UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

ECOLE DOCTORALE SCIENCES TECHNOLOGIES ET SANTÉ (547)

T H È S E

pour obtenir le titre de

Docteur en Sciences de l'Université de Reims Champagne-Ardenne

Discipline : INFORMATIQUE

Présentée par

Florian LEGENDRE

Exploitation de la logique propositionnelle pour la résolution des problèmes cryptographiques

Thèse co-dirigée par Michaël KRAJECKI et Gilles DEQUEN

préparée à l'Université de Reims Champagne-Ardenne

soutenue publiquement le 30 Juin 2014

Jury :

Président :	Bertrand MAZURE	-	Professeur à l'Université d'Artois
Rapporteur :	Serge VAUDENAY	-	Professeur à l'École Polytechnique Fédérale de Lausanne
Directeurs :	Michaël KRAJECKI	-	Professeur à l'Université de Reims Champagne-Ardenne
	Gilles DEQUEN	-	Professeur à l'Université de Picardie Jules Verne
Examinateurs :	Stéphane MANUEL	-	Enseignant-Chercheur à l'École Polytechnique Féminine
	Valérie ROUAT	-	Chercheur à DGA Maîtrise de l'Information
	Daniel SINGER	-	Maître de Conférence à l'Université de Lorraine

Préambule

Cette thèse s'est déroulée au sein du Centre de Recherche en STIC (EA 3804) de l'Université de Reims Champagne-Ardenne et du Laboratoire Modélisation, Information et Systèmes (EA 4290) de l'Université de Picardie Jules Verne. Les adresses de ces établissements sont données ci-dessous.

CReSTIC	Laboratoire MIS
UFR Sciences Exactes et Naturelles Moulin de la Housse, BP 1039 51687 Reims CEDEX 2 FRANCE Tél. : 03.26.91.33.89	UFR Sciences 33 rue Saint Leu 80039 Amiens CEDEX 1 FRANCE Tél. : 03 22 82 59 05

La thèse a été financée à 50% par la région Champagne-Ardenne et à 50% par la Direction Générale de l'Armement.

Remerciements

J'aimerai adresser les premiers remerciements à mes deux directeurs de thèse. Tout d'abord, Michaël Krajecki, professeur à l'Université de Reims Champagne-Ardenne pour son soutien permanent, la confiance qu'il m'a accordée tout au long de la thèse et les précieux conseils qui m'ont accompagnés durant 3 ans. Merci à Gilles Dequen, Professeur à l'Université de Picardie Jules Verne pour avoir partagé son précieux temps à répondre à mes nombreuses questions, pour les échanges toujours instructifs et bien sûr pour les nombreux cafés pris dans son bureau en discutant, entre autres, de recherche. Enfin, je les remercie une nouvelle fois tous les deux pour leur bonne humeur, leur sincère sympathie et leur humanisme.

Un grand merci à Bertrand Mazure et Serge Vaudenay qui m'ont fait le grand honneur d'accepter de rapporter ma thèse. Leurs nombreux conseils m'ont grandement aidé à peaufiner ce document. Les questions ont toujours été constructives et m'ont aussi permis d'avancer encore plus loin dans la compréhension de mes travaux et de ce qui gravite autour.

Merci également à Stéphane Manuel et Daniel Singer pour s'être montrés intéressés par mes recherches et pour avoir accepté de les examiner. Enfin, un grand merci à Valérie Rouat pour l'intérêt porté sur nos travaux depuis le commencement et les échanges que j'ai eu avec elle et son équipe.

Je remercie également les laboratoires MIS et CReSTIC pour leur accueil toujours chaleureux ; les secrétaires, Valérie, Marion, Ida et Géraldine, et les membres enseignants-chercheurs qui m'ont toujours dégagé beaucoup de sympathie. Merci également aux doctorants et jeunes docteurs - trop nombreux pour être tous nommés - que j'ai pu croiser au quotidien, les échanges et les moments de convivialité que nous avons pu partager ensemble. Une mention spéciale pour les quelques parties de football qui ont enrobées de plaisir nos journées d'été.

Merci aux membres du Département Mathématiques, Mécanique et Informatique de l'URCA avec qui j'ai eu la chance de collaborer. Je leur suis reconnaissant des efforts réalisés pour être me mettre le plus à l'aise possible dans mes enseignements. En particulier, merci à Christophe Jaillet pour son aide au quotidien.

Je tiens aussi à remercier l'équipe du Centre de Calcul de Champagne-Ardenne ROMEO, Arnaud Renard et Hervé Deleau, pour les nombreux services rendus entre les moments de convivialité. Merci aussi à Pascal Vander-Swalmen pour son amitié, son aide et son aiguillage dès que j'en avais besoin.

Merci également à ma famille et à ma belle-famille qui m'ont beaucoup apporté et en particulier mes parents Daniel et Yolande pour les sacrifices qu'ils ont faits pour la réussite de leurs enfants. Merci à mon frère David et mes sœurs Brunehilde et Ludivine.

Merci aussi à mes amis qui, en dehors du contexte professionnel, ont souvent fait preuve de compréhension à mon égard et ont patiemment attendu jusqu'ici pour voir leurs noms apparaître. Sans aucun ordre, l'AS Glisy, la team Cuigy, Gaëtan, Guilhem, Guillaume, Mathieu, Matthieu, Paolo, Luss, Maribelle et Benjamin, Maxime, Teddy, une nouvelle fois Pascal, Zhu, Thomas et Bénédicte.

Enfin pour finir, merci à Delphine qui m'a accompagné au quotidien, encouragé et offert sans concession toute l'attention dont j'avais besoin pour avancer. Notre amour est précieux.

« La poésie est présente dans la musique, dans la photographie, voire dans l'art culinaire ; partout où il s'agit de percer l'opacité des choses pour en faire jaillir l'essence à nos yeux. Partout où ce qui est en jeu c'est la révélation du monde. [...] Tout m'est bon pour peu qu'il lève un coin de voile sur le ciel et nous procure un frisson d'immortalité. Peu importe ce qu'on pourra vous dire, les mots et les idées peuvent changer le monde. »

Le cercle des poètes disparus

Résumé

La démocratisation des ordinateurs, des téléphones portables et surtout de l'Internet a considérablement révolutionné le monde de la communication. Les besoins en matière de cryptographie sont donc plus nombreux et la nécessité de vérifier la sûreté des algorithmes de chiffrement est vitale. Cette thèse s'intéresse à l'étude d'une nouvelle cryptanalyse, appelée cryptanalyse logique, qui repose sur l'utilisation de la logique propositionnelle pour exprimer et résoudre des problèmes cryptographiques. Plus particulièrement, les travaux présentés ici portent sur une certaine catégorie d'algorithmes utilisés dans les protocoles d'authentification et d'intégrité de données qu'on appelle fonctions de hachage cryptographiques. La cryptanalyse logique est une cryptanalyse algébrique particulière où l'expression du problème cryptographique est faite à travers le problème de satisfaisabilité, plus couramment appelé problème SAT. Il s'agit d'un problème combinatoire de décision qui est central en théorie de la complexité et qui a permis le développement de solveurs très performants pour la résolution de problèmes issus du monde industriel ou académique. Les travaux présentés dans ce document sont le fruit d'une exploration aux confins de la satisfaisabilité et de la cryptanalyse, permettant d'exhiber de nouveaux résultats et des méthodes innovantes pour l'affaiblissement de fonctions cryptographiques.

Un premier point concerne l'aspect modélisation d'un problème cryptographique en un problème SAT où nous présentons les règles qui permettent de décrire de façon simple les opérations de base constituant un algorithme cryptographique. Une partie est consacrée à la réduction des formules logiques ainsi générées et montre comment une expertise en satisfaisabilité peut enrichir la connaissance associée à un problème. Par ailleurs, nous présentons aussi plusieurs façons d'utiliser cette modélisation fine pour produire une nouvelle connaissance probabilistique sur les données du problème. Ces informations pertinentes permettent en outre d'améliorer les deux principaux points d'une attaque par cryptanalyse logique, à savoir la modélisation et la résolution. Les travaux mettent aussi en avant une faiblesse statistique sur l'utilisation de constantes de ronde.

Un second point traite des attaques en pratique. Dans ce cadre, la recherche de pré-image pour les fonctions de hachage les plus couramment utilisées consiste à donner une certaine mesure de la résistance de ces fonctions à la cryptanalyse logique. À cela s'ajoutent plusieurs attaques pour la recherche de collisions et notamment la translation de l'attaque de Stevens [Stevens 2012a] sur la fonction de hachage MD5 dans le cadre de la logique propositionnelle. Enfin, notons que ces travaux portant sur des attaques de pré-images ou de collisions dans le contexte d'une cryptanalyse logique portent un nouveau regard sur les problèmes considérés.

Mots-clés : Cryptanalyse logique, satisfaisabilité, fonction de hachage cryptographique, MD5, SHA-1, SHA-3

Exploitation of propositional logic for solving cryptographic problems

Abstract

Democratization of increasingly high-performance digital technologies and especially the Internet has considerably changed the world of communication. Consequently, needs in cryptography are more and more numerous and the necessity of verifying the security of cipher algorithms is essential.

This thesis deals with a new cryptanalysis, called logical cryptanalysis, which is based on the use of logical formalism to express and solve cryptographic problems. More precisely, works presented here focuses on a particular category of ciphers, called cryptographic hash functions, used in authentication and data integrity protocols. Logical cryptanalysis is a specific algebraic cryptanalysis where the expression of the cryptographic problem is done through the satisfiabilty problem, fluently called SAT problem. It consists in a combinatorial problem of decision which is central in complexity theory. In the past years, works led by the scientific community have allowed to develop efficient solvers for industrial and academical problems. Works presented in this thesis are the fruit of an exploration between satisfiability and cryptanalysis, and have enabled to display new results and innovative methods to weaken cryptographic functions.

The first contribution is the modeling of a cryptographic problem as a SAT problem. For this, we present some rules that lead to describe easily basic operations involved in cipher algorithms. Then, a section is dedicated to logical reasoning in order to simplify the produced SAT formulas and show how satisfiability can help to enrich a knowledge on a studied problem. Furthermore, we also present many points of view to use our smooth modeling to apply a probabilistic reasoning on all the data associated with the generated SAT formulas. This has then allowed to improve both the modeling and the solving of the problem and underlined a weakness about the use of round constants.

Second, a section is devoted to practical attacks. Within this framework, we tackled preimages of the most popular cryptographic hash functions. Moreover, the collision problem is also approached in different ways, and particularly, the one-bloc collision attack of Stevens [Stevens 2012a] on MD5 was translated within a logical context. It's interesting to remark that in both cases, logical cryptanalysis takes a new look on the considered problems.

Keywords : Logical cryptanalysis, satisfiability, cryptographic hash function, MD5, SHA-1, SHA-3

Introduction générale

Le secret est immanent à l'être humain. Depuis l'ère Antique jusqu'à aujourd'hui, les hommes se sont attelés à créer des codes pour rendre un message inintelligible pour toutes personnes à qui il n'est pas destiné. Cette pratique porte le nom de cryptographie, signifiant étymologiquement *l'écriture cachée*. Utilisée par les gouvernements et les armées, elle a souvent joué un rôle stratégique dans l'Histoire de l'Homme et le destin de ses guerres. Par ailleurs, la capacité à préserver un secret est fortement liée à celle de le découvrir. La cryptanalyse est la science qui cherche à déchiffrer de façon partielle ou entière un message codé. L'ambivalence entre cryptographie et cryptanalyse a permis à ces disciplines concurrentielles de progresser sans discontinuer. Avec l'arrivée de l'ère informatique, l'explosion de la puissance des calculs a totalement bouleversé la donne et fait se rencontrer ces deux disciplines dans le monde des sciences exactes pour donner naissance à la cryptologie, la science du secret.

Dès lors, les messages sont exprimés avec des bits et la sûreté d'un chiffrement repose sur des propriétés mathématiques cadrées par les lois de la théorie de l'information. À cela s'ajoute la démocratisation des ordinateurs et de l'Internet qui a considérablement révolutionné l'utilité des chiffrements pour tout un chacun. Les récentes révélations d'Edward Snowden ont mis en lumière l'étendue de la vigilance accrue de la NSA, avec notamment XKeyscore, un programme de surveillance de masse collectant quasiment toutes actions exercées par un utilisateur sur Internet. Actuellement, chiffrer ses communications et ses données est le seul moyen pour préserver une certaine confidentialité. Pour répondre à ce besoin, la cryptographie moderne fournit plusieurs algorithmes de chiffrement symétriques, tels que DES, AES, IDEA ou Blowfish qui nécessitent une clé secrète pour chiffrer et déchiffrer un message.

Par ailleurs la numérisation des informations et des communications a aussi apporté de nouvelles contraintes en matière de sécurité et donc de nouveaux besoins en cryptographie comme la conservation de l'intégrité des données échangées ainsi que leur authentification. L'authentification est le mécanisme permettant d'identifier une entité ou une personne tandis que l'intégrité consiste en le fait que les données n'aient pas été altérées au cours de leurs transmissions. Ces notions abstraites se concrétisent chaque jour de façon transparente et les exemples dans la vie quotidienne ne manquent pas. Prenons l'exemple d'un retrait de 100 euros sur un guichet automatique. L'authentification est réalisée au moment de la validation du code de quatre chiffres, secrets, tapé sur le clavier qui empêche une autre personne de se servir de votre carte bancaire. Et c'est l'intégrité qui assure que 100 euros seront bel et bien débités sur votre compte et non 1000, ce qui serait bien malheureux. Ces deux concepts sont généralement instaurés dans les crypto-systèmes grâce aux fonctions de hachage cryptographiques.

Le principe d'une telle fonction est de prendre en paramètre une donnée et d'en ressortir une empreinte numérique de taille fixe. Le terme d'empreinte est idéal car il permet de

faire la conjonction avec les empreintes digitales chez les humains, qui offrent la capacité de nous identifier. Toutes ces empreintes ont à peu près la même taille, qu'on soit un homme ou une femme, grand ou petit... et sont fortement différentes suivant chaque individu, de sorte qu'une empreinte digitale correspond à une seule personne. La transposition de cette idée dans le monde numérique conserve ces propriétés. À chaque donnée correspond une unique empreinte et pour deux données très ressemblantes leurs empreintes doivent être assez différentes pour ne pas pouvoir les lier. C'est de la sécurité cryptographique de la fonction de hachage que dépend la garantie de l'authentification et l'intégrité d'une donnée. Pour s'assurer que cette fonction répond à ces exigences, elle doit être résistante à toutes attaques connues quel que soit l'angle et la méthode choisis. C'est dans ce contexte que la cryptanalyse joue un rôle primordial.

Cette thèse a pour sujet la cryptanalyse logique de fonctions cryptographiques et plus particulièrement de fonctions de hachage cryptographiques. La cryptanalyse logique est une cryptanalyse algébrique particulière puisqu'elle repose sur la logique propositionnelle (algèbre de Bool) pour décrire et résoudre un problème. L'expression du problème cryptographique est réalisée à travers le problème de satisfaisabilité, plus couramment appelé problème SAT. Il s'agit d'un problème combinatoire de décision central en théorie de la complexité. Les travaux conduits par la communauté scientifique ces dernières années ont amené à considérer le problème SAT non plus comme un problème exclusivement intéressant pour ses aspects fondamentaux mais aussi comme un outil performant pour la résolution de problèmes contraints issus du monde industriel ou académique. Sa simplicité conceptuelle permet en effet d'exprimer facilement un panel de problèmes très variés et l'effervescence autour des algorithmes de résolution qui lui sont dédiés ne cesse de progresser. Les travaux présentés dans ce document sont le fruit d'une exploration aux confins de la satisfaisabilité et de la cryptanalyse afin d'exhiber de nouvelles pistes pour attaquer et affaiblir les fonctions cryptographiques.

Le chapitre I, dédié aux présentations des notions et des concepts inhérents aux domaines de cette thèse, se divise en trois parties. La première est consacrée à l'introduction de la cryptographie et notamment des fonctions de hachage cryptographiques, noyaux autour desquels gravitent nos travaux. Les méthodes courantes en cryptanalyse pour ces fonctions seront laconiquement décrites avant de donner une description détaillée des algorithmes des fonctions de hachage étudiées dans cette thèse, à savoir celles des familles MD et SHA. Dans la deuxième partie, nous mentionnons le problème de satisfaisabilité, d'abord à travers son langage lexical puis en évoquant ses méthodes de résolution, qu'elles soient complètes ou incomplètes. Dans ce dernier registre, nous présenterons plus en détail l'algorithme de résolution Walksat sur lequel nous avons expérimenté quelques modifications dans le chapitre IV. Enfin, nous terminons ce chapitre en évoquant ce qu'est la cryptanalyse logique, le domaine de prédilection de nos recherches. Cette dernière partie contient les grands principes de mise en œuvre de ce domaine et dresse un état de l'art de la discipline.

Le sujet traité dans le chapitre II concerne l'aspect modélisation d'un problème cryptographique en un problème SAT. Parce que modéliser un problème peut se faire de plusieurs façons, nous discuterons dans un premier temps des différents choix pouvant être faits pour représenter un simple circuit logique. Cette mise en bouche introduit ensuite la modélisa-

tion de fonctions de hachage. Dans un premier temps nous montrons comment décrire avec des règles logiques très simples les opérations de base constituant une telle fonction cryptographique. Les méthodes pour traduire une addition, une fonction non-linéaire et un décalage circulaire dans la logique propositionnelle sont entièrement détaillées. S'ensuit la façon de décrire un circuit logique complet pour une fonction de hachage et le générateur de formules propositionnelles qui a été conçu par nos soins. Une comparaison des formules ainsi produites avec celles de la littérature a été réalisée pour souligner l'importance des choix que nous avons faits. Un point sera aussi fait sur les caractéristiques des formules générées pour mettre en avant leurs aspects très singuliers et totalement inédits pour la communauté. La fin du chapitre est consacrée à la simplification des formules via un moteur d'inférence produit pour nos travaux. Celui-ci fait office de pré-traitement et permet non seulement de réduire la complexité du problème mais aussi de l'enrichir avec la création de ponts logiques caractérisés par la détection de relations d'équivalences. Ces nouvelles relations concluent ce chapitre et peuvent d'ailleurs tout à fait être utilisées dans un autre contexte que celui de la cryptanalyse logique.

Alors que le chapitre II s'appuie sur la découverte de structures génériques, nous avons poussé le raisonnement jusqu'alors exclusivement logique dans un univers probabiliste. Le chapitre III se penche sur le comportement logique en moyenne des formules SAT associées aux fonctions de hachage. Une première partie est dédiée à la production de statistiques sur les variables afin d'obtenir pour chacune d'elles une affectation probabiliste dans un espace de solutions. Nous expliquons ainsi le protocole mis en place pour générer ces données. À la vue de la quantité d'information que cela représente, nous montrons une façon originale pour les visualiser qui nous a permis de révéler quelques variables aux caractéristiques particulières. Nous avons d'ailleurs dégagé la cause de cette singularité en démontrant une faiblesse sur l'utilisation de constantes de ronde. L'accent s'est ensuite porté sur l'exploitation de ces informations en définissant la notion de « quasi-relations » , c'est-à-dire des relations entre certaines variables qui occurrent dans une très grande majorité des cas sans pourtant être toujours vraies. Nous donnons à ce titre des exemples concrets qui peuvent être utilisés en pratique pour réduire un problème et améliorer sa résolution. Dans le même registre, une seconde étude a été vouée à l'examen des clauses pour exhiber des structures remarquables. Deux voies ont été explorées et ont permis de donner pour chaque clause un profil d'affectation probabiliste. Le chapitre IV concerne les attaques en pratique. Il commence par décrire la méthode couramment employée pour attaquer la pré-image d'une fonction de hachage par le levier de la cryptanalyse logique. Dans une première étape, l'algorithme de hachage est décrit en un formalisme logique à travers un problème SAT. La deuxième étape consiste à affecter les variables d'une empreinte particulière dans le problème. On obtient alors une nouvelle formule qui est plus contrainte et ayant pour solution une réponse au problème de la pré-image. Puisque cette solution est une affectation de toutes les variables alors une valeur de vérité est aussi donnée à celles correspondant au message d'entrée. Le but de l'attaque est donc de trouver au moins une affectation qui satisfasse le problème. C'est la résolution de la formule par un solveur SAT qui permet cela, apparentant l'approche à un « reverse-engineering » . Cependant, même si en pratique les formules résultantes répondent au problème de la pré-image, la difficulté calculatoire est un obstacle trop important pour être surmonté. C'est donc la pré-image de versions réduites

de la fonction de hachage qui est attaquée. L'idée est de déterminer combien d'étapes sont inversibles pour donner une certaine mesure de la résistance à la pré-image. Nous présentons dans ce chapitre nos résultats en la matière pour les fonctions de hachage MD4, MD5, SHA-0 et SHA-1. Par ailleurs, la cryptanalyse logique offre un cadre idyllique pour examiner au plus près une attaque connue. La recherche de collisions consiste en le fait de trouver deux messages différents qui donnent la même empreinte. C'est aujourd'hui l'angle d'attaque privilégié par les cryptanalystes pour affaiblir (voire casser) les fonctions de hachage. Les travaux que nous avons réalisés autour de la collision sont découpés en deux parties. Dans la première, nous montrons comment la modélisation fine permet d'étudier une collision pour comprendre ses mécanismes. La seconde expose deux types d'attaques que nous avons conduites contre MD5, d'abord en la réimplantation d'un chemin différentiel donné par Stevens [Stevens 2012a] puis dans une approche personnalisée du problème par le biais de collisions locales réinjectées dans une formule globale.

Enfin, le chapitre V conclut cette thèse en proposant une appréciation de nos travaux pour la cryptanalyse logique ainsi que des perspectives de recherche attractives.

Table des matières

CHAPITRE I

Préliminaires

L'intérêt pour l'étude du problème de satisfaisabilité a considérablement augmenté au cours des dernières années en raison de sa simplicité conceptuelle et de sa capacité à exprimer un grand éventail de problèmes aussi divers que variés. Dans ce cadre, un nombre important de travaux ont mis en avant ce sujet à travers diverses implications issues du monde réel. Par exemple, la réalisation de planning [Kautz 1996], le model checking [Biere 2006], la conception de circuit (VLSI) ainsi que la cryptographie [Kullman 2002, Potlapally 2007] ont été des problèmes modélisés sous un formalisme logique pour ensuite être résolus plus efficacement par des méthodes de résolution dédiées.

Par ailleurs, le monde numérique prend chaque jour une place de plus en plus importante dans notre société et les systèmes de communication ont toujours besoin de plus de sûreté. C'est dans cet environnement que la cryptographie joue un rôle primordial, plaçant ainsi les primitives cryptographiques au cœur de la sécurité des systèmes informatiques. La capacité de ces fonctions à préserver un secret, c'est à dire un texte chiffré ou bien une clé de chiffrement, devient dès lors un intérêt majeur.

La cryptanalyse est la science qui se réfère à toutes les techniques permettant de déceler une faiblesse qui facilite la découverte d'une information mettant à mal le secret préservé. Plusieurs approches très performantes ont été proposées ces dernières années, comme par exemple la cryptanalyse différentielle [Biham 1990], la cryptanalyse linéaire [Matsui 1992] ou bien la cryptanalyse algébrique [Patarin 1995]. Dans ce contexte, une cryptanalyse algébrique particulière, appelée cryptanalyse logique [Massacci 2000], est en pleine éclosion. Elle consiste à s'appuyer sur le problème de satisfaisabilité pour d'abord modéliser un problème cryptographique en un système d'équations booléennes puis sur les méthodes de résolution automatiques pour le résoudre.

Les travaux de recherche présentés dans ce manuscrit font partie intégrante de la cryptanalyse logique. Situés aux confins du problème de satisfaisabilité et de la cryptanalyse algébrique, ce chapitre présentera les notions pour s'approprier ces deux univers.

La première section sera dédiée à la cryptographie et plus particulièrement aux fonctions de hachage cryptographiques, au cœur de nos travaux. Les fonctions MD4, MD5, SHA-1 et SHA-3 seront décrites en détail afin de garantir une compréhension complète du chapitre II. Les techniques majeures en cryptanalyse pour ces fonctions seront succinctement énumérées pour introduire dans une dernière partie la cryptanalyse logique, le domaine de prédilection de nos contributions, exposées aux chapitres III et IV.

Dans la partie suivante, nous évoquerons le problème de la satisfaisabilité et son langage de description, à savoir la logique propositionnelle. Cette dernière joue un rôle primordial dans la modélisation des fonctions de hachage cryptographiques au chapitre II

ainsi que dans l'analyse structurelle du problème SAT créé, au chapitre III. Un aperçu des méthodes de résolution les plus récentes pour le problème de satisfaisabilité sera fourni pour aider la compréhension du chapitre IV.

1 Introduction à la cryptographie

À l'origine, la cryptographie est l'étude dédiée à la protection des messages de façon à les rendre sans aucun sens pour quiconque voudrait les connaître, excepté pour leurs destinataires. Elle englobe désormais un peu plus que ce simple concept de confidentialité. C'est une branche de la *cryptologie*, la science du secret, qui emprunte divers principes issus d'un panel de disciplines dans les domaines des mathématiques et de l'informatique parmi lesquelles figurent l'arithmétique, l'algèbre, la théorie des probabilités, l'algorithmique, la théorie de l'information et la théorie de la complexité.

Depuis la période antique (scytale, chiffrement de César) jusque la moitié du XX$^{\text{ième}}$ siècle (Enigma), les méthodes utilisées en cryptographie nécessitaient la même connaissance secrète (ou clé) pour chiffrer et déchiffrer un message. En ce sens, cette cryptographie dite symétrique, a accompagné l'Homme et ses communications les plus secrètes durant plus de 2000 ans. Aujourd'hui, de redoutables algorithmes de chiffrement symétrique garantissent une sécurité très élevée comme par exemple les chiffrements par bloc DES puis AES et les chiffrements par flot RC4 et A5/1. La cryptographie symétrique garde cependant un inconvénient : la distribution de la clé.

La réponse à ce problème est arrivé en 1976 avec le paradigme d'échange de clés « Diffie-Hellman-Merkle » [Diffie 1976] donnant naissance à la cryptographie asymétrique [1]. Cette méthode de chiffrement utilise une clé publique qui peut être vue de tous et d'une clé secrète qui doit restée privée. Dans le cadre de la confidentialité, la première permet de chiffrer le message et la seconde de le déchiffrer. Par conséquent, si n'importe qui peut chiffrer un message, seul le destinataire est en mesure de le déchiffrer. Pour traiter du problème de l'authentification, la clé privée sert à signer un message et la clé publique à vérifier la signature. Ainsi, tout le monde peut s'assurer que l'émetteur est bien celui qu'il prétend être car il est le seul à connaître la clé privée. Le chiffrement asymétrique fut applicable dès 1978 avec l'apparition du RSA [Rivest 1978].

Une troisième catégorie d'algorithmes cryptographiques fit son apparition pour répondre aux problèmes de l'intégrité de données et de l'authentification (HMAC [Bellare 1996]). Il s'agit des fonctions de hachage cryptographiques, que nous nommeront par abus de langage « fonctions de hachage » . Une telle fonction prend en entrée un message de taille quelconque et fournit en sortie un message de taille fixe qu'on appelle *empreinte*. Cette thèse s'est principalement focalisée sur ce type de fonction et un détail plus fourni est proposé paragraphe 3.

De nos jours, les besoins en cryptographie sont nombreux. On recense bien sûr les problèmes liés à la *confidentialité*, mais aussi des problèmes d'*intégrité de données*, d'*authentification* et de *non-désaveu*. Les définitions précises de ces notions et leurs mises

1. Des travaux classifiés quasi identiques conduits par James H. Ellis, Clifford Cocks et Malcolm J. Williamson datant de quelques années auparavant apportaient aussi une solution à la distribution de clés.

en œuvre peuvent être trouvées dans [Schneier 2001].

La cryptanalyse est le pendant de la cryptographie. Il s'agit de la discipline qui cherche à retrouver toute information permettant de retrouver partiellement ou de façon intégrale la signification d'un message chiffré ou de la clé qui le préserve. La cryptanalyse est *de facto* née avec les premiers chiffrements et s'est diversifiée avec le temps au point de fournir une véritable boîte à outils pour quiconque voudrait percer les mystères d'un secret. Ci après nous donnons quelques définitions permettant de formaliser le contexte.

Définition 1 : *La méthode consistant à chercher le sens d'un secret est appelée* ***une attaque****.*

Définition 2 : *Une* ***attaque par force brute*** *consiste à tester toutes les combinaisons possibles pour retrouver un texte en clair ou une clé de chiffrement.*

Définition 3 : *On dit d'un algorithme de chiffrement qu'il est* ***affaibli*** *quand il existe une attaque requérant moins de ressources qu'une attaque par force brute.*

Définition 4 : *On dit d'un algorithme de chiffrement qu'il est* ***cassé*** *lorsqu'une information est trouvée sur la donnée préservée.*

Définition 5 : *Une attaque* ***sur chiffré seul*** *est une attaque où le cryptanalyste ne connaît que les messages secrets qu'il tente de déchiffrer.*

Définition 6 : *Une attaque* ***à textes clairs*** *est une attaque où le cryptanalyste connaît des textes clairs et les textes chiffrés qui correspondent.*

Définition 7 : *Une attaque* ***à textes clairs choisis*** *est une attaque à texte clairs où le cryptanalyste peut lui même choisir les textes clairs qu'il peut chiffrer grâce à l'algorithme de chiffrement.*

De nos jours, la puissance de calcul fournie par les ordinateurs (et même nos téléphones portables) couplée à des techniques basiques (statistiques d'apparition de motifs, force brute...) suffit à casser la grande majorité des chiffrements antérieurs à la cryptographie moderne. Par conséquent, depuis l'apparition de l'ère informatique, la cryptographie s'est adaptée et les techniques de cryptanalyse ont dû suivre cette évolution. Plusieurs types de cryptanalyses - liste non-exhaustive - ont depuis vu le jour :

- **La cryptanalyse différentielle** est une attaque à textes clairs choisis qui consiste en l'étude des changements provoqués dans la structure interne d'un algorithme de chiffrement par l'injection de différence(s) dans le message d'entrée. Les premiers travaux [Biham 1990] ont été réalisés par Eli Biham et Adi Shamir à la toute fin des années 1980 et elle peut être portée sur tous types d'algorithmes cryptographiques y compris les fonctions de hachage [2].

2. On notera qu'IBM aurait découvert cette technique au milieu des années 1970 mais la NSA voulait la préserver secrète pour que les Etats-Unis puissent garder un certain avantage sur ses adversaires...

- **La cryptanalyse linéaire** a pour but de représenter par une fonction linéaire une approximation de l'algorithme de chiffrement. Elle fut introduite par Mitsuru Matsui en 1992 sur FEAL [Matsui 1992] puis en 1994 sur DES [Matsui 1994] et touche principalement les algorithmes de chiffrement par bloc et par flot.
- **La cryptanalyse algébrique** est une technique mise en lumière en 1995 par Jacques Patarin sur le schéma de cryptographie multivariée de Matsumoto-Imai [Matsumoto 1988] [Patarin 1995]. Elle consiste en la réécriture de l'algorithme sous la forme d'un système d'équations polynomiales à plusieurs variables puis en sa résolution par diverses techniques (bases de Gröbner, solveurs automatiques). Les travaux présentés dans cette thèse aux chapitres II, III et IV se situent dans la sphère de la cryptanalyse logique qui est un sous domaine de la cryptanalyse algébrique.

2 Cryptographie symétrique et asymétrique

Nous commençons cette partie par de nouvelles définitions permettant de décrire quelques éléments utilisés en cryptographie.

Définition 8 : ***Une transposition*** *ou* ***permutation*** *est l'opération qui consiste à échanger de place les éléments d'un ensemble. C'est une perturbation de l'ordre.*

Exemple 1 : *Soit m le message «* ABCD *» . Par la transposition «* A *est échangé avec* C, D *est échangé avec* B *» , on obtient un nouveau message m' : «* CDAB *» .*

Définition 9 : ***Une substitution*** *est l'opération qui remplace, pour un message donné, un symbole par un autre. Contrairement à la transposition où chaque lettre garde sa nature, l'ordre des lettres n'est pas changé.*

Exemple 2 : *Soit m le message «* ABCD *» . Par la substitution «* A *devient* F, B *devient* X *» , on obtient un nouveau message m' : «* FXCD *» .*

Définition 10 : *Une* ***clé*** *de chiffrement, notée k, est une information, en principe secrète, permettant de chiffrer (respectivement déchiffrer) un message en clair (respectivement chiffré).*

Définition 11 : *L'ensemble des valeurs possibles d'une clé k est appelé* ***l'espace des clés****. Si la clé est définie en bits, alors pour une clé k de longueur l, l'espace des clés comprend 2^l clés possibles.*

2.1 Cryptographie symétrique

La cryptographie symétrique consiste en le chiffrement et le déchiffrement de messages, grâce à l'utilisation d'une clé secrète. Cette clé est fondamentale pour la sécurité d'un tel chiffrement, et sans elle, un message chiffré doit être totalement illisible pour toutes personnes à qui il n'est pas destiné.

Soient $\mathcal{E}$ un algorithme de chiffrement, k une clé de chiffrement et $\mathcal{M}$ un message en clair, un chiffrement symétrique est défini de la manière suivante :

$$\mathcal{E}_k(\mathcal{M}) = \mathcal{C}$$

Un déchiffrement est défini par l'opération suivante :

$$\mathcal{D}_k(\mathcal{C}) = \mathcal{M}$$

Un algorithme de chiffrement symétrique vérifie la propriété suivante :

$$\mathcal{D}_k(\mathcal{E}_k(\mathcal{C})) = \mathcal{M}$$

La cryptographie symétrique fut celle utilisée depuis l'ère antique jusque de nos jours et a réussi avec succès le passage entre la cryptographie classique - basée sur les opérations de transpositions et de substitutions - et la cryptographie moderne apparue après la seconde guerre mondiale, reposant sur les sciences mathématiques et informatiques. Cette thèse ne s'intéresse pas à la cryptographie classique qui regorge pourtant d'anecdotes qui ont alimentées l'Histoire et la passion. Pour plus d'informations sur ce sujet, nous invitons le lecteur à les retrouver dans l'ouvrage de Simon Singh [Singh 1999].

L'arrivée de l'informatique et la puissance des calculs automatisés ont rendus les chiffrements classiques obsolètes, entres autres car la taille des clés utilisées sont devenues trop petites, laissant l'énumération de l'espace des clés réalisable par un ordinateur. De plus, en même temps que les communications se multiplièrent, la cryptographie se démocratisa et les besoins en matière de sécurité se firent plus nombreux. La cryptographie empirique bascula alors vers la science et de nouvelles règles furent établies pour garantir la sûreté des chiffrements. Premièrement, les messages formulés avec des symboles sont désormais exprimés avec des bits. Secondement, la définition d'un bon chiffrement est donnée par les lois définies en théorie de l'information. Ce domaine devint très vite prépondérant pour la cryptographie moderne et ses lois - établies par Claude Shannon [Shannon 1949] - ont permis d'élaborer les propriétés nécessaires pour avoir un chiffrement sécurisé.

Ainsi, un chiffrement est dit sûr lorsqu'à partir d'un texte chiffré $\mathcal{C}$ et d'un texte clair $\mathcal{M}$, la probabilité que $\mathcal{M}$ ait donné $\mathcal{C}$ est la même que celle pour $\mathcal{M}$ d'exister. En d'autres termes, soit $|\mathcal{M}|$ le nombre de textes clairs possibles ; la probabilité que $\mathcal{C}$ soit le chiffré de $\mathcal{M}$ est de $\frac{1}{|\mathcal{M}|}$.

Cela implique deux choses : les clés de chiffrement sont équiprobables et aucune information du texte clair ne peut être trouvée à partir du texte chiffré. L'assurance de cette dernière propriété se fait à travers le respect des notions de confusion et de diffusion dont les définitions sont données ci-après.

Définition 12 : *La **confusion** est le fait de rompre toute relation entre le texte chiffré et sa clé de chiffrement. Elle est mise en pratique par le procédé de substitution.*

Définition 13 : *La **diffusion** vise à rendre très influents chaque bit du texte d'entrée sur la sortie. C'est l'opération de transposition qui permet de l'implanter dans un chiffrement.*

Deux méthodes existent pour établir un chiffrement symétrique : les chiffrements par flot et les chiffrements par bloc. Nous les présentons ci-après de manière informelle.

2.1.1 Chiffrement par flot

Le principe opératoire d'un chiffrement par flot (aussi appelé par flux ou par *stream*) consiste à chiffrer un texte clair bit par bit. C'est un chiffrement en continu qui n'attend pas d'avoir toutes les données pour commencer à chiffrer. Généralement, un texte clair est chiffré en l'associant à une suite secrète par une opération symétrique conservant une distribution équiprobable (généralement c'est le XOR qui est utilisé). Le masque de Vernam propose à ce titre une base solide pour cette conception et il convient d'en faire une brève présentation.

Appelé aussi masque jetable, il a été élaboré en 1926 par G. Vernam. C'est aujourd'hui le seul chiffrement qui a été démontré comme totalement sûr [Shannon 1949]. Chiffrer en utilisant le masque de Vernam consiste à générer une chaine de caractères aléatoires de la même taille que le texte en clair, puis d'effectuer un XOR entre les deux. Le déchiffrement est effectué en réalisant un XOR entre le texte chiffré et la suite secrète. Soient k une suite aléatoire, $\mathcal{M}$, un message en clair, un chiffrement/déchiffrement par masque de Vernam est défini de la manière suivante :

$$\text{Chiffrement} : \mathcal{M} \oplus k = \mathcal{C}$$

$$\text{Déchiffrement} : \mathcal{C} \oplus k = \mathcal{M}$$

Exemple 3 :

`0101 0100 0101 1000 0101 0100` *(texte clair $\mathcal{M}$)*
`0111 0101 1010 1010 1010 1000` *(suite aléatoire k)*
`0010 0001 1111 0010 1111 1100` *(texte chiffré $\mathcal{C}$)*

La production de suites totalement aléatoires est un problème certain dans l'univers déterministe de l'informatique. La parade utilisée dans les chiffrements par flot est de passer par la génération de suites pseudo-aléatoires. Pour cela soit la suite est directement créée dans l'algorithme de chiffrement, comme par exemple dans le RC4 utilisé dans les protocoles TLS et WEP ou le E0 utilisé dans le Bluetooth, ou soit elle est générée par le biais de registres à décalage à rétroaction linéaire [3], comme dans l'algorithme A5/1 utilisé dans le standard GSM pour les téléphones portables.

Les angles d'attaque contre les algorithmes de chiffrement par flot sont peu nombreux. RC4 a cependant été fragilisé par la découverte de particularités statistiques sur la suite pseudo-aléatoire [Fluhrer 2001, Klein 2008]. Quant à A5/1, il a été cassé par des méthodes proches de la brute force (utilisation de FGPA [Güneysu 2008], tables de lookup et GPUs, tables arc-en-ciel, etc.).

2.1.2 Chiffrement par bloc

Un chiffrement par bloc est un chiffrement à clé secrète qui opère par bloc de bits. Lors d'un chiffrement par bloc, le message est découpé en blocs de taille fixe qui sont successivement chiffrés. Bien souvent, le message est de taille quelconque et doit être complété

3. Le terme utilisé en anglais est **L**inear **F**eedback **S**hift **R**egister.

pour être d'une taille multiple à la taille des blocs. Plusieurs modes opératoires permettent de manier les blocs d'entrée et de sortie comme par exemple ECB (pour Electonic Code Book), CBC (pour Cipher Block Chaining) ou CFB (Cipher Feedback).Nous ne les détaillerons pas ici car ce n'est pas le propos de cette thèse. Cependant, plus d'informations sur les modes peuvent être trouvées dans [Alfred J. Menezes 1996].

Les algorithmes sont fabriqués de manière empirique avec pour objectif de résister à toutes les attaques connues tant sur le plan théorique que pratique. La taille de la clé et la taille des blocs utilisés sont intimement liés à l'algorithme de chiffrement. Deux principales méthodes de construction se distinguent : les réseaux de substitutions-permutations et les réseaux de Feistel. Chacun de ces algorithmes sont conçus de manière itérative et composés de plusieurs tours, faisant intervenir une fonction de chiffrement. Cette fonction prend pour paramètre une sous-clé dérivée de la clé secrète de l'algorithme et assure les propriétés de confusion et de diffusion.

Le chiffrement par bloc le plus célèbre basé sur les réseaux de Feistel est le DES (Data Encryption Standard) qui donna ensuite le triple-DES, défini par le NIST [4] en 1977. Le DES utilise un réseau de Feistel à 16 tours et une clé de chiffrement de 56 bits, pour chiffrer des blocs de 64 bits. La cryptanalyse linéaire a permis d'affaiblir sa complexité à environ 2^{39} [Matsui 1994, Junod 2001]. Mais c'est surtout la taille de sa clé qui est jugée trop petite. En effet, une attaque par force brute nécessite dans le pire des cas 2^{56} opérations, ce qui est devenu réalisable aujourd'hui. Parce ce qu'il est lent et qu'il repose sur le DES, le triple-DES est quant à lui jugé insuffisant pour devenir un standard. Il reste cependant encore approuvé par le NIST jusque 2030 [NIST 2012b] pour laisser le temps aux entités qui l'utilisent de transiter vers le nouveau standard. Face à l'obsolescence du DES, le NIST lança en 1997 un concours pour élire le nouveau standard en matière de chiffrement symétrique. RIJNDAEL [Daemen 1998] conçu pour résister aux attaques connues contre le DES, sorti vainqueur et renommé AES (pour Advance Encryption Standard). Cet algorithme de chiffrement utilise un réseau de substitutions-permutations matérialisées par des S-box et des P-box. Sa clé de chiffrement est paramétrable à 128, 192 et 256 bits et suivant la taille de sa clé, son réseau admet respectivement 10, 12 et 14 tours.

Le principal inconvénient des chiffrements symétriques est qu'ils utilisent une clé secrète qui doit être distribuée lors de la transmission d'un message chiffré. Il est en effet primordial d'échanger cette clé si deux interlocuteurs veulent communiquer de manière sécurisée. Cette distribution de clé est un obstacle certain et fut pendant de nombreuses années un réel problème à résoudre pour les cryptographes. La solution leur vint seulement à la fin des années 1970 avec ce qu'on appelle désormais la cryptographie asymétrique.

2.2 Cryptographie asymétrique

La cryptographie asymétrique, qu'on appelle aussi cryptographie à clé publique, est un concept trouvé en 1970 par Diffie-Hellman-Merkle [Diffie 1976]. Elle est très fortement

4. Pour **N**ational **I**nstitute of **S**tandards and **T**echnology. Il s'agit de l'agence fédérale des technologies cherchant, entre autres, à développer les standards en matière de technologie.

liée à la théorie des nombres et au problème de la factorisation des grands entiers. Les chiffrements asymétriques utilisent une clé publique qui peut être vue de tous et une clé privée qui doit restée secrète. Soient, $\mathcal{E}$ un algorithme de chiffrement, $\mathcal{M}$, un message en clair, k_A et k_B respectivement une clé publique et une clé privée générées par un algorithme paramétré par un élément de sécurité. Quand il s'agit d'assurer la confidentialité, le principe opératoire d'un algorithme de chiffrement asymétrique est le suivant :

$$\text{Chiffrement} : \mathcal{E}_{k_A}(\mathcal{M}) = \mathcal{C}$$

$$\text{Déchiffrement} : \mathcal{D}_{k_B}(\mathcal{C}) = \mathcal{M}$$

Le principe d'un envoi de courrier dans une boîte aux lettres permet d'expliquer très simplement le principe d'un chiffrement asymétrique. Pour envoyer un courrier, il suffit de connaître l'adresse - publique - du destinataire. Pour lire le message envoyé, il faut ouvrir la boîte avec une clé - privée - appartenant uniquement à celui à qui appartient la boîte aux lettres. Quand il s'agit de répondre au problème de la confidentialité, la clé publique permet de chiffrer le message et la clé privée de le déchiffrer. Par conséquent, tout le monde est en mesure de chiffrer un message mais seul le destinataire est capable de le déchiffrer. Ainsi, tout le monde peut s'assurer que le destinataire est bien celui qu'il prétend être car il est le seul à connaître la clé privée. Quand il s'agit d'assurer l'authentification, le principe opératoire d'un tel chiffrement asymétrique est alors le suivant :

$$\text{Signature} : \mathcal{E}_{k_B}(\mathcal{M}) = \mathcal{C}$$

$$\text{Vérification de signature} : \mathcal{D}_{k_A}(\mathcal{C}) = \mathcal{M}$$

Plusieurs ouvrages spécialisés [Zémor 2000, Schneier 2001, Vergnaud 2012, Ghernaouti 2013] détaillent l'apport des chiffrements asymétriques pour la cryptographie. La mise en pratique fut cependant un véritable challenge pour les cryptographes. En 1978, la réponse fut apportée par le chiffrement asymétrique RSA [Rivest 1978] qui propose une fonction à sens unique à trappes basée sur la factorisation de grands entiers. Le protocole est le suivant. Le destinataire choisit deux grands nombres premiers p et q et calcule leur produit N. Il choisit également un exposant e qui ne doit avoir aucun diviseur premier commun avec la fonction indicatrice d'Euler $\phi(\mathcal{N}) = (p-1)(q-1)$. Le couple (N,e) forme alors une clé publique, tel qu'il soit possible de chiffré un message $\mathcal{M}$ pour obtenir $\mathcal{C}$ de la façon suivante :

$$\text{Chiffrement} : \mathcal{M}^{\mathrm{e}} \; mod \; N = \mathcal{C}$$

D'après le petit théorème de Fermat, il est possible de déchiffrer $\mathcal{C}$ en ayant connaissance d'un exposant d, tel que d soit l'inverse de e modulo $\phi(\mathcal{N})$. Seule la connaissance de p, q et e permet de calculer l'exposant privé d. Ainsi, le couple (N,d) forme une clé privée, tel qu'il soit possible de déchiffrer un message $\mathcal{C}$ pour obtenir $\mathcal{M}$ de la façon suivante :

$$\text{Déchiffrement} : \mathcal{C}^{\mathrm{d}} \; mod \; N = \mathcal{M}$$

À ce jour RSA est affaibli si les nombres premiers utilisés sont « petits » puisqu'en pratique, un nombre de 768 bits à été factorisé en 2009 par un crible algébrique et beaucoup de patience, la totalité des calculs ayant duré quelques années [Kleinjung 2010]. De part ce résultat, on considère aujourd'hui qu'une clé de 1024 bits est trop juste pour certifier comme sûr un chiffrement. Plus de détails peuvent être trouvés dans [Schneier 2001, Hankerson 2003]. Récemment, d'autres travaux [Genkin 2013] ont mis en lumière une attaque sur une clé de 4096 bits par une cryptanalyse acoustique [5]. Enfin, concernant la cryptanalyse logique, quelques travaux se sont intéressés à la découverte de bits sur une clé privée RSA. Nous faisons un point sur ce sujet dans la partie 5.

Enfin, même si les travaux dans cette thèse ne les abordent pas, faisons un détour succinct sur la cryptographie par les courbes elliptiques pour être plus complet sur les chiffrements asymétriques. Dans ce contexte, l'émetteur et le destinataire choisissent publiquement trois paramètres : une courbe elliptique $E(a,b)$ de la forme $y^2 = x^3 + ax^2 + b$, un corps fini $\mathbb{F}_q$ et un point P sur la courbe précédemment définie. Ensuite, de manière privée le destinataire choisi un nombre entier k_1 et le destinataire un nombre entier k_2. L'émetteur calcule alors le point de la courbe P'_1 défini par $P'_1 = k_1 \cdot P$; le destinataire calcule le point de la courbe P'_2 défini par $P'_2 = k_2 \cdot P$. Puis, les deux interlocuteurs s'échangent leurs points P'. De cette façon, ils peuvent maintenant calculer une clé secrète définie par :

$$\mathcal{K} = P'_1 \cdot k_2 = P'_2 \cdot k_1 = k_1 \cdot k_2 \cdot P$$

Cette clé ne peut être connue que d'eux seuls car aucune autre personne ne peut connaître k_1 et k_2. En effet pour découvrir $\mathcal{K}$ à partir de E, P'_1, P'_2 et P, il faut pouvoir calculer k_1 (respectivement k_2) à partir de P et de P'_1 (respectivement P'_2), ce qui revient à résoudre le problème du logarithme discret sur la courbe elliptique associée. Or, à ce jour cela reste un problème qui n'admet pas d'algorithmes de résolution en temps sous-exponentiel. La façon la plus efficace pour attaquer le problème passe par l'algorithme BABY-STEP GIANT-STEP [6]. Nous invitons le lecteur curieux à trouver plus d'informations sur les courbes elliptiques dans [Hankerson 2003]. Pour conclure sur ce point, notons que le problème du logarithme discret est utilisé en pratique dans le chiffrement asymétrique El Gamal [Gamal 1985] lui même utilisé entres autres dans le logiciel PGP.

De part sa complexité, cette cryptographie est encore peu utilisée et est par conséquent moins étudiée que sa consœur RSA alors qu'elle présente pourtant l'avantage de pouvoir utiliser des clés plus courtes. On dénombre néanmoins quelques travaux sur des attaques par le biais de canaux auxiliaires. Du point de vue de la cryptanalyse, les courbes elliptiques ne sont pas directement attaquées. Il existe cependant quelques travaux sur la modélisation d'inversion de la multiplication qui pourrait être des prémices pour attaquer en pratique la factorisation de grands entiers [Béjar 2010].

5. Cela consiste à écouter le bruit réalisé par un chiffrement RSA au niveau des composants de la machine et à identifier les tâches que la machine est en train d'effectuer.

6. L'algorithme a été développé par Daniel Shanks [Shanks 1971]. Quelques détails peuvent aussi être trouvées dans [Sutherland 2007].

3 Fonctions de hachage cryptographiques

Nos travaux se sont essentiellement concentrés autour des fonctions de hachage cryptographiques, et plus particulièrement MD4, MD5, SHA-1 et SHA-3. Cette partie a pour vocation la présentation de quelques généralités sur les fonctions de hachage puis de donner un détail précis de leur conception. Un tour d'horizon de leur état en matière de cryptanalyse sera également proposé.

3.1 Définitions

Les fonctions de hachage sont des éléments centraux dans la cryptographie moderne, bien différentes des fonctions de chiffrement symétriques/asymétriques.

Ces fonctions cryptographiques assurent principalement deux contraintes : *l'intégrité* des données et *l'authentification* dans le cadre des HMAC [Bellare 1996]. Dans [Schneier 2001], la notion d'intégrité est présentée comme le fait que « *le destinataire d'un message doit pouvoir vérifier que celui-ci n'a pas été modifié en chemin. Un intrus doit être incapable de faire passer un faux message pour un légitime* » . Il s'agit donc de la garantie qu'une donnée n'a pas été falsifiée durant un traitement. De même, l'*authentification* est définie ainsi : « *Le destinataire d'un message doit pouvoir s'assurer de son origine. Un intrus ne doit pas être capable de se faire passer pour quelqu'un d'autre* » . Il s'agit donc d'authentifier une entité (une personne, un fichier) en vérifiant qu'elle est bien celle qu'elle prétend être. Les fonctions de hachage sont très largement utilisées dans les méthodes d'authentification (UNIX utilise l'empreinte des mots de passe), dans les signatures électroniques ainsi que dans bien d'autres protocoles cryptographiques (SSL, Code d'Authentification de Messages, etc.).

Une fonction de hachage prend en entrée un message $\mathcal{M}$ de taille arbitraire finie et donne en sortie une empreinte $\mathcal{E}$ de longueur fixe n. Les messages informatiques étant exprimés dans un univers binaire, la taille de l'empreinte sera exprimée en bits.

Plus formellement, une fonction de hachage peut s'exprimer de la manière suivante :

$$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$$

Avant de hacher un message, les fonctions de hachage étudiées dans cette thèse effectuent d'abord un pré-traitement de manière à ce que la longueur du message soit multiple de la taille d'un bloc. Ce remplissage, qu'on appelle plus couramment *bourrage* ou *padding*, consiste à ajouter des bits à la fin du message. La façon dont il est réalisé et la taille d'un bloc dépendent de la fonction de hachage.

3.2 Sécurité des fonctions de hachage

L'évaluation de la sécurité d'une fonction de hachage repose sur la résistance aux trois propriétés définies ci-dessous, en partie retranscrites de [Vergnaud 2012] :

Définition 14 : *La recherche de* ***collision*** *signifie trouver deux messages* m_1, $m_2 \in \{0,1\}^*$ *tels que* $m_1 \neq m_2$ *et* $\mathcal{H}(m_1) = \mathcal{H}(m_2)$.

Définition 15 : *Rechercher une **pré-image** ou un **antécédent** est le fait de trouver, étant donnée une empreinte $h \in \{0,1\}^n$, un message $m \in \{0,1\}^*$ tel que $\mathcal{H}(m) = h$.*

Définition 16 : *Rechercher une **seconde pré-image** ou un **second antécédent** est le fait de trouver, étant donné un message $m_1 \in \{0,1\}^*$ et son empreinte $h_1 \in \{0,1\}^n$ un deuxième message $m_2 \in \{0,1\}^*$ tel que $\mathcal{H}(m_1) = \mathcal{H}(m_2)$.*

La complexité d'une attaque contre une fonction de hachage sera quantifiée par la taille de son empreinte. Pour une recherche de pré-image, il est nécessaire que la fonction ne soit pas attaquable par une méthode requérant moins de 2^n calculs et pour une recherche de collisions moins de $2^{n/2}$ calculs, en tenant compte du paradoxe des anniversaires [Von Mises 1964]. Par conséquent, plus l'empreinte va être de grande taille, plus la fonction de hachage sera potentiellement résistante.

Le paradoxe des anniversaires est une proposition surprenante puisqu'il s'agit d'une vérité mathématique contre-intuitive. C'est aussi une estimation probabiliste qui est systématiquement appliquée en cryptanalyse permettant de réduire la complexité théorique d'une attaque par collisions à sa racine carrée. Son énoncé est le suivant :

Soit une classe de x élèves. Combien doit valoir x pour avoir 50% de chance que deux élèves soient nés le même jour ?

Considérons l'événement A_x = « toutes les x personnes ont des dates d'anniversaire différentes. » et calculons sa probabilité en fonction de x. La première personne peut être née n'importe quel jour. Pour la seconde personne, il reste 364 choix possibles, pour la troisième 363, etc. De façon générale, cette probabilité s'exprime par :

$$P(A_x) = \frac{365 \text{ x } 364 \text{ x } 363 \text{ x } \ldots}{365^{\text{x}}}$$

Pour $x = 22$ cela donne :

$$P(A_{22}) = \prod_{i=0}^{21} \frac{365 - i}{365} = 0,5243$$

et

$$P(A_{23}) = 0,4927$$

La réponse à la question est donc 23 personnes. Plusieurs ouvrages mentionnent ce paradoxe et ses retombées pour les attaques par collisions [Alfred J. Menezes 1996, Poli 2005].

Cet énoncé est remarquablement parlant et peut être repris pour exprimer plus clairement les propriétés de résistance des fonctions de hachage. Imaginons une classe d'élèves, les définitions deviendraient :

Exemple 4 : *Une **collision** signifie trouver deux élèves nés le même jour.*

Exemple 5 : *Trouver une **pré-image** s'apparente à définir une date puis de trouver un élève qui soit né à cette date.*

Exemple 6 : *Trouver une **seconde pré-image** s'apparente au fait de choisir un élève puis de trouver un second élève qui soit né le même jour.*

Il est intéressant de remarquer qu'en pratique, chaque empreinte générée est désirée dépendante uniquement d'un seul message d'entrée. Cependant, étant donné la taille de l'ensemble de départ et la taille de l'ensemble d'arrivée, il existe un nombre incommensurable de collisions et de pré-images. La sécurité d'une fonction de hachage repose donc sur le fait qu'il soit *calculatoirement impossible* d'attaquer ses propriétés cryptographiques. Ce seuil calculatoire est dépendant de la croissance des puissances de calcul et est aujourd'hui consensuellement admis inatteignable à 2^{80}.

Ces travaux de thèse se sont principalement focalisés sur les fonctions de hachage de la famille MD [7] et SHA, toutes construites sur le schéma de construction de *Merkle-Damgård* que nous présentons ci-après. Les descriptions sont assez techniques mais néanmoins nécessaires pour avoir la compréhension des chapitres suivants.

3.3 Construction de Merkle-Damgård

Les fonctions de hachage cryptographiques les plus utilisées reposent sur le schéma de construction de Merkle-Damgård [Merkle 1989, Damgård 1989]. Cette construction consiste en x itérations d'une fonction de compression $\mathcal{F}$, appliquée sur claque bloc m_i en provenance d'un message d'entrée m et sur le résultat de la précédente compression. Un mot H de n bits, qu'on appellera ensuite l'état interne, est initialisé (H_0) puis modifié à chaque compression. Le dernier état interne calculé (H_x) donne l'empreinte. La figure I.1 est une représentation de cette construction.

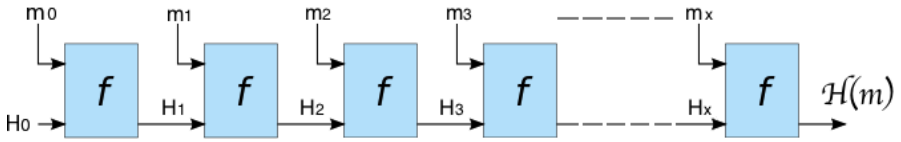

FIGURE I.1 – Construction de Merkle-Damgård

3.3.1 La famille MD

À propos de MD4 : MD4 [Rivest 1990a] a été conçu en 1990 par Ron Rivest et repose sur la construction de Merkle-Damgård. L'algorithme déroule un processus de hachage en 48 étapes où 4 mots de 32 bits appelés *états* $\in \{A,B,C,D\}$ sont modifiés à chacune des étapes. À la fin, leurs concaténations donnent une empreinte de 128 bits. Une étape est retranscrite sur la figure I.2 et représente le calcul suivant :

7. En réalité, nous évoqueront les termes « famille MD » uniquement pour aborder à la fois MD4 et MD5, même s'il existe MD2 ou bien MD6.

$$Q_i \leftarrow f_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) + M[j] + Cst_i$$

$$Q_i \leftarrow Q_i \lll r_i$$

Où :

- i est l'étape courante, $\in \{ 1,2,\ldots,48 \}$.
- Q_i est l'état courant, Q $\in \{A,B,C,D\}$.
- Q_{-3}, Q_{-2}, Q_{-1} et Q_0 sont les valeurs initiales (I.V.), telles que Q_{-3} = `0x01234567` ; Q_{-2} = `0x89ABCDEF` ; Q_{-1} = `0xFEDCBA98` ; Q_0 = `0x76543210`
- $M[j]$ est le j^{ieme} mot de 32 bits, $j \in \{ 0,1\ldots15 \}$, en provenance du message d'entrée M.
- Cst_i une constante qui a pour valeur : $i \leq 16$: `0x00000000` ; $17 \leq i \leq 32$: `0x5A827999` ; $i \geq 33$: `0x6ED9EBA1`
- f_i une fonction non-linéaire $\in$ { F, G, H }.
- $\lll r_i$ est un décalage circulaire de r_i bits sur la gauche.

Les fonctions non-linéaires sont définies par :

- F(x,y,z) = (x $\wedge$ y) $\vee$ ($\overline{x}$ $\wedge$ z) (*IF*)
- G(x,y,z) = (x $\wedge$ y) $\vee$ (x $\wedge$ z) $\vee$ (y $\wedge$ z) (*MAJ*)
- H(x,y,z) = x $\oplus$ y $\oplus$ z (*XOR*)

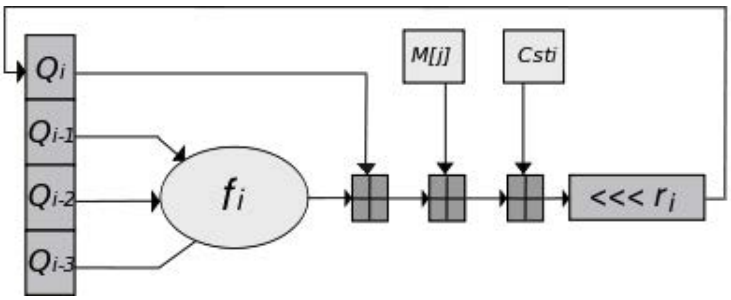

FIGURE I.2 – Une étape du processus de MD4

Le détail complet de cette fonction de hachage peut être trouvé dans sa documentation technique [Rivest 1990b]. MD4 a été affaibli très rapidement [Boer 1991] [Dobbertin 1996] jusqu'à être facilement cassé par une attaque différentielle permettant de générer des collisions en 2005 [Wang 2005] en 2^8 opérations MD4. Cette attaque a ensuite été améliorée jusqu'à une complexité évaluée à moins de deux calculs de MD4 [Sasaki 2007]. La préimage de MD4 a été attaquée en 2008 par Gaëtan Leurent [Leurent 2008] avec une attaque théorique évaluée à 2^{102}. Depuis 2011, l'IETF [8] a classé MD4 comme une fonction de hachage obsolète [Turner 2011].

8. Pour « Internet Engineering Task Force »

À propos de MD5 : MD5 [Rivest 1991] a été conçu en 1991 par Ron Rivest, comme une évolution de MD4, sur lequel a été ajouté quelques améliorations permettant de le renforcer. Cette fonction fournit aussi une empreinte de 128 bits mais le principe opératoire comporte désormais 64 étapes où les états sont initialisés [9] avant d'être modifiés à chacune des étapes par le calcul suivant :

$$Q_i \leftarrow Q_{i-4} + f_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) + M[j] + Cst_i$$

$$Q_i \leftarrow Q_i \lll r_i$$

$$Q_i \leftarrow Q_i + Q_{i-1}, i \in \{ 1,...,64 \}$$

Où :

- i est l'étape courante, $\in \{ 0,1,\ldots,64 \}$.
- Q_i est l'état courant, Q $\in \{A,B,C,D\}$.
- Q_{-3}, Q_{-2}, Q_{-1} et Q_0 sont les valeurs initiales (I.V.), telles que Q_{-3} = `0x01234567` ; Q_{-2} = `0x89ABCDEF` ; Q_{-1} = `0xFEDCBA98` ; Q_0 = `0x76543210`
- $M[j]$ est le j^{ieme} mot de 32 bits, $j \in \{ 0,1,\ldots,15 \}$, en provenance du message d'entrée M.
- Cst_i parmi 64 constantes prédéfinies.
- f_i une fonction non-linéaire $\in$ { F, G, H, I }.
- $\lll r_i$ est un décalage circulaire de s bits sur la gauche.

Les fonctions non-linéaires sont définies par :

- F(x,y,z) = (x $\wedge$ y) $\vee$ ($\overline{x}$ $\wedge$ z) (IF)
- G(x,y,z) = F(z,x,y) (IF)
- H(x,y,z) = x $\oplus$ y $\oplus$ z (XOR)
- I(x,y,z) = y $\oplus$ (x $\vee$ $\overline{z}$)

Une étape pour MD5 est représentée sur la figure I.3. À la fin du processus, une dernière addition est faite entre les états de l'étape 64 et les $I.V.$ De cela, en résulte l'empreinte.

Dès 2005, plusieurs collisions ont été trouvées pour la fonction de hachage MD5 grâce à une attaque différentielle [Wang 2005]. Depuis, plusieurs références en matière de cryptanalyse sur cette fonction ont vu le jour. Sa résistance aux collisions est donnée à 2^{39} opérations [Stevens 2012b] et sa résistance à la pré-image est évaluée à une complexité de l'ordre de $2^{122.97}$ [Bettale 2011]. MD5 reste cependant une des fonctions de hachage les plus utilisées.

9. Dans la suite, nous écriront $I.V.$ pour désigner ces variables.

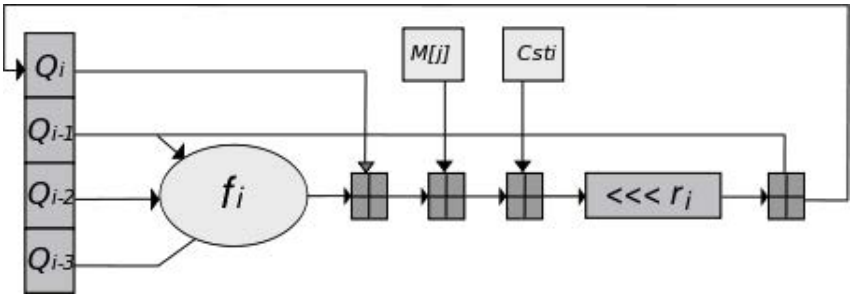

FIGURE I.3 – Une étape du processus de MD5

3.3.2 SHA-0 et SHA-1

SHA-0 a été inventé par la NSA en 1993 puis renforcé deux ans après pour donner SHA-1 afin de prévenir quelques faiblesses. Le principe opératoire est le même que celui de MD4 et consiste en un processus de hachage où cinq états $\in \{A,B,C,D,E\}$ sont initialisés puis modifiés à travers 80 étapes. L'empreinte en sortie est de taille 160 bits. Une étape de SHA-1 est représentée sur la figure I.4 et est déterminée de la façon suivante :

$$TMP \leftarrow (A_{i-1} \lll 5) + f(B_{i-1},C_{i-1},D_{i-1}) + E_{i-1} + M[i] + Cst_j$$

$$E_i \leftarrow D_{i-1}\,; D_i \leftarrow C_{i-1}\,; C_i \leftarrow (B_{i-1} \lll 30)$$

$$B_i \leftarrow A_{i-1}\,; A_i \leftarrow TMP\,; E_i \leftarrow D_{i-1}$$

Où :

- i est l'étape courante, $\in \{\ 0,1,...,79\ \}$.
- TMP est une variable temporaire.
- Cst_j est une constante parmi 4 constantes prédéfinies.
- $\lll s$, est un décalage circulaire de s bits sur la gauche.
- f est une fonction non-linéaire $\in \{$ F, G, H, I $\}$.
- $M[i]$ est le i^{ieme} mot de 32 bits, construit à partir du message d'entrée M.

Les fonctions non-linéaires sont définies par :

- F(x,y,z) = (x $\wedge$ y) $\vee$ ($\overline{x}$ $\wedge$ z) (IF)
- G(x,y,z) = x $\oplus$ y $\oplus$ z (XOR)
- H(x,y,z) = (x $\wedge$ y) $\vee$ (x $\wedge$ z) $\vee$ (y $\wedge$ z) (MAJ)
- I(x,y,z) = G(x,y,z) (XOR)

La construction des $M[i]$ est réalisée par :

- si $i < 16$
  $M[i]$ est le i^{ieme} mot de 32 bits à partir du message d'entrée.
- si $16 \leq i \leq 79$
  $M[i] \leftarrow (M[i-3] \oplus M[i-8] \oplus M[i-14]$
  $\oplus M[i-16]) \lll 1$

SHA-1 diffère de SHA-0 par le décalage circulaire ajouté dans la construction des $M[i]$ évitant ainsi de reproduire l'attaque d'Antoine Joux et Florent Chabaud [Chabaud 1998] qui consiste en la recherche de collisions par une cryptanalyse différentielle. Aujourd'hui, la meilleure attaque contre un SHA-1 complet reste une recherche de collisions en 2^{61} opérations [Stevens 2013]. Sa résistance à la pré-image est quant à elle estimée à 45 étapes [Cannière 2008] pour une pseudo-pré-image partielle. Notons aussi les travaux de [Manuel 2011] qui donnent une classification des vecteurs de perturbations utilisés dans les attaques par chemins de différentiels pour la recherche de collision.

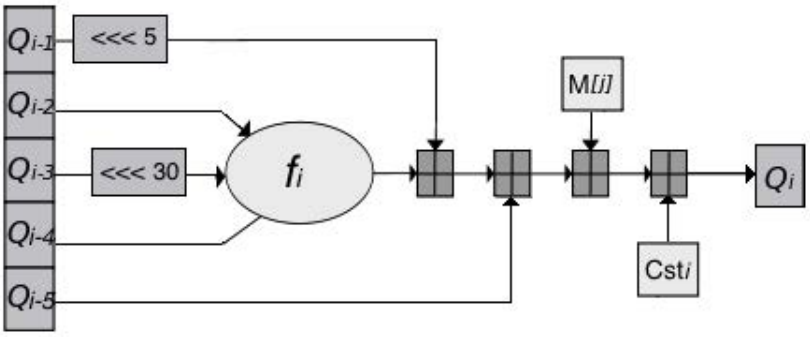

FIGURE I.4 – Une étape du processus de SHA-1

3.4 Le nouveau standard SHA-3

En 2012, le NIST a sélectionné KECCAK [Bertoni 2011] [NIST 2012a] comme étant le vainqueur du concours lancé en 2007, visant à concevoir le nouveau standard en matière de fonctions de hachage. Ainsi, KECCAK devint SHA-3 et permet de concurrencer le standard SHA-2 qui est toujours de rigueur étant donné qu'aucune attaque pertinente n'a encore été réalisée contre lui. Les spécifications qui suivent ont principalement été reprises du site conçu par les auteurs de KECCAK : $http://keccak.noekeon.org/$.

SHA-3 utilise le principe des *fonctions éponges* illustré figure I.5. L'état interne est composé de $r + c$ bits : les r premiers bits étant appelés le *bitrate* et les c bits qui suivent, la *capacité*. Une première phase est dédiée à *l'absorption* ; les premiers blocs du message d'entrée sont XOR-és ($\oplus$) dans l'état interne de la fonction. Puis une fonction de compression est appliquée sur l'état interne. Ce processus est répété autant de fois que nécessaire, jusqu'à l'absorption complète du message d'entrée. Dans une seconde phase, l'état interne est *essoré* pour obtenir des bits en sortie. Concrètement, le *bitrate* est gardé de côté. S'il y a assez de bits pour constituer l'empreinte, alors le processus s'arrête et l'empreinte finale est obtenue, sinon la fonction de compression est de nouveau appliquée sur l'état interne et de nouveau, le *bitrate* est mis de côté. L'empreinte finale sera donnée par la concaténation de tous ces bits gardés en sortie.

Techniquement, plusieurs paramètres spécifient les caractéristiques de la fonction de hachage KECCAK. Voici un résumé de ceux définis pour le standard SHA-3 :

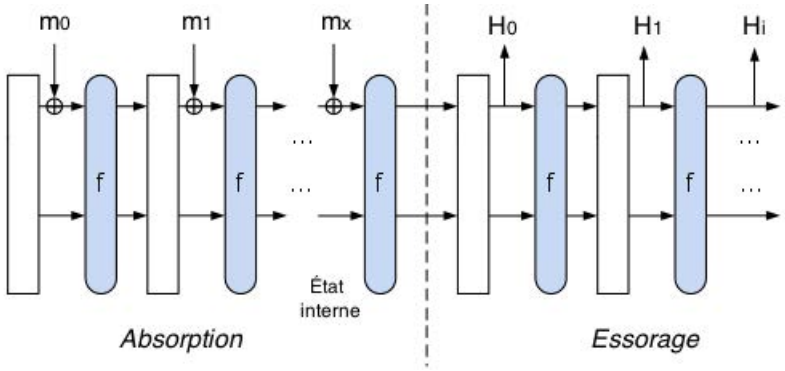

FIGURE I.5 – Principe des fonctions éponges

- L'état interne dans SHA-3 est un mot de 1600 bits, découpé en une matrice 5*5 de mots de 64 bits. Soit $a[i][j][k]$ le bit $(i*5+j)*w+k$ du message d'entrée.
- La fonction de compression comprend 24 étapes, découpées en 5 sous-fonctions :
  - θ : *Calcul de parité*

    $$a[i][j][k] = a[i][j][k] \oplus parité(a[0..4][j-1][k]) \oplus parité(a[0..4][j+1][k-1])$$

  - ρ : *Décalage de bits circulaire*

    Pour, $0 \leq t < 24$, $a[i][j][k] = a[i][j][k-(t+1)*(t+2)/2]$ avec :

    $$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

  - π : *Permutation*

    $$a[j][2i+3j] = a[i][j]$$

  - χ : *Fonction non-linéaire*

    $$a = a \oplus (\overline{b} \wedge c), \text{ avec :}$$

    $$a = a[i][j][k]; b = \overline{a[i][j+1][k]}; c = a[i][j+2][k].$$

  - ι : *Injection d'une constante*

    $$a[0][0] = a[0][0] \oplus Cst_i$$

De part sa récence et ses nombreuses évaluations pour la compétition, la fonction de hachage SHA-3 est sûre contre toutes attaques connues. Dans le cadre de la cryptanalyse logique, seuls les travaux de [Morawiecki 2013] ont permis de trouver une pré-image partielle pour deux étapes. Des challenges de pré-image partielles et de collisions partielles sont mis en jeu par les auteurs de KECCAK sur leur site [10] et permettent une veille sur les travaux en cours.

10. http ://keccak.noekeon.org/crunchy_contest.html

Cette présentation succincte de SHA-3 clos le volet cryptographie de ce premier chapitre. Nous avons ainsi dressé les contours du contexte dans lesquels évoluent nos travaux de recherche, du strict point de vue de la cryptanalyse. La partie suivante est dédiée à d'autres préliminaires concernant le second faciès de notre étude, à savoir le domaine de la satisfaisabilité.

4 Le problème de satisfaisabilité

Le problème de satisfaisabilité, ou plus couramment appelé « problème SAT » , repose sur la logique propositionnelle, introduite par George Boole [Boole 1847], il y a désormais plus de 150 ans. Cette algèbre binaire permet, de part sa simplicité, d'exprimer facilement un très grand nombre de problèmes. Une définition formelle du problème SAT est :

Etant donné une formule booléenne l'objectif de trouver une affectation qui satisfait la formule.

Définition 17 : *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. Le problème* SAT *est le problème de décision qui détermine s'il existe une interprétation des variables $\mathcal{V}$ de sorte que $\mathcal{F}$ soit consistante.*

Le problème SAT est un problème combinatoire de décision. C'est d'abord un problème académique en informatique théorique, devenu fondamental en 1971 pour la théorie de la complexité quand Stephen Cook établi la notion de NP-complétude [Cook 1971]. Dès lors, les recherches autour de la résolution du problème SAT ont été nombreuses et variées, apportant des algorithmes de résolution novateurs et rapides, à la fois pour des problèmes aléatoires fortement liés à la théorie de la complexité que pour des problèmes académiques. Ajouté à cela, les problèmes issus de cas réels, comme le sont ceux du monde industriel où les contraintes décisionnelles et d'optimisation sont importantes (planification [Kautz 1996], model checking [Biere 2006], etc.), font aussi office de nombreuses études par la communauté.

Des ouvrages reconnus [Saïs 2008] [Biere 2009] [Knuth 2011] très détaillés ont été consacrés à l'étude du problème SAT, de ses aspects fondamentaux jusqu'aux plus récentes techniques de résolution. Cette section n'aura donc pas la prétention de résumer ses longues années de recherche et se contentera uniquement de donner les clés pour définir la logique propositionnelle et par conséquent les contours du problème SAT.

4.1 Logique propositionnelle

La logique propositionnelle est un langage muni d'un alphabet comprenant un ensemble de variables $\mathcal{V}$, les connecteurs logiques « NON » ($\neg$), « ET » ($\wedge$), « OU » ($\vee$) ainsi que les constantes FAUX et VRAI. Les symboles « (» et «) » sont couramment utilisés pour annihiler toute ambiguité de priorité lors de l'évaluation d'une formule propositionnelle.

Définition 18 : ***Une proposition*** *est une affirmation mathématique résultant d'une logique bivalente ayant une valeur de vérité, pouvant soit être* VRAIE*, soit être* FAUSSE*.*

Exemple 7 : *La proposition suivante : « Le soleil est une étoile » est* VRAIE*, mais la proposition « La Terre est plate » est* FAUSSE*.*

Définition 19 : ***Une variable propositionnelle*** $v \in \mathcal{V}$*, aussi appelée* ***variable booléenne****, est une variable qui prend une valeur parmi* $\{\text{FAUX}, \text{VRAI}\}$*, pouvant respectivement être notées* $\{\text{FALSE}, \text{TRUE}\}$*,* $\{0,1\}$ *ou* $\{\bot, \top\}$*.*

Définition 20 : ***Une affectation*** *est l'action d'associer une valeur à une variable propositionnelle. Le symbole couramment utilisé pour représenter cette relation est «* $\leftarrow$ *» .*

Exemple 8 : *Soit* $a \in \mathcal{V}$ *une variable booléenne. La règle «* $a \leftarrow$ VRAI *» signifie « a est affecté à* VRAI *» .*

Définition 21 : ***Un littéral*** *est une variable propositionnelle signée positivement ou négativement. Soit* $v \in \mathcal{V}$ *une variable booléenne. Le littéral positif est noté v et le littéral négatif est noté* $\neg v$ *ou* $\overline{v}$*.*

Définition 22 : *Un littéral v est dit* ***satisfait*** *(respectivement* ***contredit****) quand il est positif et que la variable booléenne associée est affectée à* VRAI *(respectivement à* FAUX*) ou quand il est négatif et que la variable booléenne associée est affectée à* FAUX *(respectivement à* VRAI*).*

Définition 23 : *On dit qu'une variable est* ***retournée*** *ou* ***flippée*** *quand sa valeur de vérité est inversée,* i.e. *changée de* VRAI *à* FAUX *ou de* FAUX *à* VRAI*.*

Définition 24 : *Un littéral est dit monotone (ou pur) quand il ne se présente dans une formule propositionnelle que sous une signature (positive ou négative).*

Définition 25 : *Soit v un littéral.* ***Le connecteur unaire « NON »*** *change la signature du littéral et inverse la valeur affectée à la variable booléenne correspondante. Le littéral sera alors noté* $\neg v$ *ou* $\overline{v}$*. La table de vérité associée à la négation est donnée table I.1.*

v	$\overline{v}$
FAUX	VRAI
VRAI	FAUX

TABLE I.1 – Table de vérité pour la négation

Définition 26 : ***Le connecteur binaire « ET »*** *,* $\wedge$*, représente la conjonction entre deux propositions. La conjonction résultante vaudra* VRAI *si et seulement si les deux propositions sont affectées à* VRAI*. Soit A et B, deux propositions. La table de vérité associée à la conjonction est la table I.2.*

A	B	$A \wedge B$
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

TABLE I.2 – Table de vérité pour la conjonction

Définition 27 : ***Le connecteur binaire « OU »*** *, noté* $\vee$*, représente la disjonction entre deux propositions. La disjonction résultante vaudra* VRAI *si au moins une des deux propositions est affectée à* VRAI*. Soit A et B, deux propositions. La table de vérité associée à la disjonction est la table I.3.*

A	B	$A \vee B$
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

TABLE I.3 – Table de vérité pour la disjonction

Définition 28 : ***Une clause*** *est une disjonction de littéraux. Elle est dite satisfaite si et seulement si au moins un de ses littéraux est satisfait. Sinon elle est contredite.*

Exemple 9 : $a \vee b \vee \overline{c}$ *est une clause contenant trois littéraux. Cette clause est satisfaite si au moins* $a \leftarrow$ VRAI *ou* $b \leftarrow$ VRAI *ou* $c \leftarrow$ FAUX*.*

Remarque 1 : *Une clause qui ne contient pas de littéraux est une clause vide et elle est contredite.*

Remarque 2 : *Une clause est dite* ***unitaire*** *si elle ne contient qu'un seul littéral.*

Remarque 3 : *Une clause est dite* ***n-aire*** *si elle contient exactement n littéraux. On parlera aussi de n-cl pour caractériser une clause de taille n.*

Définition 29 : *Si une clause est unitaire alors la seule façon de la satisfaire est d'affecter à son unique littéral la seule et unique valeur qui la rende* VRAI*. Dans le cadre d'une résolution consistant à rechercher une solution à la formule, cette affectation sera forcée, indépendante de tout choix et appelée* ***propagation unitaire****.*

Remarque 4 : *La propagation unitaire peut déclencher d'autres propagations unitaires en cascade.*

Exemple 10 : *Soient deux clauses c1 :* (a) *et c2 :* $(\overline{a} \vee b)$*. c1 ne peut être satisfaite qu'en affectant la variable a à* VRAI*. Par conséquence, c2 ne peut être satisfaite qu'en affectant la variable b à* VRAI*, sans qu'il n'y ait d'autres choix possibles.*

Définition 30 : ***Une formule $\mathcal{F}$ sous Forme Normale Conjonctive*** *(*CNF*) est une conjonction de clauses.*

Exemple 11 : *$\mathcal{F} : (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee \overline{b})$ est une formule* CNF *contenant deux clauses.*

Définition 31 : *Une formule $\mathcal{F}$ est dite* ***satisfaite*** *si et seulement si toutes ses clauses sont satisfaites. Sinon elle est* ***contredite*** *ou* ***insatisfaite****.*

Remarque 5 : *Une formule propositionnelle qui ne contient pas de clauses est satisfaite.*

Définition 32 : *Soit une formule propositionnelle $\mathcal{F}$ munie des clauses $\mathcal{C}$ et des variables $\mathcal{V}$. $\mathcal{F}$ est dite* ***simplifiée*** *dès lors qu'une variable $v \in \mathcal{V}$ est affectée à* VRAI *(respectivement* FAUX*). Dans ce cas, les clauses $c \in \mathcal{C}$ contenant le littéral v (respectivement $\overline{v}$) sont satisfaites ; les clauses $c' \in \mathcal{C}$ contenant le littéral $\overline{v}$ (respectivement v) sont réduites tel que $c' \leftarrow c' \setminus \{\overline{v}\}$. Les autres clauses restent inchangées.*

Définition 33 : *Une* ***inférence logique*** *est une règle reliant des propositions qui aboutissent à une déduction logique, dite conclusion, préservant la vérité.*

Définition 34 : *Une formule propositionnelle est considérée* ***irréductible*** *quand plus aucune propagation unitaire ne peut être réalisée ou déduite et qu'aucun littéral monotone ne subsiste.*

Définition 35 : ***Une interprétation*** *(complète) I est une fonction qui associe à chaque variable de $\mathcal{V}$ une valeur dans* { VRAI, FAUX }*. Cette interprétation est dite partielle si au moins une variable n'a pas de valeur affectée. Donner une interprétation complète pour $\mathcal{V}$ permet de donner une interprétation à $\mathcal{F}$.*

- *Si I permet de satisfaire $\mathcal{F}$ alors I est* ***un modèle*** *ou* ***une solution*** *de $\mathcal{F}$ et on note $I \models \mathcal{F}$.*
- *Si I ne satisfait pas $\mathcal{F}$ alors I est* ***un contre-modèle*** *de $\mathcal{F}$ et on note $I \not\models \mathcal{F}$.*

Définition 36 : *Une formule est dite* ***consistante*** *si elle admet au moins un modèle. Sinon elle est* ***inconsistante****.*

Remarque 6 : *Dans ce manuscrit, on appellera formule* SAT *une formule que l'on sait consistante et une formule* UNSAT *une formule que l'on sait inconsistante.*

4.2 Problème SAT

Nous donnons ici une définition plus formelle pour le problème SAT :
« *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. Le problème* SAT *est le problème de décision qui détermine s'l existe une interprétation des variables $\mathcal{V}$ de sorte que $\mathcal{F}$ soit consistante.* »

Définition 37 : *Un problème* SAT *exprimé uniquement avec des clauses de deux littéraux est appelé problème 2-*SAT*. Ce problème se résout en temps polynomial [Aspvall 1979].*

Définition 38 : *Un problème* SAT *exprimé avec au moins une clause contenant k littéraux, $k > 2$, est appelé problème k-*SAT. *Ce problème est classé dans la catégorie NP-complet.*

Remarque 7 : *Tout problème* SAT *peut être réduit polynomialement à un problème 3-*SAT.

Le problème SAT a été le premier problème prouvé NP-complet, en 1971 par Stephen Arthur Cook. De ce fait c'est un problème très important dans le domaine de la théorie de la complexité. De plus, de nombreux algorithmes de résolution ont vu le jour au fil des années et sont décrits dans ce qu'on appelle des **prouveurs** SAT ou des **solveurs** SAT. La communauté distingue deux classes de techniques pour résoudre le problème SAT : *les méthodes complètes* et *les méthodes incomplètes*. Il est important pour la suite de détailler ces deux approches.

4.3 Les méthodes complètes

Les approches complètes garantissent, dans un espace illimité, une réponse au problème de décision dans un temps de calcul fini. Cependant, ce temps de réponse est exponentiellement dépendant du nombre de variables du problème considéré. En effet, ces méthodes consistent en une énumération systématique et exhaustive des affectations grâce à une structure d'arbre binaire. Ainsi, pour une formule propositionnelle inconsistante de n variables, dans le pire des cas toutes les affectations doivent être interprétées avant d'avoir une réponse au problème de décision, ce qui est inévitablement très coûteux puisque 2^n interprétations sont possibles. En pratique, si un problème est de grande taille, difficile et qu'il n'admet pas de solution alors il sera impossible de le savoir dans un temps raisonnable. Par ailleurs, si un problème difficile de grande taille admet une solution, alors il devient possible d'obtenir cette réponse dans un temps raisonnable, l'algorithme n'ayant pas à explorer tout l'arbre de recherche. Par difficile, nous entendons ici qu'il n'existe pas de sous-formules dans le problème initial pouvant amener très rapidement à la contradiction.

4.3.1 L'algorithme DPLL

L'algorithme de résolution DPLL [Davis 1962] [11] datant de 1962 est une amélioration de la procédure DP [Davis 1960] conçue deux ans auparavant. DPLL un algorithme récursif qui consiste à déterminer si une formule propositionnelle admet une solution, en exploitant la notion de backtracking, la propagation unitaire (qui n'existait pas dans [Davis 1960]) et la détection des littéraux purs.

Concrètement, à chaque étape de la résolution, une variable est choisie en fonction d'une heuristique de branchement et une valeur de vérité lui est assignée. De ce choix, diverses inférences logiques vont simplifier la formule jusqu'à obtenir une formule irréductible. Dès lors, une interprétation de cette formule déterminera si la formule est satisfaite ou non. Si ce n'est pas le cas, une nouvelle étape sera exécutée.

11. du nom de leurs auteurs Martin **D**avis, Hilary **P**utnam, George **L**ogemann et Donald **L**oveland.

Lorsqu'une clause est vide, et donc contredite, l'algorithme exécute un backtracking, c'est-à-dire qu'il désaffecte la dernière variable assignée dans la pile d'exécution puis l'affecte à la valeur opposée. Si une clause vide apparaît de nouveau, le backtracking « remonte » dans l'arbre de recherche jusqu'à la variable précédemment affectée. Et ainsi de suite.

L'algorithme 1 décrit la procédure DPLL et la figure I.6 montre l'aperçu d'un arbre binaire de recherche pour cette procédure. Ici le backtracking se fait de nœud en nœud.

Algorithme 1: Procédure DPLL

Données : $\mathcal{F}$: une formule CNF, $\mathcal{C}$ ses clauses, $\mathcal{V}$ ses variables

si *(*$\mathcal{F} = \emptyset$ *)* **alors**
  Retourne VRAI
fin
si *(*$\exists\, c \in \mathcal{C} / c = \emptyset$*)* **alors**
  Retourne FAUX
fin
si *(*$\exists\, c \in \mathcal{C} / c$ *est unitaire)* **alors**
  $\mathcal{F} \leftarrow$PROGAGER($\mathcal{F}, v \in c$)
fin
si *(*$\exists\, v \in \mathcal{V} / v$ *est pur)* **alors**
  $\mathcal{F} \leftarrow$PROGAGER($\mathcal{F}, v$)
fin
CHOISIR($v \in V$) *// Choix heuristique*
si *(*DPLL*(*v*)* = VRAI*)* **alors**
  Retourne VRAI
fin
sinon
  Retourne DPLL($\overline{v}$)
fin

De nombreux solveurs SAT complets sont basés sur la procédure DPLL et se différencient principalement par une spécification au niveau de l'heuristique de branchement. Quelques ouvrages spécialisés permettent d'approfondir le sujet en détail comme par exemple [Edelkamp 2012].

Ainsi, au cours des dernières années, plusieurs heuristiques efficaces ont permis d'améliorer significativement la résolution du problème SAT. Citons simplement quelques une d'entre elles qui ont prouvé leurs efficacités à travers différents solveurs comme par exemple MOMS [Freeman 1995] et ses déclinaisons, UP [Li 1997] dans SATZ, VSIDS [Moskewicz 2001] dans CHAFF, BSH [Dequen 2001] dans KCNFS [Dequen 2006], etc.

Notons aussi d'autres types d'évolutions comme l'apport de structures paresseuses [Moskewicz 2001] pour la propagation unitaire et l'utilisation de clauses

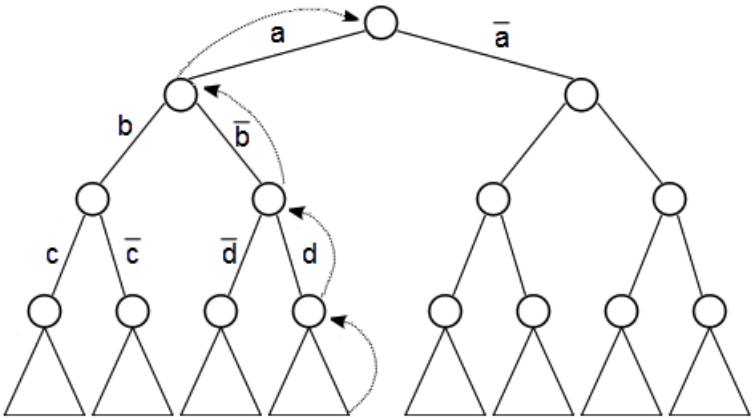

FIGURE I.6 – Arbre de résolution pour la procédure DPLL

conflits [Zhang 2001]. Ces avancées ont notamment permis l'éclosion de nouveaux solveurs mettant en avant le principe de l'analyse de conflits.

4.3.2 L'analyse de conflits

L'analyse de conflits repose sur les conflits découverts lors de la résolution. En effet, quand une clause est contredite alors l'ensemble des affectations qui a amené à ce conflit peut être retranscrit sous forme logique. Considérons une formule propositionnelle $\mathcal{F}$ et imaginons qu'assigner deux variables a et b à VRAI implique un conflit. Sous une forme logique, cela se traduit par : $(a \wedge b) \Rightarrow$ FAUX. Puisque nous recherchons un modèle pour $\mathcal{F}$, alors il est nécessaire d'éviter d'avoir de nouveau a et b affectées à VRAI en même temps. Pour cela, une nouvelle clause représentant la négation du conflit, qu'on appelle clause conflit, peut être ajoutée à la formule $\mathcal{F}$. Ainsi, dans notre exemple, la clause conflit $(\overline{a} \vee \overline{b})$ sera ajoutée.

Le fait de considérer l'affectation des variables qui ont conduit au conflit, et non plus seulement la dernière, est appelé **backtracking non-chronologique** ou **backjumping**. L'ajout de clauses conflit est quant à lui plus communément appelé **apprentissage**. Ces deux notions ont permis l'émergence de nouveaux solveurs de type CDCL (pour **C**onflict **D**riven **C**lause **L**earning). La figure I.7 décrit de façon schématique la procédure CDCL et son backjumping.

Les solveurs CDCL se sont notamment avérés extrêmement intéressants pour la résolution de problèmes de grandes tailles, tels les problèmes issus du monde réel, comme par exemple le problème de plannification [Kautz 1996], la conception de circuit [Silva 2000], le model checking [Biere 2006] ou bien la cryptographie [Massacci 2000, Kullman 2002, Potlapally 2007]. Quelques solveurs CDCL réputés, ou qui ont servis de base pour d'autres solveurs sont GRASP [Silva 1999], CHAFF [Moskewicz 2001], MINISAT [Sörensson 2005], GLUCOSE [Audemard 2013], LINGELING [Biere 2010]. À noter le solveur de type CDCL CRYPTOMINISAT 2 dédié aux problèmes cryptographiques. Ce solveur étant fortement lié à notre thématique de recherche, nous proposons de nous arrêter un instant sur sa spécification.

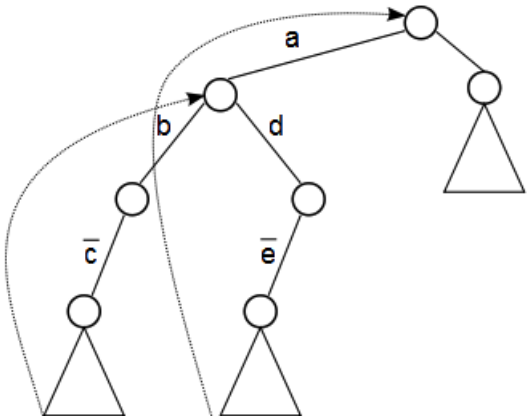

FIGURE I.7 – Arbre de résolution pour la procédure CDCL

4.3.3 À propos de CryptoMinisat 2

CRYPTOMINISAT 2 [Soos 2009] [Soos 2010a] est un solveur récent puisqu'il a été conçu par Mate Soos dans sa première version il y a un peu plus de 4 ans et qu'il est toujours en perpétuelle évolution [12]. Ce solveur, de type CDCL a comme base le solveur MINISAT. Une de ses particularités réside dans sa façon de gérer séparément les contraintes de type XOR ($\oplus$), c'est-à-dire les contraintes qui ne peuvent être satisfaites que par un nombre impair de littéraux. Pour cela, le format DIMACS [13] a été étendu pour exprimer les contraintes XOR en précédant cette contrainte du caractère « x » . CRYPTOMINISAT 2 a la capacité de repérer ces clauses et utilise l'élimination gaussienne pour améliorer la résolution de ces contraintes spécifiques. Plus d'informations peuvent être trouvées dans [Soos 2010b].

Pour conduire à bien nos travaux de recherche, nous avons privilégié l'utilisation de LINGELING dans sa version parallèle nommée PLINGELING. Le choix du solveur pour résoudre nos problèmes cryptographiques a été réalisé en testant leurs performances sur nos formules CNF [14]. Bien qu'il semblerait que ce choix soit déterminant pour optimiser la qualité de nos solutions, nous verrons plus tard que finalement la plupart des meilleurs solveurs actuels - y compris CRYPTOMINISAT 2 - admettent des performances équivalentes sur nos problèmes.

4.4 Approche incomplète

4.4.1 Principes

Les méthodes incomplètes sont ainsi nommées car elle ne parcourent pas l'entièreté de l'espace de recherche. Les plus courantes reposent sur des métaheuristiques, telles la recherche locale, le recuit simulé, les algorithmes génétiques etc. Dans ce cadre, le résultat

12. On notera cependant que la dernière version CRYPTOMINISAT 3 n'est plus spécialisée pour les instances cryptographiques.
13. Ce format est utilisé pour représenter une CNF. Il est celui utilisé par convention par les solveurs SAT.
14. Ces tests ne sont pas décris dans cette thèse car ils dépassent le propos, même si une étude approfondie pourrait paraître intéressante.

ne peut pas être garanti en un temps fini et se situe par conséquent parmi les trois états suivants : { « vrai », « faux », « on ne sait pas » }. Concevoir un algorithme que l'on saura incomplet peut permettre l'utilisation de méthodes stochastiques, utilisant par conséquent des variables aléatoires. Dans le cas de la recherche locale on parlera alors de recherche locale stochastique et de solveurs **SLS** (plus d'informations dans [Hoos 2004]). L'algorithme de résolution pouvant « boucler » dans l'espace de recherche, son calcul est borné par une limite de temps et/ou de nombre de *flips*.

La relaxation de la garantie d'une réponse en un temps fini laisse ces méthodes se comporter *comme* des algorithmes polynomiaux. Alors qu'un solveur complet cherchera à contenir l'explosion combinatoire en évitant de descendre trop profondément dans l'arbre de recherche, un solveur incomplet n'aura pas à se restreindre car la garantie de réponse dans un temps raisonnable n'est plus. Dès lors, la complexité en temps de la résolution devient polynomiale puisqu'elle est déterminée soit par une limite de temps, soit par une limite en nombre d'opérations.

Ne pas avoir de réponse en un temps fini n'empêche cependant pas d'avoir une vue sur l'état d'une solution courante. Ainsi, même si le problème SAT est un problème de décision, les méthodes incomplètes permettent de donner une réponse au problème d'optimisation associé suivant : « *existe t'il une interprétation des variables $\mathcal{V}$ de sorte à ce qu'au moins n clauses $\in$ $\mathcal{F}$ soient satisfaites ?* » . Dans ce contexte, le nombre de clauses satisfaites (ou insatisfaites) permet en pratique d'avoir un indicateur sur l'état de la configuration donnée aux variables. Cet indicateur est intéressant quand on structure l'ensemble des affectations possibles en fonction de leur voisinage pour les algorithmes de recherche locale, puisqu'il permet de lui affilier une topologie du paysage de recherche. Ce paysage est un ensemble de sommets définis par la valeur des configurations, où les points culminants représentent une configuration qui a une valeur éloignée de la solution et un creux une configuration ayant une valeur proche de la solution. Dans ce dernier cas, on parlera alors d'optimum local et les creux les plus bas dans le paysage intégral représentent les optima globaux (voir figure I.8). Deux critères définissant la complexité du problème peuvent alors être donnés : soit en évaluant le coût de passage entre un sommet et un optimum global soit en comptant le nombre d'optima locaux (non-globaux). En effet, plus les optima locaux sont nombreux, plus le paysage sera « rugueux » et par conséquent trouver une solution sera difficile. Dans le cas contraire on parlera plutôt de paysages convexes.

Parmi les solveurs incomplets pour la résolution de SAT, nous pouvons citer GSAT, WALKSAT [Selman 1993], SPARROW [Tompkins 2011], CCASAT [Cai 2011], SATTIME [Li 2012] qui sont les principaux solveurs servant souvent de base pour d'autres implémentations. Durant nos expérimentations au chapitre IV, nous avons mis en place certaines stratégies sur le solveur WALKSAT afin d'améliorer son efficacité. La partie suivante est donc dédiée à sa présentation.

4.4.2 Walksat

WALKSAT est un solveur SLS qui a la particularité d'être performant pour la recherche locale de solution et qui permet une prise en main très facile. Un algorithme détaillant

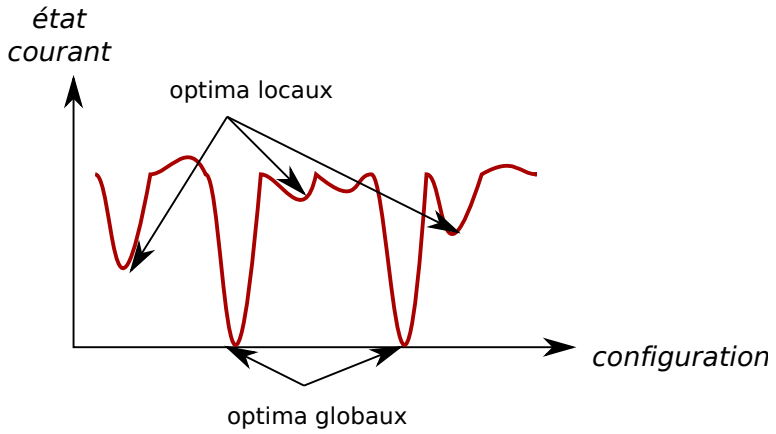

FIGURE I.8 – Notion de paysage de recherche

WALKSAT est donné dans le cadre Algorithme 2, où :

- INITIALISATION est une procédure qui affecte aléatoirement une valeur de vérité pour chaque variable dans {VRAI, FAUX}
- RANDOM($\mathcal{S}$) retourne aléatoirement un élément s parmi l'ensemble $\mathcal{S}$
- FLIP : inverse la valeur de vérité d'une variable.
- MISE_À_JOUR : met à jour l'ensemble des clauses non satisfaites

Algorithme 2: Procédure WALKSAT

```
   Données : F : une formule CNF, V ses variables
1  pour (i = 1 à MAX_TRY) faire
2      INITIALISATION(V)
       C = clauses insatisfaites
       pour (j = 1 à NB_FLIP) faire
3          si (C = ∅) alors
4              Retourne VRAI
5          fin
6          c = RANDOM(C) // une clause est choisie au hasard parmi les insatisfaites
           v = HEURISTIQUE(c)
           FLIP(v)
           MISE_À_JOUR(C)
7      fin
8  fin
9  Retourne FAUX
```

La première étape de WALKSAT est une étape d'initialisation où une valeur aléatoire parmi {VRAI, FAUX} est affectée à chaque variable. Ainsi, une part des clauses devient satisfaite et une autre part insatisfaite. L'algorithme choisit ensuite, au hasard, une clause

Algorithme 3: Heuristique BEST

Données : c : une clause, r un réel $\in$ [0,1], paramétré par l'algorithme

```
1 p = RANDOM([0,1]) // une probabilité comprise entre 0 et 1
  si (p > r ) alors
2 |   v = RANDOM(c) // une variable est choisie au hasard parmi celles composant la
  |   clause c
3 fin
4 sinon
5 |   v = la variable qui, si flippée, minimalise le rapport (nombre de clauses
  |   insatisfaites / nombre de clauses satisfaites)
6 fin
7 Retourne v
```

insatisfaite, puis l'heuristique entre en jeu pour sélectionner une variable dans cette clause. La valeur de vérité de la variable est alors flippée.

L'heuristique étant le cœur de l'algorithme de résolution, c'est elle qui va stratégiquement guider les choix et faire progresser la recherche. WALKSAT intègre plusieurs d'entre elles ; certaines sont très classiques, comme par exemple RANDOM, une recherche aléatoire, TABU, une recherche tabou qui s'interdit certains choix, ou bien BEST qui choisit dans une clause insatisfaite, la variable qui améliorera au mieux la solution. Enfin, NOVELTY est une heuristique un peu plus cossue qui permet de placer WALKSAT comme un solveur très efficace pour les problèmes aléatoires. Même si cette heuristique se décline en plusieurs variantes [15] nous ne traiterons que de la version la plus élaborée, RNOVELTY+, sommairement décrite dans l'algorithme 4. L'utilisation de cette heuristique dans WALKSAT se fait en remplaçant les lignes de 6 à 12 dans l'algorithme 2 par ces nouvelles lignes.

L'heuristique de choix RNOVELTY+ est la suivante. Soit p une probabilité. Suivant p, deux variables sont successivement élues de sorte qu'une fois flippées (de façon indépendante !), elles soient chacune celles qui minimisent le nombre de clauses insatisfaites. Parmi ces deux variables, une seule va être choisie en fonction de deux critères : la qualité d'amélioration de la solution et la récence de retournement des variables.

Ce processus est répété un nombre de fois paramétré à l'avance. Si une solution est trouvée, alors l'algorithme s'arrête et retourne la solution. Si aucune solution n'est trouvée alors l'algorithme recommence à partir de l'étape d'initialisation.

4.5 Les méthodes hybrides

Comme nous l'avons vu, les méthodes complètes et incomplètes présentent des avantages et des inconvénients. Il faut savoir que la communauté a trouvé naturel de les hybrider afin d'en tirer le meilleur. Par exemple, cela peut se faire via un échange d'information entre les deux méthodes : les clauses apprises dans une approche CDCL peuvent aider une

15. NOVELTY+, RNOVELTY et RNOVELTY+, chacune adoptant un certain paramétrage.

recherche locale à s'échapper des optimum locaux et la recherche locale peut aider la méthode complète étendant l'affectation partielle des variables. Ce paragraphe a pour simple objectif de les évoquer mais quelques références - liste non-exhaustive - peuvent intéresser le lecteur : [Crawford 1996], [Habet 2002], [Ferris 2004], [Fang 2004], [Balint 2009], [Audemard 2009], [Audemard 2010], [Letombe 2012].

Algorithme 4: Principe heuristique de RNOVELTY+

Données : r un réel $\in [0,1]$, paramétré par l'algorithme
c : une clause

p = RANDOM([0,1]) *// une probabilité comprise entre 0 et 1*
pour *(k = 1 à $|c|$)* **faire**
    On note respectivement v_1 et v_2 les variables qui, si indépendamment retournées, apporte respectivement la plus grande et la seconde plus grande différence entre le nombre de clauses satisfaites et le nombre de clauses insatisfaites.
fin
si *(v_1 n'est pas la plus récemment retournée)* **alors**
    $v = v_1$
fin
sinon
    si *($d_1 - d_2 \geq 2$)* **alors**
        $v = v_1$
    fin
    sinon
        si *($p > r$)* **alors**
            $v = v_1$
        fin
        sinon
            $v = v_2$
        fin
    fin
fin
Retourne v

4.6 Extensions du problème SAT

Le problème SAT se décline sous plusieurs variantes et extensions, chacune très étudiées par la communauté pour leurs intérêts théoriques et pratiques. Dans cette partie nous nous focaliserons uniquement sur les problèmes XOR-SAT, NAE-SAT, MAX-SAT et sur les problèmes de type CSP, qui nous intéresseront dans le chapitre III.

4.6.1 XOR-SAT

Définition 39 : *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. Le problème* XOR-SAT *est le problème de décision qui détermine si il existe une interprétation des variables $\mathcal{V}$ qui rende $\mathcal{F}$ consistante, de sorte que les clauses ne puissent être satisfaites que par un nombre impair de littéraux.*

4.6.2 NAE-SAT *(Not All Equal SAT)*

Définition 40 : *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. Le problème* NAE-SAT *est le problème de décision qui détermine si il existe une interprétation des variables $\mathcal{V}$ qui rende $\mathcal{F}$ consistante, de sorte que les clauses ne puissent pas être satisfaites par tous leurs littéraux.*

4.6.3 MAX-SAT

Définition 41 : *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. Le problème* MAX-SAT *est le problème d'optimisation qui essaie de trouver une interprétation des variables $\mathcal{V}$ de sorte à ce qu'un maximum de clauses soient satisfaites.*

Remarque 8 : *Le problème* SAT *est le cas particulier du problème* MAX-SAT *où la solution optimale est la satisfaction de toutes les clauses.*

Le problème MAX-SAT admet différentes déclinaisons.

- MAX-SAT *partiel* est le problème MAX-SAT où les clauses sont séparées entre des *clauses dures* qui **doivent** être satisfaites et des *clauses souples* que l'on cherche à satisfaire au maximum.
- MAX-SAT *pondéré* qui consiste à différencier les clauses par un poids qui leur est propre. L'objectif est de satisfaire les clauses de sorte à maximiser le poids total.
- MAX-SAT *partiel pondéré* combine les deux précédents concepts.

4.6.4 CIRCUIT-SAT

Définition 42 : *Soit $\mathcal{F}$ une formule propositionnelle, munie d'un ensemble de variables $\mathcal{V}$ et d'un ensemble de clauses $\mathcal{C}$. $\mathcal{F}$ modélise* ***un circuit booléen*** *si un sous-ensemble $\mathcal{V}' \subset \mathcal{V}$ suffit à définir une interprétation à $\mathcal{F}$. $\mathcal{V}'$ est aussi nommé l'ensemble* ***des variables d'entrée*** *de la formule.*

Définition 43 : *Soit $\mathcal{F}_c$ une formule propositionnelle modélisant un circuit booléen et $\mathcal{V}'$ ses variables d'entrée. Le problème de satisfaisabilité pour un circuit booléen (aussi nommé* CIRCUIT-SAT, CSAT*) est le problème de décision consistant à déterminer s'il existe une interprétation des variables $\mathcal{V}'$ qui rende $\mathcal{F}_c$ consistante [Barrington 2000].*

4.7 La « SAT competition »

Tous les ans un concours est organisé pour déterminer quel est le meilleur solveur SAT de sa catégorie (aléatoire, académique ou industrielle), ceci en fonction de la nature des formules propositionnelles (SAT ou UNSAT). Suivant les années, la traditionnelle compétition n'est pas exactement la même au niveau des règles et a pris différents noms : Le « DIMACS Challenge » (1992-1993), la « Beijing Competition » (1996), la « SAT competition » (2002 à 2005 puis, 2007, 2009, 2011 et 2013), la « SAT race » (2006, 2008, 2010) et enfin la « SAT challenge » en 2012. Ces compétitions ont toutes la même vocation : distinguer les solveurs les plus rapides, les plus efficaces et mettre en avant leurs techniques novatrices. Toutes les spécifications inhérentes à chaque solveur soumis en 2013 peuvent être retrouvées dans [Balint 2013]. L'organisation et l'ensemble des résultats est publié sur le site internet de la « SAT competition » : $http://www.satcompetition.org$.

Parmi les solveurs séquentiels, la « SAT competition » 2013 a distingué :

- **Catégorie formules aléatoires** :
  - Formules SAT : PROBSAT SC13 basé sur SPARROW
  - Formules SAT + UNSAT : CSHCRANDMC, solveur portfolio basé sur (CLASP, SATTIME)
- **Catégorie formules académiques** :
  - Formules SAT : GLUCOSE 2.3
  - Formules SAT + UNSAT : BREAKIDGLUCOSE 1, basé sur GLUCOSE
  - Formules UNSAT : RISS3G CERT, basé sur GLUCOSE
- **Catégorie formules industrielles** :
  - Formules SAT : LINGELING AQW, basé sur LINGELING
  - Formules SAT + UNSAT : LINGELING AQW, basé sur LINGELING
  - Formules UNSAT : GLUCOSE 2.3

En ce qui concerne les solveurs parallèles, la « SAT competition » 2013 ne s'appliquait uniquement que pour les benchmarks SAT + UNSAT et sur des formules industrielles ou académiques. Ce sont les versions parallèles de LINGELING qui ont à chaque fois pris le dessus. Enfin, un *track* ouvert mêlant divers type de formules a vu le solveur CSHCRANDMC l'emporter.

4.8 Conclusion

Avant de choisir quel était le solveur le mieux adapté à nos expérimentations, nous nous sommes d'abord demandés s'il était préférable d'envisager une approche complète ou incomplète de la résolution. En prenant du recul sur l'ensemble de nos travaux, notre ligne directrice a souvent été la modélisation puis la résolution d'un ou plusieurs problèmes donnés. En outre, l'objectif est donc de résoudre nos instances et ce, *peu importe le temps que*

cela mettra tant qu'il reste raisonnable. Cette philosophie va à l'encontre d'une méthode de benchmarking où nous testerions chaque solveur pour élire à quelques minutes près le plus rapide sur nos problèmes. D'autant plus que, d'après les expérimentations que nous avons faites, l'ensemble des meilleurs solveurs actuels apportent des résultats équivalents sur nos instances SAT.

Le temps de résolution étant secondaire, tous les progrès dans les techniques de prétraitement, comme par exemple l'hyper-résolution [Bacchus 2003] ou l'élimination de clauses et de variables [Eén 2005], la vivification [Piette 2008], etc.deviennent alors très intéressantes. Il est vrai qu'une mise en pratique de ces traitements peut consommer un temps de calcul non-négligeable mais elles permettent tout de même de réduire la taille des formules et potentiellement le temps de résolution des solveurs SAT, et ce notamment pour les instances non aléatoires.

Il devient dès lors difficile de décider quel solveur utiliser pour nos travaux. D'un côté - et nous le verrons par la suite - nos formules CNF sont des formules qui admettent toujours une solution et par conséquent garantir le résultat de l'incomplétude n'a plus aucun sens. Ajouté à cela, nos formules admettent en réalité un très grand nombre de solutions (dans un espace certes, très très grand) et les solveurs incomplets semblent plus adaptés dès lors qu'il s'agit d'en trouver au moins une. Dans le chapitre IV nous verrons d'ailleurs comment un solveur de type WALKSAT se comporte sur nos formules.

D'un autre côté, nos instances contiennent un nombre de variables et de clauses conséquents. Ajouté à cela, le problème modélisé procure une formule très structurée, très proche des problèmes dits industriels. De ces deux caractéristiques l'utilisation de solveurs CDCL paraît être l'approche la plus adaptée. C'est la raison pour laquelle, dans cette thèse, nous avons aussi choisi d'utiliser LINGELING dans sa version parallèle pour tenter de résoudre nos problèmes cryptographiques (chapitre IV). Même si notre étude n'avait pas pour objectif l'élection du meilleur solveur de la littérature sur nos formules propositionnelles, nous dresserons une analyse assez précise sur ce qui nous semble être *la bonne* façon d'aborder la résolution de tels problèmes. Mais avant tout, présentons la discipline dans laquelle nos travaux évoluent, à savoir la cryptanalyse logique.

5 Cryptanalyse logique

5.1 Définition

Dans cette thèse, nous traitons de cryptanalyse algébrique et plus précisément de cryptanalyse logique. Le principe est le suivant : dans une première phase l'algorithme est réécrit en un système d'équations puis ce système est résolu grâce à un solveur automatique. Dans le cas de la cryptanalyse logique, le système d'équation est un problème SAT où chaque équation, représentée par une clause, est définie dans un système binaire.

Cette cryptanalyse a été décrite pour la première fois au début des années 2000 dans les travaux de Fabio Massacci [Massacci 2000]. L'idée était de profiter de la puissance des nouveaux solveurs SAT pour dédier une partie des calculs fastidieux d'une cryptanalyse à un outil automatique. En pratique, l'algorithme de chiffrement symétrique DES a d'abord été modélisé sous la forme d'un problème SAT. Puis, en affectant les variables qui corres-

pondaient à des messages d'entrées et à des sorties, il a été possible d'en déduire plusieurs bits sur la clé de chiffrement pour un DES réduit à 3 (sur 16) tours. D'un point de vue plus classique, ceci correspond simplement à une attaque à textes clairs connus.

5.2 État de l'art

Beaucoup de travaux ont émergés dans le domaine de la cryptanalyse logique et peuvent être classés suivant quatre catégories :

1. La création d'instances difficiles pour nourrir les benchmarks des compétitions de solveurs SAT.
2. Accélérer une partie des calculs pour une cryptanalyse particulière (cryptanalyse différentielle et algébrique, mais aussi attaques par canaux auxiliaires).
3. Donner une mesure de la résistance à l'inversion des primitives cryptographiques
   3.1 Retrouver les bits de la clé pour un chiffrement symétrique/asymétrique
   3.2 Attaquer la pré-image de fonctions de hachage
4. Se servir du formalisme SAT pour appendre de nouvelles informations sur le problème modélisé

Dans cette thèse nous ne nous sommes pas explicitement concentrés sur le premier point, même si le générateur de formules propositionnelles que nous avons mis en place (voir chapitre II) pourrait être utilisé dans le but de créer de véritables challenges pour la communauté SAT. Des travaux de ce genre peuvent être trouvés dans [Jovanovic 2005, Joseph Bebel 2013, Nossum 2013, Soos 2013].

Pour le point 2, nous retrouvons bien sûr les travaux précurseurs de Fabio Massacci [Massacci 2000] donnés en introduction. Dans cette approche, nous pouvons classer les résultats très intéressants apportés par Ilya Mironov et Lintao Zhang dans [Mironov 2006]. Leur étude avait pour objectif la modélisation d'une attaque par chemin différentiel sur les fonctions de hachage de la famille MD ainsi que sur SHA. L'intérêt principal réside dans le fait que les calculs mis en jeu pour résoudre les équations complexes de la cryptanalyse ont pu être dédiés à un solveur SAT. Dans la même idée, le lecteur peut aussi s'intéresser aux chemins différentiels trouvés sur SHA-1 par Cameron McDonald *et al* [McDonald 2009]. Des attaques par le biais de la cryptanalyse logique sur RSA, AES, KEELOQ et d'autres fonctions cryptographiques moins répandues peuvent être trouvées dans [Massacci 1999, Fiorini 2003, Courtois 2008a, Kamal 2010, Faizullin 2009]. Notons aussi une approche MAX-SAT dans les récents travaux de [Liao 2013].

Le point 3 se divise en deux sous-ensembles : tout ce qui concerne les algorithmes de chiffrement symétriques/asymétriques et ce qui touche aux fonctions de hachage. Nos travaux ne traitent pas du premier point. Plusieurs études très intéressantes dans ce domaine sont listées dans la première partie du tableau I.4 et promettent un futur optimiste pour la discipline.

Concernant les attaques de pré-images pour les fonctions de hachage, de plus en plus de travaux ont jailli ces dernières années et permettent désormais de couvrir l'ensemble du panel des fonctions de hachage les plus utilisées. Dans cette thèse, nous avons amélioré le nombre d'étapes inversées pour MD5 et fait aussi bien que les résultats existants sur MD4, SHA-0 et SHA-1. Dans certains cas, la cryptanalyse logique apporte de meilleures inversions en pratique que d'autres types de cryptanalyse. Elle fait ainsi office de complément aux techniques cryptanalytiques existantes. Nous avons répertorié les inversions de fonctions de hachage grâce à une approche SAT dans la seconde partie du tableau I.4.

Donner une mesure de la résistance des fonctions de hachage à l'inversion se fait de la manière suivante. Dans une première phase l'algorithme de chiffrement est représenté sous un formalisme logique à travers un problème SAT. Pour cela, on peut s'appuyer sur deux méthodes différentes : via des outils génériques automatiques (par exemple CRYPTLOGVER [16]) ou à la main, en créant un générateur dédié à une fonction particulière. La deuxième phase consiste à fixer une empreinte particulière dans le problème. Cela se fait assez simplement car il suffit d'affecter les variables qui lui correspondent dans la formule propositionnelle. Une nouvelle formule est alors obtenue et chacune de ses solutions correspond à une affectation complète des variables, y compris celles du message d'entrée. Trouver au moins l'une d'elles se fait dans une troisième phase grâce à la résolution de la CNF par un solveur SAT.

Même si les objectifs sont communs, les principaux travaux de cette lignée sont assez hétéroclites dans le sens où les méthodes de modélisation et les solveurs utilisés sont différents. De plus, les fonctions de hachage ont aussi leurs propres spécificités et dans certains cas, la cryptanalyse logique peut être améliorée en utilisant d'autres cryptanalyses. Nous avons rassemblé dans le tableau I.4 l'aperçu de toutes ces attaques de pré-images, à notre connaissance.

La cryptanalyse logique est très prometteuse pour ce type de travaux mais elle possède aussi ses limites. En effet, les solveurs SAT utilisés pour s'attaquer à ces problèmes cryptographiques sont ceux conçus pour des problèmes SAT de types industriels. De ce point de vue, ils sont faits pour répondre aux problèmes de grandes tailles mais ne sont pas spécialisés pour les instances cryptographiques, contenant des structures qui leurs sont particulièrement singulières. À ce jour, et du point de vue des cryptanalystes, le solveur SAT est une boîte noire apportant paradoxalement une réponse à un problème qu'il ne connaît pas. *A contrario*, la vision que peut avoir un expert en résolution SAT est une spécialisation de cette boîte noire afin de la rendre plus intelligente en fonction du problème attaqué.

6 Objectifs de nos travaux

La cryptanalyse algébrique regorge de travaux très intéressants pour la communauté et a pu mettre en lumière de considérables faiblesses pour tout un lot de fonctions cryptographiques. Dans cet environnement, la cryptanalyse logique n'est pas le sous-domaine le plus

16. http ://www.ecrypt.eu.org/tools/cryptlogver

Fonction	Attaque	Référence
BIVIUM	opt 141/177 bits de l'état interne	[Eibach 2008]
	opt 143/177 bits de l'état interne	[McDonald 2007]
	opt 142/177 bits de l'état interne	[Soos 2010b]
TRIVIUM	opt 24/177bits de l'état interne	[Soos 2010b]
	état retrouvé avec injection de fautes	[Mohamed 2011]
GOST	Diverses attaques	[Courtois 2011]
CRYPTO-1	200 s	[Courtois 2008b]
	40 s	[Soos 2009]
HITAG2	$2^{14.5}$ s	[Soos 2009]
RSA	opt 634/1024 bits sur la clé	[Patsakis 2013]
MD4	39/40 étapes	[De 2007]
	39/40 étapes	**[Legendre 2012b]**
MD5	26/64 étapes	[De 2007]
	28/64 étapes	**[Legendre 2012b]**
MD6	11/(80 à 168) tours	[Rivest 2008]
SHA-1	19/80 étapes / empreinte partielle	[Srebrny 2008]
	23/80 étapes	[Nossum 2012]
	23/80 étapes	**[Legendre 2012a]**
SHA-3	2/24 étapes / empreinte partielle	[Homsirikamol 2012]

TABLE I.4 – Inversion de fonctions cryptographiques réalisées dans le cadre de la cryptanalyse logique

privilégié de la cryptanalyse algébrique mais commence tout de même à trouver sa place. Sa commodité, son originalité et l'expertise du domaine de la satisfaisabilité permettent en effet de la considérer comme une nouvelle alternative plausible pour avancer dans l'étude des cryptosystèmes.

Le premier objectif de cette thèse est de proposer des solutions de « reverse-engineering » basées sur une modélisation des problèmes sous une forme logique puis sur leurs résolutions par le biais des outils issus du domaine de la satisfaisabilité. Bien que similaire à l'utilisation classique de la force brute, l'exploitation du formalisme logique permet de bénéficier des techniques récentes de résolution du problème SAT ayant toutes pour objectif d'abaisser sa complexité calculatoire. Dans ce contexte, nous avons d'une part représenté plusieurs fonctions de hachage cryptographiques sous la forme d'un problème SAT et avons tenté de les inverser. Dans les faits, nous avons attaqué la pré-image de versions réduites de ces fonctions. Ainsi, nous obtenons des bornes en termes de nombre d'étapes au delà desquelles il est calculatoirement très difficile de passer outre. Pour résumer notre pensée, nous avons utilisé SAT comme un outil de mesure de la résistance à la pré-image pour MD et SHA. Par ailleurs, nous avons conduit des travaux portant sur la recherche de collisions sur la fonction de hachage MD5. Les buts ont été doubles : examiner les collisions sur un bloc et tenter de reproduire une attaque par le biais de la cryptanalyse logique. L'intérêt réside une nouvelle fois dans l'illustration pratique de la facilité à expri-

mer une cryptanalyse compliquée.

Ajouté à cela, la nature intrinsèque de la modélisation logique que nous souhaitons étudier impose la résolution de formules disposant nécessairement d'au moins une solution et offre ainsi la possibilité d'imaginer des méthodes complètes et incomplètes de résolution spécifiquement dédiées, notamment basées sur des métaheuristiques. C'est dans ce cadre que nous avons ouvert plusieurs pistes pour analyser la structure des instances CNF générées afin d'en dégager des faiblesses cryptographiques exploitables par un solveur SAT.

Le second objectif fut donc d'exploiter le format de la logique booléenne et de la modélisation extrêmement détaillée du problème pour étudier les problèmes sous-jacents. Puisque chaque bit du processus de hachage est représenté par une variable et chaque opération par un ensemble de règles d'inférence il est possible d'avoir un rendu complet du comportement de tous ces éléments. Par conséquent, d'une part, nous avons porté notre étude sur un raisonnement mécanique permettant d'extraire des relations logiques entre les variables et les clauses de la formule propositionnelle. D'autre part, nous avons étendu ce raisonnement à travers une approche probabiliste et mis en avant de nouvelles relations pertinentes dues à l'utilisation des constantes de rondes.

Un trait intéressant concernant ces faiblesses est qu'elles sont exploitables dans d'autres types de cryptanalyse pour peu que le problème soit considéré sous sa forme la plus détaillée. Un des objectifs finaux a donc été de construire un solveur SAT dédié à nos problèmes afin d'user au mieux de ces défauts constatés. Ajouté à cela, la spécificité structurelle du problème cryptographique nous laisse croire en une amélioration de la résolution via une meilleure compréhension des clauses traitées. C'est pourquoi nous avons approfondi notre étude et dégagé des profils de satisfaisabilité pour chaque clause de la formule propositionnelle.

CHAPITRE II

Modélisation booléenne de problèmes cryptographiques

La première étape d'une attaque basée sur la cryptanalyse logique est la modélisation du problème cryptographique sous un formalisme logique. Un même problème peut admettre plusieurs modélisations. Par exemple, certaines peuvent privilégier une minimalisation du nombre de variables là où d'autres préfèrerons minimiser la taille des clauses. Nous verrons dans ce chapitre que plusieurs méthodes peuvent être adoptées pour la modélisation d'un circuit logique et qu'il est bien difficile de définir les critères de choix entre ces méthodes. À notre connaissance, peu de travaux sont d'ailleurs consacrés à fournir une réponse précise pour la modélisation de problèmes structurés. Notre expérience nous a amenés à nous poser des questions sur ce sujet et à creuser quelques pistes pour tenter d'y répondre. Nous constaterons dans ce chapitre que ces critères dépendent fortement de la vision stratégique que nous voulons accorder au problème. Un parallèle peut être fait avec un potier qui devrait concevoir un pot sous la contrainte d'avoir une bonne contenance. Pour cela il peut opter pour la réalisation d'un pot allongé, mais fragile, d'un pot large, mais peu pratique ou d'un pot de forme normale mais avec une contenance réduite. Pour nos problèmes logiques, les critères sont le nombre de variables, le nombre de clauses, la taille de ces clauses et surtout la précision à laquelle le problème de base doit être représenté.

Pour illustrer cette problématique, ce chapitre débute par un cas élémentaire présentant deux modes génériques de production de contraintes pour un circuit logique simple. Cependant, notre questionnement sur ce qu'il nous semble être une « bonne modélisation » est reprise tout au long du chapitre et nous verrons les conséquences directement induites par les choix que nous pouvons faire sur des cas concrets.

La principale contribution en termes de modélisation de problèmes cryptographiques est présentée en deuxième partie. Ici nous exposons la méthode utilisée pour la modélisation de fonctions de hachage en problèmes SAT ainsi que les formules ainsi créées. Pour cela, nous avons choisi de procéder en deux étapes. En premier lieu, une description logique sous forme clausale est donnée pour chaque opération de base inhérente aux algorithmes cryptographiques. Puis ces multiples représentations sont concaténées pour obtenir une description logique de l'ensemble. Seront donc transcrites ici les modélisations choisies pour chaque opération et les raisons qui nous ont poussées à procéder ainsi. Nous ferons aussi un point sur les formules générées, leurs caractéristiques statistiques et leurs propriétés comportementales, au sens logique du terme. Précisons que les modélisations proposées sont complètes et robustes.

La fin du chapitre est consacrée à la simplification de nos formules CNF via un prétraitement conçu par nos soins et dédié à ce type de problèmes. Nous verrons alors quels sont les enjeux d'un tel procédé et de quelle façon nous avons enrichi les déductions logiques pour rendre la réduction du problème astucieuse, au point d'apprendre de nouvelles informations utiles pour la résolution. Les nouvelles caractéristiques de nos formules ainsi qu'un ensemble de relations logiques entre certaines variables conclurons ce chapitre.

1 Représenter un circuit logique

Un circuit logique est une succession de portes logiques. Intuitivement, traduire un tel circuit dans un langage formel se fait en modélisant de façon successive chacune de ses portes. Prenons l'exemple du circuit logique figure II.1 où les lettres sont des variables d'entrées, le symbole « & » le ET logique, le symbole « ≥ 1 » le OU logique et la porte « 1 » avec un point la NEGATION logique.

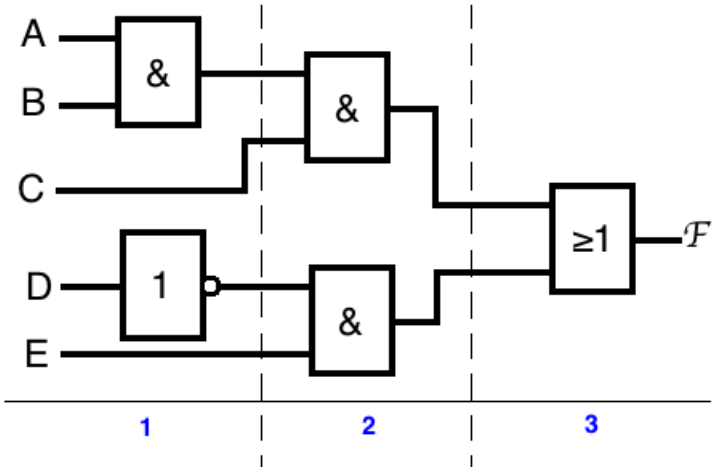

FIGURE II.1 – Un circuit logique, découpé en 3 étapes successives

Pour représenter la première étape, il convient de définir les portes « A ET B » et la NEGATION de D séparément. Puis, la seconde étape consiste à les deux portes ET, toujours séparément, et enfin, troisième étape, la porte OU. Sa modélisation sous une formule logique $\mathcal{F}$ sera donc donnée par les étapes de construction suivantes :

1. $\mathcal{F} : (A \wedge B)$
   $\overline{D}$
2. $\mathcal{F} : ((A \wedge B) \wedge C)$
   $(\overline{D} \wedge E)$
3. $\mathcal{F} : ((A \wedge B) \wedge C) \vee (\overline{D} \wedge E)$

Cette méthode est universelle et permet de modéliser tout circuit logique. Elle pose cependant un souci dès lors qu'on cherche à mettre la formule résultante sous une forme normale conjonctive. En effet, pour parvenir à cela, il est nécessaire de passer par les lois

définies par Augustus De Morgan au 19ième siècle et notamment par la loi de distributivité du « ET » logique. Or, faire ainsi peut aboutir à une formule exponentiellement grande en fonction du nombre de variables. Reprenons notre exemple. Sa modélisation sous CNF sera donnée par :

$$\mathcal{F} : (A \vee \overline{D}) \wedge (A \vee E) \wedge (B \vee \overline{D}) \wedge (B \vee E) \wedge (C \vee \overline{D}) \wedge (C \vee E)$$

Pour pallier à cela, Grigorii Samuilovich Tseitin [Samuilovich 1968] propose d'encoder les portes logiques servant en électronique sous forme normale conjonctive, sans passer systématiquement par les lois de De Morgan. L'idée est de modéliser chaque porte par une équation booléenne de sorte à considérer leurs entrées et leurs sorties respectives de façon indépendante. Cela permet en plus d'avoir une valeur de vérité pour chaque sortie de porte, ce que ne permettait pas la précédente représentation. Suivant ce qu'on appelle les transformations de Tseitin, et en considérant de nouvelles variables Si à la sortie de chaque porte, le circuit figure II.1 devient le circuit logique II.2.

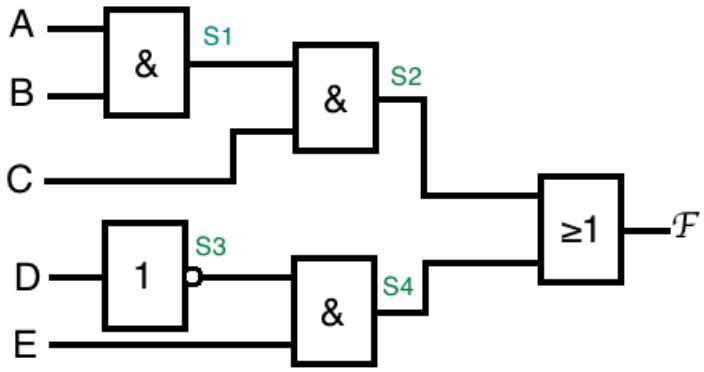

FIGURE II.2 – Un circuit logique, où chaque sortie de porte logique est identifiée en une nouvelle variable Si

Sa représentation sous forme logique est donnée par :

$$\mathcal{F} : (A \wedge B \Rightarrow S1) \wedge (S1 \wedge C \Rightarrow S2) \wedge (D \Rightarrow \overline{S3}) \wedge (S3 \wedge E \Rightarrow S4) \wedge (S2 \vee S4)$$

En appliquant ensuite les lois de la logique propositionnelle, le circuit logique s'exprimera sous CNF comme suit :

$$\mathcal{F} : (\overline{A} \vee \overline{B} \vee S1) \wedge (\overline{S1} \vee \overline{C} \vee S2) \wedge (\overline{D} \vee S3) \wedge (\overline{S3} \vee \overline{E} \vee S4) \wedge (S2 \vee S4)$$

Quand les deux méthodes sont comparées, il est facile de se rendre compte que pour un même circuit, deux modélisations complètement différentes sont possibles. Le tableau II.1 montre quelques statistiques associées aux deux présentées ici.

	Nb variables	Nb clauses	2-cl	3-cl
De Morgan	5	6	6	0
Tseitin	9	5	2	3

TABLE II.1 – Statistiques à propos des différentes modélisations du circuit II.1

Dans la première modélisation, moins de variables sont dénombrées par rapport à la seconde. Ceci est un avantage qui paraît non-négligeable puisque l'énumération de tous les cas possibles sera moins conséquente. Dans le cas présent, la modélisation comptant cinq variables signifie qu'on aura 2^5 affectations différentes. Au niveau des clauses, nous pouvons remarquer qu'elles sont plus nombreuses mais qu'elles sont toutes de taille 2. Or, le fait d'avoir un nombre conséquent de clauses de taille 2 dans une modélisation est du pain béni pour un solveur SAT puisque cela permet une propagation extrêmement efficace. Pour rappel, le problème 2-SAT est d'ailleurs polynomial.

À l'inverse, la seconde modélisation propose plus de variables et moins de clauses de taille 2 et paraît donc *a priori* moins intéressante que la première. Cependant, elle présente aussi moins de contraintes, ce qui est bénéfique pour la résolution car le problème est plus souple et moins imposant en taille. Ajouté à cela, dans le cas où il existe une solution au problème modélisé, de grandes clauses sont plus facilement satisfiables que de petites. En effet, si une clause comporte k littéraux, alors elle est satisfiable par 2^{k-1} affectations possibles. Par conséquent, plus k est grand, plus les chances de satisfaire la clause sont nombreuses.

Dès lors, choisir la façon dont un problème sera modélisé n'est pas chose aisée. Dans le premier cas, la vitesse de résolution et la taille de l'espace de recherche sont privilégiés. Dans le second, l'accent est mis sur la taille de la représentation du problème et sur une plus grande élasticité des contraintes. En ce qui nous concerne, nous avons dans un premier temps opté pour une hybridation des deux approches. Les petits circuits que nous avons eu à modéliser ont été représentés uniquement avec De Morgan mais chaque sortie de circuit a été appréciée séparément, à la manière de Tseitin. Par exemple modéliser une addition a été fait en considérant l'addition dans son intégralité. Et lier deux résultats d'additions a été réalisé en considérant leurs sorties respectives. Ce premier choix a été fait pour avoir à la fois une représentation détaillée du problème - jusqu'à considérer les retenues des additions - et une représentation cherchant à minimiser le nombre de variables. Dans un second temps, nous avons comparé les autres modélisations à travers diverses expérimentations en pratique, présentées chapitre IV.

2 Modélisation de fonctions de hachage cryptographiques

Modéliser une fonction de hachage sous une CNF peut se faire de deux façons : par le biais d'outils automatiques ou « à la main » en créant son propre générateur. C'est cette dernière voie que nous avons suivie même si elle requiert une connaissance technique de l'algorithme.

Dans cette partie nous nous concentrons sur notre procédé de génération de formules

SAT pour les fonctions de hachage MD4, MD5 et SHA-1. Chaque opération de l'algorithme sera considérée comme une grande porte logique à modéliser. Nous donnons donc un détail rigoureux pour chacune d'elles, tant pour expliquer les techniques adoptées que leurs justifications.

2.1 L'opération logique OU-exclusif ($\oplus$)

L'opérateur qui revient très souvent en cryptographie [1] est le « OU-exclusif » (aussi appelé « XOR ») noté $\oplus$. Cette porte logique prend deux entrées, et donne une sortie en suivant la table de vérité II.2.

A	B	$X : A \oplus B$
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	FAUX

TABLE II.2 – Table de vérité pour l'opération OU-exclusif

Modéliser cette opération est réalisée en extrayant tous les cas possibles décrits dans sa table de vérité, chaque ligne correspondant à une équation booléenne. Prenons la première d'entre elles. De gauche à droite, on peut lire : $A \leftarrow$ FAUX et $B \leftarrow$ FAUX implique $X \leftarrow$ FAUX. Sous un formalisme logique, cette relation est exprimée par :

$$(\overline{A} \wedge \overline{B}) \Rightarrow \overline{X}$$

Et sous forme normale conjonctive :

$$A \vee B \vee \overline{X}$$

Par conséquent, en étendant ce principe, l'ensemble des clauses décrivant le OU-exclusif est donnée par :

$$(A \vee B \vee \overline{X}) \wedge (A \vee \overline{B} \vee X) \wedge (\overline{A} \vee B \vee X) \wedge (\overline{A} \vee \overline{B} \vee \overline{X})$$

2.2 L'addition modulaire

L'addition modulaire de n opérandes est une opération assez courante dans les systèmes cryptographiques. On la retrouve d'ailleurs dans les fonctions de hachage modélisées dans cette thèse. Plus exactement, c'est une addition arithmétique binaire où les opérandes sont des mots limités par un nombre fixé de bits, ce qui signifie qu'au delà de leurs tailles, ils sont tronqués. D'un point de vue mathématique, cette addition est donc définie dans les corps de Galois $GF(2^m)$.

1. Petite digression pour rappeler que le XOR est l'unique opération qui intervient dans un masque de Vernam, seul algorithme de chiffrement prouvé sûr.

Afin de présenter la modélisation d'une telle addition sous CNF, cette partie commence par la représentation du cas le plus simple : l'addition modulaire de deux opérandes d'un bit.

2.2.1 L'addition à deux opérandes

Pour transformer cette opération du monde arithmétique vers un formalisme logique, nous avons choisi d'exprimer directement les règles logiques associées à l'addition arithmétique classique sous un ensemble de clauses SAT. La spécificité d'une addition est que ses opérandes génèrent une somme mais peuvent aussi générer une ou plusieurs retenues. En ce sens cette opération peut être découpée en deux :

- Opérandes $\Rightarrow$ Somme
- Opérandes $\Rightarrow$ Retenue(s)

Prenant en compte ces considérations, la première démarche que nous avons adoptée a été de chercher un modèle générique pour définir l'addition sur 1 bit. Une représentation de ce modèle est présenté figure II.3, où s_i correspond à la somme de a_i et b_i. La flèche représente la probable génération d'une retenue notée c_{i+1}.

Puisque cette retenue doit nécessairement être considérée comme une nouvelle opérande au rang suivant, il faut prendre en compte qu'une retenue a pu être générée au rang précédent. Et cette retenue doit être considérée comme un opérande au rang courant. De ce fait, l'addition modulaire à deux opérandes est nécessairement une addition modulaire à trois opérandes : a_i, b_i et c_i, avec c_i provenant du rang $i-1$ (sauf pour le rang $i=0$).

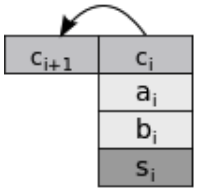

FIGURE II.3 – Modèle d'une addition à deux opérandes d'un bit

La table de vérité II.4 associée à ce modèle peut ensuite être érigée. Elle représente tous les cas possibles qui sont produits quand toutes les combinaisons en entrée sont énumérées. Autrement dit, elle décrit l'ensemble des règles d'inférence qui définissent un raisonnement correct de l'addition à deux opérandes sur 1 bit. La figure II.4 exprime cette table de vérité. Les cases blanches correspondent aux entrées du circuit additionnel et les cases grises la retenue et la somme. On remarquera au passage que cette dernière est équivalente à un OU-exclusif.

C'est à partir de cette table que les équations booléennes, *i.e.* les clauses, vont être générées. Comme pour le OU-exclusif, chaque ligne est lue de gauche à droite pour en déduire des relations d'implication. Par exemple soient c_i, a_i, b_i, c_{i+1}, s_i cinq variables booléennes ; la seconde ligne de la table figure II.4 peut être lue comme $c_i \leftarrow$ FAUX et $a_i \leftarrow$ FAUX et $b_i \leftarrow$ VRAI impliquent $c_{i+1} \leftarrow$ FAUX et $s_i \leftarrow$VRAI. Ceci peut être formulé

c_i	a_i	b_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

FIGURE II.4 – Table de vérité pour l'addition de deux opérandes d'un bit

comme suit :

$$(\overline{c_i} \wedge \overline{a_i} \wedge b_i \Rightarrow \overline{c_{i+1}}) \wedge (\overline{c_i} \wedge \overline{a_i} \wedge b_i \Rightarrow s_i)$$

Et donc, sous CNF :

$$(c_i \vee a_i \vee \overline{b_i} \vee \overline{c_{i+1}}) \wedge (c_i \vee a_i \vee \overline{b_i} \vee\ s)$$

L'ensemble des clauses pour cette addition est le suivant :

$$(c_i \vee a_i \vee b_i \vee \overline{c_{i+1}}) \wedge (c_i \vee a_i \vee b_i \vee \overline{s}) \wedge (c_i \vee a_i \vee \overline{b_i} \vee \overline{c_{i+1}}) \wedge (c_i \vee a_i \vee \overline{b_i} \vee\ s) \wedge$$

$$(c_i \vee \overline{a_i} \vee b_i \vee \overline{c_{i+1}}) \wedge (c_i \vee \overline{a_i} \vee b_i \vee\ s) \wedge (c_i \vee \overline{a_i} \vee \overline{b_i} \vee c_{i+1}) \wedge (c_i \vee \overline{a_i} \vee \overline{b_i} \vee\ \overline{s}) \wedge$$

$$(\overline{c_i} \vee a_i \vee b_i \vee \overline{c_{i+1}}) \wedge (\overline{c_i} \vee a_i \vee b_i \vee\ s) \wedge (\overline{c_i} \vee a_i \vee \overline{b_i} \vee c_{i+1}) \wedge (\overline{c_i} \vee a_i \vee \overline{b_i} \vee\ \overline{s}) \wedge$$

$$(\overline{c_i} \vee \overline{a_i} \vee b_i \vee c_{i+1}) \wedge (\overline{c_i} \vee \overline{a_i} \vee b_i \vee\ \overline{s}) \wedge (\overline{c_i} \vee \overline{a_i} \vee \overline{b_i} \vee c_{i+1}) \wedge (\overline{c_i} \vee \overline{a_i} \vee \overline{b_i} \vee\ s)$$

Arrêtons nous un instant sur l'apport du formalisme logique pour décrire une addition. Imaginons une addition à trous de deux opérandes à 4 bits où quelques variables éparpillées sont affectées. Le but est de retrouver le maximum d'information sur toutes les variables manquantes. Une telle addition est représentée figure II.5.

rang	*3*	*2*	*1*	*0*
c	**1**	-	-	**0**
a	-	**1**	-	-
+ *b*	-	-	**1**	-
= *s*	**1**	**1**	-	**1**

FIGURE II.5 – Addition à trous de deux opérandes de 4 bits

Si cette addition était observée avec l'œil du mathématicien évoluant dans un univers décimal, alors voici ce qu'il découvrirait. Assez facilement, $a \geq 100_{(2)}$ signifie $a \geq 4_{(10)}$ et $b \geq 10_{(2)}$ signifie $b \geq 2_{(10)}$. $s \in \{((1)1101)_{(2)}, ((1)1111)_{(2)}\}$ signifie que $s = 13_{(10)}$ ou $15_{(10)}$ ou $29_{(10)}$ ou $31_{(10)}$. Soit a soit b est impair car s est impair. Enfin, le vecteur des

retenues est illisible car il est dépendant de a, b, et de lui-même au niveau de chaque bit. Et ce sont finalement les seules choses qu'il peut avancer.

À l'inverse, évoluer dans un champs binaire permet d'avoir une représentation très détaillée des retenues alors que ce n'est pas le cas à plus grande échelle. De plus, le raisonnement logique permet de s'intéresser à chaque variable et il est possible d'élargir les déductions. Dans ce cas précis, à partir de c_0 et de s_0, il devient possible de déduire $c_1 = 0$ et à partir de c_3, a_2 et s_3, est inféré $c_2 = b_2 = 1$. Par conséquent, seules quatre solutions restent possibles :

$$(a,b) \in \{(0110,0111)_{(2)},(1110,1111)_{(2)},(0111,0110)_{(2)},(1111,1110)_{(2)}\}$$

La figure II.6 montre ces différentes déductions logiques.

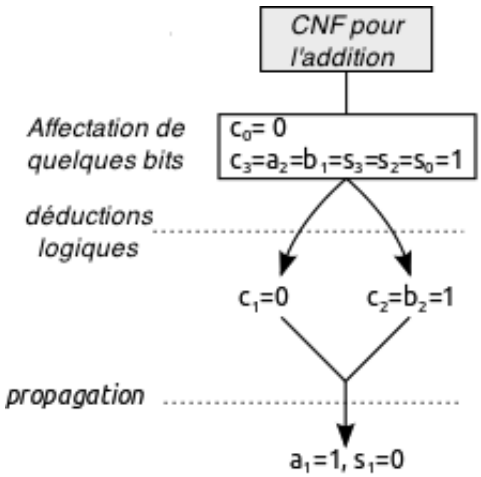

FIGURE II.6 – Déductions logiques dans l'addition à trous figure II.5

2.2.2 L'addition à n opérandes

Représenter une addition à n opérandes se fait exactement en suivant le même cheminement que pour l'addition à deux opérandes. Ce n'est finalement qu'une extension du procédé. La seule différence réside dans la génération des retenues puisque plus il y a d'opérandes plus elles seront nombreuses. Nous avons pris l'exemple d'une addition à quatre opérandes et dressé son modèle sur la figure II.7. Dans le cas où ces quatre opérandes sont à 1 alors la somme peut valoir 4, ce qui se traduit en binaire par $100_{(2)}$, *i.e.* la somme vaut 0, puis une première retenue vaut 0 et une seconde vaut 1. Le modèle étant générique, chaque rang doit prendre en compte les retenues générées aux rangs précédents. L'addition sur quatre opérandes devient nécessairement une addition sur six opérandes. Nous appelons ce modèle **l'encodage global**.

Comme nous l'avons vu précédemment, une addition binaire à n opérandes est finalement une addition à $n+k$ opérandes, où k est le nombre de retenues pouvant être produites.

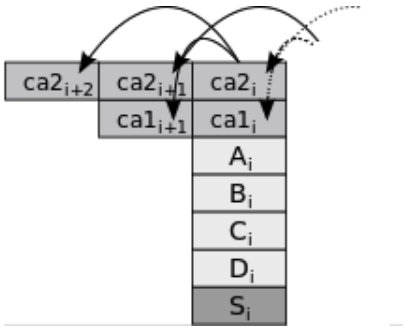

FIGURE II.7 – Modèle pour l'encodage global d'une addition sur quatre opérandes

En généralisant ce principe, on peut déterminer le nombre de retenues générées (k) en fonction du nombre d'opérandes (n) par la résolution du système d'équations suivant :

$$\begin{cases} 2^k - k + 1 \leq n \\ 2^{k+1} - k > n \end{cases}$$

La modélisation brute, c'est-à-dire sans aucune simplification, d'une addition à quatre opérandes de 32 bits nécessite 224 variables et 6144 clauses. Pour une addition à cinq opérandes, il faudra le même nombre de clauses mais 256 variables. On notera tout de même deux choses pour les retenues qui n'ont pas été prises en compte dans ces chiffres. Sur les premiers rangs, les retenues sont inexistantes car par définition, il n'y a pas de rang précédent le rang 0. Sur les derniers rangs, il est inutile de générer des retenues qui ne changent rien au problème, l'addition étant modulaire, elles sont tronquées.

Les algorithmes pour la famille MD utilisent des additions sur quatre opérandes quand celui pour SHA-1 en utilise cinq.

Un autre point très curieux sur la modélisation d'une addition à n opérandes est qu'elle est en réalité imaginable sous plusieurs faciès. En effet, le modèle présenté ici propose de considérer toutes les opérandes d'un seul et même bloc. Or, il existe au moins deux autres manières de le concevoir :

- **l'encodage découpé**. Une addition sur n opérandes peut tout simplement être découpée en $\frac{n}{2}$ additions sur deux opérandes. Les résultats intermédiaires sont ensuite eux aussi sommés deux à deux jusqu'à obtenir la somme finale. En effet, la somme $a_0 + a_1 + a_2 + a_3 + ... + a_i + a_{i+1} + ... + a_{n-1} + a_n$ peut être vue comme étant $(a_0 + a_1) + (a_2 + a_3) + ... + (a_i + a_{i+1}) + ... + (a_{n-1} + a_n)$. Dès lors, il suffit de reprendre le modèle défini pour une addition sur deux opérandes défini au point 2.2.1 puis de le dupliquer. La figure II.7 illustre cette pratique sur une addition où quatre opérandes, A, B, C, D sont additionnées deux à deux. C'est ensuite le résultat de leurs additions, notés S1 et S2 qui sont ajoutés pour donner la somme finale S. En pratique, procéder ainsi pour établir une formule SAT définissant cette addition produira plus de variables et plus de clauses. Notons enfin que l'ordre dans lequel les opérandes vont être couplées deux à deux peut influencer la modélisation. Imaginons

par exemple un cas où les valeurs des bits des opérandes A et C ne sont pas équiprobablement réparties dans $\{0,1\}$. Dans ce cas il est possible que le fait de modéliser (A+C) et (B+D) au lieu de faire (A+B) puis (C+D), puisse changer la contrainte exercée par les clauses générées.

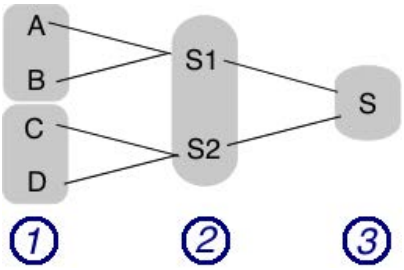

FIGURE II.8 – Découpage d'une opération à quatre opérandes en trois additions à deux opérandes

- **l'encodage hybride**. Une addition booléenne peut être vue comme la production d'une somme et d'une ou plusieurs retenues. Partant de ce principe, une représentation possible est de séparer d'un côté uniquement la somme, qui se comportera alors comme un OU-exclusif et de l'autre côté les deux vecteurs de retenues. Enfin, une addition finale sommera le résultat du OU-exclusif avec le résultat de la somme des vecteurs. La figure II.9 présente cette modélisation sur 2 bits, pour une addition à quatre opérandes. D'un point de vue comptable, le nombre de clauses, de variables et leurs tailles en moyenne se situeront entre l'encodage global et l'encodage découpé.

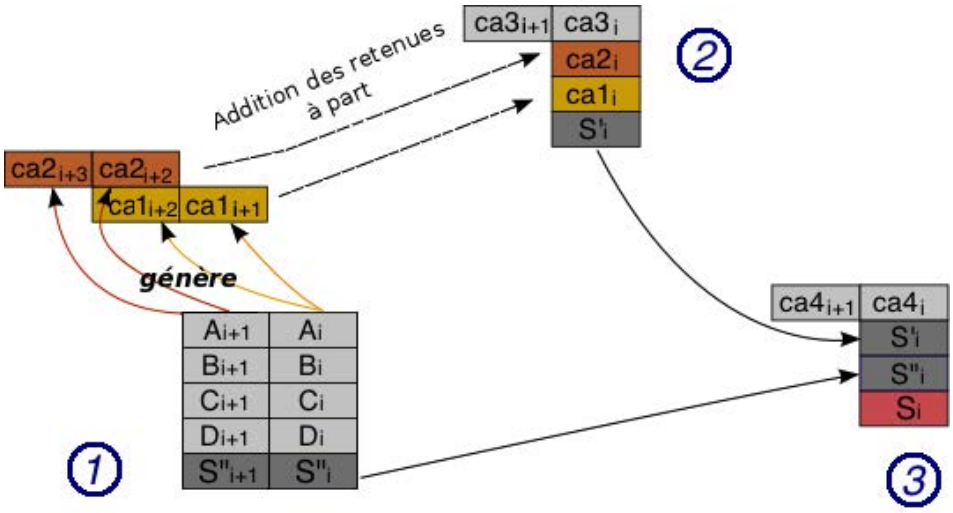

FIGURE II.9 – Encodage hybride d'une addition à quatre opérandes sur deux bits

Dans le tableau II.3 sont regroupées les statistiques associées aux différentes modélisations présentées pour l'addition à quatre opérandes de 32 bits. Nous pouvons remarquer que ces trois méthodes permettent d'avoir trois visions totalement différentes pour un même

problème. Ce point est assez curieux et peut permettre d'envisager quelques perspectives parallèles intéressantes. Avoir plusieurs perceptions peut en effet autoriser une résolution où chaque solveur échangerait ses informations avec les autres.

Encodage	Nb variables	Nb clauses	4-cl	5-cl	6-cl	7-cl
Global	224	6144	0	0	0	6144
Découpé	320	1536	1536	0	0	0
Hybride	352	2560	1024	1536	0	0

TABLE II.3 – Statistiques à propos des trois modélisations brutes pour une addition à quatre opérandes de 32 bits

Pour conclure sur la modélisation de l'addition, il est à noter que la problématique du choix de la bonne modélisation reste entier et il est difficile de trancher pour l'une ou l'autre. En ce qui concerne nos travaux autour de la modélisation de fonctions cryptographiques de hachage, nous avons généré pour MD5 des formules SAT reprenant toutes ces différentes possibilités. Une étude - qu'on peut retrouver chapitre IV paragraphe 2.3 - a ensuite été conduite pour évaluer comment se comportaient les solveurs SAT par rapport à ces modélisations.

2.3 Les fonctions non-linéaires

Les fonctions non-linéaires sont présentes à chaque étape du processus de hachage, que ce soit pour la famille MD ou SHA-1. Elles sont certes souvent difficiles à inverser mais permettent surtout de donner naissance à un certain chaos. Elles participent donc à la confusion de l'information, et un changement en entrée propage un important désordre en sortie.

Bien souvent, ces fonctions sont exprimées sous forme logique. Proposer un modèle pour une telle fonction se fait donc sans peine. Pour cela, il suffit de considérer les entrées de la fonction comme des variables, son unique sortie comme une nouvelle variable et de représenter tous les cas possibles. Par exemple, prenons la fonction non-linéaire « IF » , et a,b,c trois variables booléennes. f est définie par : $f(a,b,c) = (a \wedge b) \vee (\overline{a} \wedge c)$. Réécrire cette fonction directement sous forme logique se fait en traduisant l'égalité sous deux implications :

$$(f \Rightarrow (a \wedge b) \vee (\overline{a} \wedge c)) \ \wedge \ ((a \wedge b) \vee (\overline{a} \wedge c) \Rightarrow f)$$

La traduction sous forme normale conjonctive donne :

$$(\overline{f} \vee (a \wedge b) \vee (\overline{a} \wedge c)) \ \wedge (\overline{(a \wedge b) \vee (\overline{a} \wedge c)} \vee f)$$

Puis,

$$(a \vee b \vee c \vee \overline{f}) \wedge (a \vee b \vee \overline{c} \vee \overline{f}) \wedge (a \vee \overline{b} \vee \overline{c} \vee f) \wedge (a \vee \overline{b} \vee c \vee f) \wedge$$
$$(\overline{a} \vee \overline{b} \vee c \vee \overline{f}) \wedge \ (\overline{a} \vee \overline{b} \vee \overline{c} \vee f) \ \wedge \ (\overline{a} \vee b \vee c \vee f) \ \wedge (\overline{a} \vee b \vee c \vee \overline{f})$$

En considérant des mots de 32 bits, la modélisation brute d'une fonction non-linéaire à trois entrées nécessite 128 variables et 256 clauses.

2.4 Le décalage de bits circulaire

Un décalage d'un bit sur la gauche, noté $\ll_1$, est une opération qui consiste à déplacer les bits d'un mot d'une place vers la gauche. Soit m le mot binaire, défini dans $GF(2^8)$ tel que $m = 10011010_{(2)}$. Un décalage d'un bit vers la gauche donnera : $m_{\ll 1} = 00110100_{(2)}$.

Quand ce décalage est circulaire il est noté $\lll 1$ et le bit de poids fort prend la place laissée vacante par le bit de poids faible décalé. Soit m le mot binaire défini dans $GF(2^8)$, tel que $m = 10011010_{(2)}$. Un décalage circulaire d'un bit vers la gauche donnera : $m_{\lll 1} = 00110101_{(2)}$.

Un décalage circulaire de r bits sur la gauche correspond à r décalages circulaires d'un bit sur la gauche. Si m est décalé de trois bits alors $m_{\lll 3} = 11010100_{(2)}$. On remarquera qu'un décalage circulaire de r bits sur la droite pour un mot de taille l est un décalage circulaire sur la gauche de $l - r$ bits.

Le décalage circulaire est un élément clé de tout algorithme de hachage car il permet la diffusion de l'information dans plusieurs mots internes. De plus, d'un point de vue algorithmique il fixe la séquentialité des opérations. Prenons par exemple l'algorithme 5 suivant qui permet de décaler A, puis d'obtenir S la somme de A et de B.

Algorithme 5: Addition avec un opérande décalée

Données : A, B, S, quatre mots de 4 bits

1 S = 0000
A = 0100
B = 1111

A = A $\lll$ 1 *// A vaudra 1000*
S = A + B
Afficher S *// l'algorithme affichera 0111*

Il est impossible de nier le décalage de A puis de réécrire l'algorithme 5 sans en changer le résultat final S = 0111 (excepté par l'ajout de nouvelles variables mais ce n'est pas le propos). En effet, si le jeu consiste à annuler le décalage sur A et à essayer de le rattraper pour retrouver tout de même S, peu d'alternatives s'offrent à nous :

- Premier choix, décaler B. Ce choix n'aurait aucun effet car tous les bits de B sont équivalents.
- Second choix, décaler S. Voyons tout d'abord combien vaudrait S si A n'était pas décalé et appelons S' cette nouvelle somme. S' = A + B, donc S' = 0100 + 1111 = 0011.

Malheureusement, décaler ensuite S' n'amènera rien car cette somme ne contient que deux bits valant 1 quand la somme recherchée (S) en contient trois.

La conclusion sur ce point est que la modélisation d'un décalage de bits circulaire ne peut se faire que d'une seule façon, ce qui n'était par exemple pas le cas pour l'addition sur n opérandes. Reste le choix de la méthode : un encodage entrée/sortie indépendant *à la* Tseitin, ou un encodage direct, façon De Morgan.

Au regard des transformations de Tseitin, réaliser le décalage circulaire d'un bit vers la gauche c'est considérer un mot A, qui, décalé, donnerait le mot B. Dans ce cas, la modélisation serait réalisée par l'ajout de classes d'équivalences entre chacune des variables correspondantes entre les mots A et B.

Soient $a \in A$ et $b \in B$ deux variables, alors l'ajout d'une équivalence entre a et b se fait en ajoutant deux clauses binaires : $(\overline{a} \vee b) \wedge (a \vee \overline{b})$. Pour modéliser un décalage circulaire de cette façon sur un mot de 32 bits, il faudrait donc 32 nouvelles variables et 64 2-cl.

Penser aux lois de De Morgan simplifie cette tâche puisque les variables d'entrées du décalage circulaire seront les mêmes qu'à la sortie. En effet, soit A un mot de quatre bits tel que A s'exprime $a_0a_1a_2a_3$. Modéliser $A_{\lll 1}$ peut alors simplement être fait en réécrivant A : $a_1a_2a_3a_0$. Pour ce faire, il n'y a ni besoin de nouvelles variables ni besoin de clauses. Le procédé n'est cependant pas magique, et implanter un décalage circulaire de cette façon dans la modélisation d'un algorithme cryptographique doit nécessairement passer par un effort technique.

C'est cette dernière façon de faire que nous avons mise en place. Le gain en variables et en clauses est certes précieux, mais c'est aussi une façon d'anticiper le pré-traitement de formules SAT présenté dans ce chapitre, au point 3, qui de toute façon éliminerait les classes d'équivalences.

Avant de conclure sur le décalage circulaire, nous pouvons remarquer une chose qui rend une nouvelle fois originale l'approche SAT. Comme nous l'avons précédemment vu, le décalage circulaire implique une séquentialité algorithmiquement insécable. Sa représentation en un formalisme SAT est pourtant sans aucun ordre ; chaque clause modélise uniquement une règle qui définit les contours d'un problème. En ce sens, la modélisation d'un algorithme cryptographique en un problème SAT est un paquet de clauses contraignant un espace de recherche de façon à rendre correct l'espace des solutions, et ce, en perdant absolument la notion de séquentialité. Ceci est vrai pour l'étape de modélisation mais aussi pour l'étape de résolution où le solveur choisira des clauses à traiter sans se soucier de leur sens.

2.5 À propos de SHA-3

Comme présenté dans le chapitre I au paragraphe 3.4, la fonction de hachage cryptographique SHA-3 repose sur le schéma de construction des fonctions éponges. Nous avons cadré notre travail de modélisation sur une seule absorption et un seul essorage, en prenant en entrée de la fonction un message de taille fixe maximale de 576 bits (18 blocs de 32 bits). La modélisation s'est donc principalement focalisée sur les cinq sous-fonctions de

la fonction de compression.

Le calcul de parité fait par θ consiste en une multitude d'opérations XOR ($\oplus$). Il suffit donc de traduire l'ensemble des XOR sous forme normale conjonctive pour encoder cette fonction θ dans un problème SAT. Cette étape de modélisation ne présente pas de difficulté majeure mais la succession d'opérations XOR va générer un très grand nombre de clauses. En effet, une opération XOR entre deux opérandes produit quatre clauses ; deux opérations XOR entre trois opérandes produit huit clauses, et ainsi de suite. θ nécessite dix XOR et produit, pour chaque bit, 2^{11} clauses.

Les sous-fonctions restantes utilisent des principes déjà abordés dans les parties précédentes. ρ consiste en un décalage circulaire, une opération décrite dans ce chapitre, point 2.4. Par conséquent, nous avons à nouveau appliqué notre méthode pour modéliser cette sous-fonction. π est une permutation de bits. Retranscrire cette opération dans un contexte logique se fait simplement en permutant les variables concernées dans la formule propositionnelle. La sous-fonction χ est une fonction non-linéaire comprenant une opération XOR et une conjonction entre deux variables. Sa modélisation sous forme normale conjonctive rappelle celle adoptée dans ce chapitre, paragraphe 2.3. Par conséquent, nous avons suivi la même méthode pour traduire cette sous-fonction sous un ensemble de clauses. Enfin, la dernière sous-fonction ι, consiste en l'injection d'une constante via une nouvelle opération XOR. Il suffit donc de considérer de nouvelles variables pour définir la constante, de les affecter aux valeurs données par la définition du standard SHA-3 puis de générer les clauses en conséquence.

Les travaux autour de SHA-3 étant récents, nous n'aborderons pas dans cette thèse les attaques contre cette fonction de hachage.

2.6 Modélisation complète

La modélisation complète d'un processus de hachage se fait en concaténant les briques de base présentées ci-avant. Cet engrenage est réalisé grâce au bon chaînage entre les variables d'entrées et les variables de sortie des différents blocs. Ainsi, nous obtenons un très grand circuit logique, qui a pour commencement les variables du message d'entrée et pour fin l'empreinte. Evidemment, sous la forme d'un problème SAT, il n'y a qu'un grand ensemble de clauses qui ne respectent aucun ordre.

2.6.1 Création de formules par CRYPTLOGVER

Aujourd'hui, les transformations de Tseitin forment la méthode privilégiée pour représenter des circuits logiques en un formalisme logique. Le grand avantage est que le lot de règles définies par ces transformations permet de couvrir l'ensemble des portes logiques et la méthode peut facilement être implémentée de façon automatique. La littérature ne compte pas beaucoup d'outils ouverts et disponibles à tous permettant la modélisation de problèmes cryptographiques en un problème SAT. À notre connaissance, seul CRYPTLOGVER répond à cela et il est intéressant de s'arrêter un instant là-dessus.

CRYPTLOGVER a été conçu par Paweł Morawiecki, Marian Srebrny, et Mateusz Srebrny en 2013 afin de modéliser des algorithmes cryptographiques sous la forme de problèmes SAT. L'outil est sous licence Creative Commons Attribution et peut donc être *librement utilisée, à la condition de l'attribuer à l'auteur en citant son nom* [2]. Le programme a été ajouté à l'« ECRYPT II Tools for Cryptography » [3] qui comme son nom l'indique est un site web dont la vocation est de pourvoir la communauté d'outils pour la cryptographie et la cryptanalyse. CRYPTLOGVER a été réalisé de sorte à reproduire des attaques basées SAT solveurs. Le logiciel prend en entrée un algorithme écrit dans un langage de description (SystemVerilog en l'occurrence) et donne en sortie la CNF correspondante. Cette CNF étant bien sûr exploitable par un solveur SAT.

2.6.2 Notre générateur de formules

En pratique, pour chaque fonction de hachage, nous avons conçu un générateur de formules. Il est possible de sortir des formules SAT représentant tout le processus de hachage, mais aussi des formules répondant a un besoin spécifique. Ces programmes proposent de paramétrer la formule générée suivant diverses fonctionnalités :

- *Fixer l'étape de départ*. L'utilisateur peut faire le choix de créer une formule SAT représentant le processus de hachage à partir d'une certaine étape i. Dans ce cas, l'état interne initial (128 bits) est formé des quatre mots de 32 bits théoriquement calculés aux étapes i-4, i-3, i-2 et i-1.
- *Fixer l'étape de sortie*. Permet d'avoir des formules SAT pour des versions réduites de la fonction de hachage jusqu'à une étape j. Il est possible de combiner cette fonction avec la précédente et ainsi obtenir des formules SAT représentant le processus de hachage de l'étape i à l'étape j.
- *Fixer l'empreinte, totalement ou partiellement*. Ceci permet de créer des formules répondant au problème de la pré-image.
- *Fixer une partie du message d'entrée*. Pour le cas où on recherche une pré-image particulière et que l'on possède une certaine connaissance sur celle-ci.
- *Fixer la taille du message d'entrée*. Pour le cas où on recherche une pré-image particulière, type mot de passe.
- *Mode shell*. Ce mode permet d'avoir une interface d'interaction homme-machine. Ainsi, de nouvelles fonctionnalités sont disponibles et permettent certaines opérations en direct sur une instance particulière.
  - *Hachage*. Cela permet à l'utilisateur de saisir un message et d'obtenir une empreinte.
  - *Afficher le processus de hachage*. Cette option montre à l'utilisateur la valeur prise par chaque variable du processus de hachage, classée par catégorie (retenue, somme, états interne, etc.) et ordonnée par étape. Il est par exemple visuellement très facile de voir si le 26$^{\text{ème}}$ bit du mot décrivant le résultat de

2. http ://creativecommons.org/licenses/by/3.0/
3. http ://www.ecrypt.eu.org/tools/

la fonction non-linéaire étape 48 est fixé à 0, à 1 ou indéterminé. L'annexe 1 détaille une interface de visualisation réalisée en java permettant à l'utilisateur d'intervenir en direct avec la formule.

- *Flip*. Permet de changer la valeur d'une variable de 0 à 1 ou de 1 à 0.
- *Liste clauses non satisfaites*. Permet de voir les clauses qui ne sont pas satisfaites et/ou les littéraux qui les composent. De plus, nous pouvons aussi voir les variables libres dans ces clauses.
- *Liste des clauses unitaires*. Visualise toutes les clauses ne comprenant qu'un littéral.
- *Verrou empreinte*. Permet de bloquer les variables d'une empreinte. C'est une fonction qui peut par exemple être utilisée avec le *Flip* pour voir les changements provoqués dans le processus interne.
- *Statistiques littéraux*. Fonctionnalité qui fournit des statistiques sur la fréquence d'apparition d'un littéral. Il est aussi envisageable de voir dans quel type de clauses, en termes de taille, ce littéral apparaît.

2.6.3 Spécificités des formules SAT cryptographiques

Les formules créées sont très spécifiques du point de vue de la satisfaisabilité. Elles représentent un problème réel puisqu'il s'agit d'un problème cryptographique. Ajouté à cela, les primitives modélisées sont particulières car ce sont des fonctions de hachage, avec leurs propriétés mathématiques singulières. Toutes ces raisons combinées donnent les caractéristiques suivantes :

- **Ces formules sont toujours satisfaisables**. Résoudre ce genre de formules par un solveur SAT retourne une solution en moins d'une seconde. Celle-ci représente un modèle tel que l'affectation donnée à chaque variable du processus de hachage le rende correct. Parmi ces variables, sont incluses celles correspondant au message d'entrée et celles correspondant à l'empreinte.
  Si nous nous amusons à reconstituer en ASCII le message puis à le hacher dans l'algorithme cryptographique classique alors nous retombons bel et bien sur l'empreinte obtenue par l'affectation des variables de sortie données par le solveur SAT.
- En moyenne, **un faible pourcentage des variables (512 variables sur le nombre total de variables) suffisent à affecter l'ensemble des variables du processus**. Cette spécificité est tout à fait corrélée à la nature cryptographique du problème modélisé et rapproche nos instances brutes des problèmes dits CIRCUIT-SAT. En effet, propager un message dans une fonction de hachage permet de « descendre » dans le processus de hachage jusqu'à l'empreinte. Ici, affecter les variables correspondant au message d'entrée a pour conséquence d'affecter par propagation unitaire l'ensemble des variables du processus de hachage. Cette caractéristique signifie aussi qu'explorer seulement un faible pourcentage de l'arbre de recherche suffit à énumérer toutes les solutions, si le les variables du message d'entrée sont connues. Notons enfin que dans le cas où un chiffrement symétrique serait modélisé, affecter les variables

correspondant à une entrée ainsi que celles coïncidant avec la clé de chiffrement permettrait aussi de propager toutes les autres variables.

– **Chercher une solution particulière est très difficile** et par très difficile, nous entendons calculatoirement impossible. En effet, si nous prenons l'exemple qui consiste à affecter les variables de l'empreinte, alors chercher une solution avec un solveur SAT ne donnera rien. Cependant, pour une fonction de hachage caractérisée par une entrée de 512 bits et une empreinte de 128 bits, il existe $2^{(512-128)}$ solutions, soient 2^{384}. Tout l'intérêt du solveur SAT, est désormais d'en trouver au moins une. Remarquons que cette singularité est tout à fait attachée aux fonctions de hachage et n'est plus valide dès lors qu'on aborde un chiffrement symétrique ou asymétrique. Effectivement pour une attaque à textes clairs la clé recherchée est forcément unique. C'est cette principale raison qui a conduit nos travaux vers les fonctions de hachage, car le solveur SAT a théoriquement plus de chances de trouver une solution parmi plusieurs.

Dans le tableau II.4 nous donnons quelques statistiques à propos des formules SAT représentant les diverses fonctions de hachage que nous avons modélisées grâce à nos générateurs. Ces formules sont brutes, c'est-à-dire qu'elle n'ont pas été simplifiées, et aucune variable n'a été instanciée, si ce n'est les constantes. Étant donné la nature des fonctions de hachage et le fait qu'elles puissent prendre en entrée un message de n'importe quelle taille, nous avons dû limiter cette taille à une longueur fixe pour la modélisation. Cette borne a été définie à 1 seul bloc de 512 bits.

À la vue de ces statistiques, nous remarquons que la fonction qui demande le moins de ressources pour être modélisée est MD4, suivie de MD5 et enfin SHA. La différence entre les deux fonctions de la famille MD s'explique par l'ajout d'étapes et par l'addition à deux opérandes qui existe uniquement dans MD5. Quant à SHA, il y a encore une fois un nombre d'étapes plus important, mais c'est surtout l'addition à cinq opérandes - engendrant deux fois plus de clauses qu'une addition à quatre opérandes - qui produit beaucoup plus de clauses. Enfin, nous noterons que les modélisations de SHA-0 et SHA-1 nécessitent exactement le même nombre de clauses, de variables et de littéraux puisque ces primitives ne se différencient que par un décalage circulaire dans la diversification du message.

Fonction	Clauses	Variables	Littéraux
MD4	144 885	6 690	824 378
MD5	224 653	12 749	1 236 634
SHA-0	491 791	12 779	3 283 930
SHA-1	491 791	12 779	3 283 930
SHA-3	869 120	92 160	3 930 880

TABLE II.4 – Statistiques pour les formules SAT obtenues par rapport à la fonction de hachage modélisée

Dans cette thèse, seule la phase de modélisation de SHA-3 en un problème SAT est abordée. Les attaques sur cette fonction de hachage sont des travaux en cours. Remarquons tout de même que les statistiques en termes de nombre de clauses et de variables sont très différents de ces congénères. Cela s'explique principalement par le schéma de construction de la fonction éponge qui diffère beaucoup de celui de Merkle-Damgård.

2.6.4 Aparté sur les différentes modélisations de MD5

Dans la table II.5, nous donnons une comparaison des statistiques pour les différents encodage proposés dans cette thèse, pour la fonction de hachage MD5. Comme nous l'avons précédemment affirmé, les formules SAT générées ont des caractéristiques assez divergentes, que ce soit pour le nombre de variables, le nombre de clauses et leurs tailles. Nous pouvons remarquer par exemple qu'entre l'encodage « Global » et l'encodage « Hybride » , il y a une différence d'environ 70 000 clauses et 7 500 variables, ce qui est plutôt conséquent. De même, si nous nous arrêtons un instant sur le nombre de clauses de taille 2, l'encodage « Découpé » en comporte environ 33 000 quand les autres modélisations en ont au maximum 1 748. La différence est colossale et nous verrons dans le chapitre IV qu'elle a peut-être son importance lors de la résolution pratique.

Nous nous sommes aussi intéressés à certains ratios entre le nombre de variables, le nombre de clauses, et le nombre de littéraux, afin de voir si les modélisations comportaient de grandes différences au niveau structurel. Ces données sont souvent mises en avant dans le domaine de la satisfaisabilité dès lors que l'on cherche à étudier des formules SAT difficiles à résoudre. Le ratio Littéraux/Variables par exemple permet d'avoir une connaissance intéressante puisqu'il donne un aperçu de la fréquence moyenne d'apparition de chaque variable dans les clauses. Concernant l'encodage « Global » , ce taux est de 97 alors qu'il est d'environ 30 pour les autres modélisations. Cela signifie que lorsqu'une variable est affectée dans l'encodage « Global » , alors beaucoup plus de clauses sont directement concernées. Encore une fois, cette information peut jouer un rôle de premier rang pour la résolution des formules SAT.

Encodage	Global	Découpé	Hybride
Clauses	224 653	158 678	154 920
Variables	12 749	18 773	20 423
Littéraux	1 236 634	548 980	612 028
2-cl	381	33 438	1 748
3-cl	1 904	18 856	4 156
4-cl	48 720	106 384	149 016
5-cl	6 608	0	0
6-cl	167 040	0	0

TABLE II.5 – Statistiques pour les formules SAT obtenues en fonction de l'encodage de l'addition à quatre opérandes

Encodage	Global	Découpé	Hybride
Variables / Clauses	0,06	0,12	0,13
Littéraux / Clauses	5,50	3,46	3,95
Littéraux / Variables	97,00	29,24	29,97
Coef décroissant	3,15	26,31	10,30

TABLE II.6 – Différents ratios sur les statistiques à propos de diverses modélisations de fonctions de hachage

Le choix réalisé pour nos modélisations a suivi la direction poussant à minimiser le nombre de variables. Pour cela, nous avons considéré la fonction de hachage comme un assemblage de petits circuits logiques, eux mêmes reposant sur une représentation qui leur est propre. Cette dernière est bien souvent réalisée de façon exhaustive, en exploitant la table de vérité du circuit logique pour en dégager des clauses. Cette table de vérité est quant à elle créée en considérant *l'ensemble* des conséquences engendrées par toutes les entrées possibles.

De ce fait, la représentation est exhaustive et est en discordance avec l'idée d'avoir un problème exprimé dans sa plus simple expression. Il nous est alors paru important d'affiner la modélisation pour optimiser le nombre de variables et de clauses. À nos yeux ceci peut être tout à fait crucial pour une future résolution et nous a poussés à pré-traiter nos formules.

3 Simplification du problème

Dans cette section nous présentons quelques raisonnements logiques qui permettent de réduire le nombre de variables et/ou de clauses d'une formule SAT. Ces déductions logiques sont mécaniques et toujours vraies quelle que soit l'affectation donnée aux variables, sans altérer la décidabilité de la formule résultante. Une manière plus formelle de l'exprimer est la suivante. Soit $\mathcal{F}$ une formule propositionnelle et $\mathcal{F}'$ sa formule simplifiée, tout modèle (resp. contre-modèle) de $\mathcal{F}'$ est aussi un modèle (resp. contre-modèle) pour $\mathcal{F}$.

La complexité d'une formule SAT non-aléatoire est difficile à évaluer mais peut relativement dépendre de trois paramètres : le nombre de variables, le nombre de clauses et la taille de ces clauses. À travers le développement de nouveaux solveurs, la communauté SAT a mis en lumière plusieurs techniques issues du raisonnement logique pour tenter de compresser au mieux les contraintes et restreindre la taille de la formule CNF. Quelques outils de pré-traitement statiques ont été produits pour répondre à ce besoin, comme par exemple HYPBINRES [Bacchus 2003] ou SATELITE [Eén 2005]. Généralement ce genre de programme est très efficace et permet d'obtenir des formules profondément simplifiées. Cependant, les techniques utilisées dans ces outils sont uniquement focalisées sur la réduction du problème, sans prendre en compte qu'un raisonnement logique peut aussi permettre l'enrichissement de celui-ci par l'ajout de nouvelles connaissances pertinentes pouvant faciliter sa future résolution.

Ce manque nous a conduits à développer notre propre pré-traitement en ayant pour objectif de répondre à ces deux besoins. Cette partie sera tout d'abord dédiée à la présentation

des simplifications logiques implantées dans notre pré-traitement. Puis, quelques résultats obtenus sur nos formules seront annoncés et complétés par un ensemble concret d'équivalences entre plusieurs variables qui pourraient être utilisées pour toute autre cryptanalyse, notamment différentielle ou algébrique.

3.1 Déductions logiques classiques

Dès que les bases de la logique propositionnelle ont été posées, plusieurs règles d'inférence ont été mises en avant et certaines d'entre elles donnent la possibilité de factoriser plusieurs informations en une. Ici, nous présentons quelques unes de ces règles. Dans le contexte des problèmes structurés, les déductions logiques, aussi simples soient elles, permettent de simplifier le problème de façon significative. À titre d'exemple, l'addition de quatre opérandes sur 1 bit nécessite 192 clauses et 1344 littéraux. La version simplifiée comporte seulement 128 clauses, plus petites et 832 littéraux, ce qui représente un gain de 36% sur le nombre de clauses et 32% sur le nombre de littéraux. Les gains sont équivalents quand l'addition est étendue à 32 bits et sont donc très intéressants. Nous donnons ci-après quelques définitions concernant ces méthodes de simplification. Soit $\mathcal{F}<\mathcal{V},\mathcal{C}>$, une formule propositionnelle.

Définition 44 : *On appelle **redondance** deux informations qui sont en tout point identiques.*

Exemple 12 : *Soit $c \in \mathcal{C}$ telle que $c = (a \vee a \vee b)$. Puisque a est présent deux fois dans la même clause sous le même signe alors il y a redondance d'information et un littéral a peut être supprimé. La clause c devient $c = (a \vee b)$.*

Exemple 13 : *Soient deux clauses $c1, c2 \in \mathcal{C}$ telles que $c1 = (a \vee b)$ et $c2 = (a \vee b)$. $c1$ et $c2$ sont identiques donc il y a redondance d'information et une des deux clauses peut être supprimée.*

Définition 45 : *Dans un contexte logique, **une tautologie** est une proposition qui est toujours* VRAIE*, quelle que soit l'interprétation donnée aux variables d'une formule propositionnelle.*

Exemple 14 : *Soit $v \in \mathcal{V}$ une variable booléenne et soit $c \in \mathcal{C}$ la clause $(v \vee \bar{v})$. La clause c est toujours* VRAIE*, quelle que soit la valeur affectée à v. C'est donc une tautologie.*

En logique propositionnelle, il est inutile de garder des tautologies puisqu'elles représentent une information qui n'a aucune influence sur la décidabilité du problème. Le prétraitement d'une formule SAT les repèrera pour tout simplement les éliminer.

Définition 46 : *On appelle **subsomption** l'inclusion logique d'une proposition dans une autre, quelles que soient les valeurs de vérité assignées aux variables de la formule. Soit $p1$ et $p2$ deux propositions telles que $p1 \Rightarrow p2$, on dit alors que $p1$ **subsume** $p2$.*

Exemple 15 : *Soient deux clauses $c1, c2 \in \mathcal{C}$ telles que $c1 = (a \vee b \vee c)$ et $c2 = (a \vee b)$, quelles que soient les valeurs prises par a, b et c, si $c2$ est* VRAIE *alors nécessairement $c1$ l'est aussi. Par conséquent $c2 \Rightarrow c1$ et donc $c2$ subsume $c1$.*

D'un point de vue logique, si la formule propositionnelle est satisfaisable, alors toutes les clauses doivent être satisfaites. Dans ce sens, il suffit de satisfaire les clauses subsumantes pour aussi satisfaire les subsumées. Ces dernières deviennent donc obsolètes.

À l'inverse, si la formule est insatisfaisable, alors si une clause subsumante est contredite, peu importe l'état de la clause subsumée, elle restera contredite. Cela signifie que peu importe la décidabilité de la formule SAT, la clause subsumante aura toujours la priorité logique sur la clause subsumée. Autrement dit, il est inutile de garder les clauses subsumées qui peuvent donc être supprimées.

Définition 47 : *Lorsque deux clauses possèdent un littéral l en commun mais exprimés sous deux signatures opposées, alors les clauses peuvent par* ***résolution*** *[Robinson 1965] générer une nouvelle clause composée de tous leurs littéraux respectifs, excepté l et $\overline{l}$. Cette clause est appelée* ***résolvante****.*

Exemple 16 : *Soient deux clauses $c1 = (a \vee b \vee c)$ et $c2 = (\overline{b} \vee d \vee e)$. Puisque $c1$ et $c2$ possèdent le littéral b en commun écrit sous deux signes différents, une troisième clause $c3$ peut être générée de sorte que $c3 = (a \vee c \vee d \vee e)$.*

La résolution permet de créer de nouvelles clauses. Dans le cadre d'une simplification, ce processus n'est donc *a priori* pas adapté pour faire baisser le nombre de clauses et de variables. Cependant, dans certains cas, il permet tout de même de réduire le problème.

i) Si la résolvante permet de simplifier d'autres clauses il peut être intéressant de la générer. Considérons l'exemple suivant où $c1$ et $c2$ sont deux clauses telles que $c1 = (a \vee b \vee c \vee d)$ et $c2 = (\overline{b} \vee c \vee d)$. Par résolution, on obtient la résolvante $c3 = (a \vee c \vee d)$. Cette clause est très intéressante, car elle subsume $c1$. Cela signifie qu'en ajoutant $c3, c1$ peut être supprimée. Cette opération est appelée **résolution-subsumante** et dans le cas présent le gain total est un littéral. Remarquons que dans certaines situations, la résolvante peut subsumer les deux clauses et le gain est alors d'une clause et un littéral.

ii) La résolution est aussi très intéressante quand elle produit des clauses de taille 2 qui permettent ensuite d'accélérer la phase de résolution. C'est le cas quand la résolvante provient de deux clauses de taille 2, ou lorsqu'une double résolution-subsumante est effective sur deux clauses de taille 3. Par exemple si $c1 = (a \vee b \vee c)$ et $c2 = (\overline{a} \vee b \vee c)$, la résolvante $c3 = (b \vee c)$ est de taille 2 et subsume $c1$ et $c2$.

3.2 *Look-ahead* statique

Nous avons aussi mis en place un traitement local utilisé dans les solveurs et appelé **look-ahead**. Il consiste à évaluer les conséquences logiques d'un branchement de variable dans l'arbre de recherche. En d'autres mots, cela permet de voir comment se comportent, au sens logique, les variables et les clauses quand une variable est affectée à VRAI ou à FAUX. De cette évaluation peut découler de nombreuses informations très intéressantes comme par exemple des relations d'équivalences entre les variables, des littéraux qui sont

assignés ou bien la découverte de clauses binaires. Nous proposons d'illustrer ces propos à travers les situations présentées ci-dessous :

- **Découvrir des littéraux fixés**.
  - Affecter une valeur de vérité à une variable peut amener à une contradiction dans la formule. On peut donc en déduire que cette affectation ne doit jamais avoir lieu et par conséquent l'affectation inverse doit nécessairement être faite.

    Exemple : Si $(a \leftarrow \text{VRAI}) \Rightarrow \text{FAUX}$ alors $a \leftarrow \text{FAUX}$.

  - Si quelle que soit la valeur de vérité d'une variable, elle implique toujours la même affectation d'une valeur de vérité à une autre variable, alors cette dernière peut toujours être affectée de la sorte.

    Exemple : Si $(a \Rightarrow \text{b})$ et $(\overline{a} \Rightarrow \text{b})$ alors $b \leftarrow \text{VRAI}$.

- **Nouvelles clauses binaires**.
  - Si donner une valeur de vérité à une variable oblige une autre variable à prendre une certaine valeur alors une clause binaire peut être ajoutée. Il s'agit tout simplement de la relation d'implication.

    Exemple : Si $a \Rightarrow b$ alors $(\overline{a} \vee b)$.

  - Si donner une valeur de vérité à deux variables amène à une contradiction alors une clause binaire peut être ajoutée pour traduire ce conflit. Il en est de même pour les clauses ternaires.

    Exemple : Si $(\overline{a} \wedge \overline{b}) \Rightarrow \text{FALSE}$ alors la clause $(a \vee b)$ est ajoutée.

  - Si affecter une valeur de vérité à une variable implique affecter une valeur à d'autres variables et affecter la valeur de vérité opposée implique affecter d'autres variables, alors toutes les combinaisons entre ces affectations respectives peuvent être ajoutées sous la forme de clauses binaires.

    Exemple : Si $a \Rightarrow (x_0 \wedge x_1 \wedge \cdots \wedge x_n)$ et $\overline{a} \Rightarrow (y_0 \wedge y_1 \wedge \cdots \wedge y_m)$ alors toutes les clauses binaires de la forme $(x_i \vee y_j)$, avec $1 \leq i \leq n$ et $1 \leq j \leq m$, sont ajoutées à la formule propositionnelle.

- **Nouvelles clauses subsumantes**. En enrichissant le principe du look-ahead, il est possible de vérifier plusieurs implications dans le but de produire des clauses subsumantes. Soit une clause $c = (x_0 \vee x_1 \vee \ldots \vee x_i \vee \ldots \vee x_k)$.
  - Si $(\overline{x_0} \wedge \overline{x_1} \wedge \ldots \wedge \overline{x_i}) \Rightarrow \text{FAUX}$ alors c peut être remplacée par $(x_0 \vee x_1 \vee \ldots \vee x_i)$. En effet, si l'affectation d'une partie d'une clause amène à un conflit, alors nécessairement cette affectation doit être évitée pour rechercher un modèle. Pour cela une clause est générée et ajoutée au problème. Or, cette nouvelle clause va nécessairement subsumer la clause de base, qui sera donc retirée du problème. Par cette opération, le problème est un peu plus contraint ce qui est une bonne chose en terme de résolution car une branche de l'arbre de recherche est coupée.

– Si $(x_0 \wedge \overline{x_1} \wedge \ldots \wedge \overline{x_i}) \Rightarrow$ FAUX alors c peut être remplacée par $(x_1 \vee \ldots \vee x_i)$. De même que précédemment, à l'exception près que la clause générée ici ne subsume pas la clause de base. Cependant, elle permet de passer par une étape de résolution-subsumante.

4 Détecter des équivalences

L'existence d'équivalences logiques dans un processus de hachage correct signifie qu'au moins deux bits (cela peut être plus) sont liés par leurs valeurs respectives dans n'importe quel modèle. En pratique et de façon formelle, ceci peut être vu comme une variable qui a toujours la même valeur (ou la valeur opposée) qu'une autre. Quand cela arrive, les deux bits représentent la même information et seulement un seul d'entre eux a besoin d'être considéré dans le processus. Une telle relation est notée $\Leftrightarrow$ et correspond à une relation d'implication bidirectionnelle.

Détecter des équivalences peut se faire de plusieurs manières. Dans certaines situations, elles sont déjà dans la formule propositionnelle et il suffit de les repérer. Mais des fois, il arrive qu'elles ne soient pas explicitement exprimées et il est alors extrêmement intéressant de les extraire. En réalité, cela constitue la création de ponts logiques qui ne sauraient être débusqués autrement.

Nous proposons dans l'exemple qui suit de montrer comment détecter ce genre d'équivalences. Soit $\mathcal{F}$ une formule CNF possédant les clauses suivantes :

$$\begin{array}{llll} c_1 = (a \vee \overline{b}) & c_2 = (\overline{a} \vee \overline{f}) & c_3 = (b \vee \overline{c}) & c_4 = (c \vee \overline{d}) \\ c_5 = (f \vee \overline{g}) & c_6 = (g \vee \overline{h}) & c_7 = (a \vee d \vee \overline{e}) & c_8 = (\overline{a} \vee h \vee e) \end{array}$$

La détection d'équivalences est ainsi réalisée :

1) Si a est fixé à FAUX, il peut être déduit des clauses c_1, c_3, c_4 et c_7 que b, c, d puis e doivent aussi être affectées à FAUX pour ne pas contredire la formule. Par conséquent, a vaut FAUX implique e vaut FAUX et on a formellement $\overline{a} \Rightarrow \overline{e}$. Ceci est matérialisé par l'ajout de la clause $(a \vee \overline{e})$. Une clause étant commutative, on peut réécrire $(\overline{e} \vee a)$. Or cette clause peut se traduire par la relation $e \Rightarrow a$.

2) De la même façon, si a est VRAI alors f, g, h doivent être affectées à FAUX et e doit être VRAI. Donc $a \Rightarrow e$

3) Par conséquent, $e \Rightarrow a$ et $a \Rightarrow e$, donc $a \Leftrightarrow e$.

Faire émerger les relations d'équivalences est un processus très avantageux pour la simplification d'un problème. En reprenant notre exemple, cela signifie que peu importe les solutions pour la formule $\mathcal{F}$, a et e possèdent nécessairement la même valeur de vérité. De plus, il est désormais possible de remplacer toutes occurrences de e par a (ou l'inverse) dans l'espoir d'engendrer une cascade de nouvelles simplifications. Dans notre exemple, cette substitution permet de rendre c_7 et c_8 obsolètes, puisque $a \vee \overline{a}$ est une tautologie.

La détection d'équivalences a aussi été implantée dans notre pré-traitement. Dans cette thèse, nous ne présentons que nos résultats en matière de pré-traitement relativement à une formule SAT répondant à un encodage « Global » . Notre pré-traitement est bien évidemment conçu pour être appliquable à toutes formules SAT, y compris celles en dehors du contexte cryptographique.

Nous proposons dans le tableau II.7 un ensemble de classes d'équivalences issues des formules SAT représentant MD5 et SHA-1. La notation utilisée est celle présentée dans le chapitre I, partie 3.3.1. Ajouté à cela, à l'intérieur des crochets, la position du bit dans le mot associé. De plus, tC_i représente le vecteur de retenues généré lors de l'addition à deux opérandes à l'étape i ; fC_i, sC_i et S_i sont respectivement les vecteurs de premières et secondes retenues puis le vecteur des sommes, tous trois générés dans l'addition à quatre (ou cinq pour SHA) opérandes à l'étape i. À titre d'exemple, nous avons pour MD5 la relation $\overline{S_0[25]} \Leftrightarrow Q_1[0]$ qui signifie littéralement *la variable associée au 25ième bit de la somme à quatre opérandes au tour 0 est équivalente au bit 0 de l'état* Q_1 . Notons enfin que ces équivalences sont toujours vraies et donc exploitables dans un contexte qui peut aller au delà de la simple cryptanalyse logique.

Équivalences dans MD5
$\overline{tC_0[0]} \Leftrightarrow Q_1[0], f_1[0] \Leftrightarrow Q_1[0], \overline{S_0[25]} \Leftrightarrow Q_1[0]$
$\overline{S_0[0]} \Leftrightarrow fC_0[0], \overline{f_1[2]} \Leftrightarrow Q_1[2], \overline{f_1[4]} \Leftrightarrow Q_1[4], \overline{f_1[5]} \Leftrightarrow Q_1[5]$
$\overline{f_1[6]} \Leftrightarrow Q_1[6], \overline{f_1[12]} \Leftrightarrow Q_1[12], \overline{f_1[1]} \Leftrightarrow Q_1[1]$
$M_0[0] \Leftrightarrow fC_0[0], f_1[8] \Leftrightarrow Q_1[8], f_1[9] \Leftrightarrow Q_1[9], \overline{f_1[10]} \Leftrightarrow Q_1[10]$
$f_1[13] \Leftrightarrow Q_1[13], \overline{f_1[14]} \Leftrightarrow Q_1[14], f_1[16] \Leftrightarrow Q_1[16], \overline{f_1[17]} \Leftrightarrow Q_1[17]$
$f_1[18] \Leftrightarrow Q_1[18], \overline{f_1[20]} \Leftrightarrow Q_1[20], \overline{f_1[21]} \Leftrightarrow Q_1[21], f_1[22] \Leftrightarrow Q_1[22]$
$f_1[24] \Leftrightarrow Q_1[24], f_1[25] \Leftrightarrow Q_1[25], f_1[26] \Leftrightarrow Q_1[26], \overline{f_1[28]} \Leftrightarrow Q_1[28]$
$f_1[29] \Leftrightarrow Q_1[29], f_1[30] \Leftrightarrow Q_1[30], \overline{Hb[0]} \Leftrightarrow cHb[0], \overline{Ha[0]} \Leftrightarrow cHa[0]$
$\overline{cHd[1]} \Leftrightarrow Hd[1], \overline{Q_{62}[1]} \Leftrightarrow Hd[1], \overline{cHc[1]} \Leftrightarrow Hc[1], \overline{Q_{63}[1]} \Leftrightarrow Hc[1]$
$\overline{Q_{64}[0]} \Leftrightarrow Hb[0], \overline{Q_{61}[0]} \Leftrightarrow Ha[0], Hd[0] \Leftrightarrow Q_{62}[0], Hc[0] \Leftrightarrow Q_{63}[0]$

Équivalences dans SHA-1
$\overline{sC_0[28]} \Leftrightarrow fC_0[28], \overline{sC_0[27]} \Leftrightarrow fC_0[27], \overline{sC_0[25]} \Leftrightarrow fC_0[25]$
$\overline{sC_0[21]} \Leftrightarrow fC_0[21], \overline{sC_0[20]} \Leftrightarrow fC_0[20], \overline{sC_0[19]} \Leftrightarrow fC_0[19]$
$\overline{sC_0[24]} \Leftrightarrow sC_0[17], \overline{sC_0[15]} \Leftrightarrow fC_0[15], \overline{sC_0[12]} \Leftrightarrow fC_0[12]$

TABLE II.7 – Équivalences détectées dans les processus de hachage de MD5 et SHA-1.

5 Pré-traitement dédié

Nous avons programmé un pré-traitement parallèle en OPENMP qui reprend toutes les simplifications logiques présentées ci-avant. Les traitements mis en jeu sont très gourmands en calcul et sur des formules assez imposantes - comme le sont celles que nous avons générées - un calcul séquentiel peut aisément prendre plusieurs semaines.

MD4				
	Clauses	**Variables**	**Littéraux**	**2-cl**
Avant	144 885	6 690	824 378	2 317
Après	108 081	6 673	531 100	3 154
Gain	-25,4%	-17	-35,6%	+36,1%
MD5				
	Clauses	**Variables**	**Littéraux**	**2-cl**
Avant	224 653	12 749	1 236 634	381
Après	171 235	12 721	814 096	1 305
Gain	-23,8%	-28	-34,2%	+242,5%
SHA-0 / SHA-1				
	Clauses	**Variables**	**Littéraux**	**2-cl**
Avant	491 791	12 779	3 283 930	259
Après	375 195	12 771	2 210 708	908
Gain	-23,7%	-8	-32,7%	+250,6%

TABLE II.8 – Comparaison entre les statistiques à propos du nombre de variables et de clauses, avant et après l'application de notre pré-traitement.

Appliqué sur nos formules, nous avons réussi à diminuer de façon drastique leurs caractéristiques. Le tableau II.8 montre tout l'intérêt d'avoir simplifié nos instances relativement à un encodage « Global » . En observant ces chiffres, nous pouvons voir que le nombre de clauses est réduit d'un quart quand le nombre de littéraux a lui chuté d'un tiers. Mais, ce qui est d'autant plus intéressant c'est le nombre de clauses de taille 2 qui a été multiplié par plus de deux pour MD5 et SHA-1. Cela souligne les clauses de taille 3 qui ont été réduites mais aussi la création de ponts logiques qui peuvent permettre d'accélérer la résolution. L'apport de ce pré-traitement a d'ailleurs été mesuré à travers les expérimentations décrites au chapitre IV.

Dans la table II.9, nous donnons une comparaison de notre pré-traitement avec COPROCESSOR [Manthey 2012] (paramétrage par défaut), un outil récent issu de la communauté SAT, sur nos formules brutes pour MD5 et SHA-1. Nous pouvons ainsi remarquer l'apport important de clauses de tailles 2 dans notre méthode qui tend à penser que nos formules logiques sont plus consistantes notamment grâce à l'apport de ponts logiques.

Pré-traitements sur MD5						
	Clauses	**2-cl**	**3-cl**	**4-cl**	**5-cl**	**6-cl**
CNF de base	224 653	381	1 904	48 720	6 608	167 040
COPROCESSOR	183 620	832	28 142	40 409	55 755	58 482
Dans cette thèse	**171 235**	**1 305**	28 119	40 408	42 921	58 482

Pré-traitements sur SHA-1							
	Clauses	**2-cl**	**3-cl**	**4-cl**	**5-cl**	**6-cl**	**7-cl**
CNF de base	491 791	259	1 356	20 176	38 960	13 440	417 600
COPROCESSOR	416 421	410	8 489	55 410	70 923	135 078	146 111
Dans cette thèse	**375 195**	**908**	8 192	33 461	91 507	94 952	146 175

TABLE II.9 – Comparaison des statistiques obtenues avec notre pré-traitement et avec COPROCESSOR [Manthey 2012] paramétré par défaut sur les formules logiques pour MD5 et SHA-1

6 Conclusion

Dans ce chapitre nous proposons une méthode générique pour modéliser des problèmes cryptographiques. Cette approche s'appuie sur la représentation des opérations complexes (additions, fonctions non linéaires, etc.) sous forme logique de façon à les concaténer ensuite grâce à un bon encodage des variables. Ce procédé a pour avantage d'avoir des formules avec un nombre de variables relativement restreint tout en préservant un détail assez précis, permettant une analyse fine du processus de hachage. Nous verrons d'ailleurs dans la suite de cette thèse diverses applications en cryptanalyse logique rendues possibles grâce à ce modèle grain fin. Ce travail de modélisation a été valorisé par une contribution dans [Legendre 2012a].

Ajouté à cela, nous avons mis en place un pré-traitement efficace qui permet de compresser fortement une formule propositionnelle, mais qui présente en plus l'avantage d'apporter des ponts logiques sous forme de clauses binaires. Pour faire le lien avec le monde électronique, nos formules sont des circuits logiques où les câblages inutiles ont été retirés et où des pistes conductrices faisant office de raccourcis ont été ajoutées avec l'aide d'un pré-traitement spécifique.

CHAPITRE III

Analyse structurelle

La modélisation de fonctions de hachage cryptographiques permet la génération de formules SAT. Ces formules ont des spécificités très intéressantes du point de vue de la satisfaisabilité notamment parce qu'elles transcendent des propriétés cryptographiques singulières. Transposé en un formalisme logique le problème de départ conserve son sens et sa structure initiale.

L'usage le plus courant pour ce genre de formules propositionnelles est de les étendre à la représentation du problème de la pré-image. Pour ce faire, c'est très simple puisqu'il suffit de donner une affectation à toutes les (ou certaines) variables qui correspondent à l'empreinte dans la formule. De cette instanciation découle une formule SAT extrêmement difficile à résoudre. Bien souvent, devant cette difficulté ce sont des versions réduites du processus de hachage qui sont encodées.

En fonction du nombre de variables fixées dans l'empreinte et du nombre d'étapes modélisées, il est aisé de paramétrer des formules aux profils très variés. Ces formules sont ensuite utilisées pour deux aspects : donner une mesure de la résistance à la pré-image pour la fonction de hachage ou créer des formules logiques compliquées à résoudre dans le but de nourrir les benchmarks des compétitions SAT.

Dans le cadre de cette thèse, nous nous sommes seulement intéressés à l'aspect cryptanalyse. Les instances SAT représentant une seule ronde de la fonction de hachage sont triviales car les variables du message d'entrée sont toutes libres. Ces formules sont donc résolubles en une fraction de seconde par un solveur SAT.

La difficulté arrive en ajoutant des étapes qui reprennent des variables du message d'entrée déjà utilisées dans des étapes précédentes. Le problème qui est alors posé est de savoir à partir de combien d'étapes additionnées il devient impossible de résoudre la formule SAT. Cette frontière entre l'espace où la recherche de solution est praticable et celle où elle ne l'est plus est déterminée par la capacité du solveur à trouver un modèle dans un temps raisonnable. Les expérimentations effectuées durant cette thèse (voir chapitre IV), nous ont permis de vérifier que la résolution de ce genre de problème a une complexité en temps qui croit exponentiellement en fonction du nombre d'étapes modélisées.

Un des objectifs annoncés durant cette thèse a été de donner notre mesure de la résistance à la pré-image des fonctions de hachage étudiées. Cela signifie que nous avons déterminé un seuil d'étapes à partir duquel il n'est plus possible d'inverser la fonction par une approche SAT. Quand ce seuil est atteint, il devient naturel de se concentrer autour de la résolution de la formule pour laquelle le solveur SAT ne trouve pas de solution. C'est dans cette optique que nous avons tenté d'apprendre de nouvelles informations structurelles sur le problème sous-jacent.

Pour ce faire, nous nous sommes penchés sur une approche probabiliste du problème. Ainsi, dans une première phase, des données probabilistes sur les valeurs de vérité affectées aux variables ont été produites. Ces informations sont contenues dans une base de données permettant d'apprécier le comportement en moyenne des variables du processus de hachage. Ce pool d'informations a ensuite permis d'extraire des probabilités insolites pouvant être utilisées à des fins cryptanalytiques. Cette partie commence donc par présenter ces travaux et la découverte de relations probabilistes entre certaines variables, définies par la notion de « quasi-relations » .

Une seconde étude a été vouée à l'examen des clauses pour exhiber des structures remarquables. Deux pistes différentes ont été approfondies. Tout d'abord, nous avons tenté de savoir si les clauses pouvaient être classées dans une catégorie particulière de problèmes SAT simplement en observant la table de vérité des opérations modélisées. Par ailleurs, un nouveau raisonnement probabiliste sur nos formules a été appliqué pour dégager des informations conjecturales sur la façon dont les clauses sont satisfaites lorsqu'une solution au problème est trouvée. Ceci nous a permis de définir un profil d'affectations correctes pour chaque clause du processus de hachage.

1 Comportement des variables

Afin de comprendre de quelles façons étaient affectées les variables dans une solution type, nous avons produit des données probabilistes. Ces informations représentent plus précisément l'affectation en moyenne, comprise entre 0 et 1, donnée à chaque variable d'un processus de hachage qui s'est correctement déroulé.

Une des caractéristiques liées aux fonctions de hachage est que les empreintes générées en sortie doivent être assez confuses pour paraître aléatoires. Le chaos produit au cœur du processus de hachage doit lui aussi être assez important pour qu'aucune information intermédiaire ne puisse aider à retrouver une information secrète. En ce sens, cela laisse entendre que tous les états internes d'une telle fonction doivent aussi sembler aléatoires. Dans cette section, nous commençons par présenter le protocole de génération de nos probabilités. Puis, nous montrons à travers une image une vue représentative du processus de hachage moyen. Nous verrons que cette méthodologie nous a permis de cibler aisément les variables qui prennent un comportement tout à fait surprenant et donc qui nous intéressent fortement. Nous concluons par l'utilisation de ces variables spécifiques pour la définition de relations probabilistes pouvant grandement aider une résolution SAT du problème.

1.1 Quelques remarques sur les probabilités

Cette partie est dédiée à la notation utilisée pour écrire nos probabilités.

- Soit n, le nombre de variables de la formule. Soit v, une variable propositionnelle numérotée dans $\mathbb{N}_*$. On notera $P(v)$ **la probabilité** pour que la variable v soit à 1 et $P(-v)$ la probabilité pour que la variable v soit à 0. On a donc $P(v) + P(-v) = 1$. Dans la suite, on parlera de probabilités *simples* pour faire allusion à ce type de probabilités.

- Soient les variables propositionnelles v et w numérotées dans $\mathbb{N}_*$. La probabilité d'avoir simultanément $v = 1$ et $w = 1$ est donnée par $P(v \wedge w)$.
- Soit $P(v \mid w)$, **la probabilité conditionnelle**, correspondant à la probabilité d'avoir v sachant w. Suivant les valeurs de v et w dans $\{0,1\}$, quatre probabilités peuvent correspondrent : $\{ P(-v \mid -w), P(-v \mid w), P(v \mid -w), P(v \mid w) \}$.

Remarque 9 : *En théorie des probabilités, $P(v \mid w)$ est définie par :*

$$P(v \mid w) = \frac{P(v \wedge w)}{P(w)}$$

Remarque 10 : *Puisque $P(w) = P(w \wedge w)$ alors :*

$$P(v \mid w) = \frac{P(v \wedge w)}{P(w \wedge w)}$$

La remarque 10 signifie qu'il est possible de définir à la fois les probabilités simples et les probabilités conditionnelles uniquement avec la seule information de type $P(v \wedge w)$.

1.2 Représentation des probabilités

Soit $\mathcal{M}_{(2n+1)\text{x}(2n+1)}$, une matrice carrée contenant les probabilités, avec n le nombre de variables dans la formule SAT représentant la fonction de hachage étudiée.

Soient $v, w \in [\ 1,\ n\ ]$, $\mathcal{M}_{v,w} = P(v \wedge w)$. Nous représentons cette matrice sous la forme d'un repère orthonormé $\mathcal{R}$ qui a pour centre $\mathcal{M}_{0,0}$. La figure III.1 montre ce repère où le point de coordonnées (v, w) représente la probabilité $P(v \wedge w)$.

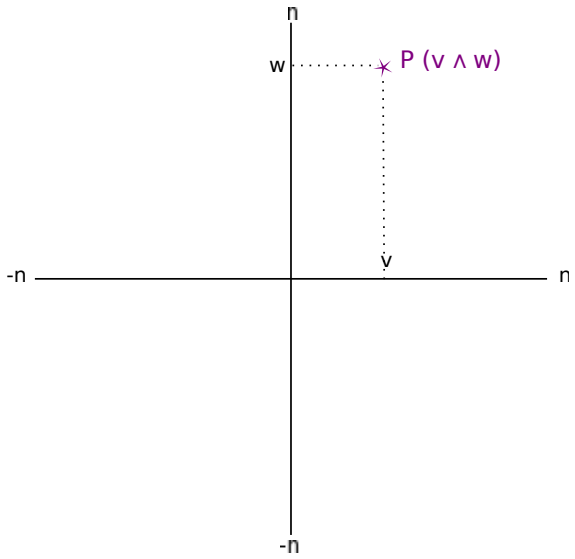

FIGURE III.1 – Représentation de la probabilité $P(v \wedge w)$

Remarque 11 : *On dénombre quatre cas possibles, suivant que v ou w soient positifs ou négatifs, donnés par :* $\{ P(-v \wedge -w), P(-v \wedge w), P(v \wedge -w), P(v \wedge w) \}$.

La figure III.2 est un repère sur lequel sont positionnés tous ces cas.

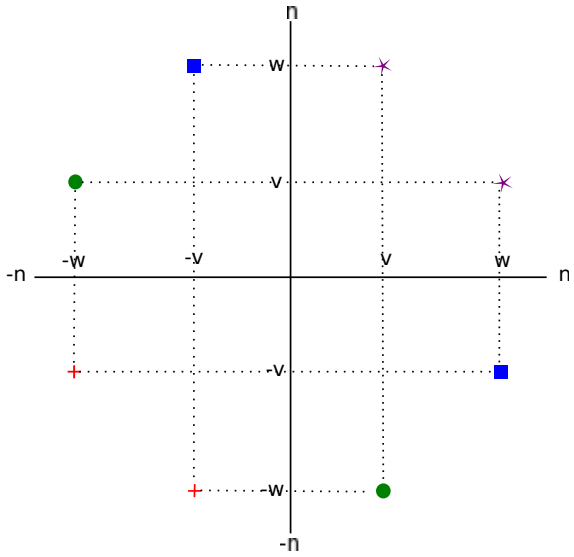

FIGURE III.2 – Représentation de tous les cas possibles pour deux variables v et w

Remarque 12 : *Pour tous* $v, w \in [\ 1,\ n\ ], P(v \wedge w) = P(w \wedge v)$.

Plus précisément, on a les égalités suivantes :

$$\begin{aligned} P(v \wedge w) &= P(w \wedge v) \\ P(v \wedge -w) &= P(-w \wedge v) \\ P(-v \wedge -w) &= P(-w \wedge -v) \\ P(-v \wedge w) &= P(w \wedge -v) \end{aligned}$$

Ces égalités ont été posées sur le repère III.2 par un code symbole/couleur. En regardant de plus près, nous pouvons remarquer une symétrie axiale d'axe $y = f(x)$ entre les points qui portent le même sens. Autrement dit, l'information contenue dans la moitié de la matrice est redondante et peut donc être supprimée. La représentation devient donc une matrice triangulaire (voir figure III.3) et le gain de données en mémoire est divisé par deux. La partie blanche et l'axe de symétrie correspond à la matrice triangulaire conservée, la partie grisée aux données retirées [1].

1. Il est possible de supprimer l'axe $y = f(x)$ de la matrice conservée pour gagner en taille sans pour autant perdre d'information. Mais cet axe est pratique à l'usage car il correspond exactement aux probabilités simples.

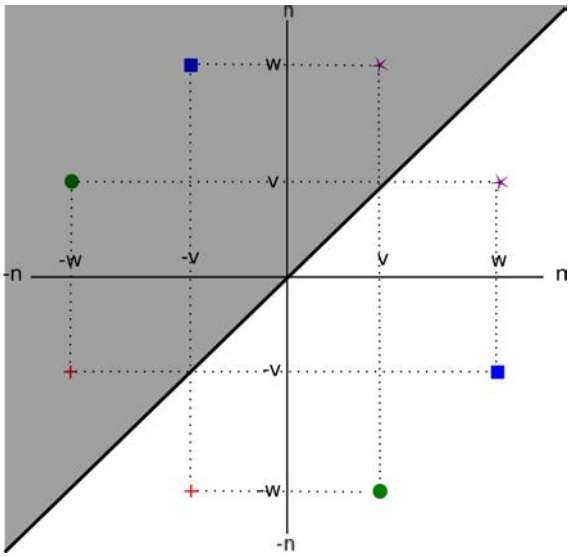

FIGURE III.3 – Représentation des probabilités par une matrice triangulaire

La représentation finale utilisée pour contenir l'ensemble des probabilités nécessaires - qu'elles soient simples ou conditionnelles - est une matrice triangulaire contenant $\frac{(N^2+N)}{2}$ éléments, avec N = $2n+1$ où n est le nombre de variables de la formule propositionnelle. Réduit, il faut donc $2n^2+3n+1$ données.

En pratique, pour représenter les probabilités issues de la formule SAT pour MD5, il faut $2 \cdot 12721^2 + 3 \cdot 12721 + 1 = 323\,685\,846$ éléments. Une probabilité étant représentée sur quatre octets, la base de données a donc une taille estimée à environ 1,3 Go.

Même si ce travail n'a pas été étendu dans cette thèse, le lecteur avisé remarquera qu'il est encore possible de réduire cet espace par deux en ne gardant, par exemple, que les probabilités de type $P(v \wedge w)$ et $P(-v \wedge -w)$. Dans ce cas, $P(v \wedge -w)$ et $P(-v \wedge w)$ sont calculées de la manière suivante :

- $P(-v \wedge w) = P(w \wedge w) - P(w \wedge v)$
- $P(v \wedge -w) = P(-w \wedge -w) - P(-v \wedge -w)$

1.3 La base de données probabilistes

Produire des données probabilistes sur les valeurs d'affectation des variables d'un problème SAT est rendu possible quand affecter une certaine partie des variables propage une valeur de vérité pour toutes les autres. C'est le cas pour les instances SAT représentant une primitive cryptographique puisque dans le cas d'un chiffrement symétrique il suffit d'affecter un texte clair et une clé pour avoir un texte chiffré et dans le cas des fonctions de hachage, seul un texte clair suffit à propager toutes les variables du processus jusqu'à l'empreinte. Dans ce contexte ces instances sont CIRCUIT-SAT.

Suivant ce principe, nous avons mis au point un programme en langage C qui affecte aléatoirement les variables correspondant à un message d'entrée et qui permet de garder en mémoire l'affectation donnée à toutes les variables du processus de hachage. L'accumula-

tion de ces traces permet ensuite de produire des données statistiques sur la façon dont sont assignées les variables. Générer des données probabilistes requiert une attention toute particulière. Il faut en effet que les probabilités soient fiables et statistiquement non-biaisées pour qu'elles gardent tout leur sens. L'algorithme 6 mis au point pour cela a donc été conçu pour répondre aux exigences suivantes : les messages en entrée doivent être composés de 512 bits déterminés aléatoirement et le nombre d'instances doit être assez conséquent.

Algorithme 6: Pseudo algorithme de génération de données probabilistes pour des formules propositionnelles représentant un processus de hachage cryptographique.

```
   Données : F : une formule CNF, V ses variables
             m un message de 512 bits
1  pour (i = 1 à 1000000) faire
2      Générer m aléatoire
       T ← Propager m dans F // T est la trace d'affectation de chaque variable
       pour (v ∈ V de 1 à n) faire
3          si (T(v) = 1) alors
4              pour (w de 1 à n) faire
5                  si ((T(w) = 1) et (w < v)) alors
6                      M_{v,w} = M_{v,w} + 1
7                  fin
8                  si (T(w) = 0) alors
9                      M_{v,-w} = M_{v,-w} + 1
10                 fin
11                 sinon
12                     si (T(v) = 0) alors
13                         pour w de 1 à n faire
14                             si ((T(w) = 0) et (w > v)) alors
15                                 M_{-v,-w} = M_{-v,-w} + 1
16                             fin
17                         fin
18                     fin
19                 fin
20             fin
21         fin
22     fin
23 fin
```

Nous donnons dans l'algorithme 6 le protocole de génération des statistiques. Le programme prend en entrée une formule SAT et les variables qui correspondent à un message d'entrée. Ce paramétrage très simple lui offre la particularité d'être facilement portable sur n'importe quelle formule SAT représentant un problème CIRCUIT-SAT et *in extenso* les instances cryptographiques. Dans nos travaux, nous l'avons appliqué sur MD4, MD5 et SHA.

La portabilité du protocole pourra aussi nous permettre de générer des probabilités pour l'étude d'autres fonctions dans le futur. Pour finir, nous aimerions souligner qu'il est aisé de paramétrer des messages d'entrées types. Par exemple, une contrainte pourrait être « *l'entrée est un message de longueur comprise entre 6 et 10 caractères alphanumériques* » . Dans ce cas, ce qui est intéressant c'est d'observer s'il existe un biais nouveau par rapport à des entrées absolument aléatoires.

1.4 Accéder aux probabilités

La matrice triangulaire est pratique pour représenter l'ensemble des probabilités mais n'est pas manipulable aisément. Faire une requête pour accéder à une probabilité est un exercice périlleux dans le sens où pour avoir connaissance d'une probabilité se situant dans la moitié supprimée il faut accéder à son image dans la symétrie. En effet, si $v < w$, alors le calcul de $P(v \mid w)$, nécessite $P(v \wedge w)$ qui n'existe plus. Sur la figure III.3 cela correspond à l'étoile violette de coordonnées (v, w) partie grisée. Il faut donc rediriger la recherche vers $P(w \wedge v)$.

En pratique, la connexion entre la matrice de probabilités et la donnée souhaitée est réalisée à travers ces deux règles :

- Le calcul de $P(v)$ est réalisé en pointant sur $\mathcal{M}_{v,v}$. Toutes ces valeurs sont en fait sur l'axe de symétrie sur le repère.
- Le calcul de $P(v \mid w)$ dépend des valeurs de v et w. Si $v > w$ alors il faut pointer à la fois sur $\mathcal{M}_{w,w}$ et $\mathcal{M}_{v,w}$ sinon il faut pointer sur $\mathcal{M}_{w,w}$ et $\mathcal{M}_{w,v}$.

Pour faciliter l'utilisation de cette base de données, nous avons mis au point une bibliothèque statique reprenant ces règles à travers plusieurs fonctions.

La première utilisation faite de ces probabilités a été de vérifier si les messages en entrée étaient bien aléatoires dans le but de confirmer la conformité des données. Pour cela, l'attention a été portée sur les probabilités des variables qui correspondent au message d'entrée et qui sont donc censées avoir pour valeur $\frac{1}{2}$.

Nos probabilités ont été générées en instanciant un million de messages aléatoires en entrée. Concernant MD4, l'écart observé le plus grand par rapport à $\frac{1}{2}$ est de 0.0046 et l'écart moyen est de 0.0001. Pour MD5 le plus gros écart à $\frac{1}{2}$ est de 0.0042, et la moyenne des écarts est de 0.001. Enfin, pour SHA-1, l'écart observé le plus grand est de 0.0047 et en moyenne il est de 0.0001. Ces chiffres montrent une marge d'erreur très réduite permettant d'affirmer que les messages en entrée sont bien aléatoires et par conséquent que les probabilités sont très proches de leurs points fixes.

2 Représentation visuelle des probabilités

La base de données de probabilités produite n'est intéressante que si elle apporte des renseignements utiles à la compréhension structurelle des formules SAT étudiées et par conséquent aux problèmes sous-jacents. Il est donc important de détacher des objectifs

précis avant d'analyser autant d'informations. Dans nos travaux, nous nous sommes focalisés autour de deux questions, traitées dans ce document séparément.

1. Existe-il des variables pour lesquelles la probabilité qu'elles soient assignées à 0 (ou 1) en pratique soit différente de la probabilité théorique attendue ?
2. Existe-t-il des liens probabilistes entre certaines variables du processus de hachage ?

2.1 À propos des probabilités simples

Pour savoir s'il existe des variables pour lesquelles la probabilité en pratique est différente de leurs probabilités théoriques, nous avions d'abord à calculer toutes les probabilités théoriques, puis à faire une comparaison avec les probabilités issues de la base de données. Mais avant tout cela, il ne faut pas oublier que nous traitons avec des formules représentant une fonction de hachage et que les variables ont un véritable sens. En effet, chacune d'elles dépend d'un certain mot de 32 bits dans le processus de hachage, de la place du bit représenté dans ce mot et de l'étape dans laquelle apparaît ce mot. Autrement dit, trois critères distincts qui doivent être considérés quand nous commençons à étudier le comportement des variables associées. La nature du mot, c'est-à-dire ce qu'il représente, est ce qui conditionne le plus les probabilités théoriques de ses bits. Par exemple, les mots identifiants les états internes ont des probabilités théoriques d'être à 1 de $\frac{1}{2}$ alors que les mots déterminant les retenues de premier rang dans une addition ont une probabilité plus élevée.

Dans le cadre de notre étude, nous avons été confronté à des instances décrites avec un très grand nombre de variables. Concernant MD5, 12 721 variables doivent par exemple être comparées puis analysées. Pour ce faire, nous avons choisi de passer par une représentation visuelle et avons dessiné une image où chaque pixel correspond à la comparaison des probabilités théoriques et pratiques pour une variable donnée. Un pixel représente donc un différentiel.

Détaillons l'image III.4 qui correspond au processus de hachage MD5. En ordonnée, sont représentées les étapes qui vont de 0 à 63 et en abscisse nous retrouvons six colonnes qui correspondent à six catégories de mots : les états (Q), la retenue de premier (colonne fC) et de second rang (colonne sC) pour l'addition à quatre opérandes, la retenue qui intervient dans l'addition à deux opérandes (tC), la fonction non-linéaire (colonne f) puis la somme de quatre opérandes (S). La largeur d'une colonne représente un mot de 32 bits, du bit de poids fort au bit de poids faible.

Les pixels sont en niveau de gris, le gris moyen signifiant que la probabilité théorique et en pratique sont identiques. Quand un pixel est foncé sa probabilité pratique est plus petite que sa probabilité théorique. Plus le pixel est noir et plus la probabilité se rapproche de 0. À l'inverse, quand un pixel est clair, sa probabilité pratique est plus grande que la probabilité attendue en théorie, et plus il est blanc, plus la probabilité pratique se rapproche de 1.

Nous observons deux phénomènes : les mots correspondants aux états, aux fonctions non-linéaires, à la retenue dans l'addition à deux opérandes et à la somme de quatre opérandes sont de couleur gris moyen. Nous pouvons en conclure que la distribution est équitablement répartie et par conséquent relève d'un comportement aléatoire. Par contre pour les

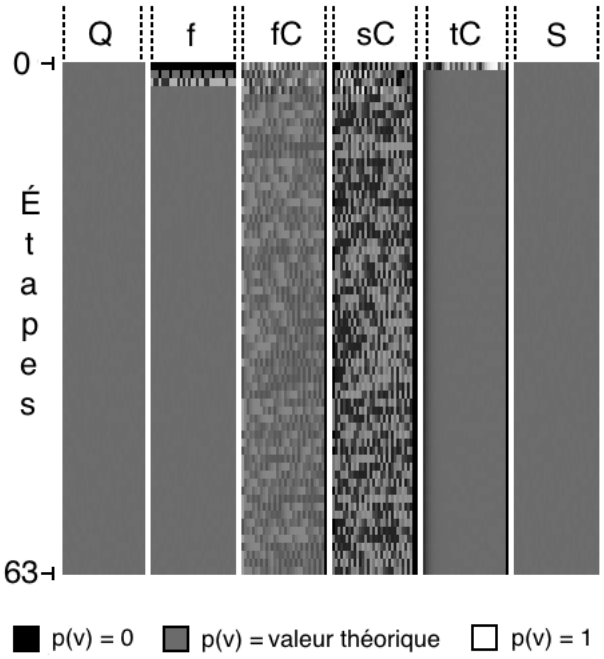

FIGURE III.4 – Comparaison imagée entre les probabilités théoriques et pratiques pour le processus de hachage de la fonction MD5

retenues de premier et de second rang dans l'addition à quatre opérandes, cela est différent. En effet, dans le premier cas, les probabilités tendent plutôt vers le blanc et dans le second cas, vers le noir.

Cela met en évidence le fait que les retenues n'ont pas une distribution pratique proche des probabilités attendues. De plus, nous remarquons que ces probabilités ne sont pas identiques suivant l'endroit où elles se situent dans un même mot, puisque le mot n'est pas d'un gris uni. Il n'est même pas possible de conclure sur une probabilité moyenne suivant un mot car cela est vraiment spécifique pour chaque bit de ce mot. La modélisation fine montre ici tout son intérêt.

Les tableaux III.1 et III.2 montrent un exemple où il est possible d'observer les probabilités théoriques et pratiques pour les retenues de l'addition à quatre opérandes sur 32 bits à l'étape 17 ainsi que leur différentiel noté Δ. Les bits de poids forts sur les retenues ne sont pas modélisés dans notre formule, ce qui explique l'absence de probabilités en pratique sur ces bits.

C'est évidemment la ligne Δ qui est la plus intéressante. En observant le différentiel, nous voyons qu'il y a un écart d'environ 7% pour les premières retenues et d'environ 15% - avec un pic à 21% - pour les retenues de second rang. Le cas présenté est pour l'étape 17, mais généralement, le phénomène reste le même.

Bit	0	1	2	3	4	5	6	7	8	9	10
Théorie	0.63	0.63	0.62	0.59	0.59	0.58	0.58	0.58	0.58	0.58	0.58
Pratique	0.50	0.63	0.66	0.66	0.66	0.67	0.56	0.63	0.49	0.60	0.59
Δ	0.13	0.00	0.04	0.07	0.07	0.09	0.02	0.05	0.09	0.02	0.01
Bit	11	12	13	14	15	16	17	18	19	20	21
Théorie	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58
Pratique	0.64	0.53	0.58	0.59	0.51	0.60	0.64	0.66	0.66	0.66	0.67
Δ	0.08	0.05	0.00	0.01	0.06	0.02	0.06	0.08	0.08	0.08	0.09
Bit	22	23	24	25	26	27	28	29	30	31	
Théorie	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.58	
Pratique	0.56	0.63	0.65	0.66	0.66	0.67	0.67	0.67	0.56	x	
Δ	0.02	0.05	0.07	0.08	0.08	0.09	0.09	0.09	0.02	x	

TABLE III.1 – Comparaison entre les probabilités théoriques et celles obtenues en pratique, pour la retenue de premier rang étape 17.

Bit	0	1	2	3	4	5	6	7	8	9	10
Théorie	0.06	0.22	0.24	0.29	0.29	0.30	0.30	0.31	0.31	0.31	0.31
Pratique	0.00	0.06	0.08	0.10	0.10	0.11	0.41	0.10	0.52	0.35	0.23
Δ	0.06	0.16	0.16	0.19	0.19	0.19	0.11	0.21	0.21	0.04	0.08
Bit	11	12	13	14	15	16	17	18	19	20	21
Théorie	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31
Pratique	0.17	0.45	0.38	0.21	0.49	0.23	0.20	0.11	0.13	0.11	0.12
Δ	0.14	0.14	0.07	0.10	0.18	0.08	0.11	0.20	0.18	0.20	0.19
Bit	22	23	24	25	26	27	28	29	30	31	
Théorie	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31	
Pratique	0.42	0.11	0.18	0.11	0.13	0.11	0.12	0.11	x	x	
Δ	0.11	0.20	0.13	0.20	0.18	0.20	0.19	0.20	x	x	

TABLE III.2 – Comparaison entre les probabilités théoriques et celles obtenues en pratique, pour la retenue de second rang étape 17.

2.2 Faiblesse de la constante de ronde

Nous nous sommes alors interrogés sur les raisons qui expliquent ces trop grands écarts différentiels. Les vecteurs de retenues étant générés dans l'addition à quatre opérandes, nous nous sommes intéressées à chacune d'elles. En se focalisant sur l'addition de MD5, trois des quatre opérandes ont un comportement tout à fait aléatoires puisque les états, les fonctions non-linéaires ainsi que le message d'entrée sont aléatoires. L'image III.4 nous le montre. Reste donc la constante qui est invoquée à chaque étape et notre hypothèse a été de la considérer comme responsable de ces perturbations. Pour vérifier cette idée nous avons calculé, pour chaque étape, les probabilités pour les retenues de premier et second rang d'être à 1 sachant que les 32 bits de la constante sont fixés. Nous avons mis à jour nos tableaux comparatifs avec cette fois-ci les probabilités théoriques en fonction de la

constante (voir tableaux III.3 et III.4).

Bit	0	1	2	3	4	5	6	7	8	9	10
Th. + cst	0.50	0.63	0.66	0.66	0.66	0.66	0.55	0.63	0.44	0.60	0.62
Pratique	0.50	0.63	0.66	0.66	0.66	0.67	0.56	0.63	0.49	0.60	0.59
Δ	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.05	0.00	0.03
Bit	11	12	13	14	15	16	17	18	19	20	21
Th. + cst	0.64	0.51	0.56	0.63	0.45	0.62	0.62	0.65	0.65	0.65	0.65
Pratique	0.64	0.53	0.58	0.59	0.51	0.60	0.64	0.66	0.66	0.66	0.67
Δ	0.00	0.02	0.02	0.04	0.06	0.02	0.02	0.01	0.01	0.01	0.02
Bit	22	23	24	25	26	27	28	29	30	31	
Th. + cst	0.54	0.63	0.63	0.65	0.65	0.65	0.65	0.65	0.54	0.57	
Pratique	0.56	0.63	0.65	0.66	0.66	0.67	0.67	0.67	0.56	x	
Δ	0.02	0.00	0.02	0.01	0.01	0.02	0.02	0.02	0.02	x	

TABLE III.3 – Comparaison entre les probabilités théoriques en considérant la constante et celles obtenues en pratique, pour la retenue de premier rang étape 17.

Bit	0	1	2	3	4	5	6	7	8	9	10
Th. + cst	0.00	0.06	0.08	0.10	0.10	0.11	0.41	0.10	0.52	0.33	0.22
Pratique	0.00	0.06	0.08	0.10	0.10	0.11	0.41	0.10	0.52	0.35	0.23
Δ	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01
Bit	11	12	13	14	15	16	17	18	19	20	21
Th. + cst	0.17	0.45	0.38	0.19	0.50	0.10	0.22	0.11	0.14	0.11	0.12
Pratique	0.17	0.45	0.38	0.21	0.49	0.23	0.20	0.11	0.13	0.11	0.12
Δ	0.00	0.00	0.00	0.02	0.01	0.13	0.02	0.00	0.01	0.00	0.00
Bit	22	23	24	25	26	27	28	29	30	31	
Th. + cst	0.41	0.10	0.20	0.11	0.14	0.11	0.12	0.11	0.42	0.37	
Pratique	0.42	0.11	0.18	0.11	0.13	0.11	0.12	0.11	x	x	
Δ	0.01	0.01	0.02	0.00	0.01	0.00	0.00	0.00	x	x	

TABLE III.4 – Comparaison entre les probabilités théoriques en considérant la constante et celles obtenues en pratique, pour la retenue de second rang étape 17.

Dans ces nouveaux tableaux, ces probabilités prennent en compte la constante et le différentiel s'est alors totalement réduit au point d'être quasiment toujours à 0. La conclusion de cette expérience est que ce sont les constantes qui apportent un biais sur les probabilités des retenues dans l'addition à quatre opérandes. Afin d'être complet, nous avons tracé sur le graphique III.5, les probabilités théoriques, les probabilités théoriques prenant en compte la constante et enfin les probabilités observées en pratique, pour les retenues de second rang à l'étape 17. Nous voyons clairement que la courbe des probabilités en pratique suit celle des probabilités théoriques qui tient compte de la constante.

Une parenthèse peut être ouverte pour remarquer que dans SHA, il n'est pas question d'addition à quatre opérandes, mais d'une addition à cinq opérandes. Ainsi, avec les retenues, cela devient une addition sur sept opérandes. Là où cette remarque est intéressante, c'est que l'énumération de toutes les valeurs que peuvent prendre les sept opérandes montre que la retenue de premier rang devient équiprobable. Ceci est expliqué par le fait qu'une telle addition permet de combler totalement l'espace de représentation de l'addition et qu'il n'y a plus de cas impossible. Par exemple, la somme d'une addition à quatre opérandes binaires sur un bit dans MD5 peut prendre une valeur comprise entre 0 et 6, mais pas la valeur 7. De ce fait, la retenue de premier rang apparaît dans trois cas sur sept (quand la somme vaut 2, 3, 6). Lors d'une addition sur cinq opérandes, la somme peut valoir 7, et générer une retenue de premier rang. La retenue apparaitra alors dans quatre cas sur huit (quand la somme vaut 2, 3, 6, 7), ce qui la rend équiprobable.

Ce principe est généralisable pour toutes additions à 2^k-1 opérandes, retenues comprises [2]. Par conséquent, l'addition à deux opérandes, à cinq opérandes, à douze opérandes, (etc.), sont de bons choix, contrairement à l'addition à quatre opérandes pourtant utilisée dans MD⋆. La conséquence principale, est que les probabilités pour les retenues de second rang s'éloignent beaucoup moins de leurs valeurs théoriques. Ce qui minimise la faiblesse cryptographique par un comportement plus proche de l'aléatoire.

2.3 Les probabilités conditionnelles

Le dénombrement des probabilités simples est donné par le nombre de variables d'une formule, puisque chaque variable a une et une seule probabilité. À l'opposé, pour une variable donnée, il existe une probabilité conditionnelle pour chaque autre variable de la formule. Cela signifie que la représentation imagée des probabilités conditionnelles est possible uniquement si on fixe une variable qui cadre l'information désirée. Par exemple, dans nos travaux, nous avons observé la probabilité d'avoir une variable fixée à 1 sachant la variable précédente dans le même mot. Ceci est exprimé par $P(v \mid v-1)$.

Appliqué sur MD5, nous avons dressé l'image III.6, reprenant le même schéma descriptif que l'image des probabilités simples. Nous remarquons que les pixels concernant les mots représentant les retenues ont un contraste bien plus élevé. Ce phénomène montre

2. et donc à $2^k - k$ opérandes sans prendre en compte les retenues

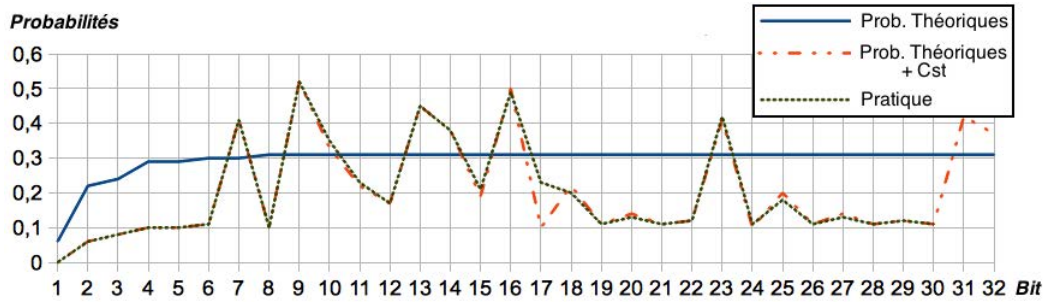

FIGURE III.5 – Comparaison entre les probabilités pratiques, théoriques avec et sans prise en compte des constantes pour les retenues de second rang étape 17

que les différences entre les probabilités théoriques et pratiques sont un peu plus éloignées. Ce phénomène est intuitivement expliqué par le fait que les retenues sont fortement liées entre elles. Par conséquent, à titre d'exemple, sachant qu'une retenue est à 0, il est aisé d'imaginer que la retenue suivante aura plus de chance d'être à 0 qu'à la normale. Ces probabilités conditionnelles pourraient aussi être utilisées pour générer un réseau bayésien sur nos instances.

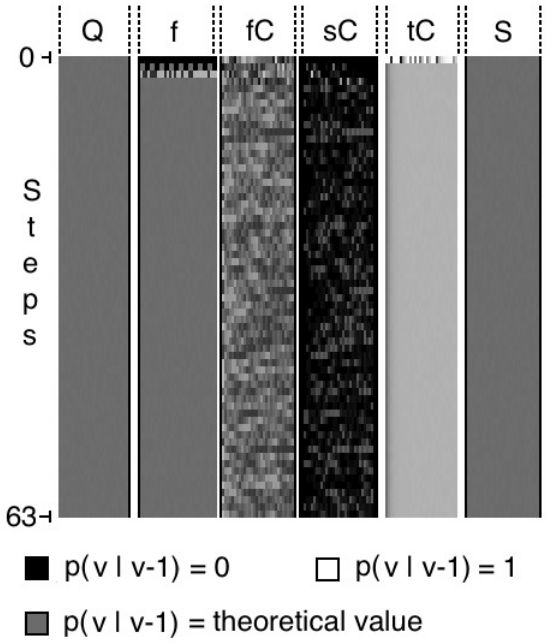

FIGURE III.6 – Comparaison imagée entre les probabilités conditionnelles de type P(v | v - 1) théoriques et pratiques pour le processus de hachage de la fonction MD5

Conclusion sur la représentation des probabilités

Modéliser une fonction de hachage sous la forme d'un problème SAT permet d'avoir une modélisation très fine du processus de hachage. De ce fait, il est possible d'étudier bit par bit ce qu'il se passe lors d'un hachage et par extension lors de plusieurs milliers de hachages. Ceci peut alors permettre d'étudier le comportement structurel de la formule de façon probabiliste. Dans ces travaux, nous avons contribué en la génération d'une base de données probabilistes à propos des variables contenues dans les processus de hachage des fonctions MD4, MD5 et SHA. La représentation de ces informations sous forme d'images nous a permis de comparer les probabilités calculées avec celles attendues en pratique. Cette approche offre la capacité d'extraire plusieurs informations sur un comportement non-aléatoire au niveau des retenues de l'addition à quatre opérandes et d'en comprendre les raisons. Ces informations sont pertinentes car elles apportent des faiblesses au niveau de la fonction de hachage et peuvent être exploitées pour résoudre plus efficacement les instances SAT représentant ces fonctions. Cela signifie que les constantes utilisées pour générer du chaos lors du hachage engendrent aussi une meilleure connaissance des vecteurs

de retenues de l'addition, ce qui affaiblit la robustesse probabilistique d'une affectation de variable dans un cadre général. Dans la partie suivante, nous montrons comment exploiter cette faiblesse au sens de la cryptanalyse logique et dans le chapitre IV nous verrons une application pratique à travers l'exemple d'un solveur SAT dédié au guidage de l'heuristique de choix de variable.

3 Exploitation des probabilités en pratique

Dans cette partie nous présenterons tout d'abord les notions utilisées pour décrire les liens probabilistes unissant certaines variables du processus de hachage. Nous verrons ensuite comment découvrir ces relations sur les fonctions de hachage MD5 et SHA-1.

3.1 Relations remarquables

Comme nous l'avons vu, la base de données de probabilités peut être utilisée afin de connaître la probabilité d'une variable d'être assignée à 0 ou 1. De la même façon, il est aussi possible de connaître la probabilité conditionnelle pour une variable d'être à 0 ou 1 sachant n'importe quelle autre variable du processus de hachage. En réalité, l'ensemble de ces informations peut être manié pour effectuer un raisonnement probabiliste sur la formule SAT pouvant donner lieu à l'extraction de nouvelles règles logiques. Ces règles peuvent occurrer de façon déterministe - dans ce cas nous parlerons d'événements factuels - ou de façon probabiliste.

3.1.1 Événements factuels

L'identification des variables toujours assignées à la même valeur est simplement réalisé en auscultant les probabilités. En effet, il suffit de repérer les variables qui ont une probabilité égale à 0 ou 1. À titre d'exemple, dans le processus de hachage MD5, la variable correspondante à la seconde retenue, étape 1, bit 29 a une probabilité d'être à 1 qui est nulle, et peut donc être fixée à 0 de façon générique.

De même, il est aussi possible de découvrir des relations d'équivalences entre les variables. Soient v et w deux variables.

- Si $p(w \mid v) = 1$ alors $v \Rightarrow w$; en d'autres termes, si $p(v) = p(v \wedge w)$ alors $v \Rightarrow w$. En effet, si la probabilité d'avoir $v = 1$ est la même que la probabilité d'avoir à la fois v et w qui valent 1 alors cela signifie qu'à chaque fois que v vaut 1, w vaut également 1.
- Si $p(v \mid w) = p(w \mid v) = 1$, alors $v \Leftrightarrow w$.

En réalité, ces événements factuels sont aussi découverts dans le pré-traitement expliqué dans ce chapitre partie 3 puisqu'il s'agit de faits génériques, valables pour tout hachage.

3.1.2 Événements probabilistes

Un événement probabiliste peut concerner la façon dont une variable est statistiquement assignée ou bien une relation probabiliste, c'est-à-dire l'apparition d'une connexion

entre deux variables qui n'existe que dans certains cas. Afin de déceler un événement probabiliste, nous associons un seuil (très proche de 1) à partir duquel cet événement est considéré comme intéressant ; quand il a une grande chance d'occurrer. Soit s ce seuil.

- On dit qu'une variable v est **quasi-assignée** si quelle que soit l'instance, la variable a une probabilité $\mathrm{p}(v) \geq s$.
- On appelle **quasi-implication** la relation entre deux variables telles que $\mathrm{p}(w \mid v) \geq s$ et notons cette relation $v \underset{\geq s}{\Rightarrow} w$.
- On dénote **quasi-équivalence** une relation entre deux variables telle que $\mathrm{p}(w \mid v) \geq s$ et $\mathrm{p}(v \mid w) \geq s$ et notons cette relation $v \underset{\geq s^2}{\Leftrightarrow} w$.

Recherche d'événements sur MD5 **et** SHA-1

En pratique, nous avons conçu un programme permettant de traquer dans la base de données toutes les probabilités au dessus d'un seuil paramétré par l'utilisateur. Ainsi, le programme applique un traitement spécifique qui donne en sortie toutes les relations que nous venons de définir. Par exemple, en définissant un seuil $s = 0,99$, nous avons trouvé $\mathrm{p}(M0[14] \mid f_2[21]) = 0.9969$ pour le processus de hachage MD5. Ceci signifie que nous avons la relation $f_2[21] \underset{\geq 0,99}{\Rightarrow} M0[14]$.

Cette information peut ensuite être retranscrite sous forme de clause pour être ajoutée à la formule propositionnelle. En reprenant le précédent exemple, cela se concrétiserait par l'introduction d'une nouvelle clause : $(\overline{f_2[21]} \vee M0[14])$. En procédant ainsi il ne faut pas omettre que l'espace des solutions se contracte puisque cette clause ne sera pas satisfaite pour l'ensemble des processus de hachage possibles. Elle ne le sera « que » dans 99,69% des cas. Nous proposons d'ouvrir une parenthèse pour approfondir ce point.

L'utilisation de relations probabilistes qui ont de très grandes chances d'occurrer est tout de même à prendre avec des pincettes. En effet, inclure une clause ou assigner une variable qui a une probabilité p d'exister réduit l'espace des solutions d'un pourcentage équivalent à p. En inclure une deuxième ayant une probabilité p', réduit l'espace d'un facteur $p \cdot p'$. Et ainsi de suite... Cela signifie qu'utiliser ce genre de relations force l'espace des solutions à converger vers l'ensemble vide.

Qu'en est-il de l'espace de recherche ? Du point de vue de la résolution SAT, la fixation d'une variable ou l'ajout d'une clause permet de diviser par deux cet espace. Autrement dit, faire état de l'apport d'une information probabiliste est une affaire de pari. D'un côté, l'espace de recherche - immensément grand - est réduit de moitié et de l'autre côté l'espace de solution est en partie rogné. En pratique, toute la question est de savoir si ça vaut le coup de sacrifier des solutions (et si oui, combien ?) contre une restriction de l'espace de recherche par l'utilisation d'événements probabilistes. Intuitivement, nous serions tenté de répondre par l'affirmative mais jusqu'à aujourd'hui il est difficile d'évaluer cette pensée, même si les expérimentations présentées au chapitre IV sont plutôt prometteuses.

Variables quasi-assignées dans MD5
$cHa[7] \simeq 0\ car\ p(cHa[7]) = 0.0043, cHa[6] \simeq 0\ car\ p(cHa[6]) = 0.0082$ $cHc[7] \simeq 1\ car\ p(cHc[7]) = 0.9922, fC_0[28] \simeq 1\ car\ p(fC_0[28]) = 0.9904$ $sC_0[28] \simeq 0\ car\ p(sC_0[28]) = 0.0096$
Quasi-implications dans MD5
$M_0[14] \underset{\geq 0,99}{\Rightarrow} Q_1[21], Q_1[21] \underset{\geq 0,99}{\Rightarrow} M_0[14], Ha[8] \underset{\geq 0,99}{\Rightarrow} Q_{16}[8]$ $Q_{61}[8] \underset{\geq 0,99}{\Rightarrow} Ha[8], cHa[8] \underset{\geq 0,99}{\Rightarrow} Q_{61}[8], Q_{64}[28] \underset{\geq 0,99}{\Rightarrow} cHb[30]$ $Q_{61}[8] \underset{\geq 0,99}{\Rightarrow} Ha[8], Ha[8] \underset{\geq 0,99}{\Rightarrow} Q_{61}[8], cHa[8] \underset{\geq 0,99}{\Rightarrow} Ha[8]$ $Hb[28] \underset{\geq 0,99}{\Rightarrow} cHb[30], cHa[8] \underset{\geq 0,99}{\Rightarrow} Q_{61}[8], Ha[8] \underset{\geq 0,99}{\Rightarrow} cHa[8]$
Quasi-équivalences dans MD5
$\overline{Q_1[21]} \underset{\geq 0,98}{\Leftrightarrow} M_0[14], Q_{61}[8] \underset{\geq 0,98}{\Leftrightarrow} Ha[8], Q_{61}[8] \underset{\geq 0,98}{\Leftrightarrow} cHa[8]$ $Ha[8] \underset{\geq 0,98}{\Leftrightarrow} Q_{61}[8], Ha[8] \underset{\geq 0,98}{\Leftrightarrow} cHa[8], cHa[8] \underset{\geq 0,98}{\Leftrightarrow} Q_{61}[8], M_0[14] \underset{\geq 0,98}{\Leftrightarrow} f_1[21]$ $cHa[8] \underset{\geq 0,98}{\Leftrightarrow} Ha[8], f_1[21] \underset{\geq 0,98}{\Leftrightarrow} M_0[14], M_0[14] \underset{\geq 0,98}{\Leftrightarrow} Q_1[21]$

TABLE III.5 – Faits et relations probabilistes détectés dans le processus de hachage de MD5. La notation utilisée est celle présentée dans le chapitre 1, partie 3.3.1 et pour le tableau II.7.

Revenons à la découverte d'événements probabilistes en pratique. Concrètement, un extrait des résultats de nos expérimentations sur MD5 et SHA-1 est donné dans les tableaux III.5 et III.6. Y apparaissent beaucoup de relations pour les premières étapes, permettant ainsi de mieux anticiper le comportement de ces variables par rapport à leur voisinage. De plus, les images présentées dans ce chapitre partie 2.1 mettaient en avant des probabilités inattendues uniquement au niveau des retenues. Ici, nous pouvons observer des relations qui font intervenir d'autres types de variable, comme par exemple des bits provenant des états, du message d'entrée ou bien des fonctions non-linéaires. Ces relations sont évidemment très intéressantes pour être exploitées en pratique. Dans ces travaux, elles l'ont été dans le cadre de la cryptanalyse logique, mais il est facile d'imaginer une réutilisation dans diverses attaques, algébriques ou différentielles notamment.

La valeur de seuil choisi jusque là était de 0,99 et permettait d'illustrer convenablement le propos. Ce seuil est bien sûr paramétrable et change la quantité et la qualité des informations probabilistes dégagées de la base de données. Le tableau III.7 montre le nombre de relations extraites de nos données statistiques pour MD5 et SHA-1 suivant la valeur du seuil. Il est facile d'observer que plus le seuil est bas, plus il laisse de souplesse à la découverte d'événements probabilistes.

Variables quasi-assignées dans SHA-1
$cHa[7] \simeq 0\ car\ p(cHa[7]) = 0.0043, cHa[6] \simeq 0\ car\ p(cHa[6]) = 0.0082$ $cHc[7] \simeq 1\ car\ p(cHc[7]) = 0.9922, fC_0[28] \simeq 1\ car\ p(fC_0[28]) = 0.9904$ $sC_0[28] \simeq 0\ car\ p(sC_0[28]) = 0.0096$
Quasi-implications dans SHA-1
$\overline{M_0[8]} \underset{\geq 0,99}{\Rightarrow} sC_1[14], M_1[13] \underset{\geq 0,99}{\Rightarrow} sC_1[14], M_1[10] \underset{\geq 0,99}{\Rightarrow} sC_1[14]$ $M_2[25] \underset{\geq 0,99}{\Rightarrow} sC_2[27], M_3[27] \underset{\geq 0,99}{\Rightarrow} sC_3[27]$
Quasi-équivalences dans SHA-1
$sC_0[28] \underset{\geq 0,98}{\Leftrightarrow} sC_0[17], sC_0[27] \underset{\geq 0,98}{\Leftrightarrow} fC_0[28], sC_0[24] \underset{\geq 0,98}{\Leftrightarrow} cHc[7]$ $sC_0[24] \underset{\geq 0,98}{\Leftrightarrow} fC_0[28], sC_0[17] \underset{\geq 0,98}{\Leftrightarrow} cHc[7], sC_0[17] \underset{\geq 0,98}{\Leftrightarrow} fC_0[28]$ $fC_1[16] \underset{\geq 0,98}{\Leftrightarrow} sC_1[16], sC_1[16] \underset{\geq 0,98}{\Leftrightarrow} fC_1[16]$

TABLE III.6 – Faits et relations probabilistes détectés dans le processus de hachage de SHA-1. La notation utilisée est celle présentée dans le chapitre 1, partie 3.3.2 et pour le tableau II.7.

Pour finir sur ce point, nous avons montré dans cette partie comment utiliser notre base de probabilités pour découvrir des informations concrètes sur et entre les variables qui composent nos formules propositionnelles. Nous montrerons dans le chapitre IV comment se servir de ces informations pour la recherche de pré-image.

4 Spécification des clauses

Avoir une connaissance sur les variables d'un problème SAT est fortement intéressant pour améliorer sa résolution. Par ailleurs, il est aussi très profitable d'avoir des informations sur la façon dont les clauses sont satisfaites. Cette partie expose les travaux que nous avons réalisés en ce sens. Dans un premier temps, nous dresserons un portrait des clauses composant nos formules à travers un raisonnement déductif, puis dans un second temps nous montrerons comment un nouveau raisonnement probabiliste peut permettre de conclure en l'élaboration de profils de résolution.

4.1 Classe de l'instanciation SAT de MD5

Dans le chapitre I, nous avons présenté des classes du problème SAT, comme par exemple NAE-SAT ou XOR-SAT, pour lesquelles la littérature offre des méthodes permettant de résoudre le problème plus efficacement que SAT. Nous avons donc eu pour idée d'observer

	MD5		SHA-1	
Seuil	**Variables quasi-assignées**	**Quasi équivalences**	**Variables quasi-assignées**	**Quasi équivalences**
1	0	0	0	0
0.995	2	10	1	8
0.990	5	29	5	44
0.985	9	79	8	88
0.980	12	105	13	169
0.975	16	162	15	210
0.970	17	186	16	241
0.965	21	228	21	348
0.960	23	252	23	382
0.955	25	271	24	426
0.950	28	367	26	486
0.940	31	707	30	617
0.930	46	1193	77	1642
0.920	70	1773	84	2274
0.910	85	2477	85	2479
0.900	162	4260	98	3569

TABLE III.7 – Nombre de variables quasi-assignées et de quasi-équivalences détectées dans le processus de hachage de MD5 et SHA-1.

si les formules SAT associées aux problèmes cryptographiques faisaient partie de ces catégories.

Comprendre les problèmes formulés à travers nos modélisations est important pour extraire un maximum d'informations sur leur structures internes. Dans ce contexte, cette problématique s'est focalisée sur la formule propositionnelle représentant la fonction de hachage MD5 pour savoir s'il était possible de l'assimiler à une forme XOR-SAT ou NAE-SAT. Pour ce faire, chaque composante a été étudiée une à une pour vérifier si elles se situaient dans une de ces deux classes particulières. La fonction de hachage MD5 est principalement composée d'une addition sur deux opérandes, d'une addition sur quatre opérandes (pouvant être écrite sous la forme d'additions à deux opérandes) et de fonctions non-linéaires.

L'addition

Pour classifier l'addition comme étant NAE-SAT ou XOR-SAT, il faut regarder la table de vérité associée à cette opération et chercher des contre-exemples. Un contre-exemple pour une clause XOR-SAT est une ligne où on trouve un nombre pair de variables d'entrées à 0 (ou 1) et une sortie à 1 (respectivement 0) et un contre-exemple pour une clause NAE-SAT se manifeste par paire puisqu'il suffit de rechercher deux lignes où les variables d'entrées sont opposées provoquant néanmoins la même sortie. Pour suivre le raisonnement qui suit, il faut se référer à la table de vérité de l'addition (Chapitre I , table II.4).

De cette table représentative de l'exhaustivité des cas possibles, l'ensemble des affectations des variables a_i, b_i, c_i, c_{i+1} qui sont solutions des clauses générées montrent que les clauses peuvent être satisfaites par 1, 2 ou 3 littéraux, mais jamais par 4. En d'autres termes, ce paquet de clauses n'est pas XOR-SAT (seules certaines le sont) mais est NAE-SAT puisqu'il n'existe aucune solution qui fasse entrer en jeu une clause totalement satisfaite. Puisque la modélisation est générique, ce raisonnement sur un bit peut s'étendre aux additions sur 32 bits et notre conclusion est que les additions sont modélisées sous un ensemble de clauses NAE-SAT.

Les fonctions non-linéaires

En ce qui concerne les fonctions non-linéaires dans MD5, c'est assez hétéroclite car elles sont au nombre de quatre et assez différentes les unes des autres. En procédant comme pour l'addition, seule la fonction H a pu être classée comme étant XOR-SAT et NAE-SAT. Les autres fonctions possèdent des clauses XOR-SAT et/ou NAE-SAT mais jamais leur groupement ne forme un ensemble appartenant à une classe particulière.

Bilan de la classification

Malheureusement, puisqu'il n'y a pas 100% des clauses qui puissent se ranger dans une classification particulière, nous ne pouvons pas affirmer que la formule représentant le processus de hachage MD5 est exclusivement XOR-SAT ou NAE-SAT. Nous avons cependant représenté sur le diagramme III.7 la répartition détaillée du nombre de clauses XOR-SAT et NAE-SAT pour notre formule propositionnelle. On y voit notamment que le nombre de clauses non NAE-SAT n'est finalement qu'une infime part de la modélisation.

Au delà de ces deux classifications, nous avons approfondi le sujet en nous questionnant sur la réelle façon dont les clauses sont satisfaites lors d'un hachage. Pour cela, notre idée a été une nouvelle fois de passer par un raisonnement probabiliste pour de nouveau générer des statistiques.

4.2 Raisonnement probabiliste sur les clauses

La génération des statistiques pour connaître la façon dont les clauses sont satisfaites a été réalisée en utilisant la même méthodologie que la production de statistiques sur les variables. Cette fois, quand une clause est satisfaite, l'information gardée est par quels littéraux elle a été satisfaite.

Imaginons la clause $(a \vee b \vee c)$. Il y a sept affectations différentes qui satisfassent cette clause :

$$\{\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{a,b,c\}\}$$

Pourtant, lors d'un hachage, seule une de ces affectations est « choisie » . C'est cette affectation qui est gardée pour produire des données statistiques. Par conséquent, pour un nombre de hachages conséquent, nous obtenons un profil moyen de satisfaction de cette

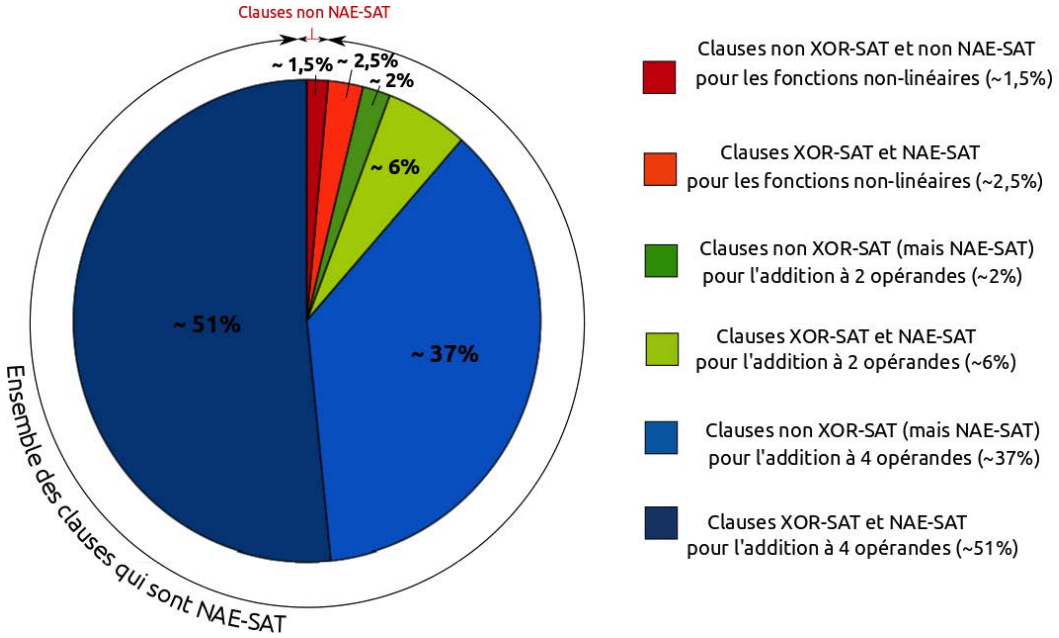

FIGURE III.7 – Répartition des clauses suivant la modélisation SAT de MD5

clause. Pour l'exemple précédent, cela pourrait être :

$$\{a\} : 10\%, \{b\} : 0\%, \{c\} : 20\%, \{a,b\} : 0\%, \{a,c\} : 20\%, \{a,b,c\} : 50\%$$

Ainsi, il est possible de savoir exactement la façon dont la clause est satisfaite. Cette information est extrêmement intéressante pour une résolution SAT puisqu'elle permet d'orienter la recherche de solution en choisissant les satisfactions les plus courantes. De plus, quand un pourcentage est à 0 cela signifie que la clause ne peut pas être satisfaite de cette façon pour trouver une solution au problème en entier. En pratique il est donc possible de personnaliser le solveur en lui spécifiant cette nouvelle information dont il ne peut avoir connaissance seul. Ce point sur le profilage de clauses pour s'orienter vers une résolution de la formule revêt une importance non-négligeable car il permet de se diriger vers des techniques très efficaces, non affiliées directement au problème SAT.

À ce titre, le parallèle avec les techniques utilisées dans la résolution de problèmes de satisfaction de contraintes est intéressant.

Les problèmes de satisfaction de contraintes ou CSP (pour *Constraint Satisfaction Problem*) sont des problèmes où chaque variable peut prendre une valeur parmi un domaine de valeurs prédéfinies, dans le discret ou le réel. Les contraintes sont des égalités ou des inégalités, quantifiées, entre les variables, reliées par les opérateurs mathématiques classiques (addition, multiplication). SAT peut être vu comme un CSP spécifique où chaque variable possède la même plage de valeurs possibles, à savoir { VRAI, FAUX }. Les contraintes sont des égalités qui doivent valoir VRAI et les opérateurs mathématiques deviennent, dans le binaire, le « ET » logique ($\wedge$) pour la multiplication et le « OU » logique ($\vee$) pour l'addition.

Dans un solveur complet, le backtracking induit une exploration exhaustive de l'espace de recherche. Par conséquent, le temps de résolution devient rapidement contraignant dès que l'on traite de problèmes de taille importante. Plusieurs techniques ont donc été étudiées afin d'accroître l'efficacité de résolution des solveurs CSP. Parmi elles, le filtrage par consistance partielle est une des techniques les plus utilisées pour augmenter l'efficacité de la résolution. Cela consiste simplement à supprimer du domaine de recherche les valeurs prohibées. Or, dans notre cas, les profils d'affectation correspondent tout à fait à un filtrage de 2-consistance, appelée aussi consistance d'arc. Ce lien entre cryptanalyse et résolution est donc un atout indéniable pour nos travaux.

5 Conclusion

Ce chapitre autour de l'analyse structurelle des formules SAT cryptographiques a permis de mettre en avant quelques avantages liés à de la modélisation logique du problème. À travers elle, deux bases de données ont pu être produites pour contenir des informations sur les comportements probabilistes des variables et des clauses. Une première partie a ainsi été dédiée à la genèse puis à la visualisation de probabilités sur les variables de fonctions de hachage. Ces travaux ont notamment permis d'exhiber l'existence de la faiblesse des constantes de rondes et ont fait office d'une contribution dans [Legendre 2013]. De plus, dans une deuxième partie, les outils informatiques développés pour exercer des requêtes sur ces bases de données ont été présentés. Ils ont permis l'obtention d'informations pertinentes comme des (quasi-)équivalences et des variables (quasi-)assignées, pouvant être réutilisées dans d'autres cryptanalyses. Enfin, une étude sur la classification des formules associées aux problèmes cryptographiques a été introduite. Ici, nous avons montré que les clauses adoptent un profil de satisfaction non-équiprobable. Certes, ce profil est corrélé au fait que le problème étudié est un problème structuré, mais il rend possible une spécification précise de la méthode de résolution. En pratique, un solveur SAT peut être aidé lors de sa recherche par l'apport de ce profil, à la fois en évitant les cas impossibles, et en orientant sa recherche vers les cas les plus probables. La génération des profils de satisfaction des clauses est un travail réalisé durant cette thèse tandis que leur utilisation en pratique fait l'objet de travaux en cours.

CHAPITRE IV

Cryptanalyse logique : attaques en pratique

1 Introduction sur les attaques

Attaquer la pré-image d'une fonction de hachage par le levier de la cryptanalyse logique est réalisé de la manière suivante. Dans une première étape l'algorithme de hachage est représenté en un formalisme logique à travers un problème SAT. Nous avons vu dans le chapitre II la conception d'un générateur de formules dédié à cette production de formules propositionnelles pour les fonctions de hachage MD4, MD5 et SHA. La deuxième étape consiste à figer une empreinte particulière dans le problème. Cela se fait en donnant une valeur de vérité aux variables correspondantes dans la formule. Une nouvelle formule contenant moins de clauses (mais plus contrainte) est ainsi obtenue et a pour solution une réponse au problème de la pré-image. Cette solution est une affectation complète des variables comprenant celles du message d'entrée. L'enjeu d'une telle attaque est donc de trouver au moins une affectation qui satisfasse le problème. Cela se fait dans une troisième étape grâce à la résolution de la formule par un solveur SAT.

En pratique, même si les formules résultantes répondent au problème de la pré-image, il reste calculatoirement impossible de trouver une solution en un temps raisonnable. La méthode consiste donc à calculer la pré-image de versions réduites de la fonction de hachage. La démarche est incrémentale : on commence par chercher à résoudre une ronde, puis une ronde et une étape, une ronde et deux étapes, etc. L'idée est de trouver la frontière entre les instances qu'un solveur est capable de résoudre et celles où il n'y arrive pas. Quand cette limite est trouvée alors une certaine mesure de la résistance de la fonction de hachage à la pré-image peut être donnée.

Dans cette thèse, nous avons amélioré le nombre d'étapes inversées pour MD5 et fait aussi bien que les résultats existants sur MD4, SHA-0 et SHA-1. Nos expérimentations sont présentées dans la première partie de ce chapitre.

La recherche de collisions est l'autre angle d'attaque étudié dans cette thèse. Deux parties sont consacrées à ce sujet : la première montre comment le cadre de la cryptanalyse logique permet d'étudier une collision en pratique, la seconde explicite deux types d'attaques différentes. La première attaque consiste en la réimplantation d'un chemin différentiel donné par Stevens [Stevens 2012a] dans une modélisation originale de la collision. La seconde est une approche personnalisée du problème par le biais de collisions locales réinjectées dans une formule globale.

2 Pré-images en pratique

La résistance à la pré-image des fonctions de hachage a été réalisée en deux temps. Tout d'abord, nous nous sommes appliqués à savoir quel était le dernier nombre d'étapes que nous étions en mesure d'inverser. Pour cela, nous avons mis en pratique la méthodologie précédemment énoncée en introduction. Par exemple dans le cas de la fonction de hachage MD5, nous avons généré, puis pré-traité, toutes les formules de l'étape 17 à l'étape 64, en fixant les variables de l'empreinte à une valeur choisie, qui ne peut être le fruit du hasard. Puis, nous avons résolu ces formules en commençant par la plus simple jusqu'à la plus difficile. La dernière formule à être résolue nous donne la limite pour l'inversion.

Le tableau IV.1 donne une comparaison entre les meilleurs résultats en matière de cryptanalyse logique à notre connaissance et ceux issus de nos travaux. Des exemples de pré-images sont données en annexe 2.

	Meilleures attaques en pratique	**Dans cette thèse**
MD4	28 puis 39 étapes dans [De 2007]	31 puis 39 étapes [Legendre 2012b]
MD5	26 étapes dans [De 2007]	28 étapes [Legendre 2012b]
SHA-0	23 étapes dans [Nossum 2012]	23 étapes [Legendre 2012a]
SHA-1	23 étapes dans [Nossum 2012]	23 étapes [Legendre 2012a]

TABLE IV.1 – Meilleures attaques en pratique pour la recherche de pré-image sur des versions réduites des fonctions de hachage MD⋆ et SHA-⋆

Nos meilleurs résultats ont été obtenus grâce au solveur SAT complet PLINGELING appliqué sur une machine Westmere-EP de 12 cœurs [1]. En pratique, nous avons testé plusieurs solveurs SAT parmi les plus performants lors des récentes compétitions de solveurs sur les benchmarks de problèmes structurés. Sur nos instances, ils se sont tous comportés de la même façon en trouvant la solution dans une fourchette de temps équivalente. Ce phénomène était assez attendu car ces solveurs ne sont pas spécialement conçus pour la résolution d'instances cryptographiques. Le choix du solveur a donc été fait par défaut, parmi ceux utilisables en parallèle avec une prise en main aisée.

Concernant la pré-image de MD5, une remarque qui pourrait être faite consiste à dire que l'amélioration du nombre d'étapes dépend uniquement de la qualité du solveur qui est plus récent que celui utilisé dans la littérature. Cette critique étant fondée, nous avons tenté de résoudre la pré-image sur 28 étapes avec le solveur utilisé dans [De 2007], à savoir MINISAT. Il s'est avéré qu'avec ce solveur nous résolvions encore plus d'étapes que ceux présentés dans leurs travaux. Cela signifie que c'est bien notre modélisation (pré-traitement compris) qui a permis l'amélioration du nombre d'étapes inversées et non simplement la résolution.

Pour conclure, il peut être noté qu'autant d'étapes ont été inversées pour SHA-0 et SHA-1 alors que ces fonctions n'ont pas la même résistance à la recherche d'antécédents pour d'autres types de cryptanalyses. En effet, du strict point de vue de la cryptanalyse logique,

1. Grâce au *centre de calcul ROMEO en Champagne-Ardenne*, https ://romeo.univ-reims.fr.

ces fonctions ont une résistance équivalente, ce qui peut paraitre assez surprenant.

2.1 Expérimentations par rapport aux fonctions de hachage

Pour être un peu plus fin sur les chiffres énoncés en termes de nombre d'étapes inversées, nous nous sommes intéressés à l'évolution du temps de résolution en fonction du nombre d'étapes modélisées. Pour chaque nombre d'étapes attaquées [2], nous avons résolu 25 instances sur un MacBookPro 1.7 GHz à l'aide du solveur séquentiel LINGELING afin d'établir un temps moyen de résolution. Ces observations ont été retranscrites dans le graphe IV.1 qui utilise une échelle logarithmique.

Sur ce graphe, nous remarquons que quelle que soit la fonction traitée, les temps de calcul augmentent de façon exponentielle et montrent la difficulté croissante de l'attaque en pratique [3]. Il est facile d'imaginer que faire le « pas » de plus pour résoudre une étape supplémentaire est un véritable challenge. Ceci étant il faut raison garder car ces résultats dépendent de facteurs non-maitrisés. Par exemple, la modélisation du problème peut ne pas être optimale et la résolution n'est pas dédiée au problème. Il est donc possible d'avoir une meilleure estimation de la robustesse de la fonction par une amélioration de ces deux critères. Ajouté à cela, toute faille sur la primitive étudiée peut être importée pour aider à briser plus d'étapes. La valeur ajoutée d'un expert en cryptanalyse peut donc jouer un rôle essentiel et les mesures d'inversions trouvées par les expérimentations présentées ci-avant ne reflètent que très peu tous ces aspects. Le cas de l'importation de l'attaque algébrique de Dobbertin [Dobbertin 1996] sur MD4 dans notre formule propositionnelle en est le parfait exemple.

2.2 Import de l'attaque de Dobbertin

En 1996, Hans Dobbertin montre que les deux premières rondes de la fonction de hachage MD4 sont facilement inversibles simplement en affectant certains blocs du message d'entrée à des valeurs spécifiques. Cette condition permet de mettre quelques états internes à 0x0000000, provoquant l'inutilité des fonctions non-linéaires aux mêmes endroits, ce qui entraîne ensuite la fixation d'autres états internes. Pour résumer, cette attaque permet de supprimer plusieurs étapes du hachage tout en gardant une certaine liberté dans le message d'entrée.

Dans le cadre de la cryptanalyse logique, l'affectation d'une variable est quelque chose de très facile à faire. L'enchaînement de conséquences, comme elles sont décrites ci-avant, correspond en logique à de la propagation unitaire. Autrement dit, c'est un processus qui va s'exercer de lui-même dès que l'affectation aura été faite. Il a donc été aisé pour nous de reproduire l'attaque de Dobbertin dans le contexte de la cryptanalyse logique.

Reprenant notre méthodologie, nous avons généré des formules CNF pour des versions réduites de la fonction MD4, dans lesquelles nous avons fixé l'empreinte et les variables d'une partie du message d'entrée comme indiqué dans [Dobbertin 1996]. Procédant ainsi,

2. exception faite de la dernière étape inversée qui peut avoir une formule très longue à résoudre
3. Par contre, la taille des instances, en terme de clauses et de variables, n'augmente que linéairement.

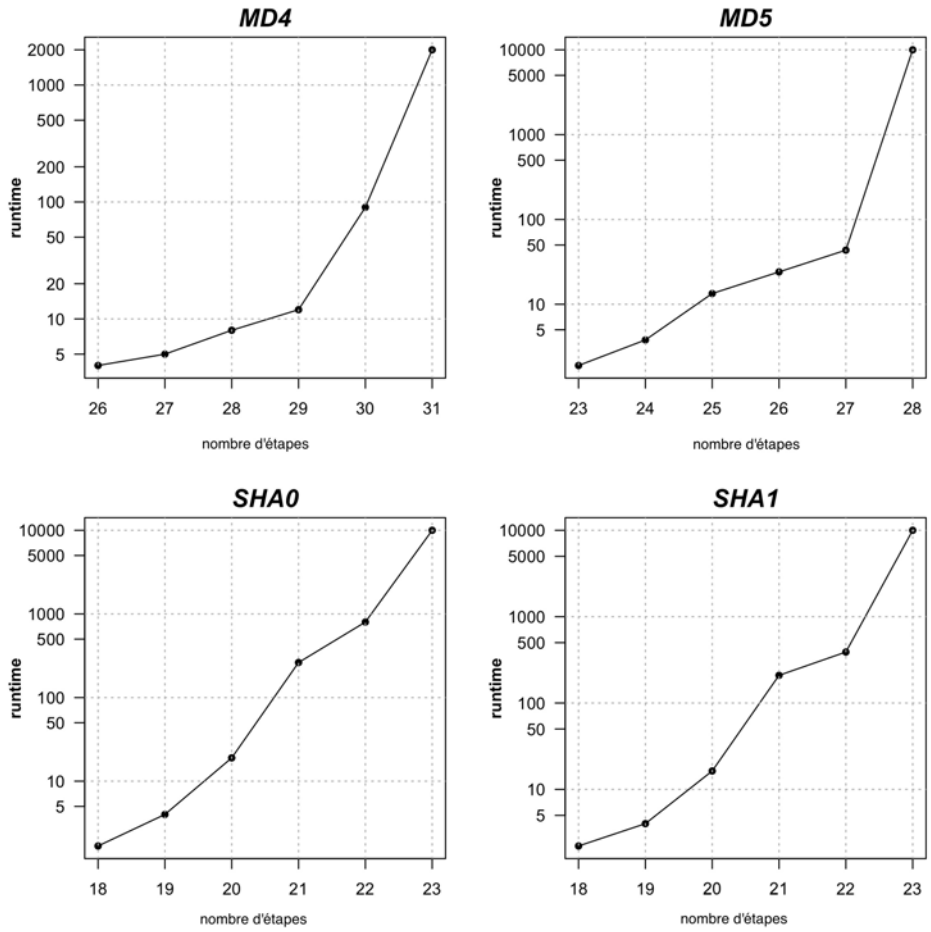

FIGURE IV.1 – Temps de calcul pour la résolution d'instances représentant des versions réduites des fonctions de hachage MD4, MD5, SHA-0 et SHA-1, en fonction du nombre d'étapes inversées

nous avons réussi à inverser jusqu'à 39 étapes, inversant donc 8 étapes de plus qu'auparavant. Ci-après quelques détails sur l'attaque sont donnés.

Soit Q_i, l'état modifié à l'étape $i \in \{ 1, ..., 64 \}$, et M_j le j^{ieme} sous-bloc de 32 bits issu du message d'entrée, $j \in \{ 0, ..., 15 \}$. Sont fixés :

- Q_{14}, Q_{15}, Q_{17} = `0x00000000`
- M_1, M_2, M_4, M_5, M_6, M_8, M_9, M_{10} = `0xa57d8667`

Par propagation unitaire, les états suivants sont aussi fixés :

- Q_{18}, Q_{19}, Q_{21}, Q_{22}, Q_{23}, Q_{25}, Q_{26}, Q_{27} = `0x00000000`

Ci-après l'exemple d'une pré-image trouvée pour une version réduite de la fonction de hachage MD4 à 39 étapes.

Empreinte fixée :
`0x00000000 0x00000000 0x00000000 0x00000000`
Message trouvé :
`0x321838cd 0x67867da5 0x67867da5 0x4bd844ff 0x67867da5 0x67867da5`
`0x67867da5 0x60babe30 0x67867da5 0x67867da5 0x67867da5 0x2e731890`
`0xb84655eb 0x1094c071 0xce0cfe36 0x0252233c`

Transposer cette attaque dans un contexte logique n'est pas une idée novatrice puisque cela a déjà été produit dans [De 2007] avec le même résultat final. Cependant, nous avons pu confirmer que les temps de calculs sur la figure IV.1 - qui auraient prédit une résolution pour 39 étapes de MD4 en plusieurs années de calculs - peuvent nettement être confondu par une faille dans la fonction. Nous avons d'ailleurs de nouveau calculé ces temps de calculs pour MD4 en reprenant les instances comportant moins de 39 étapes tout en considérant l'attaque de Dobbertin. Les résultats sont sur la figure IV.2.

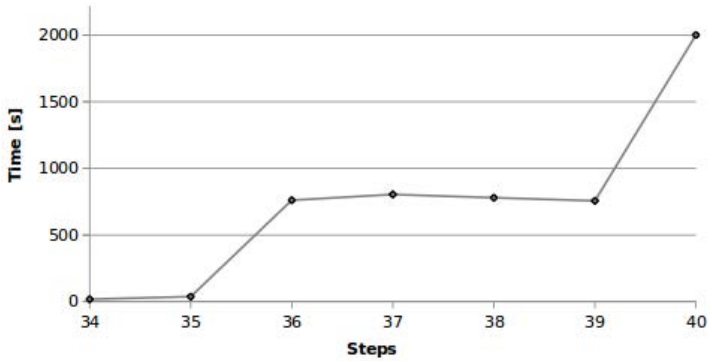

FIGURE IV.2 – Temps de calcul pour la résolution d'instances représentant des versions réduites de la fonction de hachage MD4 avec l'apport de l'attaque décrite dans [Dobbertin 1996], en fonction du nombre d'étapes inversées

Étrangement, la courbe d'évolution des temps de calculs n'a pas une croissance linéaire, mais plutôt par paliers. Des sauts sont visibles entre les étapes 35 et 36 puis entre les étapes 39 et 40 (restée irrésolue après 2000s). Ce phénomène est expliqué par le fait que l'espace de recherche est corrélé aux étapes affectées par l'attaque de Dobbertin. En réalité, cette attaque fixe les sous-blocs du message d'entrée qui apparaissent aux étapes 34, 35, 37, 38, 39 puis 41 etc.Nous pouvons remarquer que les étapes 36 et 40 - les étapes de sauts - ne sont pas concernées. Par conséquence, les contraintes sont plus nombreuses et le ratio (espace de solution)/(espace de recherche) décroit spontanément.

Cette observation est vérifiable en la comparant avec le nombre de clauses pour chacune des instances étudiées, visible sur le graphe IV.3. Les étapes finales admettent une progression quasi-linéaire pour le nombre de clauses car aucun des sous-blocs de 32 bits du message d'entrée utilisé dans l'attaque de Dobbertin n'intervient dans leurs calculs.

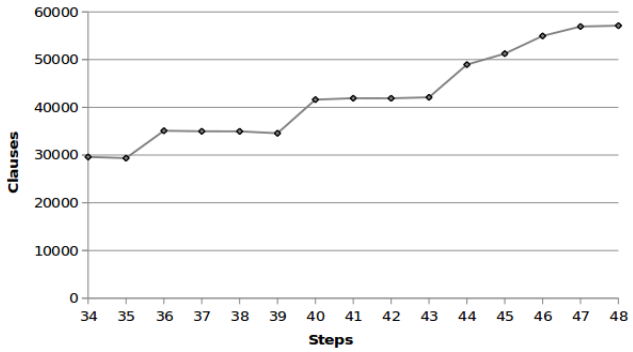

FIGURE IV.3 – Nombre de clauses composant les instances représentant des versions réduites de la fonction de hachage MD4 avec l'apport de l'attaque décrite dans [Dobbertin 1996] en fonction du nombre d'étapes.

2.3 Expérimentations par rapport aux modélisations de l'addition

Au chapitre II partie 2.2.2, nous avons montré qu'il existe plusieurs façons pour modéliser une addition à n opérandes sous la forme de clauses SAT. Dans ce cadre, nous avons modifié le générateur de formules décrit chapitre II partie 2.6.2 pour la fonction de hachage MD5 afin de générer des formules SAT correspondant à chacune des modélisations présentées. Cette partie est dédiée à l'étude de ces différentes modélisations. En effet, nous avons choisi de reconduire nos attaques sur la recherche d'antécédent sur MD5 en fonction du nombre d'étapes modélisées. Pour cela, les conditions ont été reproduites à l'identique, c'est-à-dire sur 25 instances sur un MacBookPro 1.7 GHz à l'aide du solveur séquentiel LINGELING. Le tableau IV.2 montre quelques statistiques sur les temps de calcul pour la résolution de ces instances en fonction du choix de la modélisation de l'addition à quatre opérandes. Nous pouvons simplement remarquer que l'encodage découpé (les additions à quatre opérandes sont modélisées uniquement avec des additions à deux opérandes) paraît meilleur sur les instances facilement résolubles. Cependant, seul l'encodage global a permis d'inverser 28 étapes. Nous expliquons cela par le nombre de variables trop conséquent dans l'encodage découpé qui tend à demander beaucoup plus (et ici trop) de mémoire lors d'une résolution avec un solveur CDCL.

2.4 Apport du pré-traitement et des informations probabilistes

Dans cette partie, nous donnons quelques résultats pratiques qui montrent comment SAT peut aider à accélérer la résolution d'un problème. Nous présentons dans le tableau IV.3 les temps de calculs obtenus sur des attaques de pré-images pour des instances réduites de la fonction de hachage MD5. La colonne *Simplification* indique si l'instance a été pré-traitée ou non et la colonne *Probas* indique si des informations probabilistes ont été ajoutées à la formule. Dans ce cas, il a été fait en sorte que ces informations ne consument pas plus de 50% de l'espace de solution. L'ensemble de ces expérimentations a été réalisé grâce à la version parallèle de LINGELING nommée PLINGELING [Biere 2010] appliquée

Nb d'étapes	Encodage	Temps
25 étapes	global	de 0 à 1 s
	découpé	de 0 à 1 s
	hybride	0 s
26 étapes	global	de 4 à 10 s
	découpé	de 7 à 9 s
	hybride	de 10 à 16 s
27 étapes	global	de 6 à 13 s
	découpé	de 6 à 13 s
	hybride	$\simeq$ 12 s
28 étapes	global	de 30 m à 24 h
	découpé	irrésolue en 24h
	hybride	irrésolue en 24h

TABLE IV.2 – Temps de calcul pour la recherche de premier antécédent sur des versions réduites de la fonction de hachage MD5 en fonction de la modélisation choisie

sur deux processeurs Intel Ivy Bridge 8 cores 2,6 GHz[4]. Dans ce tableau, nous pouvons observer que les temps de calculs obtenus sur une formule SAT pré-traitée sont meilleurs que ceux où elle ne l'est pas. D'autre part, la remarque est aussi valable pour les formules qui ont subies un apport probabiliste. Cela montre que purifier nos instances peut s'avérer efficace en vue d'améliorer une résolution pratique. Ajouté à cela, le pari pris de réduire à la fois l'espace de recherche et l'espace de solution par des données probabiliste montre qu'il est possible d'améliorer une nouvelle fois les temps de calcul. Néanmoins, il faut reconnaitre que les améliorations restent très légères et qu'un travail plus conséquent sur la bonne utilisation de ces informations probabilistes pourrait les faire fructifier davantage. Pour conclure sur ce point, ces expérimentations montrent tout de même qu'un travail en amont sur la modélisation peut jouer un rôle non-négligeable sur la résolution en pratique des instances SAT cryptographiques.

2.5 Expérimentations par rapport aux solveurs SAT

Pour clore ce point sur les attaques de pré-images, notons que nos travaux de modélisation en amont ont permis d'améliorer ou d'égaler les meilleurs résultats existants en cryptanalyse logique pour l'inversion des fonctions étudiées. Cette contribution a fait office d'une publication dans [Legendre 2012b] et une réadaptation de la méthode sur SHA-3 est en cours.

Par ailleurs, même si la cryptanalyse logique offre un cadre pratique adapté à ce genre d'expérimentations, les estimations données pour la résistance à la pré-image sont liées à l'outil de mesure. Dans ce contexte, cet outil de mesure est défini par la qualité de la modélisation et la qualité du solveur SAT, qui eux mêmes dépendent de facteurs en amont (mo-

4. Grâce au centre de calculs de Champagne-Ardenne *ROMEO*, https ://romeo.univ-reims.fr

SAT SOLVEUR			PLINGELING
FORMULE	SIMPLIFICATION ?	PROBAS AJOUTÉES ?	TEMPS (S)
md5-25-steps	non	non	3.8
md5-25-steps	oui	non	2.3
md5-25-steps	oui	oui	**1.8**
md5-26-steps	non	non	8.3
md5-26-steps	oui	non	5.4
md5-26-steps	oui	oui	**4.0**
md5-27-steps	non	non	8.8
md5-27-steps	oui	non	5.2
md5-27-steps	oui	oui	**4.1**

TABLE IV.3 – Temps de calcul en moyenne pour 100 résolutions de formules SAT, en fonction de la primitive modélisée, de sa simplification logique et de l'apport de données probabilistes.

délisation « handmade » ou automatique, résolution dédiée etc.). Or, nous avons montré par l'amélioration du nombre d'étapes inversées sur MD5 et par la faille de [Dobbertin 1996] sur MD4 qu'une mesure peut être améliorée, quand bien même les modèles de prévision dérivent exponentiellement. La récence du domaine de recherche et des attaques conduites en ce sens promettent une inversion plus importante du nombre d'étapes en pratique dans les années à venir.

3 Étude de la collision de Xie et Feng (2010)

Le contexte est le suivant. Une formule SAT représentant le processus de hachage de MD5 pour un bloc de 512 bits en entrée a été générée, et le message d'entrée et l'empreinte restent indéfinis. L'idée ici est d'utiliser cette formule pour étudier une collision donnée par la littérature. Dans ce cadre, nous nous sommes intéressés de plus près à la collision trouvée par Tao Xie et Dengguo Feng dans [Xie 2010] qui a la particularité d'être trouvée sur un bloc message de 512 bits en entrée, ce qui la rapproche de notre CNF, contrairement aux collisions multi-blocs. Nous montrons dans cette partie comment le formalisme donné par la cryptanalyse logique permet d'étudier les caractéristiques spécifiques d'une collision afin d'en déduire des informations intéressantes.

Fin 2010, Tao Xie et Dengguo Feng ont publié le résultat d'une attaque par collision avec en entrée deux messages de 512 bits, n'ayant que deux bits de différences. C'est la première collision un bloc sur MD5 qui a été trouvée. Voici ce qu'ils ont obtenu :

Message 1 :

```
0x6165300e 0x87a79a55 0xf7c60bd0 0x34febd0b 0x6503cf04 0x854f709e
0xfb0fc034 0x874c9c65 0x2f94cc40 0x15a12deb 0x5c15f4a3 0x490786bb
0x6d658673 0xa4341f7d 0x8fd75920 0xefd18d5a
```

Message 2 :
0x6165300e 0x87a79a55 0xf7c60bd0 0x34febd0b 0x6503cf04 0x854f74 9e
0xfb0fc034 0x874c9c65 0x2f94cc40 0x15a12deb 0xdc15f4a3 0x490786bb
0x6d658673 0xa4341f7d 0x8fd75920 0xefd18d5a

Hash_MD5(Message 1) = Hash_MD5(Message 2) :
0xf999c8c9 0xf7939ab6 0x84f3c481 0x1457cb23

Vérification de la collision La première chose sur laquelle nous nous sommes penchés a été de reproduire la collision. Pour ce faire, leurs messages hexadécimaux ont été traduits sous un langage ASCII pour avoir des 0 et des 1, puis traduit en affectation de variables. L'affectation d'un des deux messages a ainsi pu donner, par propagation unitaire, une valeur de vérité pour chaque variable de la formule. Les variables correspondant à l'empreinte ont ensuite été dégagées pour être traduites en hexadécimal. En procédant ainsi pour les deux messages d'entrées, les empreintes ont pu être identifiées et la collision confirmée.

Ce qui est intéressant dans cette démarche, c'est que pour chacune des deux affectations données, nous avons un rendu complet sur toutes les variables, qui peut être étudié pour y dégager des spécificités.

Étude statistique La première voie que nous avons suivie a été de compter et d'identifier les bits qui sont différents entre la trace d'exécution de leur premier message et celles provoquées par trois autres messages bien choisis.

Soit *M1* le premier message donné par [Xie 2010] et *M2* le second. *M"* est un message dérivé de *M1* avec une distance de Hamming extrêmement faible. Enfin *A* représente le message d'entrée « a » . Sur le tableau IV.4 le résultat de diverses comparaisons entre la trace d'exécution de *M1* et *M2*, puis de *M1* et *M"* et enfin de *M1* et *A*.

Différentes statistiques	M1 versus M2	M1 versus M''	M1 versus A
Comparaisons (totales)	12721	12721	12721
Nb 0 dans M1	6595	6595	6595
Nb 0 dans le second message	6540	6707	6999
Nb 1 dans M1	6126	6126	6126
Nb 1 dans le second message	6181	6014	5722
Nb 0 devenus 1	**437**	2583	2775
Nb 1 devenus 0	**382**	2694	3178
Différences 0/1	**819**	5277	5953
% de différences	**6,4**	41,5	46,7

TABLE IV.4 – Statistiques sur les différences observées entre les traces d'exécution pour M1 avec d'autres messages d'entrée.

Analyse En termes de probabilité, puisque le processus de hachage donne des mots aléatoires, environ un bit sur deux doit être à une valeur opposée entre deux traces d'exécution différentes ; ce qui signifie un pourcentage de différences d'environ 50%. En réalité, ce taux est plus faible à cause des secondes retenues plus souvent à 0 qu'à 1, comme nous l'avons montré au chapitre III. Globalement, le tableau montre que les statistiques sont à peu près équivalentes entre une trace d'exécution pour M1 et celle pour un autre message, quelqu'il soit.

Seules deux composantes ont un rapport vraiment différent : le nombre de 0 passés à 1 et le nombre de 1 passés à 0 (qui sont corrélés). En effet, alors que la comparaison entre *M1* et *A* et *M1* et *M''* montrent plus de 5 200 passages de 0 à 1 (ou inverse), le différentiel entre *M1* et *M2* ne compte qu'environ 800 changements. C'est donc tout naturellement que nous avons approfondi notre étude sur ces différences ciblées.

Localisation des différences Pour repérer où se situaient les différences, nous avons mis au point un petit programme qui permet de connaître exactement quel bit est touché par un changement. De la même façon, ce programme permet de tirer une analyse simpliste et quantifiée des différents changements en fonction des 64 étapes. La figure IV.4 est une représentation du nombre de ces changements.

Le cadre de la cryptanalyse logique offre aussi la possibilité de connaître avec une précision exacte quel bit à été touché et dans quelle partie du hachage. Par exemple, une différence se situe à l'étape 34, portant sur le 15$^{\text{ème}}$ bit du résultat de la fonction non-linéaire.

À ce titre, nous avons changé notre point de vue pour, non plus observer où se situaient ces différences en fonction des étapes, mais plutôt en fonction de la signification du mot dans lesquelles nous les trouvons. Le tableau IV.5 référence les statistiques obtenues en comparant les deux messages de [Xie 2010] (colonnes M1 et M2) avec une statistique moyenne réalisée sur des hachages aléatoires (troisième colonne). Les deux colonnes Δ **M1** et Δ **M2** montrent le nombre de différences entre les messages servant à la collision par rapport à la moyenne.

Les différences se situent principalement sur les retenues de second rang, avec notamment un nombre de bit à 1 beaucoup plus important (environ 9-10%). Cela montre l'importance de ces retenues dans la recherche de collisions puisqu'il est facile d'imaginer qu'elles servent à absorber les différences engendrées. Pour faire écho au chapitre 2 partie 2.2, il paraît clair qu'une façon d'atténuer ce phénomène serait de ne pas utiliser d'addition sur quatre opérandes ou alors d'éviter l'utilisation de constante si elle n'apporte rien en terme de sécurité, en reportant notamment la valeur ajoutée de l'injection de constantes à un autre mécanisme.

Pour conclure sur cette partie, notons que le contexte détaillé de la cryptanalyse logique permet d'étudier très précisément les traces d'exécutions d'une attaque par collisions. Les travaux présentés ici ont notamment permis de déchiffrer et de mesurer les caractéristiques d'une attaque par chemin différentiel. Dans ce cadre, le problème SAT agit finalement comme un outil d'analyse et non plus seulement comme un outil de conception. Enfin,

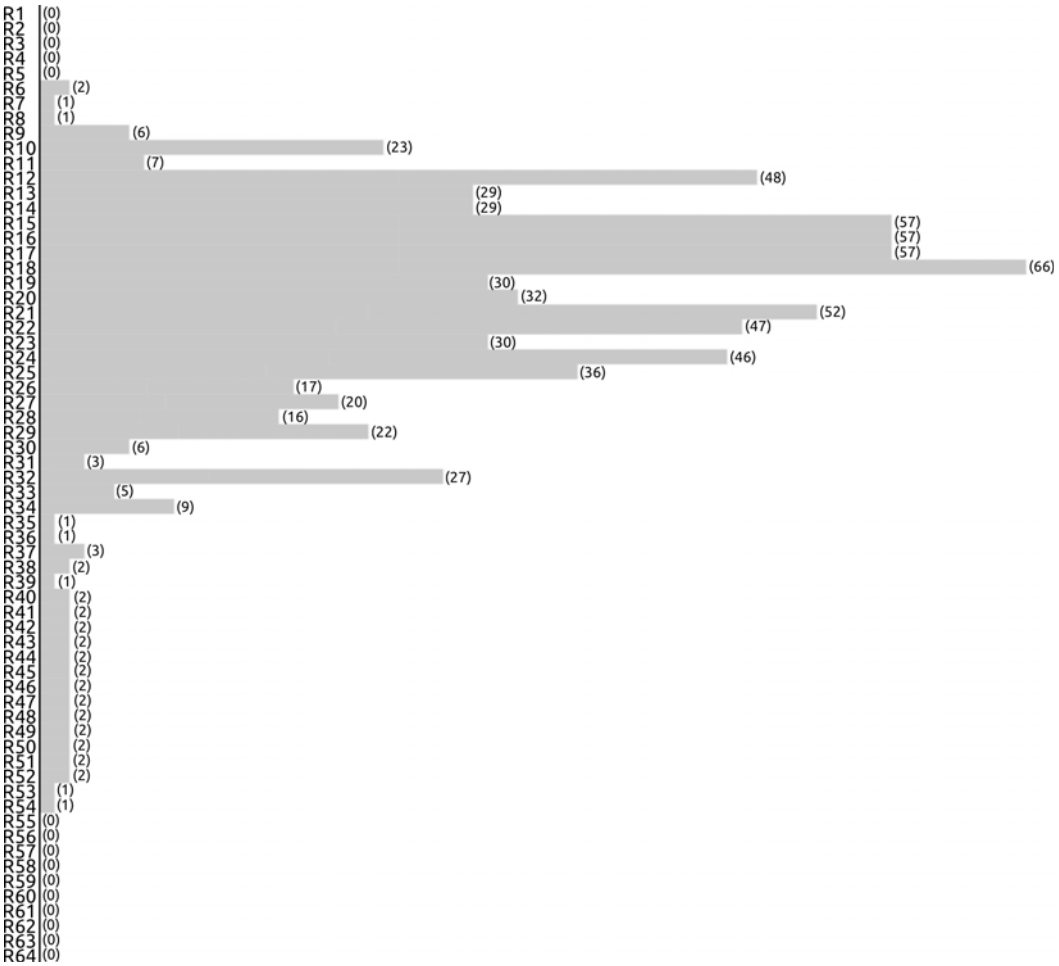

FIGURE IV.4 – Répartition des bits différents sur une collision un bloc, en fonction des 64 étapes du processus de hachage

la formule SAT peut aussi servir dans l'autre sens, pour implanter directement un chemin différentiel et attaquer la collision. C'est ce que nous montrons dans les parties suivantes.

4 Attaquer la collision

4.1 Principe et modélisation

Il est possible de modéliser le problème de la collision en une formule SAT en réutilisant les formules représentant un processus de hachage. L'idée est de produire deux formules, de les concaténer et de préciser que les empreintes doivent être identiques. Il faut aussi ajouter la condition suivante : deux variables représentant le même bit dans les deux entrées doivent nécessairement être à des valeurs opposées. La formule ainsi obtenue a pour solution une affectation des variables telle qu'elle donne deux messages d'entrées différents ayant la même empreinte, c'est-à-dire une collision. La figure IV.5 représente schématiquement ce concept.

Localisation	M1	M2	Moyenne	Δ M1	Δ M2
Q_i à 0	1038	1008	1029	8,7	21,3
Q_i à 1	1010	1040	1019	8,7	21,3
S_i à 0	1013	1002	1029	16,3	27,3
S_i à 1	1035	1046	1019	16,3	27,3
fC_i à 0	798	809	820	21,9	10,9
fC_i à 1	1186	1175	1164	21,9	10,9
sC_i à 0	1368	1357	1409	**40,7**	**51,7**
sC_i à 1	547	558	506	**40,7**	**51,7**
f_i à 0	1002	997	1005	3,2	8,1
f_i à 1	1006	1011	1003	3,2	8,1
tC_ià 0	1010	1003	1026	16,5	23,5
tC_i à 0	974	981	958	16,5	23,5

TABLE IV.5 – Récapitulatif des différences entre les traces d'exécution pour M1 avec d'autres messages. La notation utilisée est la suivante : Q_i, les états ; fC_i et sC_i, la retenue de premier et de second rang pour l'addition à quatre opérandes ; tC_i la retenue qui intervient dans l'addition à deux opérandes ; f_i la fonction non-linéaire ; S_i la somme de quatre opérandes.

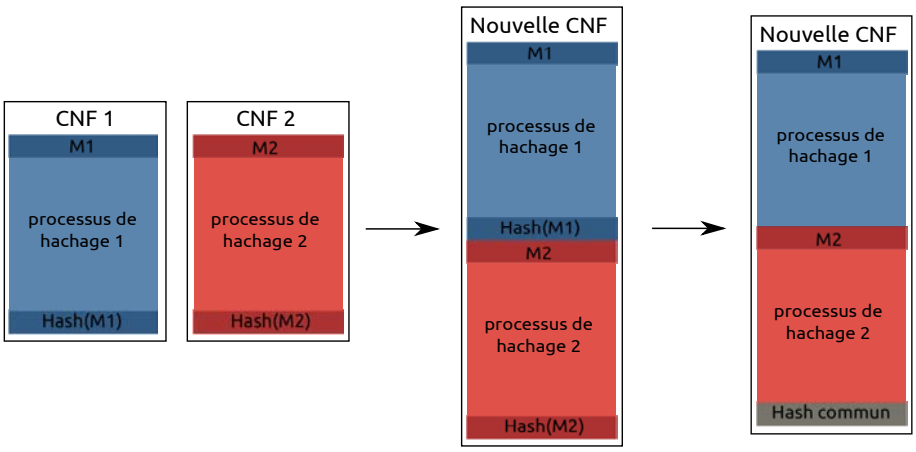

FIGURE IV.5 – Construction d'une formule pour la recherche de collisions sur MD5

La formule SAT proposée pour cette modélisation comporte 451 355 clauses et 25 370 variables. C'est à peu près deux fois plus que la modélisation du problème de recherche d'une pré-image. Pourtant, cette formule représente un problème théoriquement plus simple. En pratique, nous n'avons pas encore réussi à résoudre ce problème via une approche basée sur SAT.

Tout comme nos attaques sur la pré-image, nous avons donc cherché à résoudre le problème de recherche de collisions sur des versions réduites de la fonction de hachage. Malheureusement, tenter de résoudre des versions réduites pour la collision reste un problème compliqué pour le solveur parce qu'il cherche une solution de la mauvaise façon. En effet, quand une telle formule lui est proposée, le solveur satisfait un processus de hachage,

empreinte comprise, puis cherche à satisfaire le second. Cela signifie qu'il agit exactement comme si on lui proposait de rechercher une seconde pré-image. Ici encore, nous pouvons soulever le besoin d'avoir des méthodes de résolution entièrement dédiées au problème traité. En terme de résultats, la recherche de collisions telle qu'elle est présentée ne permet donc pas d'attaquer plus d'étapes que la pré-image.

Ce souci pourrait néanmoins être corrigé en affectant des priorités sur les variables au niveau de l'heuristique du solveur, de sorte à ce qu'il essaie d'abord de satisfaire les contraintes concernant des mots ciblés à l'avance par un cryptanalyste et non celles sur le message d'entrée. À défaut d'avoir une réponse garantie sur cette idée, nous avons préféré choisir une autre voie, celle donnée par la cryptanalyse différentielle.

4.2 Implanter un chemin différentiel de Stevens

Dans ce contexte, nous nous sommes intéressés aux travaux de Marc Stevens [Stevens 2012a] dans lesquels une collision un bloc sur MD5 a été trouvée. Son étude étant assez détaillée, nous avons accès aux conditions suffisantes sur les bits des états qui sont calculés à chaque étape, de sorte à ce qu'ils garantissent la découverte d'une collision avec une bonne probabilité. Ces conditions, disponibles en annexe 2, sont très facilement transposables sous un format logique. Dans la table IV.6, sont recensées les différents cas conditionnels ainsi que leurs traductions logiques. P1 et P2 sont les deux processus de hachage.

Condition	*Signification*	*Sous forme logique*
0	Le bit est à 0 dans P1 et P2	Affectation des variables à 0 et 0
1	Le bit est à 1 dans P1 et P2	Affectation des variables à 1 et 1
+	Le bit est à 1 dans P1 et à 0 dans P2	Affectation des variables à 1 et 0
-	Le bit est à 0 dans P1 et à 1 dans P2	Affectation des variables à 0 et 1
$\wedge$	Le bit a la même valeur qu'à l'étape i-1	Ajout de 2 clauses pour l'équivalence
!	Le bit a la valeur opposée à celle étape i-1	Ajout de 2 clauses pour l'inégalité

TABLE IV.6 – Conditions suffisantes pour suivre un chemin différentiel dans une recherche de collisions et traduction sous forme logique

Le travail que nous avons réalisé a été de transposer toutes les conditions données par [Stevens 2012a] dans notre formule pour la collision. Ce sont minutieusement 122 clauses binaires et 1 030 affectations de variables qui ont été ajoutées à la formule. Après pré-traitement, nous avons obtenu une formule SAT composée de 271 780 clauses et 23 064 variables représentant la collision, avec la contrainte de la recherche d'un chemin différentiel. Ce nouveau problème modélisé a été passé à un solveur SAT mais demeure encore irrésolu à ce jour.

4.3 Approche par collisions locales

La partie 3 montrait qu'une collision par chemin différentiel a pour caractéristique d'avoir une région fortement contrainte dans *le haut* du processus de hachage, et une région quelque peu contrainte dans les dernières étapes. Avant, après et entre les deux, les contraintes sont très peu nombreuses puisque les états Q_i intervenant dans les deux processus de hachage sont très semblables (le différentiel est proche de 0).

Partant de ce principe, une attaque sur la collision peut naïvement être mise en œuvre. La première étape consiste à trouver une collision locale pour les étapes se situant vers la fin du processus. Puis, injecter cette collision locale dans une formule complète pour enfin la résoudre. La solution d'une telle formule, où les contraintes sont principalement situées dans le *haut* du hachage répond au problème de la collision.

1ère étape : recherche d'une collision locale

Par la suite nous appelons une collision locale de x à y, une collision sur une version réduite du processus de hachage, allant de l'étape x à l'étape y. Soient P_1 et P_2 deux processus de hachage, qui ont respectivement pour entrée les messages m_1 et m_2. Soit, $P_j^{(x,y)}$ la partie du processus de hachage P_j entre les étapes x et y ; et soit Q_i^{Pj}, l'état Q à l'étape i dans le processus de hachage P_j. Nous considérons trois conditions :

- $Q_x^{P_1} = Q_x^{P_2}$
- $Q_y^{P_1} = Q_y^{P_2}$
- $\exists\, q$ tel que $x < q < y$, tel que $Q_q^{P_1} \neq Q_q^{P_2}$

Enfin, nous appelons $\mathcal{F}_{col}^{(x,y)}$, la formule SAT modélisant une collision locale entre les étapes x et y, construite à partir de $P_1^{(x,y)}$ et de $P_2^{(x,y)}$. Pour que cette sous-formule soit importable dans une formule complète il faut s'assurer que les empreintes finales soient identiques. Pour cela, il est nécessaire d'anticiper le fait que les sous-blocs issus de m_1 et m_2 intervenant après l'étape y soient identiques entre $P_1^{(y,63)}$ et $P_2^{(y,63)}$ en ajoutant des contraintes d'équivalences entre les variables qui correspondent.

Une solution sur $\mathcal{F}_{col}^{(x,y)}$, est une affectation des variables présentes dans $P_1^{(x,y)}$ et $P_2^{(x,y)}$ parmi lesquelles figurent des variables issues du message d'entrée. Ceci est représenté sur la figure IV.6 (à titre représentatif), avec en gris les variables communes entre P_1 et P_2, en orange les variables propres à P_1 et en vert les variables propres à P_2.

2ème étape : de $\mathcal{F}_{col}^{(x,y)}$ vers $\mathcal{F}_{col}^{(0,63)}$

Quand une solution est trouvée sur $\mathcal{F}_{col}^{(x,y)}$ il faut l'exporter dans $\mathcal{F}_{col}^{(0,63)}$. Pour cela, il suffit simplement de reprendre la solution sur $\mathcal{F}_{col}^{(x,y)}$ et d'affecter les variables correspondantes dans $\mathcal{F}_{col}^{(0,63)}$. La représentation IV.7 montre de façon schématique l'état de la formule $\mathcal{F}_{col}^{(0,63)}$, où les parties colorées sont les variables affectées. Celles avant l'étape x et après l'étape y sont celles provenant de m_1 et m_2 et fixées dans la collision locale.

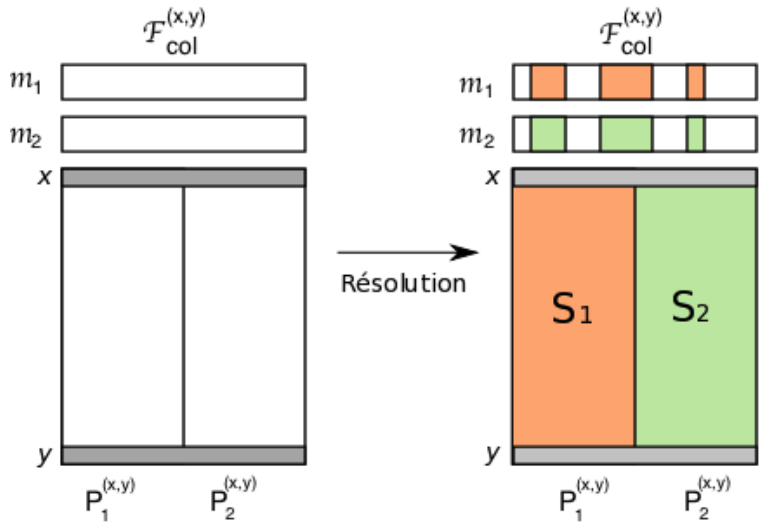

FIGURE IV.6 – Construction et idée de la résolution d'une formule pour la recherche de collisions locales sur MD5

3ème étape : recherche d'une solution sur $\mathcal{F}_{col}^{(0,63)}$

Cette dernière étape concerne la résolution de l'instance ainsi créée. Soit $m_{(i,j)}$ le $j^{\text{ème}}$ sous-bloc de 32 bits (appelés mots par la suite) du message d'entrée i. À ce stade, il est possible de classer les $m_{(i,j)}$ en trois catégories :

- **C1** : Les mots libres, qui **n'apparaissent pas** dans la collision locale mais **qui apparaissent** après la collision locale (ces mots sont identiques entre P_1 et P_2).
- **C2** : Les mots fixés et non-identiques, qui **apparaissent** dans la collision locale et **qui n'apparaissent pas** après la collision locale.
- **C3** : Les mots fixés et identiques, qui **apparaissent** dans la collision locale et aussi après celle-ci.

Ce sont les mots libres qui donnent une mesure de l'espace de recherche de solution ; plus ils sont nombreux, plus cet espace est grand. Par conséquent, toute la difficulté repose sur la bonne définition des étapes x et y pour la recherche de collisions locales car ce sont elles qui encadrent la complexité pratique du problème par l'engendrement des espaces de recherches et de solutions.

Plus concrètement, nous avons travaillé sur la fonction de hachage MD5 qui comporte 4 rondes de 16 étapes, soient 64 étapes au total, numérotées par la suite de 0 à 63. Fixer y puis x détermine le nombre de mots intervenant après la collision locale puis le nombre de mots libres en entrée. Dans le cadre de nos travaux, nous avons d'abord choisi $y = 53$. Ceci signifie que les mots apparaissant entre les étapes 53 et 63, c'est-à-dire $\{m_{(i,1)}, m_{(i,2)}, m_{(i,4)}, m_{(i,6)}, m_{(i,8)}, m_{(i,9)}, m_{(i,10)}, m_{(i,11)}, m_{(i,13)}, m_{(i,15)}\}$, sont forcément identiques entre P_1 et P_2. Les six mots restants dans la dernière ronde, donnant l'ensemble $\{m_{(i,0)}, m_{(i,3)}, m_{(i,5)}, m_{(i,7)}, m_{(i,12)}, m_{(i,14)}\}$ appartiennent à C2 et font donc nécessairement

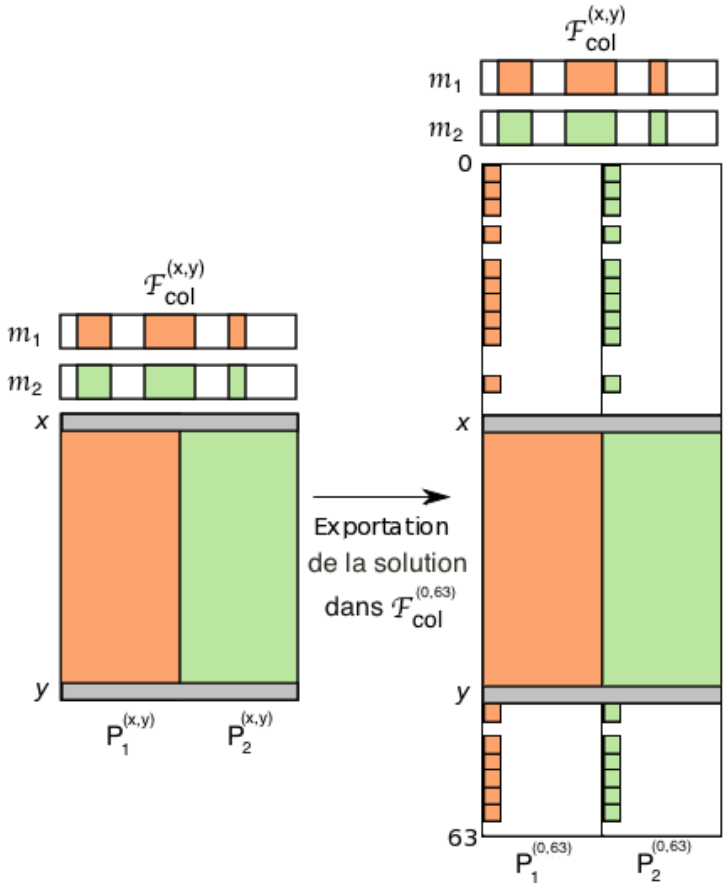

FIGURE IV.7 – Import d'une collision locale dans la résolution d'une formule pour la recherche de collisions locales sur MD5

partis de la collision locale.

Par ailleurs, nous avons choisi $x = 36$ ce qui permet de garder deux mots de libres en entrée, tel que $\{m_{(i,8)}, m_{(i,11)}\} \in$ C1. Par conséquent, $\{m_{(i,1)}, m_{(i,2)}, m_{(i,4)}, m_{(i,6)}, m_{(i,9)}, m_{(i,10)}, m_{(i,13)}, m_{(i,15)}\} \in$ C3. Tout ceci est représenté sur la figure IV.8 où des couleurs représentent C1, C2 et C3 et permettent d'y voir plus clair.

En pratique, la formule représentant la collision locale $\mathcal{F}_{col}^{(x,y)}$ se résout instantanément. Trouver une collision se résume donc à trouver une solution sur $\mathcal{F}_{col}^{(0,63)}$. Cependant, il faut noter que les mots libres sont identiques entre les deux processus de hachage P_1 et P_2. Ceci signifie qu'attaquer la collision peut se réduire à trouver une solution uniquement sur un seul des deux hachages. C'est avec cet angle que nous avons attaqué le problème. La formule comporte 4 741 variables et 55 820 clauses mais est restée irrésolue en 24 heures de calculs.

Cette attaque reste prometteuse car le solveur - non spécialisé - utilisé pour la mettre en œuvre a été limité en temps et en mémoire. De plus, elle montre une nouvelle fois comment

la cryptanalyse logique peut offrir un cadre pratique pour conduire des attaques originales contre une fonction de hachage notamment grâce à l'apport conceptuel simple de la logique propositionnelle.

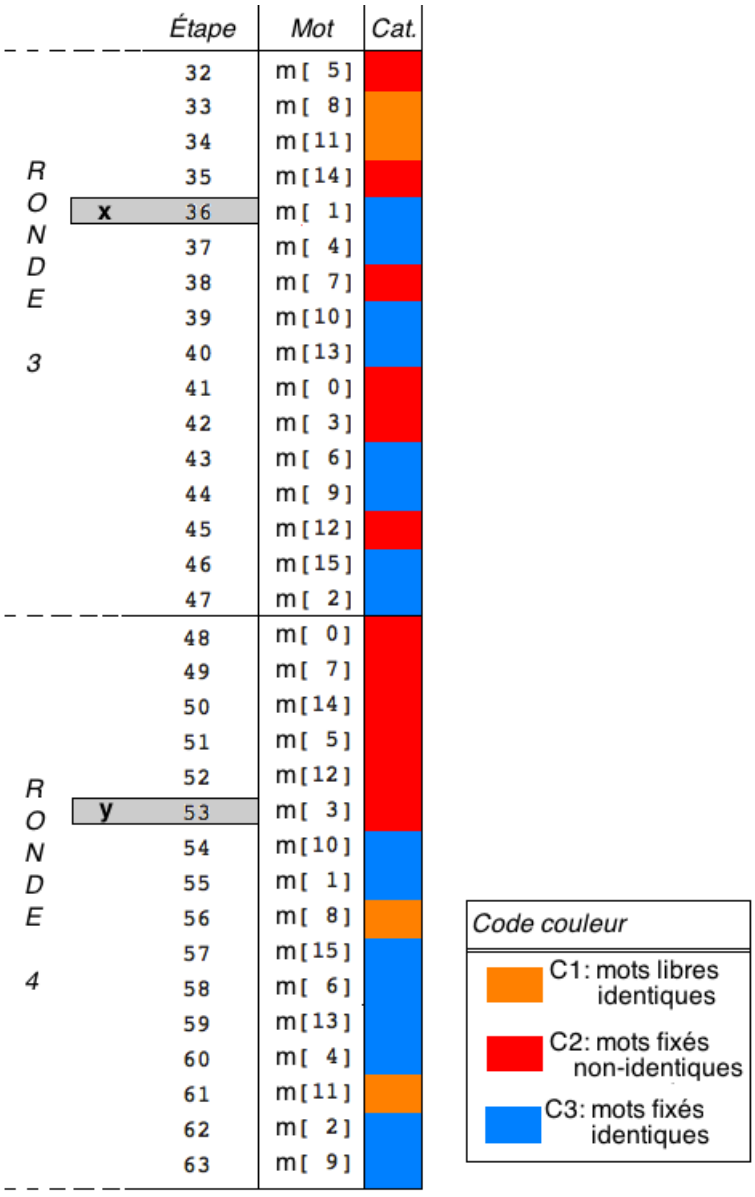

FIGURE IV.8 – Zoom sur les dernières étapes de la fonction de hachage MD5 et identification des mots du message d'entrée suivant leurs catégories $\in \{ \text{C1}, \text{C2}, \text{C3} \}$

5 Spécification d'une approche incomplète

Dans le cadre de cette thèse, nous nous sommes intéressés à la formalisation logique de problèmes cryptographiques pour leur résolution à l'aide de solveur SAT. Cette partie est

dédiée à l'aspect résolution et donne quelques pistes sur une spécification du solveur. Dans ce contexte, nous nous sommes focalisés sur une résolution incomplète de la formule. La relaxation de la complétude offre la garantie d'une réponse en un temps fini mais propose surtout un cadre pratique pour mesurer l'état d'une solution courante à travers divers paramètres, tels que le nombre de clauses insatisfaites par exemple.

Par ailleurs, quand un problème SAT est formalisé puis donné à un solveur pour le résoudre, chaque variable a *a priori* autant de chances d'être à 0 ou 1 dans une solution. Or, les travaux présentés au chapitre III montrent que certaines variables dans nos problèmes cryptographiques ont une affectation moyenne non-équiprobable dans une solution. Par conséquent, il convient d'utiliser ces connaissances acquises pour améliorer la résolution d'un solveur incomplet. Dans ce contexte, nous avons présenté dans cette partie des expérimentations sur le solveur WALKSAT dont la description est donnée chapitre I partie 4.4.

Notons enfin un point intéressant dans cette approche. Le problème SAT est un problème de décision : pour un problème donné, soit il existe une solution, soit il n'en existe pas. Dans le cadre de nos formules, il y a toujours des solutions, mais elles sont difficiles à trouver. Avoir une approche incomplète sur ces problèmes permet d'évaluer l'état courant, en comptant les clauses insatisfaites. Le terme « évaluer » est bien sûr à prendre avec des pincettes puisqu'il est tout à fait possible de baisser le nombre de clauses insatisfaites tout en s'éloignant de la solution la plus proche (voir problème DISTANCE-SAT [Bailleux 1999]). L'algorithme de résolution peut alors être envisagé comme un solveur traitant d'un problème d'optimisation consistant à baisser au maximum la borne de clause insatisfaites ; l'optimum étant 0 (puisqu'il y a au moins une solution).

5.1 Expérimentations

Les travaux autour des probabilités sur les variables ont permis de dégager des caractéristiques intéressantes sur certaines catégories de bits, notamment celles représentant les secondes retenues dans l'addition à quatre opérandes. Par conséquent, l'objectif fut d'exploiter ces caractéristiques dans le solveur incomplet WALKSAT.

WALKSAT est un solveur performant pour la recherche locale de solution. Il intègre deux principales heuristiques nommées *Best* et *Novelty*, détaillées dans le chapitre I.

L'algorithme de résolution comporte deux phases. Dans une première étape, chaque variable est initialisée de façon statique à une valeur parmi {VRAI, FAUX}. S'ensuit une seconde étape de « descente » où l'algorithme boucle sur une heuristique (traitement dynamique) pour améliorer petit à petit la borne de clauses insatisfaites. Enfin, il faut noter que cet algorithme s'arrête quand il a parcouru *n restarts*[5] durant lesquels il a exécuté *m* flips, *n* et *m* étant définis par l'utilisateur.

5.2 À propos de l'initialisation

L'observation d'un apport probabiliste sur la résolution s'est d'abord portée sur l'initialisation. Le tableau IV.7 est un récapitulatif des bornes inférieures obtenues pour des

5. Un *restart* est une nouvelle résolution, à partir de l'étape d'initialisation.

instances correspondant à une attaque sur la pré-image d'une version réduite de MD4, suivant différentes stratégies. La colonne *borne inf* correspond au minimum de clauses insatisfaites et la colonne *borne moyenne* est la borne minimale obtenue en moyenne durant l'expérimentation. À noter que les termes « + probas » signifient que les variables correspondant aux retenues ont été traitées différemment du reste en étant fixées suivant la base de données probabilistes. Enfin, la ligne « Hachage correct » est une initialisation qui prend en paramètre la trace d'exécution d'un message aléatoire qui a été haché.

Stratégie	*Borne inf*	*Borne moyenne*
Aléatoire	59 cl	82,5
Tout à 0	55 cl	80,8
Tout à 1	56 cl	82,5
Aléatoire + probas	55 cl	82,1
Tout à 0 + probas	50 cl	80,2
Tout à 1 + probas	57 cl	81,2
Hachage correct	**23**	**23,2**

TABLE IV.7 – Nombre de clauses insatisfaites au minimum et en moyenne en fonction de la stratégie d'initialisation. Les paramètres donnés à WALKSAT sont 1M de flips et 100 restarts.

Aucune stratégie ne se démarque vraiment des autres, exceptée celle initialisant les variables avec un processus de hachage correct. Cela peut paraître compréhensible puisque dès l'initialisation, la plupart des contraintes du problème sont déjà satisfaites. La seule différence entre cette initialisation et le problème étudié est l'empreinte fixée qui diffère. Quand nous observons où se situent les contraintes insatisfaites, nous nous rendons compte qu'elles contiennent des variables identifiées dans les sous-blocs du message d'entrée qui interviennent dans les dernières étapes modélisées. Autrement dit, cette méthode d'initialisation permet d'avoir un nombre de contraintes insatisfaites très bas, mais représente un processus de hachage « cassé » , que le solveur n'arrive pas à réparer.

5.3 À propos de l'heuristique

Différentes heuristiques sont présentes dans WALKSAT et permettent de tester différentes approches pour un problème donné. C'est ce que nous avons donc fait en comparant les résultats d'attaques sur la pré-image d'un MD4 suivant l'utilisation des heuristiques *Best*, *Novelty* et ses déclinaisons, *Random* (heuristique aléatoire), et *Tabu*, une heuristique à base de liste Tabu. L'expérimentation, présentée dans la table IV.8 en prenant une stratégie d'initialisation aléatoire.

L'heuristique *RNovelty(+)* apporte indéniablement le meilleur résultat. La suite logique de cette expérimentation a été de voir comment ajouter nos probabilités à cette heuristique. *RNovelty+* intègre deux idées pour le choix de variable. Soit p une probabilité. Suivant p,

Stratégie	*Borne inf*	*Borne moyenne*
Random	1054 cl	1091,2 cl
Tabu (10)	61 cl	76,6 cl
Best	242 cl	272,9
Novelty	144 cl	162,7 cl
Novelty+	142 cl	164,3 cl
RNovelty	**52 cl**	**67,8 cl**
RNovelty+	**52 cl**	**67,3 cl**

TABLE IV.8 – Nombre de clauses insatisfaites au minimum et en moyenne en fonction de la stratégie heuristique. Les paramètres donnés à WALKSAT étaient 1M de flips et 100 restart, et 10 variables dans la liste Tabu

deux variables sont successivement choisies de sorte qu'une fois retournées, elles soient chacune celles qui minimisent le nombre de clauses insatisfaites. Parmi ces deux variables, l'élue est choisie en fonction de deux critères : la qualité d'amélioration de la solution et sa récence de retournement (*flip*). Nous rappelons que l'algorithme décrivant cette heuristique est donné chapitre I partie 4.4.2.

En pratique, une touche probabiliste à été ajoutée à cette heuristique qui a été renommée pour l'occasion *RNovelty+p*. L'idée est qu'il est sûrement bon de retourner une variable qui est statistiquement plus souvent à 1 que à 0 (respectivement à 0 que à 1) alors qu'elle est positionnée à 0 (respectivement 1). L'algorithme 7 décrit cette nouvelle heuristique.

Le résultat de la comparaison entre cette nouvelle heuristique et *RNovelty+* est donné table IV.9.

Les bornes inférieures et moyennes obtenues avec *RNovelty+p* permettent de conclure sur le fait que les probabilités apportent une nette amélioration par rapport aux autres heuristiques présentes dans WALKSAT. Avant d'analyser cela, faisons un aparté sur la méthode.

Sur cette heuristique, au niveau de l'apport probabiliste, nous avions au préalable implémenté deux facteurs α et β tel que $\alpha + \beta = 1$ (et donc $\beta = 1 - \alpha$) afin de paramétrer un indice de confiance. L'heuristique était un peu plus étoffée et faisait intervenir des variables de types $\alpha \cdot p(v)$ et $\beta \cdot p(\overline{v})$. Cependant, après plusieurs expérimentations, il en est ressorti que les meilleures valeurs pour α et β étaient respectivement de 1 et 0 ; le facteur 0 de β annulant finalement plusieurs paramètres et simplifiant l'heuristique. Cette parenthèse était très intéressante car l'expérience montrait la pertinence des données probabilistes dans le sens où elles se devaient d'être stables pour être influentes.

Pour conclure sur cette heuristique, il faut remarquer que même si la méthode semble prometteuse, le nombre de clauses insatisfaites au minimum reste supérieur au nombre de clauses insatisfaites simplement obtenu en changeant l'initialisation de départ.

Algorithme 7: Principe heuristique de RNOVELTY+ avec probabilités intégrées

Données : c : une clause

pi = RANDOM([0,1]) *// une probabilité comprise entre 0 et 1, paramétré par l'algorithme*

pour *(k = 1 à $|c|$)* **faire**

    On note respectivement v_1 et v_2 les variables qui, si indépendamment retournées, apportent respectivement la plus grande et la seconde plus grande différences entre le nombre de clauses satisfaites et le nombre de clauses insatisfaites. On note ces différences d_1 et d_2.

fin

si *(v_1 n'est pas la plus récemment retournée)* **alors**

    $v = v_1$

fin

sinon

    si $(v_1 = 1)$ &$(pi > p(v_1))$ **alors**

        $v = v_1$

    fin

    si $(v_1 = 0)$ &$(pi < p(v_1))$ **alors**

        $v = v_1$

    fin

    si $(v_2 = 1)$ &$(pi > p(v_2))$ **alors**

        $v = v_2$

    fin

    si $(v_2 = 0)$ &$(pi < p(v_2))$ **alors**

        $v = v_2$

    fin

fin

Retourne v

Heuristique	*Borne inf*	*Borne moyenne*
RNovelty+	52 cl	67,2 cl
RNovelty+p	**19 cl**	**26,7 cl**

TABLE IV.9 – Nombre de clauses insatisfaites au minimum et en moyenne en fonction de l'intégration des probabilités. Les paramètres donnés à WALKSAT étaient 1M de flips et 100 restarts.

L'ensemble de ces tests a aussi été réalisé pour les fonctions de hachage MD5, SHA-0 et SHA-1. Nous donnons sur la figure IV.9 les résultats obtenus sur des instances portant sur la pré-image d'un MD5 complet en fonction du nombre de bits fixés dans l'empreinte. Les deux derniers points sur la courbe sont confondus. Ils correspondent respectivement à 126 et 127 bits fixés dans l'empreinte et comme nous pouvons le voir, aucune clause ne reste insatisfaite dans ce cas, ce qui montre une pré-image (très) partielle sur MD5. Ce résultat n'est évidemment pas extraordinaire du point de vue de la cryptanalyse, car un message sur quatre répond à une pré-image partielle de 2 bits.

Cependant, la courbe des bornes minimales en moyenne montre très bien une décroissance des clauses insatisfaites en fonction du nombre de bits fixés. C'est intéressant dans le sens où le nombre de clauses insatisfaites ne signifie pourtant pas que le solveur est proche de la solution dans une approche incomplète. En effet, dans le paysage de recherche de ce problème, la borne minimale représente un minimum local qui peut être extrêmement éloigné d'un optimum global. Dans cette situation, le solveur peut avoir besoin de passer dans un état où beaucoup de clauses sont insatisfaites (un pic dans le paysage de recherche) avant de tomber sur une solution plus acceptable. La qualité de la recherche d'une solution dans un paysage si rugueux devient alors difficile.

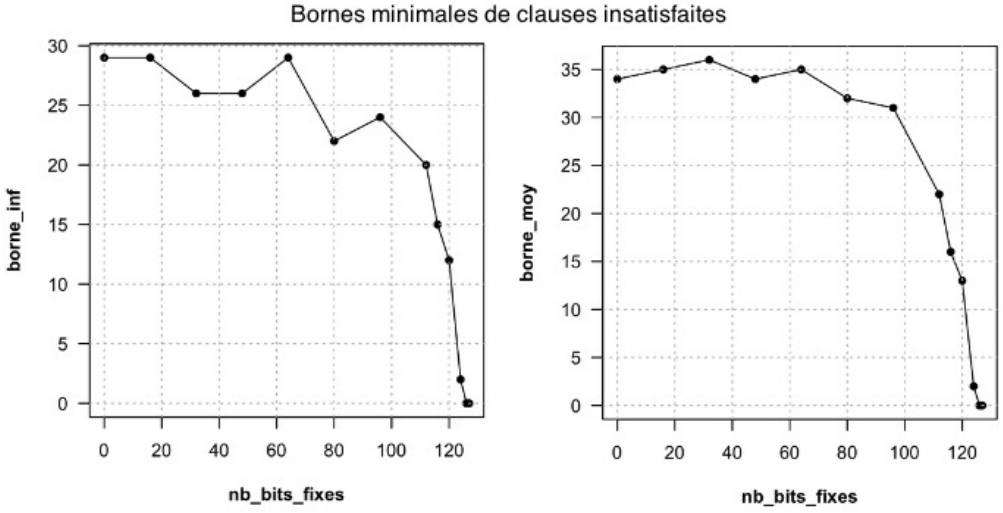

FIGURE IV.9 – Bornes inférieures et en moyenne de clauses insatisfaites obtenues avec un solveur incomplet pour la pré-image de MD5 en fonction du nombre de bits fixés dans l'empreinte. Résultats obtenus avec WALKSAT paramétré pour 100 000 flips et 100 restarts.

Les travaux autour de la spécification de l'heuristique d'un solveur SAT incomplet pour la résolution d'instances cryptographiques ont permis d'évaluer l'apport probabiliste sur les variables. Une autre voie est de tenter d'améliorer la qualité du solveur en enrichissant sa recherche locale par une métaheuristique.

5.4 Métaheuristique

Une stratégie heuristique a été mise en place pour atteindre un minimum local de façon efficace. Le but maintenant est d'avoir une stratégie locale d'intensification qui permet, à coup sûr, d'atteindre une borne minimale au moins aussi bonne. Pour cela, nous nous sommes appuyés sur l'élaboration d'une métaheuristique [6]. Ainsi, un optimum local est trouvé dans un premier temps par une méthode heuristique classique comme par exemple *RNovelty+p*. Dans un second temps la recherche est intensifiée dans la région de l'optimum local par une recherche locale itérative. Le protocole est le suivant :

1.1) Initialisation du problème. On obtient S la solution courante

1.2) Application de *Rnovelty+p* sur S. On obtient n le nombre de clauses insatisfaites

1.3) Application d'une recherche locale sur S

2.1) **Perturbation** de S pour obtenir S'

2.2) Application d'une **recherche locale** sur S'. On obtient n' le nombre de clauses insatisfaites.

2.3) Si $n' < n$ alors on remplace S par S'

Dans un cadre pratique, la perturbation sur S (point 2.1) a été faite en introduisant une notion de poids sur les clauses. Ce poids est établi en fonction des probabilités sur les variables composant la clause, comme décrit dans l'algorithme 8. La perturbation sur S consiste à inverser la valeur de vérité de tous les littéraux de la clause ayant le plus gros poids. La solution courante comporte alors un nombre de clauses insatisfaites plus élevé qu'auparavant.

La recherche locale utilisée sur la solution courante S (point 2.2) est réalisée grâce à l'heuristique RNOVELTY+, qui n'utilise pas de probabilités pour descendre. Un exemple de résolution sur les bornes minimales est la figure IV.10. D'autres heuristiques ont été expérimentées ici, comme par exemple *Best* ou une adaptation de 2-*opt* [Croes 1958], mais aucune n'a apporté une amélioration plus convaincante que RNOVELTY+.

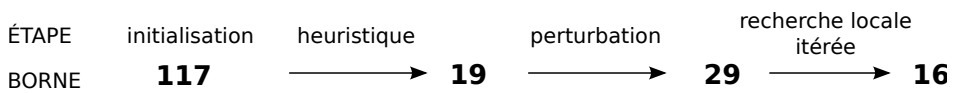

FIGURE IV.10 – Évolution de la borne minimale en fonction des étapes successives lors de l'application d'une métaheuristique.

6. schématiquement parlant, une heuristique guidant une seconde heuristique.

Algorithme 8: Pseudo algorithme de génération de poids sur les clauses

Données : $\mathcal{F}$: une formule CNF, $\mathcal{C}$ ses clauses

```
pour (chaque clause c ∈ F) faire
    POIDS(c) = 0
    pour (chaque littéral v ∈ c) faire
        si ((v = 0) & (p(v) est proche de 1)) alors
            POIDS(c) = POIDS(c) + p(v)
        fin
        si ((v = 1) & (p(v) est proche de 0)) alors
            POIDS(c) = POIDS(c) + 1-p(v)
        fin
    fin
fin
```

6 Conclusion

Ce chapitre s'est partitionné en quatre temps, avec à chaque fois des résultats ayant une portée bien différente. Dans une première partie, nous avons montré qu'il est en pratique possible d'attaquer la pré-image d'une version réduite d'une fonction de hachage. Ceci aide à donner une certaine mesure de sa résistance contre l'inversion. La modélisation du problème proposée dans cette thèse a permis d'inverser plus d'étapes que la littérature pour la fonction de hachage MD5 et au moins autant d'étapes pour MD4 et SHA.

Les deux parties suivantes sont dédiées aux attaques par collisions. Nous avons d'abord vu que la cryptanalyse logique offre un cadre idyllique pour l'analyse de collisions existantes et permet de déceler les informations sur l'attaque en question. Ensuite, deux attaques sur la collision ont été mises en pratique ; d'abord par l'implémentation du chemin différentiel de [Stevens 2012a] puis par une approche « collisions locales » . À l'heure où ces lignes sont écrites, aucune solution n'a été trouvée mais nous gardons bon espoir de passer ce cap.

Enfin le travail sur l'heuristique permet de montrer une certaine utilisation des probabilités pour la résolution d'instances cryptographiques. En toute objectivité, les résultats présentés ici ne sont que prometteurs, dans le sens où il s'agit d'une première ébauche mettant en avant quelques progrès pour une résolution incomplète dédiée. Cependant, les perspectives d'évolution restent nombreuses. Les informations dégagées sur le profil de résolution de clauses n'ont pas été exploitées ici et les méthodes de résolution (et heuristiques) peuvent encore être approfondies. Ces idées font d'ailleurs état d'un travail en cours puisqu'un solveur hybride complet/incomplet reposant sur l'expertise acquise dans cette thèse est en cours de développement.

CHAPITRE V

Conclusion

Conclusion

Les travaux et les résultats présentés dans cette thèse permettent de montrer l'étendue des ouvertures offertes par la cryptanalyse logique. Cette discipline, encore jeune, voit de plus en plus de travaux émerger, pressentant des chemins prometteurs. Cette thèse a tenté d'exploiter le potentiel de la logique propositionnelle pour un apport en cryptanalyse.

Quand il s'agit de s'intéresser aux attaques en pratique, la littérature en cryptanalyse logique est principalement portée sur la recherche d'antécédents pour les fonctions de hachage ou la découverte de clés secrètes dans les algorithmes de chiffrement symétriques et asymétriques. Le principe consiste à modéliser le processus de hachage sous la forme d'un problème SAT pour ensuite le résoudre grâce à un solveur dédié. Cette méthodologie en deux phases est souvent assez obscure pour les cryptanalystes et réside principalement en l'utilisation successive d'outils automatiques pour obtenir une certaine mesure de résistance contre des attaques dans un cadre applicatif.

Si on porte l'attention sur la modélisation de problèmes cryptographiques, les outils de la littérature montrent qu'il devient en effet aisé de représenter un algorithme cryptographique dans un format exploitable par un solveur SAT. Ceci étant, divers travaux ont aussi montré qu'un problème modélisé « à la main » permet d'avoir une meilleure analyse finale de la robustesse de la fonction représentée. Par ailleurs, il est aujourd'hui difficile pour la communauté SAT d'ordonner qualitativement deux formules CNF pour un même problème structuré. Par conséquent, les choix effectués pour générer les formules proposées dans cette thèse peuvent toujours être sujet à discussions. Ils ont néanmoins permis de contribuer à l'amélioration du nombre d'étapes inversées pour MD5 et semblent donc intéressants pour une réutilisation dans d'autres travaux. Ajouté à cela, les méthodes de simplification mises en place dans un pré-traitement spécifique ont permis de contracter les formules générées tout en révélant des relations remarquables entre certaines variables.

Au delà de ces faits, c'est bien l'action d'envisager un algorithme cryptographique sous une formule SAT qui est le plus important car il permet de reconsidérer le problème sous un angle différent. Ce style de modélisation offre un détail très précis puisque chaque bit du processus de hachage est représenté par une variable. Dans ce contexte, il est intéressant de remarquer que le problème SAT n'est plus seulement un outil de modélisation et de résolution, mais aussi un cadre pour analyser et étudier différents problèmes. C'est ce que nous avons notamment réalisé lorsque nous nous sommes penchés sur la représentation de traces d'exécution que ce soit directement dans le générateur de formules ou bien par le biais d'une interface graphique. C'est aussi le cas quand nous avons cherché à obtenir des probabilités sur les variables du processus de hachage pour extraire des comportements

singuliers. Et c'est encore le cas quand les caractéristiques d'une collision sur MD5 sont représentées sous forme d'image. En ce sens, l'extension de la cryptanalyse logique à des fins d'observations peut permettre de découvrir plusieurs informations pertinentes sur la structure et le sens logique des problèmes considérés.

Cette approche novatrice est à la croisée de la cryptanalyse et de la satisfaisabilité, et la conjonction de ces deux univers est pleinement illustrée dans deux exemples concrets. Premièrement, lorsque les probabilités sur les variables montrent une certaine faiblesse sur la constante de ronde. Il est remarquable de voir que c'est la description du problème en un formalisme logique qui permet d'avoir une image probabiliste moyenne d'un processus de hachage. Sous l'aspect cryptanalyse, cette image n'a de sens que si elle permet de débusquer des comportements particuliers pouvant identifier de possibles failles. On retrouve un intérêt à mêler ces deux vues dans les travaux concernant la mise en œuvre de l'attaque du chemin différentiel de Stevens dans une formule SAT. Une fois encore, le contexte logique permet d'apporter un cadre parfait pour mettre ce type d'attaques en pratique. Les conditions sur les bits sont en effet facilement transposables puisqu'il s'agit d'ajouter des clauses binaires et d'affecter les bonnes variables aux bonnes valeurs. La résolution du problème devient aussi plus confortable car la formule est simplement donnée à un solveur SAT.

Ce solveur est le fruit d'une communauté très active qui s'attèle à développer des moteurs d'inférence compétitifs. Ils sont adaptés pour la résolution de problèmes aux intérêts théoriques, académiques ou industriels. C'est dans cette dernière catégorie que se classent les formules générées dans cette thèse, notamment pour leurs applications en cryptographie. Au cours de nos recherches, nous avons essayé de dégager des informations pertinentes pour dédier un solveur à nos problèmes bien spécifiques. Ainsi, nous avons extrait des données sur les variables et sur les clauses qui ont permis d'aboutir sur la reconnaissance de profils d'affectations. Ces informations sont intéressantes pour spécifier un solveur afin d'améliorer sa capacité à résoudre les problèmes cryptographiques étudiés dans cette thèse. Retoucher l'heuristique est par exemple une façon de perfectionner la résolution et la piste empruntée dans nos travaux semble d'intérêt. Il faut faire en sorte que la boîte noire des cryptanalystes soit plus performante pour donner une réponse plus précise au problème posé.

Perspectives

L'ensemble des travaux présentés dans cette thèse pourrait paraître n'être qu'une infime contribution dans un océan de possibilités, tant les perspectives sont prometteuses. Les tâches à accomplir sont par conséquent nombreuses. Explorer une mise en pratique d'attaques similaires sur d'autres fonctions de hachage cryptographiques paraît déjà être un premier pas pour pouvoir comparer ces fonctions du strict point de vue de la cryptanalyse logique. Le nouveau standard SHA-3 a par exemple été attaqué par cette approche mais il est possible que le résultat puisse être amélioré. Des travaux sont en cours sur ce sujet. Quid de SHA-2, toujours en vigueur et considéré comme cryptographiquement encore très sûr. Plus largement encore, nous avons vu qu'en terme de modélisation ce sont certaines opérations de base et le nombre d'étapes d'une fonction de hachage qui influen-

çaient le nombre de variables et le nombre de clauses des formules générées. Dans cette optique, nous comptons porter un regard sur l'influence de la cryptanalyse logique relativement aux fonctions cryptographiques à bas coût ou sur des algorithmes de chiffrement par flots, tels que RC4, où notre étude sur les biais statistiques peut se révéler complémentaire aux attaques existantes.

Par ailleurs, un des travaux les plus prometteurs semble être le développement d'un solveur entièrement dédié à la résolution de problèmes cryptographiques. En ce sens, les profils de résolution sont un premier pas vers l'approfondissement d'une heuristique. Nous comptons appuyer sur ce point et mettre en place une méthode hybride mêlant méthode complète et incomplète. L'idée est la suivante. La méthode incomplète aurait pour but de satisfaire facilement un maximum de clauses, c'est-à-dire obtenir une borne minimale de clauses insatisfaites convenable. Nous considérons ensuite la formule composée de toutes ces clauses satisfaites auxquelles nous ajoutons une des clauses insatisfaites. Puis, nous appliquons une méthode complète pour résoudre cette nouvelle formule. Quand une solution est trouvée, elle est implantée dans la formule entière. Cette façon de faire sera appliquée par couches successives jusqu'à trouver une solution pour une instance complète.

Enfin, d'une manière beaucoup plus générale, la cryptanalyse logique est encore récente et se situe au croisement de deux domaines différents mais complémentaires. Il n'est pas impossible que les champs de recherche intéressés en cryptanalyse soient élargis. Nous pensons notamment aux attaques par canaux auxiliaires qui pourraient être sensibles à la modélisation sous forme logique des algorithmes de chiffrement. Dans cette voie, un des enjeux pour la cryptanalyse logique est de réussir à parler un langage commun pour conjuguer au mieux l'expertise de la satisfaisabilité et celle de la cryptanalyse, au sens le plus large.

Table des figures

Liste des tableaux

Bibliographie

[Alfred J. Menezes 1996] Scott A. Vanston Alfred J. Menezes Paul C. van Oorschot. Handbook of applied cryptography. Discrete Mathematics and Its Applications. CRC Press Inc, oct 1996. (Cité en pages 17 et 21.)

[Aspvall 1979] Bengt Aspvall, Michael F. Plass et Robert Endre Tarjan. *A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas*. Information Processing Letters, vol. 8, no. 3, pages 121–123, Mars 1979. (Cité en page 31.)

[Audemard 2009] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure et Lakhdar Sais. *Learning in Local Search*. In ICTAI, pages 417–424, 2009. (Cité en page 39.)

[Audemard 2010] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure et Lakhdar Saïs. *SatHYS : Sat Hybrid Solver*. Rapport technique, Proceedings of SAT Race 2010 : Solver and Benchmarks Descriptions, Edinburgh, Scotland, UK, jul 2010. Proceedings of SAT Race 2010 : Solver and Benchmarks Descriptions. (Cité en page 39.)

[Audemard 2013] Gilles Audemard, Jean-Marie Lagniez et Laurent Simon. *Improving Glucose for Incremental SAT Solving with Assumptions : Application to MUS Extraction*. In SAT, pages 309–317, 2013. (Cité en page 34.)

[Bacchus 2003] Fahiem Bacchus et Jonathan Winter. *Effective preprocessing with hyper-resolution and equality reduction*. In SAT, 2003. (Cité en pages 42 et 65.)

[Bailleux 1999] Olivier Bailleux et Pierre Marquis. *DISTANCE-SAT : Complexity and Algorithms*. In AAAI/IAAI, pages 642–647, 1999. (Cité en page 112.)

[Balint 2009] Adrian Balint, Michael Henn et Oliver Gableske. *A Novel Approach to Combine a SLS- and a DPLL-Solver for the Satisfiability Problem*. In SAT, pages 284–297, 2009. (Cité en page 39.)

[Balint 2013] Adrian Balint, Anton Belov, Marijn Heule et Matti Järvisalo. *Proceedings of the SAT competition 2013*, 2013. (Cité en page 41.)

[Barrington 2000] David Mix Barrington et Alexis Maciel. *Lecture 7 :NP-Complete Problems*. In AS/PCMI Summer Session 2000, Clay Mathematics Undergraduate Program, Basic Course on Computational Complexity, july 2000. (Cité en page 40.)

[Béjar 2010] Ramón Béjar, Cèsar Fernández et Francesc Guitart. *Encoding Basic Arithmetic Operations for SAT-Solvers*. In CCIA, pages 239–248, 2010. (Cité en page 19.)

[Bellare 1996] Mihir Bellare, Ran Canetti et Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. In Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag. (Cité en pages 12 et 20.)

[Bertoni 2011] Guido Bertoni, Joan Daemen, Michael Peeters et Gilles Van Assche. *The Keccak reference*, jan 2011. (Cité en page 26.)

[Bettale 2011] Luk Bettale. *Cryptanalyse algébrique : outils et applications.* In Thèse, 2011. (Cité en page 24.)

[Biere 2006] Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala et Viktor Schuppan. *Linear Encodings of Bounded LTL Model Checking*. Logical Methods in Computer Science, 2006. (Cité en pages 11, 28 et 34.)

[Biere 2009] Armin Biere, Marijn Heule, Hans van Maaren et Toby Walsh. Handbook of satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. (Cité en page 28.)

[Biere 2010] Armin Biere. *Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010*. In Tech. Rep. 10/1, FMV Reports Series, Johannes Kepler University, Altenbergerstr. Linz, Austria, pages 244–257, 2010. (Cité en pages 34 et 100.)

[Biham 1990] Eli Biham et Adi Shamir. *Differential Cryptanalysis of DES-like Cryptosystems*. In CRYPTO, pages 2–21, 1990. (Cité en pages 11 et 13.)

[Boer 1991] Bert Den Boer et Antoon Bosselaers. *An Attack on the Last Two Rounds of MD4*. In CRYPTO, pages 194–203, 1991. (Cité en page 23.)

[Boole 1847] George Boole. The mathematical analysis of logic : being an essay towards a calculus of deductive reasoning. Macmillan, Barclay, & Macmillan, Cambridge, 1847. (Cité en page 28.)

[Cai 2011] Shaowei Cai et Kaile Su. *Local Search with Configuration Checking for SAT*. In ICTAI, pages 59–66, 2011. (Cité en page 36.)

[Cannière 2008] Christophe De Cannière et Christian Rechberger. *Preimages for Reduced SHA-0 and SHA-1*. In CRYPTO, pages 179–202, 2008. (Cité en page 26.)

[Chabaud 1998] Florent Chabaud et Antoine Joux. *Differential Collisions in SHA-0*. In CRYPTO, pages 56–71, 1998. (Cité en page 26.)

[Cook 1971] Stephen A. Cook. *The Complexity of Theorem Proving Procedures*. In 3^{rd} ACM Symp. on Theory of Computing, Ohio, pages 151–158, 1971. (Cité en page 28.)

[Courtois 2008a] Nicolas Courtois, Gregory V. Bard et David Wagner. *Algebraic and Slide Attacks on KeeLoq*. In FSE, pages 97–115, 2008. (Cité en page 43.)

[Courtois 2008b] Nicolas Courtois, Karsten Nohl et Sean O'Neil. *Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards*. IACR Cryptology ePrint Archive, vol. 2008, page 166, 2008. (Cité en page 45.)

[Courtois 2011] Nicolas T. Courtois. *Algebraic Complexity Reduction and Cryptanalysis of GOST*, 2011. (Cité en page 45.)

[Crawford 1996] James M. Crawford. *Solving Satisfiability Problems Using a Combination of Systematic and Local Search*. In Rutgers University, 1996. (Cité en page 39.)

[Croes 1958] Georges A. Croes. *A method for solving traveling salesman problems*. Operations Research, vol. 5, pages 791—-812, 1958. (Cité en page 117.)

[Daemen 1998] Joan Daemen et Vincent Rijmen. *The Block Cipher Rijndael*. In CARDIS, pages 277–284, 1998. (Cité en page 17.)

[Damgård 1989] Ivan Damgård. *A Design Principle for Hash Functions*. In CRYPTO, pages 416–427, 1989. (Cité en page 22.)

[Davis 1960] Martin Davis et Hilary Putnam. *A Computing Procedure for Quantification Theory*. Journal Association for Computing Machine, no. 7, pages 201–215, 1960. (Cité en page 32.)

[Davis 1962] Martin Davis, George Logemann et Donald Loveland. *A Machine Program for Theorem-Proving*. Journal Association for Computing Machine, vol. 5, pages 394–397, 1962. (Cité en page 32.)

[De 2007] Debapratim De, Abishek Kumarasubramanian et Ramarathnam Venkatesan. *Inversion Attacks on Secure Hash Functions Using SAT Solvers*. In SAT, pages 377–382, 2007. (Cité en pages 45, 96 et 99.)

[Dequen 2001] Gilles Dequen et Olivier Dubois. *A Backbone Search Heuristic for Efficient Solving of Hard 3-SAT Formulae*. In Proc. of the 17th Internat. Joint Conf. on Artificial Intelligence, pages 248–253, Seattle, aug 2001. (Cité en page 33.)

[Dequen 2006] Gilles Dequen et Olivier Dubois. *An Efficient Approach to Solving Random k-SAT Problems*. J. Autom. Reasoning, vol. 37, no. 4, pages 261–276, 2006. (Cité en page 33.)

[Diffie 1976] Whitfield Diffie et Martin E. Hellman. *New directions in cryptography*. IEEE Transactions on Information Theory, vol. 22, no. 6, pages 644–654, 1976. (Cité en pages 12 et 17.)

[Dobbertin 1996] Hans Dobbertin. *Cryptanalysis of MD4*. In FSE, pages 53–69, 1996. (Cité en pages 23, 97, 99, 100, 102, 123 et 124.)

[Edelkamp 2012] Stefan Edelkamp et Stefan Schrödl. Heuristic search - theory and applications. Academic Press, 2012. (Cité en page 33.)

[Eén 2005] Niklas Eén et Armin Biere. *Effective Preprocessing in SAT Through Variable and Clause Elimination*. In SAT, pages 61–75, 2005. (Cité en pages 42 et 65.)

[Eibach 2008] Tobias Eibach, Enrico Pilz et Gunnar Völkel. *Attacking Bivium Using SAT Solvers*. In SAT, pages 63–76, 2008. (Cité en page 45.)

[Faizullin 2009] Rashit Tagirovich Faizullin, Ivan Gennad'evich Khnykin et V. I. Dylkeyt. *The SAT solving method as applied to cryptographic analysis of asymmetric ciphers*. CoRR, vol. abs/0907.1755, 2009. (Cité en page 43.)

[Fang 2004] Hai Fang et Wheeler Ruml. *Complete Local Search for Propositional Satisfiability*. In AAAI, pages 161–166, 2004. (Cité en page 39.)

[Ferris 2004] B. Ferris et Jon Froehlich. *WalkSAT as an Informed Heuristic to DPLL in SAT Solving*. Department of Computer Science, University of Washington, Seattle, 2004 2004. (Cité en page 39.)

[Fiorini 2003] Claudia Fiorini, Enrico Martinelli et Fabio Massacci. *How to fake an RSA signature by encoding modular root finding as a SAT problem*. Discrete Applied Mathematics, vol. 130, no. 2, pages 101–127, 2003. (Cité en page 43.)

[Fluhrer 2001] Scott R. Fluhrer, Itsik Mantin et Adi Shamir. *Weaknesses in the Key Scheduling Algorithm of RC4*. In Selected Areas in Cryptography, pages 1–24, 2001. (Cité en page 16.)

[Freeman 1995] Jon W. Freeman. *Improvements to Propositional Satisfiability*. Thesis, may 1995. (Cité en page 33.)

[Gamal 1985] Taher El Gamal. *A public key cryptosystem and a signature scheme based on discrete logarithms*. In Proceedings of CRYPTO 84 on Advances in cryptology, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc. (Cité en page 19.)

[Genkin 2013] Daniel Genkin, Adi Shamir et Eran Tromer. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*. In Cryptology ePrint Archive, Report 013/857, 2013. (Cité en page 19.)

[Ghernaouti 2013] Solange Ghernaouti. Sécurité informatique et réseaux. Dunod, 4ième édition, août 2013. (Cité en page 18.)

[Güneysu 2008] Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar et Andy Rupp. *Cryptanalysis with COPACOBANA*. IEEE Trans. Computers, vol. 57, no. 11, pages 1498–1513, 2008. (Cité en page 16.)

[Habet 2002] Djamal Habet, Chu Min Li, Laure Devendeville et Michel Vasquez. *A Hybrid Approach for SAT*. In CP, pages 172–184, 2002. (Cité en page 39.)

[Hankerson 2003] Darrel Hankerson, Alfred J. Menezes et Scott Vanstone. Guide to elliptic curve cryptography. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. (Cité en page 19.)

[Homsirikamol 2012] Ekawat Homsirikamol, Pawel Morawiecki, Marcin Rogawski et Marian Srebrny. *Security Margin Evaluation of SHA-3 Contest Finalists through SAT-Based Attacks*. In CISIM, pages 56–67, 2012. (Cité en page 45.)

[Hoos 2004] Holger H. Hoos et Thomas Stützle. Stochastic local search : Foundations and applications. Morgan Kaufmann / Elsevier, 2004. (Cité en page 36.)

[Joseph Bebel 2013] Henry Yuen Joseph Bebel. *Hard SAT instances based on factoring*. In Proceedings of the SAT competition 2013, page 102, 2013. (Cité en page 43.)

[Jovanovic 2005] Dejan Jovanovic et Predrag Janicic. *Logical Analysis of Hash Functions*. In FroCoS, pages 200–215. "Nauka", Leningrad. Otdel, 2005. (Cité en page 43.)

[Junod 2001] Pascal Junod. *On the Complexity of Matsui's Attack*. In Selected Areas in Cryptography, pages 199–211, 2001. (Cité en page 17.)

[Kamal 2010] Abdel Alim Kamal et Amr M. Youssef. *Applications of SAT Solvers to AES key Recovery from Decayed Key Schedule Images*. IACR Cryptology ePrint Archive, vol. 2010, page 324, 2010. (Cité en page 43.)

[Kautz 1996] Henry A. Kautz et Bart Selman. *Pushing the Envelope : Planning, Propositional Logic and Stochastic Search*. In Proc. of 30th national AI and 8th IAAI, 1996. (Cité en pages 11, 28 et 34.)

[Klein 2008] Andreas Klein. *Attacks on the RC4 stream cipher*. Des. Codes Cryptography, vol. 48, no. 3, pages 269–286, 2008. (Cité en page 16.)

[Kleinjung 2010] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev et Paul Zimmermann. *Factorization of a 768-Bit RSA Modulus*. In CRYPTO, pages 333–350, 2010. (Cité en page 19.)

[Knuth 2011] Donald E. Knuth. The art of computer programming. volume 4a : Combinatorial algorithms part 1. Addison-Wesley Professional, 1 édition, 2011. (Cité en page 28.)

[Kullman 2002] Oliver Kullman. *Towards an adaptive density based branching rule for SAT solvers, using a database for mixed random conjunctive normal forms built upon the Advanced Encryption Standard (AES)*. In SAT, Cincinnati, Ohio, USA, May 2002. (Cité en pages 11 et 34.)

[Legendre 2012a] Florian Legendre, Gilles Dequen et Michaël Krajecki. *Encoding Hash Functions as a SAT Problem*. In ICTAI, pages 916–921, 2012. (Cité en pages 45, 72 et 96.)

[Legendre 2012b] Florian Legendre, Gilles Dequen et Michaël Krajecki. *Inverting Thanks to SAT Solving - An Application on Reduced-step MD**. In SECRYPT, pages 339–344, 2012. (Cité en pages 45, 96 et 101.)

[Legendre 2013] Florian Legendre, Gilles Dequen et Michaël Krajecki. *From a Logical Approach to Internal States of Hash Functions - How SAT Problem Can Help to Understand SHA-* and MD**. In SECRYPT, 2013. (Cité en page 93.)

[Letombe 2012] Florian Letombe et João Marques-Silva. *Hybrid Incremental Algorithms for Boolean Satisfiability*. International Journal on Artificial Intelligence Tools, vol. 21, no. 6, 2012. (Cité en page 39.)

[Leurent 2008] Gaëtan Leurent. *MD4 is Not One-Way*. In FSE, pages 412–428, 2008. (Cité en page 23.)

[Li 1997] Chu Min Li et Anbulagan. *Heuristics Based on Unit Propagation for Satisfiability Problems*. In Proc. of the 15th Internat. Joint Conf. on Artificial Intelligence, pages 366–371, Nagoya (Japan), August 1997. IJCAI. (Cité en page 33.)

[Li 2012] Chu Min Li et Yu Li. *Satisfying versus Falsifying in Local Search for Satisfiability - (Poster Presentation)*. In SAT, pages 477–478, 2012. (Cité en page 36.)

[Liao 2013] Xiaojuan Liao, Hui Zhang, Miyuki Koshimura, Hiroshi Fujita et Ryuzo Hasegawa. *Using MaxSAT to Correct Errors in AES Key Schedule Images*. In ICTAI, pages 284–291, 2013. (Cité en page 43.)

[Manthey 2012] Norbert Manthey. *Coprocessor 2.0 - A Flexible CNF Simplifier - (Tool Presentation)*. In SAT, pages 436–441, 2012. (Cité en pages 71, 72 et 125.)

[Manuel 2011] Stéphane Manuel. *Classification and generation of disturbance vectors for collision attacks against SHA-1*. Des. Codes Cryptography, vol. 59, no. 1-3, pages 247–263, 2011. (Cité en page 26.)

[Massacci 1999] Fabio Massacci. *Using Walk-SAT and Rel-Sat for Cryptographic Key Search*. In IJCAI, pages 290–295, 1999. (Cité en page 43.)

[Massacci 2000] Fabio Massacci et Laura Marraro. *Logical Cryptanalysis as a SAT Problem*. J.Autom.Reasoning, pages 165–203, 2000. (Cité en pages 11, 34, 42 et 43.)

[Matsui 1992] Mitsuru Matsui et Atsuhiro Yamagishi. *A New Method for Known Plaintext Attack of FEAL Cipher*. In EUROCRYPT, pages 81–91, 1992. (Cité en pages 11 et 14.)

[Matsui 1994] Mitsuru Matsui. *The First Experimental Cryptanalysis of the Data Encryption Standard*. In CRYPTO, pages 1–11, 1994. (Cité en pages 14 et 17.)

[Matsumoto 1988] Tsutomu Matsumoto et Hideki Imai. *Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption*. In EUROCRYPT, pages 419–453, 1988. (Cité en page 14.)

[McDonald 2007] Cameron McDonald, Chris Charnes et Josef Pieprzyk. *An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem*. IACR Cryptology ePrint Archive, vol. 2007, page 129, 2007. (Cité en page 45.)

[McDonald 2009] Cameron McDonald, Philip Hawkes et Josef Pieprzyk. *Differential Path for SHA-1 with complexity $O(2^{52})$*. IACR Cryptology ePrint Archive, vol. 2009, page 259, 2009. (Cité en page 43.)

[Merkle 1989] Ralph Merkle. *One Way Hash Functions and DES*. In CRYPTO, pages 428–446, 1989. (Cité en page 22.)

[Mironov 2006] Ilya Mironov et Lintao Zhang. *Applications of SAT Solvers to Cryptanalysis of Hash Functions*. In SAT, pages 102–115, 2006. (Cité en page 43.)

[Mohamed 2011] Mohamed Saied Emam Mohamed, Stanislav Bulygin et Johannes Buchmann. *Using SAT Solving to Improve Differential Fault Analysis of Trivium*. In ISA, pages 62–71, 2011. (Cité en page 45.)

[Morawiecki 2013] Pawel Morawiecki et Marian Srebrny. *A SAT-based preimage analysis of reduced Keccak hash functions*. Inf. Process. Lett., vol. 113, no. 10-11, pages 392–397, 2013. (Cité en page 27.)

[Moskewicz 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang et Sharad Malik. *Chaff : Engineering an Efficient SAT Solver*. In DAC, pages 530–535, 2001. (Cité en pages 33 et 34.)

[NIST 2012a] NIST. *NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition*, oct 2012. (Cité en page 26.)

[NIST 2012b] NIST. *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, jan 2012. (Cité en page 17.)

[Nossum 2012] Vegard Nossum. *SAT-based preimage attacks on SHA-1*. Master's thesis, 2012. (Cité en pages 45 et 96.)

[Nossum 2013] Vegard Nossum. *Instance generator for encoding preimage, second-preimage, and collision attacks on SHA-1*. In Proceedings of the SAT competition 2013, pages 119–120, 2013. (Cité en page 43.)

[Patarin 1995] Jacques Patarin. *Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88*. In CRYPTO, pages 248–261, 1995. (Cité en pages 11 et 14.)

[Patsakis 2013] Constantinos Patsakis. *RSA private key reconstruction from random bits using SAT solvers*. IACR Cryptology ePrint Archive, vol. 2013, page 26, 2013. (Cité en page 45.)

[Piette 2008] Cédric Piette, Youssef Hamadi et Lakhdar Saïs. *Vivifying propositional clausal formulae*. In 18th European Conference on Artificial Intelligence(ECAI'08), pages 525–529, 2008. (Cité en page 42.)

[Poli 2005] Alain Poli et Philippe Guillot. Algèbre et protection de l'information. Hermes, mars 2005. (Cité en page 21.)

[Potlapally 2007] Nachiketh R. Potlapally, Anand Raghunathan, Srivaths Ravi, Niraj K. Jha et Ruby B. Lee. *Aiding side-channel attacks on cryptographic software with satisfiability-based analysis*. IEEE Trans. VLSI Syst., vol. 15, no. 4, pages 465–470, 2007. (Cité en pages 11 et 34.)

[Rivest 1978] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Commun. ACM, vol. 21, no. 2, pages 120–126, 1978. (Cité en pages 12 et 18.)

[Rivest 1990a] Ronald L. Rivest. *The MD4 Message Digest Algorithm*. In CRYPTO, pages 303–311, 1990. (Cité en page 22.)

[Rivest 1990b] Ronald L. Rivest. *The MD4 Message Digest Algorithm*, 1990. (Cité en page 23.)

[Rivest 1991] Ronald L. Rivest. *The MD5 Message Digest Algorithm*, 1991. (Cité en page 24.)

[Rivest 2008] Ronald L. Rivest. *The MD6 hash function - A proposal to NIST for SHA-3*, oct 2008. (Cité en page 45.)

[Robinson 1965] John Alan Robinson. *A Machine-Oriented Logic Based on the Resolution Principle*. J. ACM, vol. 12, no. 1, pages 23–41, 1965. (Cité en page 67.)

[Samuilovich 1968] Tseitin Grigorii Samuilovich. *On the Complexity of Derivation in Propositional Calculus*. In Studies in constructive mathematics and mathematical logic. Part II, pages 234–259, 1968. (Cité en page 49.)

[Sasaki 2007] Yu Sasaki, Lei Wang, Kazuo Ohta et Noboru Kunihiro. *New Message Difference for MD4*. In FSE, pages 329–348, 2007. (Cité en page 23.)

[Saïs 2008] Lakhdar Saïs. Problème SAT : progrès et défis. Hermes, apr 2008. (Cité en page 28.)

[Schneier 2001] Bruce Schneier. Applied crytography - second edition. Vuibert, London, 2nd édition, 2001. (Cité en pages 13, 18, 19 et 20.)

[Selman 1993] Bart Selman et Henry A. Kautz. *An Empirical Study of Greedy Local Search for Satisfiability Testing*. In AAAI, pages 46–51, 1993. (Cité en page 36.)

[Shanks 1971] Daniel Shanks. *Class number, a theory of factorization and genera*. In Proc Symp Pure Math, pages 415–440. AMS, Providence, R.I, 1971. (Cité en page 19.)

[Shannon 1949] Claude E. Shannon. Communication theory of secrecy systems, volume 28-4. Bell System Technical Journal, 1949. (Cité en pages 15 et 16.)

[Silva 1999] João P. Marques Silva et Karem A. Sakallah. *GRASP : A Search Algorithm for Propositional Satisfiability*. IEEE Trans. Computers, vol. 48, no. 5, pages 506–521, 1999. (Cité en page 34.)

[Silva 2000] João P. Marques Silva et Karem A. Sakallah. *Invited Tutorial : Boolean Satisfiability Algorithms and Applications in Electronic Design Automation*. In CAV, page 3, 2000. (Cité en page 34.)

[Singh 1999] Simon Singh. Histoire des codes secrets - de l'Égypte des pharaons à l'ordinateur quantique. JC Lattès, 1999. traduit par Catherine Coqueret. (Cité en page 15.)

[Soos 2009] Mate Soos, Karsten Nohl et Claude Castelluccia. *Extending SAT Solvers to Cryptographic Problems*. In SAT, pages 244–257, 2009. (Cité en pages 35 et 45.)

[Soos 2010a] Mate Soos. *CryptoMiniSat 2 : a SAT solver for cryptographic problems*. In http ://www.msoos.org/cryptominisat2/, 2010. (Cité en page 35.)

[Soos 2010b] Mate Soos. *Enhanced Gaussian Elimination in DPLL-based SAT Solvers*. In Daniel Le Berre, editeur, POS-10, volume 8 of *EPiC Series*, pages 2–14, 2010. (Cité en pages 35 et 45.)

[Soos 2013] Mate Soos. *Grain of Salt benchmarks*. In Proceedings of the SAT competition 2013, page 121, 2013. (Cité en page 43.)

[Srebrny 2008] Marian Srebrny, Mateusz Srebrny et Lidia Stepien. *SAT as a Programming Environment for Linear Algebra and Cryptanalysis*. In ISAIM, 2008. (Cité en page 45.)

[Stevens 2012a] Marc Stevens. *Single-block collision attack on MD5*. IACR Cryptology ePrint Archive, vol. 2012, page 40, 2012. (Cité en pages vi, viii, 4, 95, 107, 118, 124 et 143.)

[Stevens 2012b] Marc Stevens, Arjen K. Lenstra et Benne de Weger. *Chosen-prefix collisions for MD5 and applications*. IJACT, vol. 2, no. 4, pages 322–359, 2012. (Cité en page 24.)

[Stevens 2013] Marc Stevens. *New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis*. In EUROCRYPT, pages 245–261, 2013. (Cité en page 26.)

[Sutherland 2007] Andrew V. Sutherland. *Order computations in generic groups*. In PhD thesis, pages 49–54, M.I.T, 2007. (Cité en page 19.)

[Sörensson 2005] Niklas Sörensson et Niklas Een. *MiniSat v1.13 – A SAT Solver with Conflict-Clause Minimization*. In SAT, page Poster, 2005. (Cité en page 34.)

[Tompkins 2011] Dave A. D. Tompkins, Adrian Balint et Holger H. Hoos. *Captain Jack : New Variable Selection Heuristics in Local Search for SAT*. In SAT, pages 302–316, 2011. (Cité en page 36.)

[Turner 2011] Sean Turner et Lily Chen. *MD4 to Historic Status*, 2011. (Cité en page 23.)

[Vergnaud 2012] Damien Vergnaud. Exercices et problèmes cryptographiques. Dunod, mars 2012. (Cité en pages 18 et 20.)

[Von Mises 1964] Richard Von Mises. Mathematical theory of probability and statistics. Academic Press, 1964. (Cité en page 21.)

[Wang 2005] Xiaoyun Wang et Hongbo Yu. *How to Break MD5 and Other Hash Functions*. In EUROCRYPT, pages 19–35, 2005. (Cité en pages 23 et 24.)

[Xie 2010] Tao Xie et Dengguo Feng. *Construct MD5 Collisions Using Just A Single Block Of Message*. IACR Cryptology ePrint Archive, vol. 2010, page 643, 2010. (Cité en pages 102, 103 et 104.)

[Zhang 2001] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz et Sharad Malik. *Efficient Conflict Driven Learning in a Boolean Satisfiability Solver*. In ICCAD, pages 11–16, 2001. (Cité en page 34.)

[Zémor 2000] Gilles Zémor. Cours de cryptographie. Cassini, 2000. (Cité en page 18.)

Annexe 1 : Interface de visualisation

L'interface de visualisation

Un des travaux réalisé a été le développement d'une interface de visualisation pour la résolution d'une instance MD5. Cet annexe donne un détail des possibilités offertes par cet outil.

Principe de fonctionnement

Le fonctionnement repose sur le principe d'une application client/serveur, avec le solveur SAT qui joue le rôle du serveur et l'interface de visualisation le rôle de client. Le serveur garde les données nécessaires pour résoudre le problème et propose les moyens techniques pour le traitement de ces données. De plus, il a un rôle d'écoute auprès de l'interface et répond aux requêtes envoyées par celle-ci.

De l'autre coté, l'interface reçoit les données de la part du solveur et a pour principale mission de les afficher de manière à ce qu'elles soient lisibles pour l'utilisateur. Cet utilisateur pourra ainsi interagir avec l'interface en effectuant des requêtes pour ajouter, modifier, supprimer des données et plus généralement manipuler la formule SAT. L'interface redirige ensuite ces actions au solveur qui traite les demandes de l'utilisateur. Le principe de fonctionnement de l'interface est représentée sur la *figure V.1*.

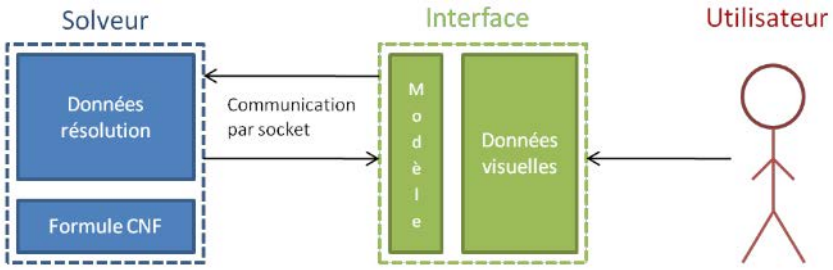

FIGURE V.1 – Principe de fonctionnement de l'interface de visualisation

Les différentes options de l'interface

L'interaction qui est créée entre l'utilisateur et l'interface passe en réalité par un panel de choix disponibles. Ces options peuvent être découpées en deux sous-ensembles suivant leurs rôles au niveau de l'interaction avec l'utilisateur, en considérant qu'il y a des options *offline* et des options *online*.

Les options *offline* Parmi les options *offline*, se trouve tout ce qui attrait au chargement de données. En effet, pour commencer, il est possible d'ouvrir et de charger une formule et/ou une solution partielle à la formule. Il est possible aussi de sauvegarder une solution courante afin de la réutiliser plus tard. Ceci permet notamment de reprendre des travaux en cours mais aussi la comparaison avec différents choix d'affectations. Ajouté à cela, des options de zoom sont proposées pour donner l'accès à une vision globale de la formule ou au contraire à une vision plus rapprochée.

Enfin, une dernière option est la notion de profil utilisateur, qui permet à un utilisateur d'importer et d'exporter l'ensemble des fonctionnalités qu'il aura pris soin de paramétrer.

Les options *online* Les options *online* vont principalement toucher au domaine de l'interaction. Pour que celle-ci soit réalisable il faut en effet des fonctions qui permettent à l'utilisateur d'agir directement sur la formule et sur sa résolution. Ainsi, ont été mises en place les diverses options suivantes :

- Possibilité de mettre le solveur en mode automatique ou en mode manuel. Ceci signifie que chaque affectation demandée par l'utilisateur se fera soit de façon autonome dès qu'une action est réalisée, soit après une validation de la part de l'utilisateur et ce, même pour plusieurs affectations simultanées.
- Changer l'état d'une variable à une des valeurs de l'ensemble $\{0, 1, \text{non déterminé}\}$
- Remettre à l'état par défaut toutes les variables de l'instance
- Revenir au coup d'avant, annuler la dernière affectation si nous sommes en mode manuel
- Bloquer/Débloquer des variables pour forcer par exemple une empreinte ou n'importe quelle variable que nous jugerions bon de garder dans son état courant.

Description de l'interface Cette interface propose un affichage spécialement conçu pour la résolution d'une formule MD5 mais peut être adaptée à une autre fonction de hachage moyennant un léger développement. Sur cette interface, les variables sont colorées en fonction de leur état courant : *Vert* signifie que la variable est fixée à 1, *rouge* à 0 et *gris* est un état non déterminé. Elles sont rangées par catégorie suivant ce qu'elles représentent dans le processus de hachage. Il est aussi possible d'avoir accès à l'état des variables représentant le message en entrée ainsi que l'empreinte. Enfin, un aperçu des clauses satisfaites, insatisfaites ou dans un état non déterminé est donné sur la droite de l'interface. La figure V.2 illustre l'ensemble de ce qui vient d'être décrit.

Utilisation de l'interface L'utilisation de l'interface en pratique nécessite le lancement du serveur puis du client en java. Une fois connecté, un profil par défaut est choisi et l'interface affiche une instance totalement vierge. L'utilisateur peut soit choisir de charger une solution préalablement sauvegardée, soit commencer de nouveaux traitements sur une nouvelle formule. Une fois cela fait, l'utilisateur se retrouve en face d'une interface avec laquelle il peut totalement interagir. En cliquant sur des variables, il peut changer leur état et visualiser directement ses conséquences sur les autres variables. L'intérêt premier

FIGURE V.2 – L'interface de visualisation, pour une formule MD5 quelconque avec quelques variables affectées à vraies (en vert) et à faux (en rouge).

va être de fournir à l'utilisateur un rendu en direct de ses choix, des propagations qui en résultent ainsi que l'état dans lequel sera chaque clause. Ainsi, l'utilisateur peut apporter des modifications à une formule pour tenter d'avancer dans la résolution d'un problème mais aussi pour essayer de comprendre quand cela coince (clauses contredites), totalement indépendamment de l'heuristique initiée dans le solveur.

Cet outil se veut être capable d'aider les cryptanalystes en automatisant les calculs et la résolution de problèmes cryptographiques. Pour le moment, il n'a pas été publié car même s'il est aujourd'hui fonctionnel, il reste dans une première version avec ces quelques inconvénients, notamment au niveau de la rapidité du traitement des informations qui n'est pas adaptée à une utilisation en temps réel.

Par ailleurs, il existe des améliorations à intégrer comme par exemple un système d'information qui pourrait suggérer les bonnes variables candidates à être modifiées. Pour cela, il faut envisager l'ajout d'infobulles sur les variables qui donneraient à l'utilisateur plus de caractéristiques sur celles-ci, avec notamment l'intégration de statistiques et de probabilités d'être à 0 ou 1. Enfin, l'interface a été réalisée de façon à ce que de nouvelles spécificités techniques puissent être ajoutées avec le temps. C'est de notre expérience d'utilisation que nous verrons le besoin d'ajouter ou non de nouveaux modules pour rendre cette interface plus attractive et plus efficace.

Annexe 2 : Pré-images pour des versions réduites de MD4, MD5 et SHA1

---

1 ronde 15 étapes sur MD4 (31 étapes)

Empreinte fixée :
`0x00000000 0x00000000 0x00000000 0x00000000`
Message retrouvé :
`0x184937d5 0x6348828c 0x65e7547c 0x0201b903 0xba4f5298 0x12edc6df`
`0xbbe4a23e 0xa4c25972 0x5d9019f8 0x40bd880b 0x352f6960 0xbcb22ec4`
`0x43e0debc 0x0a4838d4 0xdf6a3b9f 0xcec88113`

1 ronde 12 étapes sur MD5 (28 étapes)

Empreinte fixée :
`0x01234567 0x89abcdef 0xfedcba98 0x76543210`
Message retrouvé :
`0x0bd86c16 0x6dea158a 0x3fea904c 0x5930a4a1 0xf733709c 0x7e818951`
`0xdc6f481b 0x21f85c42 0x7a6b2051 0x09762af5 0xbf21286b 0xb70fe9bc`
`0xb6e76e81 0x0ba31a2c 0x71512697 0xbc2931af`

1 ronde 3 étapes sur SHA-0 (23 étapes)

Empreinte fixée :
`0x00000000 0x00000000 0x00000000 0x00000000`
Message retrouvé :
`0x4e073784 0x050bf14c 0xa5d325cb 0xf81d3ce2 0x123bd6f4 0xe71fc07f`
`0xf07fda73 0x43b05533 0x704efd95 0xc8178c49 0x02c891de 0x1f7630ae`
`0x4130f392 0x55113db5 0xacc0022f 0x3b8c154d`

1 ronde 3 étapes sur SHA-1 (23 étapes)

Empreinte fixée :
`0x00000000 0x00000000 0x00000000 0x00000000`
Message retrouvé :
`0x35691c1a 0xead7eb26 0xcac76b0e 0x00000000 0x51e43c45 0xaa8bc12a`
`0xdb8fa47c 0x00000000 0x637c1517 0x80abea2e 0x9339f44e 0x00000000`
`0x6367caee 0xbc8920ec 0x1084c8d7 0x45075a9e`sur

Annexe 3 : Conditions suffisantes pour la découverte de collisions dans MD5

t	$q_t[31] \ldots q_t[0]$
-3 – 2	....
3	010.0.00 .0001... 01.1...0 00.000..
4	00000000 00000000 00000000 00000000
5	11101011 01111000 11010001 11011100
6	...1.1.. 1....111 ..1.111. ..1...11
7	0
8	....^0
9	00000000 00000000 000000.0 0!0000+-
10	00000000 00000000 000000.0 0.00000+
11	11111111 11111111 11111101 1.11111+
12	....1..00. ......+-
13	....^.^^ .^^^.... ..1.00+. ...^^^0+
14	010.0.00 .0001.^. 01+10--0 00.0001+
15	000.0.00 .00001+. 00+00--0 001000+-
16	0001010+ 100000+1 00+0-+-0 00100001
17	.0..10.+ 0+000++0 10+10--1 01+0.000
18	.01+1-+1 -+-+-01- --+0-+++ ---010+.
19	.-0++.+1 00+011-1 ++1----- +1--0+0.
20	..-10^0. 1+010.11 011.+100 0011+.+.
21	.^11+.1. .1+-..1+ 11.^0010 0-10.^..
22	..+.-... .-+...1+ ....-... .-.-^.^.
23	....1.0. .+-^..+0 ....-... .+.-...0
24	..^.1.0. .-0....0 ....0... 1+.+...1
25	0.....-. .0-...^. ....1... 1+.+...+
26	......0. .-...... ....1... +1.1....
27	-.....-. ..^....0 ....1... .1.-...^
28	-.....+. .^.....1 ....-... ^..-....
29	0.....-. ....- ....1..0 ...+....
30	-.....-. ....0 ....0... ...+....
31	-.....1. ....1 ....- ...+....
32	......0. ...!.... ....- ....
33	!.....0. ...-.... ....!....
34	+.....0. ...-.... ....! ....
35	+.....0. ....
36	....!.... ....
37	!.....0. ....
38	+.....0. ....
39 – 56	+.....0. ....
57	1.....0. ....
58	0.....0. ....
59 – 64	....

FIGURE V.3 – Conditions suffisantes sur les bits pour la découverte de collisions dans MD5 données dans [Stevens 2012a]

Résumé : La démocratisation des ordinateurs, des téléphones portables et surtout de l'Internet a considérablement révolutionné le monde de la communication. Les besoins en matière de cryptographie sont donc plus nombreux et la nécessité de vérifier la sûreté des algorithmes de chiffrement est vitale. Cette thèse s'intéresse à l'étude d'une nouvelle cryptanalyse, appelée cryptanalyse logique, qui repose sur l'utilisation de la logique propositionnelle - à travers le problème de satisfaisabilité - pour exprimer et résoudre des problèmes cryptographiques. Plus particulièrement, les travaux présentés ici portent sur une certaine catégorie de chiffrements utilisés dans les protocoles d'authentification et d'intégrité de données qu'on appelle fonctions de hachage cryptographiques. Un premier point concerne l'aspect modélisation d'un problème cryptographique en un problème de satisfaisabilité et sa simplification logique. Sont ensuite présentées plusieurs façons pour utiliser cette modélisation fine, dont un raisonnement probabiliste sur les données du problème qui permet, entres autres, d'améliorer les deux principaux points d'une attaque par cryptanalyse logique, à savoir la modélisation et la résolution. Un second point traite des attaques menées en pratique. Dans ce cadre, la recherche de pré-image pour les fonctions de hachage les plus couramment utilisées mènent à repousser les limites de la résistance de ces fonctions à la cryptanalyse logique. À cela s'ajoute plusieurs attaques pour la recherche de collisions dans le cadre de la logique propositionnelle.

Exploitation of propositional logic for solving cryptographic problems

Abstract : Democratization of increasingly high-performance digital technologies and especially the Internet has considerably changed the world of communication. Consequently, needs in cryptography are more and more numerous and the necessity of verifying the security of cipher algorithms is essential. This thesis deals with a new cryptanalysis, called logical cryptanalysis, which is based on the use of logical formalism to express and solve cryptographic problems. More precisely, works presented here focuses on a particular category of ciphers, called cryptographic hash functions, used in authentication and data integrity protocols. The first contribution is the modeling of a cryptographic problem as a SAT problem. For this, we present some rules that lead to describe easily basic operations involved in cipher algorithms. Then, a section is dedicated to logical reasoning in order to simplify the produced SAT formulas and show how satisfiability can help to enrich a knowledge on a studied problem. Furthermore, we also present many points of view to use our smooth modeling to apply a probabilistic reasoning on all the data associated with the generated SAT formulas. This has then allowed to improve both the modeling and the solving of the problem and underlined a weakness about the use of round constants. Second, a section is devoted to practical attacks. Within this framework, we tackled preimages and the collision problem of the most popular cryptographic hash functions.

Mots-clés/Keywords : Cryptanalyse logique, satisfaisabilité, fonction de hachage cryptographique, MD5, SHA-1, SHA-3 / Logical cryptanalysis, satisfiability, cryptographic hash function

---

Adresse de l'unité de recherche : CReSTIC, UFR Sciences Exactes et Naturelles, Moulin de la Housse, BP 1039, 51687 Reims CEDEX 2, FRANCE, Tél. : 03.26.91.33.89