal{F}}.$$

Du fait de l'algorithme d'extension de domaine utilisé pour Hamsi-256, cette attaque sur la fonction de compression se traduit immédiatement par une attaque en recherche de collisions à valeurs initiales choisies sur la fonction de hachage. En effet, si $\mathcal{F}(C_0, M_0) = \mathcal{F}(C_1, M_1)$, en notant $\texttt{Hamsi}_{IV}$ la version de Hamsi-256 modifiée de telle sorte que la valeur d'initialisation soit remplacée par la valeur IV, on a $\texttt{Hamsi}_{C_0}(M_0) = \texttt{Hamsi}_{C_1}(M_1)$, les blocs de message contenant la longueur de M_0 et M_1 étant identiques.

Notons toutefois que ce modèle d'attaquant ne correspond à aucune notion de sécurité nécessaire pour des applications pratiques des fonctions de hachage. Pour certaines fonctions de hachage, il est trivial de générer des collisions dans ce modèle. C'est le cas des fonctions éponges telles que Keccak ou Luffa, ou encore de Shabal.

8.6 Recherche de secondes préimages pour Hamsi

Nous avons montré dans la section 8.5 qu'il est possible de trouver des préimages pour la fonction de compression de Hamsi-256 avec une complexité équivalente à environ $2^{245,3}$ évaluations de la fonction de compression. Cette propriété compromet la sécurité de Hamsi-256. En effet il est possible d'utiliser la recherche de préimages sur la fonction de compression pour trouver des secondes préimages sur la fonction de hachage complète. Une première méthode consiste à utiliser un algorithme de rencontre au milieu. Cette méthode permet de transformer de manière générique une recherche de préimage sur la fonction de compression en recherche de seconde préimage sur la fonction de hachage, à l'aide d'un algorithme de rencontre au milieu déséquilibré introduit par Lai et Massey dans [LM92]. La complexité de cet algorithme est alors équivalente à $2^{\frac{256+245,7}{2}+1} = 2^{251,9}$ évaluations de la fonction de compression.

Cependant, un tel algorithme ne tire pas parti de la forme spécifique de notre attaque sur la fonction de compression. En effet, une partie importante de la complexité de l'algorithme consiste à construire et à inverser des systèmes d'équations linéaires. Or si on cherche une préimage d'un élément dans un ensemble d'images et non pas d'un élément unique, la phase de génération des relations affines peut être mutualisée. Nous décrirons donc un algorithme qui tient compte de cette possibilité.

8.6.1 Préimages d'un ensemble d'éléments

Nous cherchons maintenant à évaluer la complexité de la recherche d'une préimage d'un élément quelconque d'un ensemble S de cardinal s. La complexité d'un tel algorithme est inférieure à la complexité de la recherche d'une préimage d'un élément donné. Pour cette notion, l'attaque générique consiste en une recherche probabiliste. Sa complexité est de $2^n/s$ évaluations de la fonction de compression.

Dans l'algorithme 8.3, la génération de relations affines est identique pour toutes les images $H \in S$. Seule l'affectation des bits de sortie dépend de H. On peut donc déterminer ces relations une fois pour s images, et ensuite résoudre les s systèmes d'équations induits.

Une partie de la résolution du système d'équations peut également être mutualisée. L'attaquant cherche à résoudre des systèmes d'équations de la forme $Ax = b$, où b est un vecteur constant de taille N_{eq}, x est un vecteur d'inconnues de taille r et A est une matrice binaire fixée. Il peut alors commencer par appliquer l'algorithme de Gauss en parallèle sur une base de l'espace de définition de b. La complexité de cette partie est notée T_{invert}. L'analyse effectuée dans la section précédente permet d'évaluer sa complexité à $N_{eq}N_{var}(N_{eq}+N_{var})/2$ opérations binaires. La fin de la résolution consiste alors à vérifier quelques relations linéaires (au plus N_{eq}) sur les bits de b. La complexité de cette phase est $T_{check} \leq N_{eq}^2$ opérations binaires, et peut être négligée dans les applications numériques.

La complexité de la recherche d'une préimage d'une image parmi s est définie par :

$$T_{set}(N) = \frac{2^{256-r}}{s}(T_{build} + T_{invert}) + 2^{256-r}T_{check} + 2^{256-N_{eq}}T_{\mathcal{F}} \tag{8.13}$$

Pour un système de 11 équations et 9 inconnues, on a $T^{(9)}_{invert} \approx 2^{-3,4}T_{\mathcal{F}}$. Pour trouver une préimage dans un ensemble de préimages, le test des solutions devient vite la partie la plus coûteuse en temps de calcul lorsque s augmente. De ce fait, il est avantageux d'utiliser des systèmes avec moins de variables et plus d'équations. Avec 8 variables, du fait du coût de la génération et de l'inversion de systèmes, il est préférable de limiter le nombre d'équations que l'on génère. En utilisant 14 équations, on aboutit à $T^{(8)}_{build} \approx 2^{-2,1}T_{\mathcal{F}}$ et $T^{(8)}_{invert} \approx 2^{-3,1}T_{\mathcal{F}}$. En utilisant 13 équations, on aboutit à $T^{(8)}_{build} \approx 2^{-3,1}T_{\mathcal{F}}$ et $T^{(8)}_{invert} \approx 2^{-3,6}T_{\mathcal{F}}$.

8.6.2 Secondes préimages pour Hamsi-256

Afin d'alléger l'écriture, nous utiliserons la notation suivante pour la composition de fonctions de compression :

$$\begin{aligned} \mathcal{F}_0(C) &= C \\ \forall i > 0, \mathcal{F}_i(C, m_0, \ldots, m_{i-1}) &= \mathcal{F}(\mathcal{F}_{i-1}(C, m_0, \ldots, m_{i-2}), m_i - 1) \end{aligned}$$

Nous pouvons maintenant décrire notre attaque en recherche de secondes préimages pour Hamsi-256. Contrairement aux attaques génériques, sa complexité ne dépend pas de la longueur du message initial. Lorsque celui-ci est suffisamment court, notre attaque est meilleure que l'attaque de Kelsey et Schneier, qui est la meilleure attaque générique connue en recherche de secondes préimages.

Considérons donc un message M qui contient au moins 10 blocs, et cherchons M' tel que $\mathcal{H}(M) = \mathcal{H}(M')$. Nous notons $M = m_0||\ldots||m_9||\ldots||m_\ell$, et on considère la variable de chaînage $h_{10} = \mathcal{F}_{10}(IV, m_0, \ldots, m_9)$. Dans un premier temps, nous cherchons à déterminer x préimages de h_{10} par $\mathcal{F}$, que nous notons $(h_{9,1}, m_{9,1}), \ldots, (h_{9,x}, m_{9,x})$. Pour cela nous utilisons l'algorithme 8.3, avec des systèmes de 11 équations et 9 inconnues. La complexité de cette étape est environ

$$T_1(x) = x \times (2^{247}(T^{(9)}_{build} + T^{(9)}_{solve}) + 2^{245}T_{\mathcal{F}}) \approx 2^{245,7}xT_{\mathcal{F}}\ .$$

Plutôt que d'appliquer directement l'algorithme de rencontre au milieu, nous utilisons une deuxième phase de calcul de préimages, que nous appelons phase d'accélération. En partant de $S = \{h_{9,0}, h_{9,1}, \ldots, h_{9,x-1}\}$ (où $h_{9,0} = \mathcal{F}_9(IV, m_0, \ldots, m_8)$) , nous cherchons y préimages d'un élément de S par $\mathcal{F}$. L'algorithme de rencontre au milieu est ensuite appliqué sur cet ensemble de y valeurs. La recherche de secondes préimages est résumée par la Figure 8.1.

Pour la phase d'accélération, nous utilisons des systèmes de 8 équations et 14 inconnues. La complexité de cette étape est :

$$T_2(x, y) = \left(\frac{2^{248}}{x+1}(T^{(8)}_{build} + T^{(8)}_{invert}) + 2^{242}T_{\mathcal{F}}\right) \times y \approx \left(2^{246,5}\frac{y}{x+1} + 2^{242}y\right)T_{\mathcal{F}}.$$

Finalement, en utilisant une approche probabiliste, nous cherchons $(m^*_0||\ldots||m^*_7) \neq (m_0||\ldots||m_7)$ tel que $h^*_8 = \mathcal{F}_7(IV, m^*_0, \ldots, m^*_7)$ collisionne avec l'un des $h_{8,j}$, où $h_{8,0} = \mathcal{F}_8(IV, m_0, \ldots, m_7)$. La complexité de cette étape est :

$$T_3(y) = \frac{2^{256}}{y+1}. \tag{8.14}$$

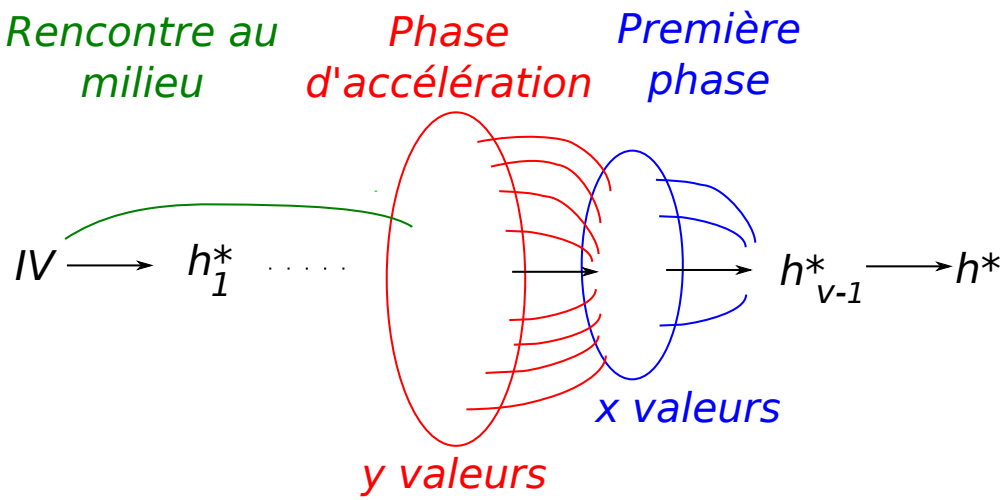


FIGURE 8.1 – Technique de rencontre au milieu avec phase d'accélération utilisée pour la recherche de secondes préimages pour Hamsi.

Notons alors $m_{8,0} = m_8$ et $m_{9,0} = m_9$. Pour j défini ci-dessus, il existe i tel que $\mathcal{F}(h_{8,j}, m_{8,j}) = h_{9,i}$. De ce fait, $\mathcal{F}_{10}(IV, m_0^*, \ldots, m_7^*, m_{8,j}, m_{9,i}) = h_{10}$, et par conséquent :

$$\mathcal{H}(m_0^* || \ldots || m_7^* || m_{8,j} || m_{9,i} || \ldots || m_\ell) = \mathcal{H}(M). \tag{8.15}$$

Nous pouvons donc calculer une seconde préimage pour M avec une complexité :

$$T(x,y) = T_1(x) + T_2(x,y) + T_3(y) \approx \left(2^{245,7} \times x + \frac{2^{246,5} y}{x+1} + 2^{242} \times y + \frac{2^{256}}{y+1}\right) T_{\mathcal{F}}. \tag{8.16}$$

Pour `Hamsi`-256, le choix optimal de x et y est le suivant. Pour $x = 10$ et $y = 72$,

$$T_1(x) \approx 2^{249} T_{\mathcal{F}} \ , \ T_2(x,y) \approx 2^{249.8} T_{\mathcal{F}} \ , \ T_3(y) \approx 2^{249.8} T_{\mathcal{F}}$$

La complexité totale de l'attaque est donc $T(x,y) \approx 2^{251,16} T_{\mathcal{F}}$.

Comparaison avec les attaques génériques. Dans le chapitre 2, nous décrivons une généralisation de l'attaque de Kelsey et Schneier sur les fonctions de hachage basées sur l'algorithme d'extension de domaine de Merkle-Damgård. Dans le cas de `Hamsi`-256, $p = 4$, $q = 8$ et $r = 2$. Cette attaque permet de trouver des secondes préimages avec une complexité $T(k) = k2^{128} + 2^{256-k}$, pour des messages d'au moins $4k + 2^k + 9$ blocs. Notre attaque est plus efficace que cette attaque générique tant que le message fait moins de 41 blocs.

Comparaison avec l'attaque de Shamir et Dinur. Dans notre attaque, nous choisissons des nombres de variables r égaux à 7 ou 8, et nous considérons des restrictions de domaine et des ensembles de bits de sortie de manière à garantir que les polynômes $Q_{i,\mathcal{V}}$ soient linéaires. Cela rend très facile la résolution des systèmes d'équations induits. Dans l'algorithme 8.1, les complexités dominantes sont T_{build} et T_{test}.

L'attaque de Dinur et Shamir [SD11], publiée à FSE 2011, n'impose pas la linéarité des $Q_{i,\mathcal{V}}$. Leur attaque utilise jusqu'à 24 variables, et des ensembles I et J tels que les polynômes $Q_{i,\mathcal{V}}$ soient

de degré au plus 3. La phase de résolution du système d'équations polynomiale est réalisée par recherche exhaustive. De manière surprenante, cette manière de procéder conduit à un meilleur compromis entre les différentes phases de l'algorithme 8.1.

Bibliographie

[Abe10] Masayuki Abe, éditeur. *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 de *Lecture Notes in Computer Science*. Springer, 2010. 198, 199

[AGK+10] Jean-Philippe Aumasson, Jian Guo, Simon Knellwolf, Krystian Matusiewicz et Willi Meier : Differential and Invertibility Properties of BLAKE. *Dans* Hong et Iwata [HI10], pages 318–332. 42

[AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier et Raphael C.-W. Phan : SHA-3 proposal BLAKE, Décembre 2010. Cf. `http://131002/blake`. 42

[AKK+10] Jean-Philippe Aumasson, Emilia Käsper, Lars Ramkilde Knudsen, Krystian Matusiewicz, Rune Ødegaard, Thomas Peyrin et Martin Schläffer : Distinguishers for the compression function and output transformation of Hamsi-256. *Dans ACISP*, volume 6168 de *LNCS*, pages 87–103. Springer, 2010. 179

[AM09] Jean-Philippe Aumasson et Willi Meier : Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009. 115, 179

[AMM09] Jean-Philippe Aumasson, Atefeh Mashatan et Willi Meier : More on Shabal's permutation. OFFICIAL COMMENT, 2009. 62, 66

[Ass10] Gilles Van Assche : A rotational distinguisher on Shabal's keyed permutation and its impact on the security proofs. Available online, 2010. 70

[Aum09] Jean-Philippe Aumasson : On the pseudorandomness of Hamsi. NIST mailing list (local link), 2009. 179

[BC04] Eli Biham et Rafi Chen : Near-Collisions of SHA-0. *Dans* Matthew K. Franklin, éditeur : *CRYPTO*, volume 3152 de *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004. 39

[BC10] Christina Boura et Anne Canteaut : Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak- and Hamsi-256. *Dans* Biryukov *et al.* [BGS11], pages 1–17. 54, 115, 179

[BCCM+08] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, María Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet et Marion Videau : SHABAL – A submission to Advanced Hash Standard. Submission to NIST, 2008. 47, 50, 61, 63, 73, 76, 119

[BCdC11] Christina Boura, Anne Canteaut et Christophe de Cannière : Higher Order Differential Properties on Keccak and Luffa. *Dans FSE*, 2011. 42, 115

[BCK96] Mihir Bellare, Ran Canetti et Hugo Krawczyk : Keying Hash Functions for Message Authentication. *Dans* Neal Koblitz, éditeur : *CRYPTO*, volume 1109 de *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996. 8

[BD06] Eli Biham et Orr Dunkelman : A Framework for Iterative Hash Functions : HAIFA. *Dans Proceedings of Second NIST Cryptographic Hash Workshop*, 2006. Cf. `http://www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`. 20, 42

[BDPA06] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Radiogatun, a belt-and-mill hash function. Presented at Second Cryptographic Hash Workshop, Santa Barbara, 24-25 août 2006. Cf. `http://radiogatun.noekeon.org/`. 57, 153, 159, 166, 169

[BDPA07] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Sponge Functions. presenté au ECRYPT Hash Workshop, Mai 2007. 19, 154

[BDPA08a] G. Bertoni, J. Daemen, M. Peeters et G. Van Assche : Keccak specifications. Submission to NIST, 2008. 42

[BDPA08b] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : On the Indifferentiability of the Sponge Construction. *Dans* Nigel P. Smart, éditeur : *EUROCRYPT*, volume 4965 de *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. 19, 24, 58, 154

[BDPA10] Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Duplexing the sponge : single-pass authenticated encryption and other applications. Presented at Second SHA-3 Conference, Santa Barbara, 23-24 août 2010. 154

[Ber08] Daniel J. Bernstein : ChaCha, a variant of Salsa20, Janvier 2008. Cf. `http://cr.yp.to/chacha.html`. 42

[Ber11] Daniel J. Bernstein : Second preimages for 6 (7 ? (8 ? ?)) rounds of Keccak ? NIST Hash Forum, 2011. Cf. `http://ehash.iaik.tugraz.at/wiki/Keccak`. 43

[BF08] Charles Bouillaguet et Pierre-Alain Fouque : Analysis of RadioGatun using Algebraic Techniques. *Dans* Liam Keliher Roberto Avanzi et Francesco Sica, éditeurs : *SAC*, Lecture Notes in Computer Science. Springer, 2008. 154

[BFL10] Charles Bouillaguet, Pierre-Alain Fouque et Gaëtan Leurent : Security Analysis of SIMD. *Dans* Biryukov *et al.* [BGS11], pages 351–368. 115, 144, 145

[BGS11] Alex Biryukov, Guang Gong et Douglas R. Stinson, éditeurs. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 de *Lecture Notes in Computer Science*. Springer, 2011. 195, 196, 197, 199

[BH05] Boaz Barak et Shai Halevi : A model and architecture for pseudo-random generation with applications to /dev/random. *Dans* Vijay Atluri, Catherine Meadows et Ari Juels, éditeurs : *ACM Conference on Computer and Communications Security*, pages 203–212. ACM, 2005. 10

[BKN09] Alex Biryukov, Dmitry Khovratovich et Ivica Nikolic : Distinguisher and Related-Key Attack on the Full AES-256. *Dans* Shai Halevi, éditeur : *CRYPTO*, volume 5677 de *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009. 154

[BR93] Mihir Bellare et Phillip Rogaway : Random Oracles are Practical : A Paradigm for Designing Efficient Protocols. *Dans ACM Conference on Computer and Communications Security*, pages 62–73, 1993. 22

[BR03] Paulo S. L. M. Barreto et Vincent Rijmen : The WHIRLPOOL hashing function, Mai 2003. Cf. `www.larc.usp.br/~pbarreto/WhirlpoolPage.html`. 29

[Bra90] Gilles Brassard, éditeur. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989,*

Proceedings, volume 435 de *Lecture Notes in Computer Science*. Springer, 1990. 197, 200

[BRS02] John Black, Phillip Rogaway et Thomas Shrimpton : Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. *Dans* Moti Yung, éditeur : *CRYPTO*, volume 2442 de *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002. 29

[BS92] Eli Biham et Adi Shamir : Differential Cryptanalysis of the Full 16-Round DES. *Dans* Ernest F. Brickell, éditeur : *CRYPTO*, volume 740 de *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992. 35

[CDGP93] Luc J. M. Claesen, Joan Daemen, Mark Genoe et G. Peeters : Subterranean : A 600 Mbit/Sec Cryptographic VLSI Chip. *Dans ICCD*, pages 610–613, 1993. 155

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud et Prashant Puniya : Merkle-Damgård Revisited : How to Construct a Hash Function. *Dans* Shoup [Sho05], pages 430–448. 18, 20, 22, 23, 58, 73, 74, 94, 95

[CJ98] Florent Chabaud et Antoine Joux : Differential Collisions in SHA-0. *Dans* Hugo Krawczyk, éditeur : *CRYPTO*, volume 1462 de *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998. 31, 35, 38

[CN08] Donghoon Chang et Mridul Nandi : Improved Indifferentiability Security Analysis of chopMD Hash Function. *Dans* Nyberg [Nyb08], pages 429–443. 18, 23, 74, 94

[Cra05] Ronald Cramer, éditeur. *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 de *Lecture Notes in Computer Science*. Springer, 2005. 200, 202

[CT10] Cagdas Calik et Meltem Sonmez Turan : Message Recovery and Pseudo-Preimage Attacks on the Compression Function of Hamsi-256. Cryptology ePrint Archive, Report 2010/057, 2010. urlhttp ://eprint.iacr.org/. 115, 179

[Dae95] Joan Daemen : Cipher and hash function design strategies based on linear and differential cryptanalysis. PhD thesis, Katholieke Universiteit Leuven, 1995. 155

[Dam89] Ivan Damgård : A Design Principle for Hash Functions. *Dans* Brassard [Bra90], pages 416–427. 13, 21

[dBB91] Bert den Boer et Antoon Bosselaers : An Attack on the Last Two Rounds of MD4. *Dans* Joan Feigenbaum, éditeur : *CRYPTO*, volume 576 de *Lecture Notes in Computer Science*, pages 194–203. Springer, 1991. 30

[dBB93] Bert den Boer et Antoon Bosselaers : Collisions for the Compression Function of MD5. *Dans EUROCRYPT*, pages 293–304, 1993. 30

[DC98] Joan Daemen et Craig S. K. Clapp : Fast Hashing and Stream Encryption with PANAMA. *Dans* Serge Vaudenay, éditeur : *FSE*, volume 1372 de *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998. 155

[Dea99] R.D. Dean : Formal aspects of mobile code security. PhD thesis, Princeton University, 1999. 17

[DGK10] Jérémie Detrey, Pierrick Gaudry et Karim Khalfallah : A Low-Area Yet Performant FPGA Implementation of Shabal. *Dans* Biryukov *et al.* [BGS11], pages 99–113. 57

[DH76] Whitfield Diffie et Martin E. Hellman : New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. 5

[DL11] Ming Duan et Xuajia Lai : Improved zero-sum distinguisher for full round Keccak-f permutation. Cryptology ePrint Archive, Report 2011/023, 2011. Cf. `http://eprint.iacr.org/`. 42

[Dob96] Hans Dobbertin : Cryptanalysis of MD4. *Dans* Dieter Gollmann, éditeur : *FSE*, volume 1039 de *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996. 30

[DR98] Joan Daemen et Vincent Rijmen : The Block Cipher Rijndael. *Dans* Jean-Jacques Quisquater et Bruce Schneier, éditeurs : *CARDIS*, volume 1820 de *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998. 4

[DRS09] Yevgeniy Dodis, Thomas Ristenpart et Thomas Shrimpton : Salvaging Merkle-Damgård for Practical Applications. *Dans* Joux [Jou09], pages 371–388. 24

[DS09] Itai Dinur et Adi Shamir : Cube Attacks on Tweakable Black Box Polynomials. *Dans EUROCRYPT*, pages 278–299, 2009. 174

[ECR] ECRYPT : eBASH, ECRYPT Benchmarking of All Submitted Hashes. Cf. `bench.cr.yp.to/ebash.html`. 56

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas et Jesse Walker : The Skein Hash Function Family, Octobre 2010. Cf. `http://www.schneier.com/skein.html`. 43

[FP09] Thomas Fuhr et Thomas Peyrin : Cryptanalysis of radiogatún. *Dans* Orr Dunkelman, éditeur : *FSE*, volume 5665 de *Lecture Notes in Computer Science*, pages 122–138. Springer, 2009. 153

[FS86] Amos Fiat et Adi Shamir : How to Prove Yourself : Practical Solutions to Identification and Signature Problems. *Dans* Andrew M. Odlyzko, éditeur : *CRYPTO*, volume 263 de *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. 22

[FT10] Julien Francq et Céline Thuillet : Unfolding Method for Shabal on Virtex-5 FPGAs : Concrete Results. Second SHA-3 conference, Santa Barbara, California, USA, August 23-24, 2010, 2010. 57

[Fuh10] Thomas Fuhr : Finding Second Preimages of Short Messages for Hamsi-256. *Dans* Abe [Abe10], pages 20–37. 173, 175

[Gam84] Taher El Gamal : A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *Dans CRYPTO*, pages 10–18, 1984. 5

[GHNS10] Xu Guo, Sinan Huang, Leyla Nazhandali et Patrick Schaumont : Fair and comprehensive evaluation of 14 second round SHA-3 ASIC implementations. Second SHA-3 conference, Santa Barbara, California, USA, August 23-24, 2010, 2010. 57

[GHR10] Kris Gaj, Ekawat Homsirikamol et Marcin Rogawski : Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. *Dans* Mangard et Standaert [MS10], pages 264–278. 57

[GKM+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer et Søren S. Thomsen : `grøstl-` a sha-3 candidate, Mars 2011. Cf. `http://www.groestl.info`. 42

[GLP08] Michael Gorski, Stefan Lucks et Thomas Peyrin : Slide Attacks on a Class of Hash Functions. *Dans* Josef Pieprzyk, éditeur : *ASIACRYPT*, volume 5350 de *Lecture Notes in Computer Science*, pages 143–160. Springer, 2008. 61

[GT10] Jian Guo et Søren S. Thomsen : Deterministic Differential Properties of the Compression Function of BMW. *Dans* Biryukov *et al.* [BGS11], pages 338–350. 115

[HI10] Seokhie Hong et Tetsu Iwata, éditeurs. *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 de *Lecture Notes in Computer Science*. Springer, 2010. 195, 199, 202

[HS08] Jonathan J. Hoch et Adi Shamir : On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. *Dans* Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir et Igor Walukiewicz, éditeurs : *ICALP (2)*, volume 5126 de *Lecture Notes in Computer Science*, pages 616–630. Springer, 2008. 98

[IS10] Takanori Isobe et Taizo Shirai : Low-weight Pseudo Collision Attack on Shabal and Preimage Attack on Reduced Shabal-512. Cryptology ePrint Archive, Report 2010/434, 2010. urlhttp ://eprint.iacr.org/. 68

[JL09] Li Ji et Xu Liangyu : Attacks on Round-Reduced BLAKE. Cryptology ePrint Archive, Report 2009/238, 2009. Cf. `http://eprint.iacr.org/`. 42

[Jou04] Antoine Joux : Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. *Dans CRYPTO*, pages 306–316, 2004. 15

[Jou09] Antoine Joux, éditeur. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 de *Lecture Notes in Computer Science*. Springer, 2009. 198, 202

[JP07] Antoine Joux et Thomas Peyrin : Hash Functions and the (Amplified) Boomerang Attack. *Dans* Alfred Menezes, éditeur : *CRYPTO*, volume 4622 de *Lecture Notes in Computer Science*, pages 244–263. Springer, 2007. 39

[Kal92] B. Kaliski : RFC 1319 : The MD2 Message Digest Algorithm, Avril 1992. Cf. `http://www.ietf.org`. 30

[KM05] Lars R. Knudsen et John Erik Mathiassen : Preimage and Collision Attacks on MD2. *Dans* Henri Gilbert et Helena Handschuh, éditeurs : *FSE*, volume 3557 de *Lecture Notes in Computer Science*, pages 255–267. Springer, 2005. 30

[KMMT10] Lars R. Knudsen, John Erik Mathiassen, Frédéric Muller et Søren S. Thomsen : Cryptanalysis of MD2. *J. Cryptology*, 23(1):72–90, 2010. 30

[KMNP10] Simon Knellwolf, Willi Meier et María Naya-Plasencia : Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. *Dans* Abe [Abe10], pages 130–145. 175

[KMT09] Lars R. Knudsen, Krystian Matusiewicz et Søren S. Thomsen : Observations on the Shabal keyed permutation. OFFICIAL COMMENT, 2009. 69, 115, 118

[KN10] Dmitry Khovratovich et Ivica Nikolic : Rotational Cryptanalysis of ARX. *Dans* Hong et Iwata [HI10], pages 333–346. 70

[KNR10] Dmitry Khovratovich, Ivica Nikolic et Christian Rechberger : Rotational Rebound Attacks on Reduced Skein. *Dans* Abe [Abe10], pages 1–19. 43, 54, 70

[KS05] John Kelsey et Bruce Schneier : Second Preimages on n-Bit Hash Functions for Much Less than Work. *Dans* Cramer [Cra05], pages 474–490. 16

[Küç09] Özgül Küçük : The Hash Function Hamsi. Submission to NIST (updated), 2009. 17, 173

[Leu08] Gaëtan Leurent : MD4 is Not One-Way. *Dans* Nyberg [Nyb08], pages 412–428. 30

[Leu09] Gaëtan Leurent : Communication personnelle, 2009. 67

[Lis06] Moses Liskov : Constructing an Ideal Hash Function from Weak Ideal Compression Functions. *Dans* Eli Biham et Amr M. Youssef, éditeurs : *Selected Areas in Cryptography*, volume 4356 de *Lecture Notes in Computer Science*, pages 358–375. Springer, 2006. 98

[LM92] Xuejia Lai et James L. Massey : Hash Function Based on Block Ciphers. *Dans EUROCRYPT*, pages 55–70, 1992. 191

[Luc05] Stefan Lucks : A Failure-Friendly Design Principle for Hash Functions. *Dans* Bimal K. Roy, éditeur : *ASIACRYPT*, volume 3788 de *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005. 18, 75

[Mer89] Ralph C. Merkle : One Way Hash Functions and DES. *Dans* Brassard [Bra90], pages 428–446. 13, 21

[MIO89] Shoji Miyaguchi, Masahiko Iwata et Kazuo Ohta : New 128-bit hash function. *Dans 4th International Joint Workshop on Computer Communications, Tokyo, Japan*, pages 279–288, July 13-15 1989. 29

[MMO85] S. M. Matyas, C. H. Meyer et J. Oseas : Generating Strong One-Way Functions With Cryptographic Algorithm, 1985. 29

[MP08] Stéphane Manuel et Thomas Peyrin : Collisions on SHA-0 in One Hour. *Dans* Nyberg [Nyb08], pages 16–35. 31

[MRH04] Ueli M. Maurer, Renato Renner et Clemens Holenstein : Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *Dans* Moni Naor, éditeur : *TCC*, volume 2951 de *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004. 22, 58, 77

[MS10] Stefan Mangard et François-Xavier Standaert, éditeurs. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 de *Lecture Notes in Computer Science*. Springer, 2010. 198, 202

[Mul04] Frédéric Muller : The MD2 Hash Function Is Not One-Way. *Dans* Pil Joong Lee, éditeur : *ASIACRYPT*, volume 3329 de *Lecture Notes in Computer Science*, pages 214–229. Springer, 2004. 30

[MVO96] A.J. Menezes, S.A. Vanstone et P.C. Van Oorschot : Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1996. 16

[Nik09] Ivica Nikolic : Near collisions for the compression function of hamsi-256. CRYPTO rump session, 2009. 179

[NIS95] FIPS 180-1 : Secure Hash Standard, Avril 1995. Cf. `http://csrc.nist.gov`. 31

[NIS99] FIPS 46-3 : Data Encryption Standard (DES), Octobre 1999. Cf. `http://csrc.nist.gov`. 27

[NIS01] FIPS 197 : Announcing the Advanced Encryption Standard, Novembre 2001. Cf. `http://csrc.nist.gov`. 4

[NIS02] FIPS 180-2 : Secure Hash Standard, Août 2002. Cf. `http://csrc.nist.gov`. 32

[NIS07a] NIST Special Publication 800-56A : Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised) , Mars 2007. Cf. `http://csrc.nist.gov`. 9

[NIS07b] NIST Special Publication 800-90 : Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised) , Mars 2007. Cf. `http://csrc.nist.gov`. 10

[NIS09] FIPS 186-3 : Digital Signature Standard (DSS), Juin 2009. Cf. `http://csrc.nist.gov`. 6

[Nov10] Peter Novotney : Distinguisher for Shabal's Permutation Function. Cryptology ePrint Archive, Report 2010/398, 2010. urlhttp ://eprint.iacr.org/. 68

[NP09] María Naya-Plasencia : *Chiffrements à flot et fonctions de hachage : conception et cryptanalyse*. Thèse de doctorat, Université Pierre et Marie Curie, Paris, 2009. 63, 66

[NP10] María Naya-Plasencia : How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607, 2010. Cf. `http://eprint.iacr.org/`. 42, 54

[Nyb08] Kaisa Nyberg, éditeur. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 de *Lecture Notes in Computer Science*. Springer, 2008. 197, 200

[Osv00] Dag Arne Osvik : Speeding up Serpent. *Dans AES Candidate Conference*, pages 317–329, 2000. 188

[Pey07] Thomas Peyrin : Cryptanalysis of Grindahl. *Dans ASIACRYPT*, pages 551–567, 2007. 166

[Pey10] Thomas Peyrin : Improved Differential Attacks for ECHO and Grøstl. *Dans* Tal Rabin, éditeur : *CRYPTO*, volume 6223 de *Lecture Notes in Computer Science*, pages 370–392. Springer, 2010. 42, 54

[PGV93] Bart Preneel, René Govaerts et Joos Vandewalle : Hash Functions Based on Block Ciphers : A Synthetic Approach. *Dans* Douglas R. Stinson, éditeur : *CRYPTO*, volume 773 de *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993. 28, 29

[RC97] N. Rogier et Pascal Chauvaud : MD2 Is not Secure without the Checksum Byte. *Des. Codes Cryptography*, 12(3):245–251, 1997. 30

[Riv92a] R. Rivest : RFC 1320 : The MD4 Message Digest Algorithm, Avril 1992. Cf. `http://www.ietf.org`. 30

[Riv92b] R. Rivest : RFC 1321 : The MD5 Message Digest Algorithm. April 1992, Avril 1992. Cf. `http://www.ietf.org`. 30

[RRPV01] Vincent Rijmen, Bart Van Rompay, Bart Preneel et Joos Vandewalle : Producing Collisions for PANAMA. *Dans FSE*, pages 37–51, 2001. 159

[RSA78] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman : A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. 5

[RSA02] PKCS# 1 v2.1 : RSA Cryptography Standard, Juin 2002. Cf. `http://www.rsa.com`. 6

[RSS11] Thomas Ristenpart, Hovav Shacham et Thomas Shrimpton : Careful with Composition : Limitations of the Indifferentiability Framework. *Dans* Kenneth G. Paterson, éditeur : *EUROCRYPT*, volume 6632 de *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011. 25

[RTV10] Vincent Rijmen, Deniz Toz et Kerem Varici : Rebound Attack on Reduced-Round Versions of JH. *Dans* Hong et Iwata [HI10], pages 286–303. 42

[SA09] Yu Sasaki et Kazumaro Aoki : Finding Preimages in Full MD5 Faster Than Exhaustive Search. *Dans* Joux [Jou09], pages 134–152. 30

[Sch11] Martin Schläffer : Updated Differential Analysis of `grøstl`, Janvier 2011. Cf. `http://www.groestl.info/`. 42

[SD11] Adi Shamir et Itai Dinur : An Algebraic Attack on Hamsi-256. *Dans FSE*, 2011. 175, 180, 187, 193

[Sho04] Victor Shoup : Sequences of games : a tool for taming complexity in security proofs, 2004. shoup@cs.nyu.edu 13166 received 30 Nov 2004, last revised 18 Jan 2006. 78, 79

[Sho05] Victor Shoup, éditeur. *Advances in Cryptology - CRYPTO 2005 : 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 de *Lecture Notes in Computer Science*. Springer, 2005. 197, 202

[Vie07] Michael Vielhaber : AIDA Algebraic IV Differential Attack Breaking One.Fivium by AIDA an Algebraic IV Differential Attack, 2007. 174

[WBG10] Christian Wenzel-Benner et Jens Gräf : XBX : eXternal Benchmarking eXtension for the SUPERCOP Crypto Benchmarking Framework. *Dans* Mangard et Standaert [MS10], pages 294–305. 57

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen et Xiuyuan Yu : Cryptanalysis of the Hash Functions MD4 and RIPEMD. *Dans* Cramer [Cra05], pages 1–18. 30

[Wu11] Hongjun Wu : The Hash Function JH, Janvier 2011. Cf. `http://www3.ntu.edu.sg/home/wuhj/research/jh`. 42

[WY05] Xiaoyun Wang et Hongbo Yu : How to Break MD5 and Other Hash Functions. *Dans* Cramer [Cra05], pages 19–35. 30

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin et Hongbo Yu : Finding Collisions in the Full SHA-1. *Dans* Shoup [Sho05], pages 17–36. 38

[WYY05b] Xiaoyun Wang, Hongbo Yu et Yiqun Lisa Yin : Efficient Collision Search Attacks on SHA-0. *Dans* Shoup [Sho05], pages 1–16. 31

[YW10] Hongbo Yu et Xiaoyun Wang : Cryptanalysis of the Compression Function of SIMD. Cryptology ePrint Archive, Report 2010/304, 2010. urlhttp ://eprint.iacr.org/. 54

Annexe A

Calcul des bornes d'indifférentiabilité du mode du Chapitre 5

A.1 Majoration de la probabilité d'occurrence des évènements Echec_1 et Echec_2.

Nous traitons ici simultanément le cas de l'évènement $\mathsf{Echec}_1 \vee \mathsf{Echec}_2$ lorsque $\mathcal{F}$ est une fonction aléatoire et celui de l'évènement $\mathsf{Echec}_1 \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4$ lorsque $\mathcal{F}$ est une permutation paramétrée aléatoire. Plus précisément, nous montrons le Lemme 10 suivant.

Lemme 10 *Soit $\mathcal{S}$ le simulateur défini pour le Jeu 4 dans la section 5.3 (resp. 5.4). Soit $\mathcal{D}$ un attaquant interagissant avec le système défini au Jeu 4 tel que $\mathcal{S}$ reçoit au plus N requêtes à $\mathcal{F}$ (resp. N requêtes à $\mathcal{F}$ ou $\mathcal{F}^{-1}$). Si $\mathcal{F}$ est une fonction aléatoire, nous avons*

$$\Pr\left[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4]\right] \leq \frac{N(N+1)}{2 \times 2^{\ell_a+\ell_h}} ,$$

et si $\mathcal{F}$ est une permutation paramétrée, nous avons

$$\Pr\left[\mathsf{Echec}_1[4] \vee \mathsf{Echec}_2[4] \vee \mathsf{Echec}_4[4]\right] \leq \frac{N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)} .$$

Cette borne peut être interprétée comme un majorant de la probabilité qu'un attaquant parvienne à générer des collisions internes sur des nœuds ayant un chemin dans le graphe, ou à créer un chemin depuis l'origine sans calculer ses arcs dans l'ordre. Ce type de propriété lui permet de différencier la construction d'un oracle aléatoire.

Démonstration : Pour tout q de $\{1, \ldots, N\}$, nous notons $X(q)$ et $E(q)$ le contenu des ensembles X et E au moment où le simulateur reçoit la $q^{ème}$ requête à $\mathcal{F}$ ou $\mathcal{F}^{-1}$. Chaque requête entraîne l'insertion d'au plus deux nœuds et une arc dans le graphe, donc $|X(q)| \leq 2q-1$ et $|E(q)| \leq q-1$. Nous notons maintenant $\mathsf{Echec}_i(q)$ l'évènement correspondant à l'indetification de l'échec Echec_i au cours du traitement de la $q^{ème}$ requête.
Nous commençons par traiter le cas où $\mathcal{F}$ est une fonction aléatoire. Nous cherchons donc à majorer la probabilité $\Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q)\right]$ que l'évènement $\mathsf{Echec}_1 \vee \mathsf{Echec}_2$ se produise au cours du traitement de la $q^{ème}$ requête au cours du Jeu 4. Soit $u = (M, A, B, C)$ cette requête. Nous considérons

$$D(u) = \{(A', B', C) \mid (A', B') \leftarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h}\} .$$

$D(u)$ représente l'ensemble des valeurs possibles du nœud y tel que $x \xrightarrow{M} y$ est l'arc créé lors de la $q^{ème}$ requête. Soit alors $\tilde{y} = (\tilde{A}, \tilde{B}, \tilde{C})$ un élément quelconque de $\mathcal{X}$. En choisissant y uniformément sur $D(u)$, nous avons

$$\begin{aligned} \Pr_{y \leftarrow D(u)}\left[y = \tilde{y}\right] &= \Pr_{y \leftarrow D(u)}\left[(A', B') = (\tilde{A}, \tilde{B})\right] \delta\left[C = \tilde{C}\right] \\ &\leq 2^{-(\ell_a+\ell_h)} \end{aligned}$$

où $\delta[\mathsf{Ev}]$ vaut 1 si l'évènement Ev est réalisé et 0 sinon. Pour toute entrée u, nous avons

$$\max_{\tilde{y}\in\mathcal{X}} \Pr_{y\leftarrow D(u)}[y=\tilde{y}] \leq 2^{-(\ell_a+\ell_h)}.$$

Nous majorons maintenant le nombre de valeurs de (A', B', C) qui génèrent un cas d'échec du simulateur. Par définition de Echec_1, il s'agit :
– des nœuds ayant un chemin dans $\mathcal{G}$ (Echec_1),
– des points de départ des arcs du graphe lorsqu'ils n'ont pas de chemin utile (Echec_2).
Pour chaque requête, au plus un tel nœud est inséré dans le graphe, qui en contient initialement un : x^0. Il y a donc au plus q valeurs de (A', B', C) qui conduisent à Echec_1, d'où

$$\begin{aligned}\Pr[\mathsf{Echec}_1(q)\vee\mathsf{Echec}_2(q)] &\leq \max_u \Pr_{y\leftarrow D(u)}[\exists\,\tilde{y}\in X(q) : y=\tilde{y}]\\ &\leq 2^{-(\ell_a+\ell_h)}q.\end{aligned}$$

Nous nous intéressons maintenant au cas où $\mathcal{F}$ est une permutation paramétrée. Supposons que la $q^{ème}$ requête concerne $\mathcal{F}$. Contrairement au cas précédent, $\mathcal{F}_{M,C}(A,B)$ n'est pas toujours tiré uniformément sur $\{0,1\}^{\ell_a+\ell_h}$, mais sur $\{0,1\}^{\ell_a+\ell_h}\setminus U_{C,M}$. La taille de $U_{C,M}$ étant au plus $q-1$, nous transposons le raisonnement précédent pour montrer que

$$\begin{aligned}\Pr[\mathsf{Echec}_1(q)\vee\mathsf{Echec}_2(q)] &\leq \frac{q}{2^{\ell_a+\ell_h}-(q-1)}\\ &\leq \frac{q}{2^{\ell_a+\ell_h}-N}\end{aligned}$$

Supposons maintenant que la $q^{ème}$ requête reçue implique un appel à $\mathcal{F}^{-1}$. Soit alors $v=(M,A,B,C)$ cette requête. L'arc créé lors de cette requête est de la forme $x \xrightarrow{M} v$ avec

$$x \leftarrow D'(v) = \{\mathsf{Insert}^{-1}[M](A,B,C) \mid (A,B)\leftarrow\{0,1\}^{\ell_a}\times\{0,1\}^{\ell_h}\setminus V_{C,M}\}\,.$$

Pour tout $\tilde{x}=(\tilde{A},\tilde{B},\tilde{C})\in\mathcal{X}$ fixé, nous avons alors

$$\Pr_{x\leftarrow D'(v)}[x=\tilde{x}] = \Pr_{x\leftarrow D'(v)}\left[\mathsf{Insert}^{-1}[M](A,B,C)=\tilde{x}\right]$$

où les probabilités sont prises sur un choix uniforme de $x\leftarrow D'(v)$. $\mathsf{Insert}[M]$ étant une bijection, les $2^{\ell_a+\ell_h}-|V_{C,M}|$ valeurs de (A,B,C) choisies avec probabilité $1/(2^{\ell_a+\ell_h}-|V_{C,M}|)$ conduisent à $2^{\ell_a+\ell_h}-|V_{C,M}|$ valeurs de $\mathsf{Insert}^{-1}[M](A,B,C)$ distinctes. On en déduit que

$$\Pr_{x\leftarrow D'(v)}\left[\mathsf{Insert}^{-1}[M](A,B,C)=\tilde{x}\right] \leq \frac{1}{2^{\ell_a+\ell_h}-(q-1)}$$

d'où

$$\begin{aligned}\Pr[\mathsf{Echec}_4(q)] &\leq \frac{q}{2^{\ell_a+\ell_h}-(q-1)}\\ &\leq \frac{q}{2^{\ell_a+\ell_h}-N}\end{aligned}$$

Nous pouvons maintenant conclure la preuve du Lemme 10. Lorsque $\mathcal{F}$ est une fonction aléatoire,

$$\begin{aligned}\Pr\left[\mathsf{Echec}_1 \vee \mathsf{Echec}_2\right] &= \sum_{q=1}^{N} \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q)\right] \\ &\leq 2^{-(\ell_a+\ell_h)} \sum_{q=1}^{N} q \\ &\leq 2^{-(\ell_a+\ell_h)} \left(N(N+1)/2\right) \\ &\leq \frac{N(N+1)}{2 \times 2^{\ell_a+\ell_h}}.\end{aligned}$$

Lorsque $\mathcal{F}$ est une permutation aléatoire, notons F_q l'évènement vérifié si la $q^{ème}$ requête est une requête à $\mathcal{F}$. Nous avons alors

$$\begin{aligned}\Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) \vee \mathsf{Echec}_4(q)\right] &= \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4(q) | F_q\right] \Pr\left[F_q\right] \\ &+ \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) \vee \mathsf{Echec}_4(q) | \overline{F_q}\right] \Pr\left[\overline{F_q}\right] \\ &\leq \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) | F_q\right] \Pr\left[F_q\right] \\ &\quad + \Pr\left[\mathsf{Echec}_4(q) | \overline{F_q}\right] \Pr\left[\overline{F_q}\right] \\ &\leq \frac{q}{2^{\ell_a+\ell_h} - N} \left(\Pr\left[F_q\right] + \Pr\left[\overline{F_q}\right]\right) \\ &\leq \frac{q}{2^{\ell_a+\ell_h} - N}.\end{aligned}$$

Nous en déduisons

$$\begin{aligned}\Pr\left[\mathsf{Echec}_1 \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4\right] &= \sum_{q=1}^{N} \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) \vee \mathsf{Echec}_4(q)\right] \\ &\leq \frac{\sum_{q=1}^{N} q}{2^{\ell_a+\ell_h} - N} \\ &\leq \frac{N(N+1)}{2 \times (2^{\ell_a+\ell_h} - N)}\ .\end{aligned}$$

□

Lemme 11 *Soit $\mathcal{S}_4$ le simulateur défini pour le Jeu 4 dans la section 5.4, pour la construction avec compteur et tours à blanc définie en Figure 5.18 . Soit $\mathcal{D}$ un attaquant interagissant avec le système défini au Jeu 4 tel que $\mathcal{S}$ reçoit au plus N requêtes à $\mathcal{F}$ ou $\mathcal{F}^{-1}$. Nous avons*

$$\Pr\left[\mathsf{Echec}_1 \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4\right] \leq \frac{2N^2}{2^{\ell_a+\ell_h} - 2N}\ ,$$

Démonstration : Le raisonnement de la preuve du lemme précédent se transpose aisément ici. Les seules différences sont les suivantes :

– A chaque requête, au plus deux nœuds ayant un chemin dans le graphe sont créés. Supposons que l'arc inséré soit $x \xrightarrow{M} y$. Si x a un w-chemin initial, $y \oplus w$ et $y \oplus (w+1)$ ont un chemin utile. Si x a un $w, n_f - 2$-chemin final (se terminant par M si $n_f > 2$), $x \xrightarrow{M} y$ et $y \oplus w \xrightarrow{M} z$ sont insérés. $y \oplus w$ et $z \oplus w$ ont un chemin utile. Dans les autres cas, un chemin est créé pour au plus un nœud.

– De même, le test conduisant à Echec_1 est effectué sur au plus deux nœuds (au lieu d'un seul) : $y \oplus w$ et $y \oplus (w+1)$ si x a un w-chemin initial, $y \oplus w$, $z \oplus w$ si x a un $w, n_f - 2$-chemin final.

En appliquant le même raisonnement que ci-dessus, nous en déduisons les relations suivantes (en tenant compte du fait que le graphe contient moins de $2N$ arcs) :

$$\begin{aligned} \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q)\right] &\leq \frac{2 \times (2q-1)}{2^{\ell_a+\ell_h} - 2N} \\ \Pr\left[\mathsf{Echec}_4(q)\right] &\leq \frac{2q-1}{2^{\ell_a+\ell_h} - 2N} \end{aligned}$$

$\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q)$ et $\mathsf{Echec}_4(q)$ concernant des types de requêtes distinctes, nous avons donc

$$\Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) \vee \mathsf{Echec}_4(q)\right] \leq \frac{4q-2}{2^{\ell_a+\ell_h} - 2N}.$$

En sommant sur q, nous obtenons

$$\begin{aligned} \Pr\left[\mathsf{Echec}_1 \vee \mathsf{Echec}_2 \vee \mathsf{Echec}_4\right] &= \sum_{q=1}^{N} \Pr\left[\mathsf{Echec}_1(q) \vee \mathsf{Echec}_2(q) \vee \mathsf{Echec}_4(q)\right] \\ &\leq \frac{\sum_{q=1}^{N} 4q-2}{2^{\ell_a+\ell_h} - 2N} \\ &\leq \frac{4N(N+1)/2 - 2N}{2^{\ell_a+\ell_h} - 2N} \\ &\leq \frac{2N^2}{2^{\ell_a+\ell_h} - 2N}\ . \end{aligned}$$

□

A.2 Majoration de la probabilité d'occurrence des évènements Echec_3 et Echec_5.

Nous évaluons maintenant un majorant de la probabilité d'occcurence des évènements Echec_3 et Echec_5 au Jeu 4 des preuves des sections 5.3 et 5.4. Nous commençons par montrer le lemme suivant.

Lemme 12 *Lorsque $\mathcal{F}$ est une fonction aléatoire, la probabilité d'occurrence de l'évènement Echec_3 après N appels à $\mathcal{F}$ satisfait*

$$\Pr\left[\mathsf{Echec}_3\right] \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right]\ ,$$

où t-Coll est l'évènement tel qu'il existe au moins t valeurs identiques dans un ensemble de N éléments tirés indépendamment et uniformément dans $\{0,1\}^{\ell_h}$. Lorsque $\mathcal{F}$ est une permutation aléatoire, la probabilité d'occurrence de l'évènement $\mathsf{Echec}_3 \vee \mathsf{Echec}_5$ après N appels à $\mathcal{F}$ ou à $\mathcal{F}^{-1}$ satisfait

$$\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5\right] \leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right].$$

Démonstration : Nous considérons le graphe construit par notre simulateur après la réponse à N requêtes. Soit r un entier fixé et r-Coll l'évènement correspondant à la situation suivante : il existe r arcs $x_i \stackrel{M_i}{\rightarrow} y_i$ dans $E_{\mathcal{I}}$, pour $1 \leq i \leq r$, telles que tous les y_i, $1 \leq i \leq r$, ont des parties B identiques. De plus, nous définissons l'évènement t-MaxColl qui est vrai si « t est la plus grande valeur de r telle que r-Coll est vérifié ». Les évènements t-MaxColl sont disjoints et l'un d'entre eux se produit forcément, et

$$\Pr\left[t\text{-MaxColl}\right] = \Pr\left[t\text{-Coll}\right] - \Pr\left[(t+1)\text{-Coll}\right],$$

d'où

$$\begin{aligned}\Pr\left[\mathsf{Echec}_3\right] &= \sum_{t=1}^{N} \Pr\left[t\text{-MaxColl}\right] \Pr\left[\mathsf{Echec}_3 \mid t\text{-MaxColl}\right] \\ &= \sum_{t=1}^{N} \big(\Pr\left[t\text{-Coll}\right] - \Pr\left[(t+1)\text{-Coll}\right]\big) \Pr\left[\mathsf{Echec}_3 \mid t\text{-MaxColl}\right]\end{aligned}$$

dans le cas où $\mathcal{F}$ est une fonction aléatoire, et

$$\begin{aligned}\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5\right] &= \sum_{t=1}^{N} \Pr\left[t\text{-MaxColl}\right] \Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5 \mid t\text{-MaxColl}\right] \\ &= \sum_{t=1}^{N} \big(\Pr\left[t\text{-Coll}\right] - \Pr\left[(t+1)\text{-Coll}\right]\big) \Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5 \mid t\text{-MaxColl}\right]\end{aligned}$$

En supposant que $(t+1)$-Coll ne se produit pas après N requêtes, nous calculons $\Pr\left[\mathsf{Echec}_3(q) | t\text{-MaxColl}\right]$, défini comme étant la probabilité pour que Echec_3 se produise lors du $q^{ème}$ appel au simulateur, pour tout $q < N$ correspondant à une requête provenant de $\mathcal{D}$.
Soit $x \in \mathcal{X}$ l'état défini par $x = \mathsf{Insert}^{-1}[M](A, B, C)$, où (M, A, B, C) est la $q^{ème}$ requête. L'état x a un chemin dans $\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{I}}$ s'il existe une séquence d'états dans le graphe $x_1, \ldots, x_k$ et une séquence de blocs de message $M_1, \ldots, M_k$ tels que

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

Supposons que x n'ait pas de chemin dans $\mathcal{G}_{\mathcal{D}}$. Dans ce cas, le dernier arc $x_{k-1} \stackrel{M_k}{\rightarrow} x$ n'appartient pas à $\mathcal{G}_{\mathcal{D}}$. En effet, soit i le plus grand entier tel que $1 \leq i \leq k$ et $x_{i-1} \stackrel{M_i}{\rightarrow} x_i$ n'est pas dans $E_{\mathcal{D}}$. Cela signifie que x_i a un chemin dans $\mathcal{G}_{\mathcal{D}} \cup \mathcal{G}_{\mathcal{I}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$. D'après la Propriété 2, ce chemin était défini au moment où x_i a été ajouté au graphe. De ce fait, si $i < k$, ce chemin était présent au moment où $x_i \stackrel{M_{i+1}}{\rightarrow} x_{i+1}$ a été ajoutée au graphe. Deux cas peuvent alors se présenter. Si l'arc $x_i \stackrel{M_{i+1}}{\rightarrow} x_{i+1}$ a été créé par un appel à $\mathcal{F}$, Echec_3 s'est produit lors de l'appel correspondant. Si l'arc $x_i \stackrel{M_{i+1}}{\rightarrow} x_{i+1}$ a été introduit lors d'un appel à $\mathcal{F}^{-1}$, Echec_4 s'est produit lors de l'appel correspondant.
Notons maintenant $(A, B, C) = \mathsf{Insert}^{-1}[M](S)$ où (M, S) est la $q^{ème}$ requête à $\mathcal{F}$, et majorons la probabilité qu'elle provoque Echec_3. L'étude ci-dessus montre que $(A, B, C) \in X_{\mathcal{I}}$ et $(A, B, C) \notin X_{\mathcal{D}}$. Comme on suppose t-MaxColl réalisé, il y a au plus t valeurs de (A', C') telles que (A', B, C') soit dans $X_{\mathcal{I}}$. Pour chacune de ces valeurs, A' a été tiré uniformément parmi 2^{ℓ_a} valeurs et est inconnu de l'attaquant. On a donc

$$\Pr\left[(A, B, C) = (A', B, C')\right] \leq 2^{-\ell_a},$$

d'où

$$\Pr\left[\mathsf{Echec}_3(q) \mid t\text{-}\mathsf{MaxColl}\right] \leq 2^{-\ell_a} t.$$

De manière similaire, nous calculons maintenant $\Pr\left[\mathsf{Echec}_5(q)|t\text{-}\mathsf{MaxColl}\right]$. Cet évènement intervient si lors de la $q^{ème}$ requête, qui correspond à une simulation de $\mathcal{F}_{C,M}^{-1}(A,B)$, il existe un arc de la forme $x \xrightarrow{M} (A,B,C)$ dans $E_{\mathcal{I}}$ mais pas dans $E_{\mathcal{D}}$. Cela implique que (A,B,C) est dans $X_{\mathcal{I}}$. Il a donc forcément un chemin dans $\mathcal{G}_{\mathcal{I}}$. (A,B,C) ne peut donc pas être dans $E_{\mathcal{D}}$, car il y aurait été inséré comme extrémité d'un autre arc que $x \xrightarrow{M} (A,B,C)$, ce qui implique une occurrence précédente d'un évènement d'échec.
(A,B,C) est donc dans $X_{\mathcal{I}}$ mais pas dans $X_{\mathcal{D}}$. En appliquant le raisonnement ci-dessus, nous en déduisons

$$\Pr\left[\mathsf{Echec}_5(q) \mid t\text{-}\mathsf{MaxColl}\right] \leq 2^{-\ell_a} t.$$

Lorsque $\mathcal{F}$ est une fonction aléatoire, nous avons

$$\Pr\left[\mathsf{Echec}_3|t\text{-}\mathsf{MaxColl}\right] \leq 2^{-\ell_a} tN,$$

et enfin

$$\begin{aligned}
\Pr\left[\mathsf{Echec}_3\right] &\leq \sum_{t=1}^{N} (\Pr\left[t\text{-}\mathsf{Coll}\right] - \Pr\left[(t+1)\text{-}\mathsf{Coll}\right]) Nt2^{-\ell_a} \\
&\leq 2^{-\ell_a} N \sum_{t=1}^{N} t(\Pr\left[t\text{-}\mathsf{Coll}\right] - \Pr\left[(t+1)\text{-}\mathsf{Coll}\right]) \\
&\leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] \ , \qquad (A.1)
\end{aligned}$$

la probabilité d'occurrence d'une $N+1$ collision étant nulle.
Lorsque $\mathcal{F}$ est une permutation paramétrée, nous avons

$$\begin{aligned}
\Pr\left[\mathsf{Echec}_3(q) \vee \mathsf{Echec}_5(q)|t\text{-}\mathsf{MaxColl}\right] &= \Pr\left[\mathsf{Echec}_3(q) \vee \mathsf{Echec}_5(q)|F_q \wedge t\text{-}\mathsf{MaxColl}\right] \Pr\left[F_q|t\text{-}\mathsf{MaxColl}\right] \\
&\quad + \Pr\left[\mathsf{Echec}_3(q) \vee \mathsf{Echec}_5(q)|\overline{F_q} \wedge t\text{-}\mathsf{MaxColl}\right] \Pr\left[\overline{F_q}|t\text{-}\mathsf{MaxColl}\right] \\
&\leq \Pr\left[\mathsf{Echec}_3(q)|F_q \wedge t\text{-}\mathsf{MaxColl}\right] \Pr\left[F_q|t\text{-}\mathsf{MaxColl}\right] \\
&\quad + \Pr\left[\mathsf{Echec}_5(q)|\overline{F_q} \wedge t\text{-}\mathsf{MaxColl}\right] \Pr\left[\overline{F_q}|t\text{-}\mathsf{MaxColl}\right] \\
&\leq 2^{-\ell_a} t(\Pr\left[F_q|t\text{-}\mathsf{MaxColl}\right] + \Pr\left[\overline{F_q}|t\text{-}\mathsf{MaxColl}\right]) \\
&\leq 2^{-\ell_a} t.
\end{aligned}$$

On en déduit

$$\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5|t\text{-}\mathsf{MaxColl}\right] \leq 2^{-\ell_a} tN,$$

et enfin

$$\begin{aligned}
\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5\right] &\leq \sum_{t=1}^{N} (\Pr\left[t\text{-}\mathsf{Coll}\right] - \Pr\left[(t+1)\text{-}\mathsf{Coll}\right]) Nt2^{-\ell_a} \\
&\leq 2^{-\ell_a} N \sum_{t=1}^{N} t(\Pr\left[t\text{-}\mathsf{Coll}\right] - \Pr\left[(t+1)\text{-}\mathsf{Coll}\right]) \\
&\leq 2^{-\ell_a} N \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}\right] \ , \qquad (A.2)
\end{aligned}$$

la probabilité d'occurrence d'une $N+1$ collision étant nulle.

□

Afin d'exprimer ces bornes plus précisément, nous montrons maintenant le Lemme 13, qui constitue un résultat général concernant la probabilité de t-collisions.

Lemme 13 *Pour t dans $\{1,\ldots,N\}$, soit t-$\mathsf{Coll}(N,k)$ l'évènement vérifié s'il existe au moins t valeurs identiques dans un ensemble de N éléments tirés aléatoirement dans un ensemble $\mathcal{E}$ suivant une distribution D satisfaisant*

$$\Pr_{x\leftarrow D}[x=\tilde{x}] \leq 2^{-k}, \text{ pour tout } \tilde{x}\in\mathcal{E}\ .$$

Nous avons

$$\sum_{t=1}^{N}\Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] \leq 2k+(1+e)N2^{-k}\ ,$$

où $e=\exp(1)\approx 2.71828$. De plus, si $N\leq 2^v$ pour un entier v donné, nous avons

$$\sum_{t=1}^{N}\Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] \leq \left\lceil \frac{k+v-\log_2(\sqrt{2\pi})}{k-v-\log_2 e}\right\rceil\ .$$

Démonstration :

Borne Générale. Pour tout $t>1$, nous avons

$$\begin{aligned}
\Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] &\leq \binom{N}{t}\max_{\{i_1,\ldots,i_t\}\subset\{1,\ldots,N\}}\Pr_{x_{i_1},\ldots,x_{i_t}\leftarrow D}\left[x_{i_1}=x_{i_2}=\ldots=x_{i_t}\right]\\
&\leq \binom{N}{t}2^{-k(t-1)}\\
&\leq \frac{N^t}{\sqrt{2\pi t}}\left(\frac{e}{t}\right)^t 2^{-k(t-1)}\ .
\end{aligned}$$

Nous posons maintenant

$$t_0 = 2k+\frac{eN}{2^k}\ ,$$

et nous montrons que pour tout $t\geq t_0$ et tout N,

$$\frac{N^t}{\sqrt{2\pi t}}\left(\frac{e}{t}\right)^t 2^{-k(t-1)} \leq 2^{-k}. \tag{A.3}$$

L'inégalité (A.3) est en effet équivalente à

$$t\left(\log_2 t+k-\log_2 N-\log_2 e\right)+\frac{1}{2}\log_2 t \geq 2k-\frac{1}{2}\log_2(2\pi).$$

Pour tout $t\geq t_0$, nous avons

$$\log_2 t \geq \log_2\left(\frac{eN}{2^k}\right)+\log_2\left(1+\frac{k2^{k+1}}{eN}\right) = \log_2 e+\log_2 N-k+\log_2\left(1+\frac{k2^{k+1}}{eN}\right)\ ,$$

d'où

$$\begin{aligned} t\left(\log_2 t + k - \log_2 N - \log_2 e\right) &\geq 2k\left(1 + \frac{eN}{k2^{k+1}}\right)\log_2\left(1 + \frac{k2^{k+1}}{eN}\right) \\ &\geq 2k \end{aligned}$$

Cette dernière inégalité est déduite du fait que pour x positif,

$$f : x \mapsto \left(1 + \frac{1}{x}\right)\log_2(1+x)$$

est une fonction croissante et $\lim_{x\to 0} f(x) = 1/\ln 2 > 1$. Comme $t \geq 2k \geq 1$,

$$t\left(\log_2 t + k - \log_2 N - \log_2 e\right) + \frac{1}{2}\log_2 t \geq 2k\ .$$

Nous en déduisons

$$\begin{aligned} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] &= \sum_{t=1}^{\lceil t_0\rceil - 1} \Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] + \sum_{t=\lceil t_0\rceil}^{N} \Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] \\ &\leq \lceil t_0\rceil - 1 + N2^{-k}\ , \end{aligned}$$

ce qui implique que

$$\sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] \leq 2k + (1+e)N2^{-k}\ .$$

Borne raffinée lorsque $N \leq 2^v$. Lorsque $N \leq 2^v$, une borne plus fine peut être établie en définissant

$$t_1 = \frac{k + v - \log_2(\sqrt{2\pi})}{k - v - \log_2 e}.$$

Pour tout $t \geq \lceil t_1\rceil$ et tout $N \leq 2^v$, nous montrons la relation suivante

$$\Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] \leq \frac{1}{N}.$$

Il suffit pour cela de montrer que :

$$t\left(\log_2 t + k - \log_2 N - \log_2 e\right) \geq \log_2 N + k - \log_2(\sqrt{2\pi}).$$

Pour tout $t \geq \lceil t_1\rceil$ et tout $N \leq 2^v$

$$\begin{aligned} t\left(\log_2 t + k - \log_2 N - \log_2 e\right) &\geq t(k - \log_2 N - \log_2 e) \\ &\geq t\left(k - v - \log_2 e\right) \\ &\geq k + v - \log_2(\sqrt{2\pi}) \\ &\geq \log_2 N + k - \log_2(\sqrt{2\pi}). \end{aligned}$$

Nous en déduisons que pour tout $N \leq 2^v$,

$$\begin{aligned} \sum_{t=1}^{N} \Pr\left[t\text{-}\mathsf{Coll}(N,k)\right] &\leq (\lceil t_1\rceil - 1) + \sum_{t=\lceil t_1\rceil}^{N} \frac{1}{N} \\ &\leq (\lceil t_1\rceil - 1) + 1 \\ &\leq \lceil t_1\rceil\ . \end{aligned}$$

□

Construction de Shabal. Dans le cas de la construction de Shabal étudiée en Section 5.5, nous obtenons le lemme suivant.

Lemme 14 *Dans le cas de la construction de Shabal, la probabilité d'occurrence de l'évènement* $\mathsf{Echec}_3 \vee \mathsf{Echec}_5$ *au Jeu* 4 *après* N *appels à* $\mathcal{F}$ *ou à* $\mathcal{F}^{-1}$ *satisfait*

$$\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5\right] \leq N^2 \max\left(2P_c 2^{-\ell_a}, \frac{1}{2^{\ell_a+\ell_h}-2N}\right),$$

où

$$P_c = \max_{M,c,C,U_{M,c}} \left(\Pr\left[\mathsf{Insert}_C[M](a,b,c) = C | (a,b) \leftarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h} \setminus U_{M,c}\right]\right).$$

Démonstration : Considérons tout d'abord l'évènement $\mathsf{Echec}_3(q)$, en notant (M,A,B,C) la $q^{ème}$ requête reçue par $\mathcal{S}$ et en supposant qu'elle s'adresse à $\mathcal{F}$ et qu'elle est émise par $\mathcal{D}$. Notons également $u = \mathsf{Insert}^{-1}[M](A,B,C)$.
Soit $x \xrightarrow{M} y$ un arc tel que x a un chemin utile μ dans $\mathcal{G}_{\mathcal{I}}$ mais pas dans $\mathcal{G}_{\mathcal{D}}$. Si x a un chemin complet dans $\mathcal{G}_{\mathcal{I}}$, alors $\mu||M$ n'est pas un chemin utile (du fait de la discontinuité sur le compteur dans les tours à blanc), et Echec3 ne se produit pas. Du point de vue de l'attaquant, les parties A et B de x suivent alors une distribution aléatoire. Nous avons donc $\Pr[x = u] \leq 1/(2^{\ell_a+\ell_h}-2N)$, d'où

$$\Pr\left[\mathsf{Echec}_3(q)\right] \leq \frac{2q-1}{2^{\ell_a+\ell_h}-2N}\ .$$

Nous cherchons maintenant à majorer la probabilité d'occurrence de Echec_5 dans le Jeu 3. Cet évènement correspond à l'émission d'une requête de la forme (M,A,B,C) à $\mathcal{F}^{-1}$ telle qu'il existe un arc $x \xrightarrow{M} (A,B,C)$ dans $E_{\mathcal{I}}$ mais pas dans $E_{\mathcal{D}}$. Plutôt que d'évaluer la probabilité d'occurrence de Echec_5 lors de chaque requête, nous majorons la probabilité d'occurrence de cet évènement dans tout le Jeu pour un arc $x \xrightarrow{M} (A,B,C)$ donné. Nous notons $\mathsf{Echec}_5(M,y)$ cet évènement.
Nous traitons alors les deux cas suivants.
Si $y = (A,B,C)$ n'a pas de chemin complet dans le graphe, α et β sont inconnus et aléatoires du point de vue de l'attaquant. Nous avons donc, pour toute requête (M',A',B',C') à $\mathcal{F}^{-1}$,

$$\Pr\left[(M',A',B',C') = (M,A,B,C)\right] \leq \frac{1}{2^{\ell_a+\ell_h}-2N}.$$

Nous avons alors

$$\Pr\left[\mathsf{Echec}_4(M,A,B,C)\right] \leq \frac{N}{2^{\ell_a+\ell_h}-2N}.$$

Supposons maintenant que $y = (A,B,C)$ ait un chemin complet dans $G_{\mathcal{I}}$. Au moment où Echec_5 se produit, x n'a pas de chemin utile dans $\mathcal{G}_{\mathcal{D}}$. En effet, dans le cas contraire, l'absence d'évènement d'échecs préalable implique que x n'a pu être inséré dans $X_{\mathcal{D}}$ que si l'attaquant a reconstruit tout le chemin de $x \oplus w$ par des appels à $\mathcal{F}$. Or, x a un $w,(n_f-1)$ chemin final dans le graphe, donc la simulation insère automatiquement l'arc $x \xrightarrow{M} y$ après l'ajout de $z \xrightarrow{M} x$ dans $E_{\mathcal{D}}$. y a alors un chemin dans $\mathcal{G}_{\mathcal{D}}$, ce qui est contraire à l'hypothèse.
Du point de vue de l'attaquant, B est connu, A est inconnu et aléatoire, et C suit la distribution de $\mathsf{Insert}[M](a,b,c)$ où c est connu et (a,b) sont uniformément distribués sur $\{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h} \setminus U_{M,c}$. Soit alors

$$P_c = \max_{M,c,C,U_{M,c}} \left(\Pr\left[\mathsf{Insert}_C[M](a,b,c) = C | (a,b) \leftarrow \{0,1\}^{\ell_a} \times \{0,1\}^{\ell_h} \setminus U_{M,c}\right]\right),$$

où $U_{M,c}$ est pris sur toutes les valeurs possibles de $E_{\mathcal{I}} \cup E_{\mathcal{D}}$. Nous avons donc $|U_{M,c}| \leq 2N$.
Nous considérons maintenant toutes les requêtes (M_i, A_i, B_i, C_i) émises à $\mathcal{F}^{-1}$ après l'insertion de $x \xrightarrow{M} y$ dans le graphe. Lors de chaque requête, nous pouvons majorer la probabilité que $C_i = C$ par P_c.
Pour toute valeur de M, B, C nous notons maintenant $n(\mu, \beta, \gamma)$ le nombre de requêtes M_i, A_i, B_i, C_i à $\mathcal{F}^{-1}$ émises par $\mathcal{D}$ vérifiant $M_i = \mu$, $B_i = \beta$ et $C_i = \gamma$. Nous notons également $m(\mu, \beta, \gamma)$ le nombres d'arcs de la forme $x \xrightarrow{\mu} (\alpha, \beta, \gamma)$ présents dans $E_{\mathcal{I}} \cup E_{\mathcal{D}}$ à la fin du Jeu et qui n'ont pas été obtenus par une requête à $\mathcal{F}^{-1}$. Si $C = \gamma$, la probabilité qu'une des requêtes à $\mathcal{F}^{-1}$ coincide avec M, A, B, C est alors au plus $n(M, B, \gamma)/(2^{\ell_a} - r(M, B, \gamma))$, puisque la valeur de A est uniforme sur les $(2^{\ell_a} - r(M, B, \gamma)$ valeurs restantes et que $n(M, B, \gamma)$ valeurs de A_i sont testées. En sommant sur la valeur de γ, nous avons donc

$$\begin{aligned} \Pr\left[\mathsf{Echec}_5(M, A, B, C)\right] &\leq \sum_{\gamma \in \{0,1\}^{\ell_c}} \Pr\left[C' = \gamma\right] \frac{n(M, B, \gamma)}{2^{\ell_a} - r(M, B, \gamma)} \\ &\leq P_c \sum_{\gamma \in \{0,1\}^{\ell_c}} \frac{n(M, B, \gamma)}{2^{\ell_a} - r(M, B, \gamma)} \end{aligned}$$

Le graphe final contenant au plus $2N$ arcs, nous avons

$$\sum_{\gamma \in \{0,1\}^{\ell_c}} n(M, B, \gamma) + r(M, B, \gamma) \leq 2N.$$

D'autre part, $n(M, B, \gamma)/(2^{\ell_a} - r(M, B, \gamma))$ est inférieur à 1, donc

$$\frac{n(M, B, \gamma)}{2^{\ell_a} - r(M, B, \gamma)} \leq \frac{n(M, B, \gamma) + r(M, B, \gamma}{2^{\ell_a}}.$$

Nous en déduisons :

$$\begin{aligned} \Pr\left[\mathsf{Echec}_5(M, A, B, C)\right] &\leq P_c \sum_{\gamma \in \{0,1\}^{\ell_c}} \frac{n(M, B, \gamma) + r(M, B, \gamma)}{2^{\ell_a}} \\ &\leq 2N P_c 2^{-\ell_a}. \end{aligned}$$

En sommant sur les valeurs de (M, A, B, C) nous avons :

$$\Pr\left[\mathsf{Echec}_5\right] \leq N^2 \max(2P_c 2^{-\ell_a}, \frac{1}{2^{\ell_a + \ell_h} - 2N})$$

Echec_3 et Echec_5 étant provoqués par des types de requêtes différents, nous avons donc

$$\Pr\left[\mathsf{Echec}_3 \vee \mathsf{Echec}_5\right] \leq N^2 \max(2P_c 2^{-\ell_a}, \frac{1}{2^{\ell_a + \ell_h} - 2N}).$$

□

Annexe B

Chemin différentiel et exemple de collision pour RadioGatún

B.1 Le chemin différentiel

Nous donnons ici une représentation du chemin différentiel utilisé pour générer des collisions pour RadioGatún[w] avec une complexité équivalente à 2^{11w} évaluations de la permutation. Pour chaque étape, la table B.1 contient une représentation de la différence sur l'état interne après l'insertion de message, et une représentation de la différence sur l'état interne après la permutation.

Le chemin différentiel étant de longueur 143 blocs, nous utilisons une notation hexadécimale pour représenter les valeurs des différences sur l'état interne. Chaque valeur de la différence sur le mill est écrite comme $\Sigma_{i=0}^{18}\delta M_i 2^i$, où $\delta M_i = 1$ si le mot i du mill contient une différence et $\delta M_i = 0$ sinon. De la même manière, nous représentons les différences sur le belt comme $\Sigma_{i=0}^{12}\delta B_{i,j}2^i$. Les représentations des lignes du belt sont données dans l'ordre $B_{\_,0}, B_{\_,1}, B_{\_,2}$.

Nous donnons également une estimation de la complexité de la recherche à chaque étape, selon le calcul effectué dans la section 7.5.2. Dans la colonne `Noeuds`, nous donnons une estimation de $log_{2^w}(A^i)$, qui est le logarithme du nombre de noeuds estimé qu'un attaquant doit parcourir à la profondeur i. Dans la colonne `Complexité`, nous estimons $log_{2^w}(\frac{C^i A^i}{2^{-K^i}})$, qui est le logarithme de la complexité estimée (en appels à la permutation) de la complexité de la recherche à la profondeur i.

Pour les grandes valeurs de w, nous pouvons approximer la complexité de la recherche par la complexité du parcours des profondeurs les plus coûteuses. Nous avons ici 4 profondeurs pour lesquelles la complexité est de 2^{11w}, de ce fait la complexité de la recherche peut être approximée par

$$T = 4 \times 2^{11w}.$$

Input Belt	Input Mill	Output Belt	Output Mill	Noeuds	Complexité
0000 0000 0000	00000	0000 0000 0000	00000	1.000	4.000
0000 0000 0000	00000	0000 0000 0000	00000	4.000	6.000
0001 0000 0000	10000	0002 0000 0000	20034	4.000	6.000
0002 0001 0000	00034	0014 0026 0000	7a065	2.000	1.678
0014 0027 0000	5a065	0028 006a 0040	30000	0.000	3.000
0029 006b 0040	00000	0052 00d6 0080	00000	0.000	3.000
0052 00d6 0080	00000	00a4 01ac 0100	00000	1.000	4.000
00a4 01ac 0100	00000	0148 0358 0200	00000	4.000	7.000
0148 0359 0200	20000	0290 06b2 0400	19000	5.000	8.000

Suite sur la page suivante

Suite de la page précédente

0291 06b3 0400	29000	0522 0d66 1800	71800	4.000	6.000
0523 0d67 1801	01800	0a46 12ce 0003	6c000	3.000	4.193
0a47 12cf 0002	1c000	148e 059f 0004	40034	2.000	4.000
148e 059f 0005	00034	090d 0b1a 000a	30000	1.000	4.000
090c 0b1b 000a	00000	1218 1636 0014	00000	4.000	7.000
1218 1636 0014	00000	0431 0c6d 0028	06000	7.000	7.193
0430 0c6d 0028	16000	0860 18da 0050	20034	5.000	7.000
0860 18db 0050	00034	10d0 1193 00a0	30464	3.000	2.000
10d0 1192 00a0	10464	05a1 0301 0100	20000	0.000	3.000
05a1 0300 0100	00000	0b42 0600 0200	00000	0.000	3.000
0b42 0600 0200	00000	1684 0c00 0400	00000	3.000	6.000
1684 0c00 0400	00000	0d09 1800 0800	02000	5.000	6.193
0d08 1801 0800	32000	1a10 1003 1000	7a440	4.000	6.000
1a11 1002 1001	0a440	1023 0005 0043	00020	0.000	3.000
1023 0005 0043	00020	0047 002a 0086	30000	0.000	3.000
0046 002b 0086	00000	008c 0056 010c	00000	0.000	3.000
008c 0056 010c	00000	0118 00ac 0218	00000	0.000	3.000
0118 00ac 0218	00000	0230 0158 0430	00000	2.000	5.000
0230 0158 0430	00000	0460 02b0 0860	00000	5.000	8.000
0460 02b0 0860	00000	08c0 0560 10c0	00000	7.000	10.000
08c0 0561 10c1	60000	1180 0ac2 0183	12390	5.000	7.000
1180 0ac2 0183	12390	0391 1484 0106	51400	1.000	2.000
0390 1484 0107	01400	0320 0909 120e	60000	0.000	3.000
0320 0909 120f	20000	0640 1212 041f	11000	2.000	5.000
0641 1212 041f	01000	0c82 0425 183e	60000	2.000	5.000
0c82 0424 183f	00000	1904 0848 107f	08000	4.000	7.000
1904 0849 107f	28000	1209 1092 00ff	30446	4.000	5.000
1209 1093 00ff	10446	0011 0123 01be	20000	0.000	3.000
0011 0122 01be	00000	0022 0244 037c	00000	1.000	4.000
0022 0244 037c	00000	0044 0488 06f8	00000	4.000	7.000
0044 0488 06f8	00000	0088 0910 0df0	00000	7.000	9.000
0089 0910 0df0	10000	0112 1220 1be0	20034	7.000	9.000
0112 1220 1be0	20034	0234 0465 17c1	21400	6.000	8.000
0234 0465 17c1	21400	0068 08ca 1f83	75000	4.000	6.000
0068 08ca 1f82	35000	00d0 1194 0f05	11000	2.000	5.000
00d1 1194 0f05	01000	01a2 0329 0e0a	60000	2.000	5.000
01a2 0328 0e0b	00000	0344 0650 1c16	00000	5.000	8.000
0344 0650 1c16	00000	0688 0ca0 182d	08000	7.000	10.000
0688 0ca1 182d	28000	0d10 1942 105b	11000	9.000	11.000
0d11 1943 105b	21000	1a22 1287 10b7	71000	8.000	11.000
1a23 1286 10b6	01000	1447 050d 116d	66800	7.000	7.193
1447 050c 116c	06800	088f 0218 02d9	2a006	4.000	6.000
088e 0219 02d9	1a006	111e 0436 05b2	60038	4.000	6.000

Suite sur la page suivante

Suite de la page précédente

111e 0437 05b3	00038	022d 084e 0b6e	30000	2.000	5.000
022c 084f 0b6e	00000	0458 109e 16dc	00000	5.000	8.000
0458 109e 16dc	00000	08b0 013d 0db9	0c000	8.000	9.193
08b1 013d 0db9	1c000	1162 027a 1b72	20034	7.000	9.000
1162 027b 1b72	00034	02d5 04d2 16e5	70001	6.000	9.000
02d4 04d3 16e4	00001	05a8 09a6 0dc9	40001	8.000	11.000
05a8 09a6 0dc9	40001	0b50 134c 1b92	51001	8.000	10.000
0b50 134d 1b92	71001	16a0 069b 0725	00035	4.000	7.000
16a0 069b 0725	00035	0d51 0d12 0e4a	30000	2.000	5.000
0d50 0d13 0e4a	00000	1aa0 1a26 1c94	00000	4.000	7.000
1aa0 1a26 1c94	00000	1541 144d 1929	0e000	7.000	7.193
1540 144c 1929	3e000	0a81 0899 1253	70200	6.000	9.000
0a80 0899 1252	20200	1500 1132 06a5	01028	3.000	6.000
1500 1132 06a5	01028	0a01 0245 1d42	50000	1.000	4.000
0a00 0245 1d43	00000	1400 048a 1a87	08000	3.000	6.000
1401 048b 1a87	38000	0803 0916 150f	10200	5.000	8.000
0802 0917 150f	20200	1004 122e 081f	01028	2.000	5.000
1004 122e 081f	01028	0009 047d 0036	50000	0.000	3.000
0008 047d 0037	00000	0010 08fa 006e	00000	2.000	5.000
0010 08fa 006e	00000	0020 11f4 00dc	00000	4.000	7.000
0020 11f5 00dd	60000	0040 03eb 01ba	041a2	4.000	6.000
0041 03eb 01ba	141a2	0000 06f6 0374	10000	0.000	3.000
0001 06f6 0374	00000	0002 0dec 06e8	00000	0.000	3.000
0002 0dec 06e8	00000	0004 1bd8 0dd0	00000	3.000	6.000
0004 1bd8 0dd0	00000	0008 17b1 1ba0	04000	6.000	7.193
0009 17b0 1ba0	34000	0012 0f61 1741	15000	6.000	8.000
0012 0f60 1741	35000	0024 1ec0 0e83	11000	4.000	6.000
0024 1ec1 1e83	31000	0048 1d83 1d07	71000	2.000	5.000
0049 1d82 0d06	01000	0092 1b05 1a0c	60000	2.000	5.000
0092 1b04 0a0d	00000	0124 1609 141a	04000	5.000	6.193
0125 1608 141a	34000	024a 0c11 0835	71000	5.000	8.000
024b 0c10 0834	01000	0496 1820 1068	64000	5.000	6.193
0496 1821 0069	04000	092c 1043 00d2	24006	4.000	4.678
092c 1042 00d3	44006	1258 0085 01a6	00038	3.000	4.000
125a 0081 01a6	00038	04b5 0102 034c	30000	2.000	5.000
04a4 0123 0344	00000	0948 0246 0688	00000	5.000	8.000
0948 0246 0688	00000	1290 048c 0d10	00000	8.000	10.000
1291 048d 0d10	30000	0523 091a 1a20	3b034	6.000	8.000
0522 091a 1a20	2b034	0a44 1234 1441	41400	3.000	4.000
0a54 1210 0440	01400	14a8 0421 0880	60000	2.000	5.000
10a8 0420 1881	00000	0151 0840 1103	0a000	4.000	7.000
0150 0841 1103	3a000	02a0 1082 0207	11000	5.000	8.000
02a1 1082 0207	01000	0542 0105 040e	60000	4.000	7.000

Suite sur la page suivante

Suite de la page précédente

0542 0104 140f	00000	0a84 0208 081f	08000	6.000	9.000
0a85 0208 081f	18000	150a 0410 103e	20034	7.000	9.000
150a 0411 103e	00034	0a15 0822 007d	70001	6.000	9.000
0a04 0807 007c	00001	1408 100e 00f8	51001	7.000	9.000
1409 100f 00f8	61001	0813 001f 01f0	30201	6.000	9.000
0812 001f 11f1	60201	1024 003e 03e3	4002a	2.000	5.000
1024 003e 01e2	0002a	0049 007c 03c4	30000	0.000	3.000
004a 005d 03cc	00000	0094 00ba 0798	00000	0.000	3.000
0094 00ba 0798	00000	0128 0174 0f30	00000	2.000	5.000
0128 0174 0f30	00000	0250 02e8 1e60	00000	5.000	8.000
0250 02e8 1e60	00000	04a0 05d0 1cc1	08000	7.000	10.000
04a1 05d1 1cc1	38000	0942 0ba2 1983	31006	7.000	10.000
0943 0ba3 1983	01006	1286 1746 1307	20444	3.000	3.000
1285 1743 0307	10444	050b 0e87 060e	20000	1.000	4.000
010b 0e82 064e	00000	0216 1d04 0c9c	00000	3.000	6.000
0216 1d04 0c9c	00000	042c 1a09 1938	04000	6.000	7.193
042c 1a08 1938	24000	0858 1411 1271	51034	4.000	7.000
0859 1411 1270	01034	10b2 0823 04e1	10020	1.000	3.000
10a2 0806 14e1	30020	0145 100c 09c3	21000	1.000	4.000
0145 102d 09c3	01000	028a 005b 1386	60000	0.000	3.000
028a 005a 0387	00000	0514 00b4 070e	00000	2.000	5.000
0514 00b4 070e	00000	0a28 0168 0e1c	00000	5.000	8.000
0a28 0168 0e1c	00000	1450 02d0 1c38	00000	8.000	10.000
1451 02d1 1c38	30000	08a3 05a2 1871	10200	9.000	11.000
08a3 05a2 1871	10200	1146 0b44 10e3	18028	5.000	8.000
1146 0b44 12e2	58028	028d 1688 05c5	01d40	4.000	5.000
028d 16a8 05cd	01d40	051a 0d51 0b9a	60000	0.000	3.000
011a 0450 1bdb	00000	0234 08a0 17b7	08000	0.000	3.000
0234 08a1 17b7	28000	0468 1142 0f6f	11000	1.000	3.000
0469 1142 0f6f	01000	08d2 0285 1ede	60000	1.000	4.000
08d2 0284 0edf	00000	11a4 0508 1dbe	00000	4.000	7.000
11a4 0508 1dbe	00000	0349 0a10 1b7d	0a000	6.000	9.000
0348 0a11 1b7d	3a000	0690 1422 16fb	11000	7.000	10.000
0691 1422 16fb	01000	0d22 0845 0df7	68000	5.000	8.000
0d22 0844 1df6	08000	1a44 1088 1bed	0b440	5.000	6.000
1a44 1088 1bec	4b440	1489 0111 17d9	50028	2.000	5.000
1088 0111 0798	00028	0111 0222 0f30	30000	1.000	4.000
0110 0203 0f38	00000	0220 0406 1e70	00000	4.000	7.000
0220 0406 1e70	00000	0440 080c 1ce1	08000	6.000	9.000
0440 080d 1ce1	28000	0880 101a 19c3	11032	5.000	7.000
0880 101a 19c2	51032	1100 0035 1385	7002a	0.000	2.000
1113 0014 0385	4002a	0227 0028 070a	30000	0.000	3.000
0224 0009 0702	00000	0448 0012 0e04	00000	1.000	4.000

Suite sur la page suivante

					Suite de la page précédente
0448 0012 0e04	00000	0890 0024 1c08	00000	4.000	7.000
0890 0024 1c08	00000	1120 0048 1811	08000	5.000	8.000
1120 0048 1810	48000	0241 0090 1021	105e2	5.000	6.000
0241 0090 1020	505e2	0482 0120 0041	40000	0.000	3.000
0000 0000 0000	00000	0000 0000 0000	00000	0.000	3.000

Table B.1 – Chemin différentiel symétrique pour une recherche de collisions sur RadioGatún

B.2 Collision pour RadioGatún[2]

Nous montrons maintenant une collision générée suivant le chemin différentiel B.1. La collision que nous avons trouvé concerne la version RadioGatún[2], soit celle basée sur l'utilisation de registres de 2 bits. Nous pouvons aisément vérifier qu'elle respecte le chemin différentiel B.1. Nous représentons les mots des messages en utilisant des valeurs entre 0 et 3, qui représentent les valeurs possibles des registres de 2 bits. Le chemin différentiel, ainsi que des statistiques sur la recherche effective de collisions, sont également données. Elles sont à comparer au nombre estimé de noeuds à parcourir à chaque étape.

Afin d'assurer un nombre minimal de points de départ pour la recherche, nous utilisons un préfixe commun de 5 blocs. Les deux messages qui collisionnent sont décrit dans la table B.2

Step i	M_0	M_1	$Noeuds$	$log_4(Noeuds)$
-5	330	330	16	2.000
-4	000	000	16	2.000
-3	000	000	16	2.000
-2	000	000	16	2.000
-1	000	000	16	2.000
0	113	113	16	2.000
1	311	311	1014	4.993
2	012	312	974	4.964
3	012	022	57	2.916
4	112	122	1	0.000
5	300	030	1	0.000
6	202	202	4	1.000
7	020	020	227	3.913
8	302	332	915	4.919
9	233	103	245	3.968
10	030	303	57	2.916
11	030	303	13	1.850
12	000	003	1	0.000
13	223	113	1	0.000
14	222	222	59	2.941
15	220	120	4	1.000
16	111	121	1	0.000
17	000	030	1	0.000

Suite sur la page suivante

Suite de la page précédente

18	010	020	1	0.000
19	031	031	5	1.161
20	001	001	69	3.054
21	033	303	18	2.085
22	020	313	1	0.000
23	000	000	1	0.000
24	000	330	1	0.000
25	222	222	1	0.000
26	103	103	43	2.713
27	110	110	2738	5.709
28	312	312	43959	7.712
29	231	202	2793	5.724
30	321	321	16	2.000
31	102	201	2	0.500
32	012	011	22	2.230
33	322	022	22	2.230
34	023	010	358	4.242
35	323	313	313	4.145
36	232	202	1	0.000
37	001	031	11	1.730
38	023	023	657	4.680
39	032	032	42041	7.680
40	220	120	42301	7.684
41	130	130	10299	6.665
42	103	103	611	4.628
43	203	200	42	2.696
44	003	303	37	2.605
45	200	233	2353	5.600
46	232	232	37597	7.599
47	023	013	601697	9.599
48	011	321	150451	8.599
49	222	111	37874	7.604
50	222	211	588	4.600
51	133	203	589	4.601
52	110	123	29	2.429
53	211	121	1798	5.406
54	031	031	115031	8.406
55	232	132	28707	7.405
56	122	112	6956	6.382
57	033	300	110762	8.379
58	122	122	110814	8.379
59	021	011	389	4.302
60	202	202	21	2.196
61	302	032	323	4.168
62	003	003	20644	7.167
63	120	210	5110	6.160
64	003	300	81	3.170
65	300	300	6	1.292
66	203	100	73	3.095

Suite sur la page suivante

Suite de la page précédente

67	133	203	1136	5.075
68	021	311	17	2.044
69	302	302	2	0.500
70	311	012	100	3.322
71	101	101	1583	5.314
72	031	002	1731	5.379
73	200	100	1	0.000
74	003	303	3	0.792
75	013	013	177	3.734
76	231	231	11317	6.733
77	032	302	11369	6.736
78	312	322	706	4.732
79	002	032	45	2.746
80	202	131	33	2.522
81	131	102	2083	5.512
82	331	001	2105	5.520
83	122	211	2088	5.514
84	201	232	505	4.490
85	333	300	123	3.471
86	301	301	33	2.522
87	032	302	2068	5.507
88	230	230	132333	8.507
89	031	301	8132	6.495
90	220	120	117	3.435
91	012	011	33	2.522
92	130	103	525	4.518
93	312	022	2068	5.507
94	100	200	578	4.587
95	020	013	9209	6.584
96	322	022	37022	7.588
97	222	212	9150	6.580
98	220	113	37059	7.589
99	201	131	9453	6.603
100	012	311	40	2.661
101	000	003	1	0.000
102	201	131	1	0.000
103	200	200	19	2.124
104	010	010	1155	5.087
105	230	230	18501	7.088
106	130	200	18326	7.081
107	310	020	60	2.953
108	330	000	6	1.292
109	201	231	84	3.196
110	103	103	5331	6.190
111	130	100	306	4.129
112	210	113	6	1.292
113	102	132	8	1.500
114	001	031	1	0.000
115	200	233	17	2.044

Suite sur la page suivante

				Suite de la page précédente
116	321	321	1027	5.002
117	112	112	65692	8.002
118	110	220	263409	9.003
119	232	232	1087	5.043
120	223	220	309	4.136
121	010	010	1	0.000
122	301	332	1	0.000
123	213	223	3	0.792
124	000	300	3	0.792
125	133	100	129	3.506
126	123	123	2007	5.485
127	323	013	7965	6.480
128	222	122	469	4.437
129	331	302	487	4.464
130	132	131	9	1.585
131	103	200	4	1.000
132	021	311	242	3.959
133	012	012	3825	5.951
134	330	300	914	4.918
135	201	202	2	0.500
136	100	230	1	0.000
137	203	133	2	0.500
138	321	321	115	3.423
139	013	013	447	4.402
140	332	331	480	4.453
141	020	023	1	0.000
142	000	003	1	0.000

Table B.2 – Exemple de collision pour RadioGatún[2]