Miroirs, Cubes et Feistel Dissymétriques

Emmanuel Volte

UNIVERSITÉ DE CERGY-PONTOISE

ÉCOLE DOCTORALE EM^2C
ÉCONOMIE, MANAGEMENT, MATHÉMATIQUES, CERGY

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Cergy-Pontoise
Mention : Mathématiques

Présentée et soutenue par

Emmanuel Volte

Miroirs, Cubes et Feistel Dissymétriques

Thèse dirigée par Jacques Patarin et Valérie Nachef

préparée à l'Université de Cergy-Pontoise

soutenue le 28 novembre 2014

Jury :

Rapporteurs :	Bart Preneel	-	KU-Leuven
	Pierre-Alain Fouque	-	Rennes I
Directeur :	Jacques Patarin	-	UVSQ
Co-Directrice :	Valérie Nachef	-	UCP
Examinateurs :	David Pointcheval	-	ENS
	Louis Goubin	-	UVSQ

Table des matières

Table des figures

Liste des tableaux

À Michèle et Dany

Remerciements

Voilà la page la plus difficile à écrire, et pourtant ce sera la plus lue ! Je commence par remercier donc ceux qui prendront la peine de continuer un peu plus leur lecture ...

Pendant ces six années où j'ai retrouvé le statut d'étudiant, j'ai croisé beaucoup de personnes bienveillantes, et ce sera difficile ici de les citer toutes. Je prie donc ceux que j'aurais oublié de me pardonner !

Je remercie en premier ma famille, pour son soutien tout au long de ses longues années ainsi que pour ses encouragements. Il y a eu des nuits courtes où je devais finaliser un article ou terminer la programmation d'un algorithme, il y a eu des vacances où j'attendais les remarques d'un shepherd pour me remettre au travail, il y a eu des moments, des réunions familiales où j'avais le regard absent, car occupé à continuer mentalement un calcul. Je ne pense pas avoir donné ainsi le goût de la recherche à mes enfants. J'espère qu'ils changeront d'avis, qu'un jour ils découvriront le plaisir de chercher. En tout cas ils étaient ravis lorsque je leur racontais mes voyages aux quatre coins du monde !

Mes premières nuits (presque) blanches, je les dois au projet CRUNCH. Joana Treger qui était en fin de thèse à l'époque, a eu la sagesse de refuser de travailler à des heures impossibles et nous nous sommes retrouvés avec Louis Goubin, William Jalby et Mickaël Ivascot pour finaliser dans la nuit le dossier à rendre le lendemain ! Je remercie en tout cas toute l'équipe d'avoir réussi à mener le projet jusqu'au bout.

Je remercie également pour leurs aides et encouragements mes collègues PRAG Jean Delcourt et Alexandre Mizrahi. En particulier, ils m'ont bien soutenu lors des affrontements avec le dragon administratif. Jean m'a également apporté une aide mathématique précieuse sur plusieurs points techniques, notamment sur les équations de Brent. Sur ce même sujet, je remercie Raika Dehy de m'avoir présenté un article d'algèbre bilinéaire qui me paraissait obscur avant son intervention. D'autres collègues et amis Smail Alili et Françoise Piquard m'ont donné de mini-cours de probabilité pour combler mes lacunes dans ce domaine. Je n'ai pas fini de les embêter avec mes questions ! On a toujours un projet en cours sur les calculs de variance ... Je n'oublie pas Patrick Courilleau qui a pris ma place à la responsabilité du Master Enseignement. Avec Patrick, on pratique l'alternance (rien à voir avec la formation en alternance qu'il a mis en place), et je dois dire que la période où il a pris entièrement cette responsabilité, n'a pas été du tout facile à gérer au vu des réformes incessantes. Merci donc pour toutes ces réunions que je n'ai pas eues à faire !

À propos de combats administratifs, un remerciement particulier au directeur du labo AGM Vladimir Georgescu, qui nous a permis, à Valérie et à moi, de participer à plusieurs conférences. Vladimir m'a toujours soutenu dans ma

démarche de recherche, et a même assisté à mes premiers exposés en cryptographie. Un grand merci aussi aux secrétaires, en particulier à nos secrétaires de département Linda, Caroline et Amélie.

Il y a également tous les thésards que j'ai rencontrés et avec qui nous avons échangé pendant ces six années. Les thésards de Versailles, entre autres Vanessa Vitse, Rodolphe Lampe, Benoît Cogliati (qui a corrigé un de mes théorèmes de Brent) ; et les thésards de Cergy : Sébastien, Lysianne ... la liste est longue ! Un élève de L2, Nicolas Thierce, a également programmé une simulation de mon protocole de zéro-knowledge sur le Rubik's cube.

Enfin, je remercie les membres de mon jury, les examinateurs David Pointcheval et Louis Goubin, et les rapporteurs Bart Preneel et Pierre-Alain Fouque pour l'honneur qu'ils me font d'avoir accepté de faire parti de mon jury et pour être venus d'aussi loin.

Il est de coutume également de remercier particulièrement ses directeurs de thèse. Il est difficile cependant de trouver les mots justes pour qualifier le groupe de recherche que nous formons avec Valérie et Jacques, d'ailleurs nous n'avons toujours pas donné de nom à notre équipe ! NaPaVo (ça sonne comme Oulipo) ? Je dois dire que Valérie et Jacques ont une conception généreuse du travail de recherche, et les nombreuses réunions que nous avons eues se passaient toujours dans la bonne humeur. Notre collaboration ne s'arrêtera pas à ce document !

Résumé

La première partie est consacrée à l'étude d'attaques génériques sur des schémas de Feistel dissymétriques. Ces attaques sont en fait des distingueurs qui calculent sur une partie des clairs-chiffrés le nombre de paires vérifiant un système d'égalités et de non-égalités sur un groupe fini. La recherche de ce type d'attaques a été automatisée et améliorée, notamment en tenant compte de goulots d'étranglement. Plus généralement, des travaux sur ce type de systèmes, que l'on désigne par les termes « théorie du miroir » sont exposés dans cette partie. En particulier, on décrit le problème de la somme de deux bijections sur un groupe fini.

La deuxième partie décrit un des candidats à la compétition SHA-3 : la fonction de hachage CRUNCH. Cette fonction reprend un schéma de Feistel dissymétrique et utilise la somme de deux bijections. De plus, un nouveau mode d'enchaînement a été utilisé.

Dans la dernière partie on traite de problème d'authentification à divulgation nulle de connaissance. D'abord avec les polynômes à plusieurs variables, puis avec un problème difficile lié aux groupes symétriques. Une illustration est donnée avec le groupe du Rubik's Cube.

Enfin une méthode originale pour tenter de trouver une solution aux équations de Brent est donnée en annexe.

Abstract

The first part is dedicated to the study of generic attacks in unbalanced Feistel schemes. All these attacks are distinguishers that counts how many number of couples (plain text, cipher text) verify a system of equalities and non-equalities on a finite groupe. With the help of algorithms we have found all the possible attacks, and some attacks with a neck bottle have been rejected automatically. More generally, we describe some works about the "mirror theory" that deals about that kind of systems. We specially describe the problem of the sum of two bijections in a finite group.

The second part describes one of the candidate of the SHA-3 competition : the hash function called CRUNCH. This function includes the sum of two bijections, and each bijection is an unbalanced Feistel Scheme. A new chaining process for long messages is given.

In the last part we deal with zero-knowledge authentication problems. The first protocol is based on multivariate polynomials. The second is linked to a difficult problem in symmetric groups. We take the example of the Rubik's cube group.

Finally, we reveal some works on Brent equations. We build an algorithm that may find one solution.

Introduction

Trois parties pour une thèse, cela va sembler beaucoup ! On peut penser que je me suis dispersé entre tous ces sujets différents. Je m'explique, donc. La troisième partie de ma thèse est consacrée aux schémas à divulgation nulle de connaissance. Pour les définir, on a souvent besoin de fonctions qui nous engagent, on les appelle « commitment » en anglais. Ce sont des fonctions qui permettent de donner une information cachée à quelqu'un en gardant la possibilité de la révéler plus tard. Pour fabriquer de telles fonctions, on se sert souvent de fonctions de hachages. Pourquoi pas CRUNCH, candidat à la compétition SHA-3 que je décris dans ma deuxième partie ? Certes il a été éliminé (trop lourd, trop lent aux goût de certains) mais sa sécurité reste entière. Elle repose en effet sur deux points essentiels, qui sont détaillés dans la première partie : d'abord CRUNCH est fabriqué à partir de la somme de deux bijections, et la théorie du miroir est là pour nous aider à justifier que cela donne bien une fonction aléatoire. Ensuite les bijections choisies pour notre fonction de hachage sont des schémas de Feistel expansifs dissymétriques, dont nous avons justement classifié et généralisé les attaques génériques.

Reprenons donc les choses dans l'ordre. Dans la première partie, nous commençons par étudier des attaques classiques sur les schémas de Feistel expansifs dissymétriques. Les schémas de Feistel ont été historiquement utilisés pour la construction du DES. Ils réalisent une bijection de $\{0,1\}^{2n}$ dans lui même. La figure 1 nous montre un exemple de schéma de Feistel classique.

Dans la figure f désigne une fonction aléatoire qui fait office de clé. Évidement, il faut enchaîner les tours pour réaliser une bijection pseudo-aléatoire. Nous aurons alors besoin d'une famille de fonctions f_1, f_2, ..., f_r où r est le nombre total de tours. Un des axes principaux de recherche sera de trouver le nombre minimum de tours pour qu'il soit impossible de distinguer une bijection

FIGURE 1 – Un tour d'un schéma de Feistel classique

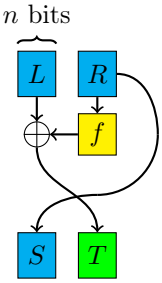
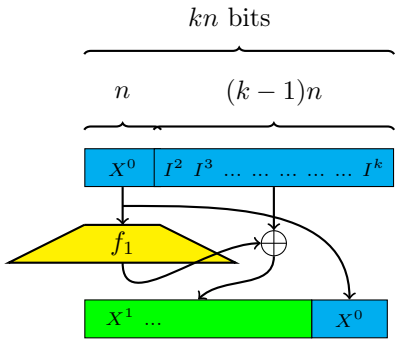


FIGURE 2 – Premier tour d'un schéma de Feistel expansif



réalisée avec ce schéma d'une bijection aléatoire. Pour cela, plusieurs travaux ont déjà été réalisés. Il y a d'une part des preuves de sécurité sur l'impossibilité théorique de distinguer les deux bijections lorsque l'on dispose d'un nombre insuffisant de messages. D'autre part il y a les constructions d'attaques génériques sur ces schémas. Par « attaques génériques » on entend des attaques qui fonctionnent pour la plupart des fonctions f_i utilisées, et non pour des fonctions particulières. Les attaques utilisées vont compter le nombre de fois que certaines configurations apparaissent lorsque l'on combine les entrées et les sorties. Autrement dit nous allons compter le nombre de solutions d'un système linéaire dont les variables sont des parties du message (dans $\{0,1\}^{2n}$) ou de son chiffré (image du message par la bijection). Ce système comporte à la fois des égalités et des non-égalités. Lorsque nous trouvons une différence notable entre le nombre de solutions pour les deux bijections, nous affirmons qu'une des deux bijections (en général celle dont le nombre est le plus grand) est la bijection construite sur les schémas de Feistel. Pour des valeurs de n petites nous pouvons même simuler ce genre d'attaque et estimer leur probabilité de réussite.

Si l'étude des schémas de Feistel classiques est pratiquement complète puisque les travaux sur la sécurité donnent des résultats proches des attaques génériques, il y a encore beaucoup à étudier sur d'autres schémas de Feistel. Nous allons détailler dans cette thèse l'étude des attaques génériques les schémas de Feistel expansifs. Cette fois ci, le message est coupé non pas en deux morceaux mais en $k \geq 3$ morceaux. Voir figure 2 pour le premier tour d'un schéma de Feistel expansif.

L'intérêt de ce genre de schémas par rapport au schémas de Feistel classiques est la taille des clés, autrement dit la quantité de mémoire que nous devons allouer pour stocker les fonctions aléatoires. Supposons k pair. Pour un tour de schéma de Feistel classique nous aurons besoin de stocker les images de $2^{kn/2}$ valeurs utilisant chacune $kn/2$ bits, donc en tout $\frac{kn}{2}2^{kn/2}$ bits. Pour un tour de schéma de Feistel expansif nous aurons besoin de stocker les images de 2^n valeurs utilisant chacune $(k-1)n$ bits, donc en tout $(k-1)n2^n$ bits. Pour $k = 128$ par exemple (valeur prise pour CRUNCH), le rapport entre ces deux valeurs est $\frac{64}{127}2^{63n}$. Evidement, il faut tenir compte du nombre de tours nécessaires pour arriver à une sécurité comparable, mais nous estimons à 6 le nombre de tours pour les schémas de Feistel classiques et $3 \times 128 = 384$ tours pour les schémas expansifs. Le rapport reste donc largement plus grand que 1.

Dans le premier chapitre nous décrirons donc des attaques génériques sur

ce genre de schémas. L'étude a été particulièrement poussée sur les attaques « rectangles » puisque toutes ces attaques pour des valeurs petites de k ($k \leq 7$) ont été générées de façon informatique, puis généralisées pour des valeurs plus grandes de k. De plus, des simulations de ces attaques ont été effectuées et ont confirmé leur efficacité. Un nouveau concept de goulot d'étranglement à été mis en évidence à cette occasion. En effet, certaines attaques nécessitaient trop de contraintes par rapport aux nombres de variables engagées et ont dû être rejetées pour cette raison. Ces travaux ont mené à une publications à ASIACRYPT 2010 [66].

Dans la suite de cette première partie, nous allons étudier les systèmes d'égalités et de non-égalités dans un groupe fini. La théorie qui étudie ce genre de systèmes a été appelée « théorie du miroir ».(à finir suivant ce qui sera mis)

Dans la seconde partie de ma thèse nous allons voir en détail ce qu'est une fonction de hachage et ce qui a pu amener le NIST à lancer une commande internationale d'un nouveau standard de fonction de hachage qui sera baptisée SHA-3. Les fonctions de hachages sont des fonctions de $\{0,1\}^*$ (suites finies de bits, autrement dit des « fichiers ») dans $\{0,1\}^n$ où n vaut 256 par exemple, qui se comportent un peu comme des fonctions aléatoires, de sorte qu'il est très difficile d'expliciter deux éléments (deux fichiers) distincts x et x' qui ont la même image, c'est ce qu'on appelle une collision. Lorsqu'un chercheur arrive à trouver une collision, soit de façon explicite, soit de façon théorique avec un algorithme utilisant moins de $2^{n/2}$ opérations – ce qui est le coût théorique d'une telle recherche –, on estime que la fonction de hachage est cassée ou bien présente une faiblesse relative. On se refuse alors de l'employer pour certaines utilisations commerciales ou liées à la sécurité. La principale difficulté dans la construction des fonctions de hachage est de justifier sa solidité, c'est à dire sa capacité à résister à des attaques menant à des collisions. L'autre préoccupation est de construire des fonctions qui se calculent assez rapidement, même pour des fichiers de taille importante (de l'ordre de plusieurs giga-octets par exemple). Enfin, la mode étant à la miniaturisation, il est également important que la fonction de hachage puisse tenir dans une carte à puce avec quelques méga-octets de mémoire seulement. Une équipe s'est alors constituée autour de Jacques Patarin pour proposer un candidat, que l'on a dénommé CRUNCH (univers se refermant sur lui-même). L'accent a été particulièrement porté sur la sécurité de la fonction. L'idée était de combiner des éléments très simples dont la sécurité sera peut être prouvée un jour. Pour fabriquer notre fonction on a donc utilisé la somme de deux bijections construites à l'aide schémas de Feistel expansifs. Le nombre de tours pour chacune des permutations a été choisi bien au delà des dernières attaques connues, afin de résister à de futures attaques. En revanche, à cause du problème d'encombrement de la mémoire, il a fallu trouver un compromis pour la génération des fonctions aléatoires. Si les deux schémas de Feistel utilisent bien des fonctions aléatoires indépendantes, nous avons dû utiliser des parties de mémoire communes pour définir les fonctions des différents tours. Cependant, aucune attaque importante a été trouvée contre CRUNCH. Les principaux griefs contre notre fonction de hachage ont été le mode d'enchaînement choisi, et la place mémoire importante utilisée, qui entraîne évidement une lenteur de notre fonction par rapport aux autres candidats. En effet, la plupart des concurrents ont utilisé des combinaisons d'opérations informatiques très rapides, mais sans aucun fondement théorique. Il est alors impossible de prouver leur sécurité. Pour sécuriser leurs fonctions, ils ont augmenté le nombre de combinaisons au fur et

à mesure que certaines attaques se concrétisaient. A noter d'ailleurs que plus d'un tiers des candidats ont été éliminés avec de sérieuses attaques (souvent des collisions). Pour corriger le premier grief, un nouveau procédé d'enchaînement à été proposé lors de la conférence du premier tour. CRUNCH a été rejeté mais ce procédé a été repris récemment par des chercheurs russes.

Dans la dernière partie, nous ferons une incursion dans le domaine de la clé publique. Je vais expliquer brièvement en quoi consiste ce procédé révolutionnaire. Imaginons que Alice souhaite recevoir un message chiffré de Bob qui se trouve à distance. Toutes leurs communications sont surveillées de près par Charlie. Alice commence par envoyer une clé à Bob avec laquelle il va chiffrer son message. Cette clé sert à chiffrer, mais pour déchiffrer on a besoin d'une autre clé, appelée clé secrète et que seule Alice connaît. Charlie voit la clé envoyée par Alice, et sait également comment chiffrer les messages. Si le message attendu est juste « oui » ou « non », Charlie peut évidement chiffrer les deux possibilités pour trouver ce que va envoyer Bob, mais il suffit à Bob de rajouter n'importe quel phrase aléatoire devant sa réponse avant de la chiffrer pour tromper Charlie. Donc Bob agit ainsi, il chiffre « KEJDHBKHKJHSVCKJHVSC OUI » puis envoie le message chiffré à Alice. Alice déchiffre le message à l'aide de sa clé privée et comprend facilement la réponse de Bob. On peut cependant imaginer un autre scénario. Charlie est vraiment méchant, il coupe la ligne de Bob, prend sa place et chiffre « LSKJLKSJBCLKJHVKSKJX NON » pour l'envoyer à Alice en se faisant passer pour Bob. On voit là que le problème de l'authentification est crucial dans ce genre de protocole. C'est justement ce à quoi on va s'intéresser dans cette partie. En 1985, un nouveau concept a vu le jour : le « zero knowledge », en français on parle de schémas à divulgation nulle de connaissance. Le principe pour s'authentifier est le suivant : Bob détient un secret. Il sait résoudre un problème publique, par exemple il sait comment colorier à l'aide de trois couleurs un graphe sans que deux sommets reliés par une arrête aient jamais la même couleur. Ce problème est réputé difficile lorsque le graphe a une solution et que sa taille est suffisamment importante : même un ordinateur ne peut trouver la solution. En revanche, il est très facile de vérifier qu'une solution est valide, cela prend à peine quelques centièmes de seconde. Bob va alors prouver à Alice qu'il détient ce secret *sans en révéler une quelconque partie*. Ainsi, il s'authentifie auprès d'Alice et peut continuer à s'authentifier avec le même secret auprès d'autres personnes.

Depuis l'invention de ce concept, plusieurs protocoles ont été trouvés autour de certains problèmes difficiles. Un des plus célèbres est le PKP (Permuted Kernel Problem). On a une matrice A publique dans un corps fini $\mathbb{Z}/p\mathbb{Z}$, et le secret est la connaissance d'un élément non nul du noyau X. On permute alors les coordonnées de X à l'aide d'une permutation secrète σ, et on rend publique X_σ, c'est à dire le vecteur X avec ses coordonnées permutées. A. Shamir a publié un protocole permettant de prouver qu'on détient ce secret sans le révéler.

Nous verrons dans cette thèse deux nouveaux protocoles d'authentification de ce type. Le premier porte sur les groupes symétriques, et notamment sur le groupe du Rubik's cube. Le secret est la factorisation d'un élément du groupe à l'aide de générateurs choisis du groupe. On verra que l'utilisation du Rubik's cube permet de visualiser plus facilement le schéma. L'étude des algorithmes de factorisation dans les groupes finis est quand à elle encore à approfondir. Nous donnerons des références récentes sur les travaux dans ce domaine. Ce protocole a donné lieu à une publication à CANS 2013 [67]. Le deuxième protocole est

une généralisation d'un protocole publié à CRYPT0 2011 sur les systèmes de polynômes de degré 2. Un protocole du même auteur sur les polynômes de degré 3 a été également publié à PKC 2012. Notre article quant à lui a été publié à LATINCRYPT 2012 [35].

Enfin, un dernier chapitre portera sur mes perspectives de recherche. Plusieurs travaux sont déjà en route, sur des sujets assez variés. Le plus ambitieux est celui sur les équations de Brent dont la résolution est un véritable défi algorithmique. Plutôt qu'essayer une méthode brutale vouée à l'échec, nous avons cherché à cerner les solutions présentant une certaine symétrie. L'algorithme a été programmé et une partie des cas a été scrutée, pour l'instant sans succès. De nombreux travaux ont été également faits sur les attaques d'autres schémas de Feistel, et l'on ambitionne de pouvoir généraliser l'automatisation des calculs pour les attaques. Une partie de ces travaux a été publié à CANS 2013 [37]. Il y a encore bien d'autres sujets de recherche intéressants, Jacques Patarin en trouve environ un nouveau par semaine et tous me plaisent ! Mais hélas, il faut du temps, beaucoup de temps !

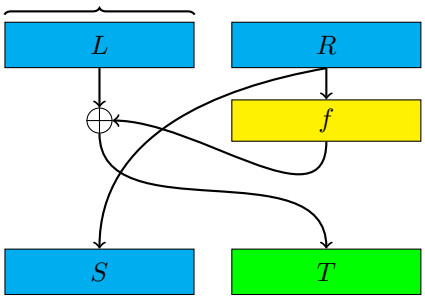
Première partie

Clé privée

Chapitre 1

Cryptanalyses de schémas de Feistel

FIGURE 1.1 – Un tour d'un schéma de Feistel classique



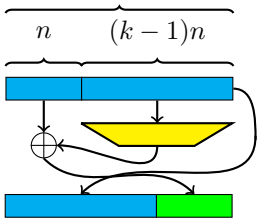
La partie cryptographie à clé secrète concerne principalement les attaques génériques sur différents schémas de Feistel. Les schémas de Feistel classiques permettent de construire des permutations de $\{0,1\}^{2n}$ vers $\{0,1\}^{2n}$ à l'aide de fonctions de $\{0,1\}^n$ vers $\{0,1\}^n$, qui sont appelées les fonctions internes (Figure 1.1) et qui constituent les clés secrètes. En itérant le procédé, on obtient plusieurs tours de schémas de Feistel. Cette construction est à la base de nombreux schémas de chiffrement (par exemple le DES). Les premiers travaux sur les schémas de Feistel ont été initiés par Luby et Rackoff [31], lorsque les fonctions internes sont aléatoires. De nombreuses variantes, utilisées dans divers schémas de chiffrement, ont ensuite été étudiées tant au niveau des attaques que de la sécurité. Une première variante consiste à prendre comme fonctions internes des permutations aléatoires au lieu de fonctions aléatoires ([27,40,65]). On a également les schémas de Feistel dissymétriques : les fonctions internes sont soit expansives ([24,43,58]), soit contractantes ([38,42]). Certains schémas de chiffrement sont basés sur les schémas alternés dans lesquelles on utilise alternativement des fonctions internes expansives et contractantes ([1,20,33]). Citons encore les schémas de Feistel de type 1, 2 ou 3 ([6,20,21,69]). Il existe d'autres constructions permettant d'obtenir des permutations pseudo-aléatoires comme par exemple le schéma de Massey et Lai utilisé dans IDEA ([30]) ou encore les schémas du type MISTY ([11,16,26,32,34,49,57,63,64]).

Nous étudions les attaques génériques sur différents schémas.

Définition 1 (Attaque générique)**.** Une attaque sur un schéma de Feistel ou une variante d'un schéma de Feistel est dite générique lorsque les fonctions internes sont supposées aléatoires.

Pour les schémas de chiffrement par blocs utilisés dans la vie de tous les jours, les fonctions internes ne sont jamais pseudo-aléatoires et il y a souvent des attaques meilleures que les attaques génériques. Cependant, les attaques génériques sont très intéressantes parce qu'elles utilisent les propriétés générales des structures et non des problèmes sur les fonctions internes. On peut donc considérer qu'elles donnent le nombre minimum de tours nécessaires pour une sécurité donnée : d'une manière générale, la sécurité avec les fonctions internes particulières est inférieure ou au mieux égale à celle obtenue avec des fonctions internes aléatoires.

FIGURE 1.2 – Un tour d'un schéma de Feistel contractant kn bits



Nous avons étudié principalement les attaques à texte clair connu (KPA) et les attaques à texte clair choisi non adaptatives (CPA-1). Étant donné la symétrie évidente entre l'entrée et la sortie de ce genre de schémas, il est facile de transformer ces attaques en s'occupant des textes chiffrés. Nous distinguons deux familles d'attaques.
- Les attaques « 2-points » qui utilisent des relations entre les entrées et les sorties pour des paires de messages.
- Les attaques « rectangles » qui utilisent des relations entre les entrées et les sorties pour des ensembles de φ messages (où φ est pair).
Selon les schémas étudiés et le nombre de tours considérés, certaines attaques seront plus performantes que d'autres.

1.0.1 Schémas avec attaques « 2-points »

Ces attaques utilisent donc des corrélations entre les entrées et les sorties pour des paires de points. Nous les avons utilisées pour les schémas de Feistel dissymétriques contractants, les schémas de Feistel généralisés de type 1, 2, 3, et les schémas alternés.

Schémas de Feistel contractants

Ces schémas permettent de construire des permutations de $\{0,1\}^{kn}$ vers $\{0,1\}^{kn}$ à l'aide de fonctions définies de $\{0,1\}^{(k-1)n}$ vers $\{0,1\}^n$. La figure 1.2 représente un tour de ce schéma. L'entrée est notée $[I_1, I_2, \ldots, I_k]$ et la sortie $[S_1, S_2, \ldots, S_k]$. Nous définissons maintenant les variables internes X_i introduites après le tour j. Après un tour, la sortie est donnée par $[I_2, I_3, \ldots, X^1]$ où $X^1 = I_1 \oplus f_1([I_2, \ldots, I_k])$. De même, on obtient :

$$X^2 = I_2 \oplus f_2([I_3, \ldots, I_k, X^1])$$

$$X^3 = I_3 \oplus f_3([I_4, \ldots, I_k, X^1, X^2])$$

et ainsi de suite. Nous notons G_k^d la permutation obtenue à partie d'un schéma de Feistel contractant avec d tours. Nous allons présenter plusieurs attaques qui permettent de distinguer une permutation aléatoire d'une permutation G_k^d. Selon le nombre de tours, il est possible de trouver des relations entre les variables d'entrée et les variables de sorties. Ces relations peuvent être satisfaites de manière aléatoire ou suite à des relations entre les variables internes qui se propagent grâce à la structure du schéma. Les attaques utilisent m paires de messages

clairs/chiffrés et on compte ensuite le nombre de paires satisfaisant les conditions données. On note ce nombre $\mathcal{N}_{G_k^d}$. On compare ensuite $\mathcal{N}_{G_k^d}$ à $\mathcal{N}_{perm}$ qui est le nombre de paires satisfaisant les relations dans le cas d'une permutation aléatoire. L'attaque est réussie s'il est possible de distinguer une permutation aléatoire d'une permutation G_k^d, c'est à dire si la différence $|E(\mathcal{N}_{G_k^d}) - E(\mathcal{N}_{perm})|$ est supérieure aux deux écart-types $\sigma_{G_k^d}$ et σ_{perm} où E dénote l'espérance. Pour calculer toutes ces valeurs, il faut tenir compte du fait que les m^2 paires obtenues ne sont pas indépendantes même si leurs dépendances mutuelles sont très faibles. Pour le calcul des écart-types, nous utilisons la formule suivante qui permet d'obtenir la variance d'une somme : Si $x_1, \ldots x_n$ sont des variables aléatoires et, si V représente la variance, alors

$$V\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} V(x_i) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left[E(x_i, x_j) - E(x_i)E(x_j)\right]$$

.

L'entrée est notée $[I_1, I_2, \ldots, I_k]$ et la sortie $[S_1, S_2, \ldots, S_k]$ où chaque I_s, S_s est un élément de $\{0,1\}^n$. Quand nous avons m messages, $I_s(i)$ représente la partie s du message d'entrés i. La même notation est utilisée pour les sorties. Avant de décrire quelques attaques, nous introduisons les notations différentielles dont nous aurons besoin. Nous utilisons des paires de textes clairs/chiffrés. Par exemple, en KPA, sur les variables d'entrée, $[\mathbf{0}, \mathbf{0}, \Delta_3^0, \Delta_4^0, \ldots, \Delta_k^0]$, signifie que la paire de messages (i, j) satisfait $I_1(i) = I_1(j)$, $I_2(i) = I_2(j)$, et $I_s(i) \oplus I_s(j) = \Delta_s^0$, $3 \leq s \leq k$. En CPA-1, la même notation signifie que nous avons choisi les I_1 tous égaux à la même valeur et les I_2 tous égaux à la même valeur. Puisque nous voulons que le relations sur les variables d'entrée se propagent, nous allons imposer des conditions sur les variables internes lors de certains tours. Quand nous imposons des conditions sur les variables internes pour obtenir le chemin différentiel, nous utilisons la notation $\mathbf{0}$ pour signifier que les variables internes correspondantes sont égales pour les messages i et j.

Nous donnons maintenant plusieurs exemples d'attaques (CPA-1 et KPA) sur les G_k^d dans le cas où $k = 3$. Tous les attaques sont décrites dans [42].

G_3^3 : 3 tours, CPA-1 avec $m = 2$ messages. Choisissons $I_2(1) = I_2(2)$, $I_3(1) = I_3(2)$ et $I_1(1) \neq I_1(2)$. L'attaque teste si $S_1(1) \oplus S_1(2) = I_1(1) \oplus I_1(2)$. Ceci apparaît avec probabilité 1 pour une permutation G_3^3, et avec probabilité $\simeq \frac{1}{2^n}$ pour une permutation aléatoire. Donc pour 3 tours, nous avons une attaque générique avec deux messages (non adaptative) et $O(1)$ calculs.

G_3^3 : 3 tours, KPA avec $m \simeq 2^n$ messages. Il est possible de transformer cette attaque CPA-1 en attaque KPA de la façon suivante. Si nous avons $m \geq 2^n$ entrées aléatoires, $[I_1(i), I_2(i), I_3(i)]$, alors (puisque $m^2 \geq 2^{2n}$) nous aurons une collision $I_2(i) = I_2(j)$ et $I_3(i) = I_3(j), i \neq j$ avec une bonne probabilité. Dès que nous avons une telle collision, nous testons si $S_1(i) \oplus S_2(j) = I_1(i) \oplus I_1(j)$. Cette attaque demande donc $O(2^n)$ messages aléatoires et $O(2^n)$ calculs.

Attaques CPA-1 et KPA sur 6 tours. L'attaque CPA-1 est décrite dans le tableau 1.1. L'attaque est construite de la façon suivante. On choisit m messages tels que $\forall i, I_3(i) = 0$. Soit $\mathcal{N}$, le nombre de paires (i, j) telles que $S_1(i) = S_1(j)$

TABLE 1.1 – CPA-1 sur G_3^6

tour	Δ_1^0	Δ_2^0	$\mathbf{0}$
1	Δ_2^0	0	$\mathbf{0}$
2	0	0	Δ_2^0
3	0	Δ_2^0	$\mathbf{0}$
4	Δ_2^0	0	$\mathbf{0}$
5	0	0	Δ_2^0
6	0	Δ_2^0	Δ_3^6

et $S_2(i) \oplus S_2(j) = I_2(i) \oplus I_2(j)$. Pour une permutation aléatoire, nous avons :

$$\mathcal{N}_{perm} \simeq \frac{m^2}{2.2^{2n}} + O(\frac{m}{2^n}), \quad \sigma_{perm} = \frac{m}{\sqrt{2} \cdot 2^n}$$

où $O(\frac{m}{2^n})$ est l'écart-type. Pour une permutation G_3^6, puisque les conditions que nous imposons sur les variables internes nous donnent

$$\mathcal{N}_{G_3^6} \simeq \frac{m^2}{2.2^{2n}} + \frac{m^2}{2 \cdot 2^{3n}}, \quad \sigma_{G_3^6} = \frac{m}{\sqrt{2} \cdot 2^n}$$

Les calculs de variance sont exposés dans [42]. Nous pouvons donc distinguer les deux permutations dès que la différence entre les deux espérances est supérieure aux écart-types, c'est à dire lorsque $\frac{m^2}{2^{3n}} \geq \frac{m}{2^n}$, c'est à dire pour $m \geq 2^{2n}$. On peut évidemment, comme précédemment, transformer cette attaque CPA-1 en KPA et cette attaque sera réussie dès que $m \geq 2^{\frac{5n}{2}}$.

Nous avons implémenté ces attaques CPA-1 et KPA sur G_3^6 pour des petites valeurs de n ($n = 6$ et $n = 8$). Les valeurs expérimentales confirment les résultats théoriques. Pour ces simulations, nous avons précédé de la façon suivante :

- On choisit de manière aléatoire une permutation G_3^6
- On choisit de manière aléatoire une permutation. Pour cela, on utilise un schéma de Feistel symétrique avec un grand nombre de tours (supérieur à 20)
- On lance l'attaque CPA-1 avec $m = 2^{2n}$, et l'attaque KPA avec $m = 2^{3n}$ ($m = 2^{\frac{5n}{2}}$ fonctionne également).
- On compte le nombre de paires de textes clairs/chiffrés qui satisfont les relations pour la permutation générée par G_3^6 et pour la permutation aléatoire
- On itère le procédé environ 1000 fois pour pouvoir évaluer les variance et les écart-types
- On calcule la variance et l'écart-type pour la permutation obtenue à l'aide du schéma et pour la permutation aléatoire

Les valeurs expérimentales pour $\mathcal{N}_{G_3^6} - \mathcal{N}_{perm}$ sont très proches des valeurs théoriques ($\frac{m^2}{2 \cdot 2^{4n}}$ en KPA et $\frac{m^2}{2 \cdot 2^{3n}}$ en CPA-1). Également, les valeurs

TABLE 1.2 – Résultats expérimentaux pour les attaques KPA et CPA-1 sur G_3^6

Attaque	n	$\mathcal{N}_{G_3^6}$	$\mathcal{N}_{perm}$	$\mathcal{N}_{G_3^6} - \mathcal{N}_{perm}$	$\frac{m^2}{2 \cdot 2^{4n}}$	$\sigma_{G_3^6}$	σ_{perm}	$\frac{m}{\sqrt{2} \cdot 2^{\frac{3n}{2}}}$
KPA	6	131006	129011	1995	2048	159	372	362.038
KPA	8	8388308	8355787	32521	32768	2862	2833	2896.309
Attaque	n	$\mathcal{N}_{G_3^6}$	$\mathcal{N}_{perm}$	$\mathcal{N}_{G_3^6} - \mathcal{N}_{perm}$	$\frac{m^2}{2 \cdot 2^{3n}}$	$\sigma_{G_3^6}$	σ_{perm}	$\frac{m}{\sqrt{2} \cdot 2^{n}}$
CPA	6	2058	2009	49	32	45	44	45.254
CPA	8	32781	32601	180	128	178	185	182.019

expérimentales σ_{perm} sont proches des valeurs attendues ($\frac{m}{\sqrt{2} \cdot 2^{\frac{3n}{2}}}$ en KPA et $\frac{m}{\sqrt{2} \cdot 2^{n}}$ en CPA-1) . Ces simulations confirment que nous pouvons distinguer une permutation G_3^6 d'une permutation aléatoire avec la complexité que nous avons donnée.

Au delà de la complexité 2^{kn}, il est possible de monter des attaques contre des générateurs de permutations, c'est à dire de distinguer un générateur de permutations aléatoires d'un générateur de permutations G_k^d. Nous pouvons construire des attaques similaires aux précédentes mais également utiliser le Théorème 1 suivant :

Théorème 1. *Soit Ψ une permutation produite par un schéma de Feistel dissymétrique sur $\{0,1\}^{\alpha+\beta} \to \{0,1\}^{\alpha+\beta}$ avec des fonctions internes de $\{0,1\}^{\beta} \to \{0,1\}^{\alpha}$. Alors si $\alpha \geq 2$ et $\beta \geq 1$, Ψ a une signature paire.*

La démonstration de ce théorème est semblable à celle donnée dans le cas des schémas de Feistel symétriques [12, 44].

Soit f une permutation de $\{0,1\}^{kn}$ vers $\{0,1\}^{kn}$. Alors avec $O(2^{kn})$ calculs sur les 2^{kn} valeurs d'entrée et de sortie de f, on peut calculer le signature de f. En effet, il suffit de calculer les cycles c_i de f, $f = \prod_{i=1}^{\alpha} c_i$ et d'utiliser la formule :

$$\text{signature}(f) = \prod_{i=1}^{\alpha} (-1)^{longueur(c_i)+1}.$$

Ainsi, il est possible de distinguer un générateur de permutations aléatoires d'un générateur de permutations G_k^d après $O(2^{kn})$ calculs.

Les tableaux 1.3 (respectivement 1.4) donnent les complexités des attaques obtenues sur les Feistel contractants pour $k = 3$ (respectivement $k \geq 4$).

Schémas de Feistel de type 1, 2, 3

Les notations sont les mêmes que pour les schémas de Feistel contractants.

Schémas de Feistel de type 1. Ils sont décrits dans la figure 1.3.

De 1 à $k-1$ tours, on utilise un seul message puisque après t tours, $1 \leq t \leq k-1$, nous avons $S_{k-t+1} = I_1$. Cette condition est donc satisfaite avec probabilité 1 pour un schéma de Feistel de type 1 et avec probabilité $\frac{1}{2^n}$ pour une permutation aléatoire. Donc avec un seul message, on peut construire une attaque KPA et CPA-1 qui permet de distinguer un schéma de Feistel de type

TABLE 1.3 – Résultats sur les G_3^d. Au-delà de 7 tours on utilise plusieurs permutations.

	KPA	CPA-1
G_3^1	1	1
G_3^2	1	1
G_3^3	2^n	2
G_3^4	2^n	$2^{n/2}$
G_3^5	$2^{3n/2}$	2^n
G_3^6	$2^{5n/2}$	2^{2n}
G_3^7	2^{3n}	2^{3n}
G_3^8	2^{4n}	2^{4n}
G_3^9	2^{6n}	2^{6n}
G_3^{10}	2^{7n}	2^{7n}
G_3^{11}	2^{8n}	2^{8n}
G_3^{12}	2^{10n}	2^{10n}
$G_3^d, d \geq 12$	$2^{(d+\lfloor \frac{d}{3} \rfloor - 6)}$	$2^{(d+\lfloor \frac{d}{3} \rfloor - 6)}$

TABLE 1.4 – Résultats sur les G_k^d pour $k \geq 4$. Au-delà de $2k$ tours on utilise plusieurs permutations.

	KPA	CPA-1
G_k^d, $1 \leq d \leq k-1$	1	1
G_k^k	$2^{\frac{n(k-1)}{2}}$	2
G_k^{k+1}	$2^{\frac{n(k-1)}{2}}$	$2^{\frac{n}{2}}$
G_k^{k+2}	$2^{\frac{k}{2}n}$	$2^{\frac{3}{2}n}$
G_k^{k+3}	$2^{(\frac{k+1}{2})n}$	$2^{\frac{5}{2}n}$
G_k^{k+i}, $1 \leq i < k$	$2^{(\frac{k+i-2}{2})n}$	$2^{(\frac{2i-1}{2})n}$
G_k^{2k}	$2^{(2k-4)n}$	$2^{(2k-4)n}$
G_k^d, $d \geq 2k$	$2^{(d+(k-2)\lfloor \frac{d}{k} \rfloor - 2k)n}$	$2^{(d+(k-2)\lfloor \frac{d}{k} \rfloor - 2k)n}$

1 d'une permutation aléatoire.
Nous donnons le modèle utilisé pour les attaques KPA. Les conditions après $pk-2$ tours ($p \geq 3$) sont données par

$$(1) \begin{cases} S_2(i) = S_2(j) \\ I_1(i) \oplus I_1(j) = S_3(i) \oplus S_3(j) \end{cases}$$

On compte le nombre d'indices (i, j) tels que ces conditions soient satisfaites. Soit $\mathcal{N}_{perm}$ (respectivement $\mathcal{N}_{scheme}$) le nombre ainsi obtenu pour une permutation aléatoire (respectivement pour un schéma de Feistel de type 1). Avec une permutation aléatoire, les conditions apparaissent au hasard et la calcul de l'espérance donne $E(\mathcal{N}_{perm}) \simeq \frac{m^2}{2^{2n}}$ et $E(\mathcal{N}_{scheme}) \simeq \frac{m^2}{2^{2n}} + \frac{m^2}{2^{(p-1)n}}$ pour un schéma de Feistel de type 1 en tenant compte des conditions sur les variables internes. Pour $p \geq 4$, il faut faire les calculs de variances. Les écart-types sont donnés par $\sigma(\mathcal{N}_{perm}) \simeq \sqrt{E(\mathcal{N}_{perm})}$ et $\sigma(\mathcal{N}_{scheme}) \simeq \sqrt{E(\mathcal{N}_{scheme})} \simeq \sqrt{E(\mathcal{N}_{perm})}$ quand $p \geq 4$. Ceci montre que l'on peut distinguer un schéma de Feistel de

FIGURE 1.3 – Un tour pour les schémas de Feistel de type 1

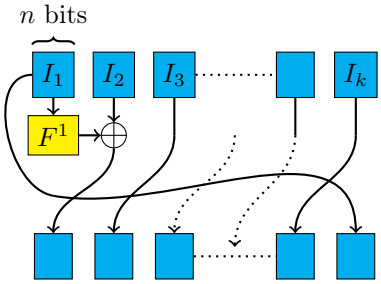


TABLE 1.5 – KPA pour les schémas de Feistel de type 1

round	Δ_1^0	Δ_2^0	Δ_3^0	...	Δ_{k-1}^0	Δ_k^0
1				...		Δ_1^0
2				...	Δ_1^0	
⋮						
$k-1$	**0**	Δ_1^0		...		
k	Δ_1^0			...		0
$k+1$				...		Δ_1^0
⋮						
$pk-2$		0	Δ_1^0	...		
$pk-1$	**0**	Δ_1^0		...		
pk	Δ_1^0			...		0
⋮						
$(p+1)k-2$		0	Δ_1^0	...		

type 1 d'une permutation aléatoire dès que $\frac{m^2}{2^{(p-1)n}} \geq \frac{m}{2^n}$. On a la condition $m \geq 2^{(p-2)n}$. Le nombre maximum de messages étant 2^{kn}, ces attaques sont valides pour $p-2 \leq k$ et donc avec $p = k+2$, nous pouvons attaquer jusqu'à $(k+2)k-2 = k^2+2k-2$ tours.

Nous passons maintenant aux attaques CPA-1. Comme nous l'avons vu, jusqu'au rang $k-1$, un seul message suffit. Soit t, $0 \leq t \leq k-2$. On construit une attaque CPA-1 sur $k+t$ tours avec deux messages de la façon suivante : on choisit deux messages tels que $I_s(1) = I_s(2)$ pour $1 \leq s \leq t+1$. On vérifie ensuite si $I_{t+2}(1) \oplus I_{t+2}(2) = S_{t+1}(1) \oplus S_{t+1}(2)$. Avec une permutation aléatoire, cette condition est satisfaite avec une probabilité égale à $\frac{1}{2^n}$ et avec un schéma de type 1, la probabilité est 1. On arrive ainsi jusqu'au tour $2k-2$. On continue ainsi les attaques en utilisant les bonnes conditions sur les variables d'entrée. La table 1.6 donne un attaque CPA-1 sur $3k-2$ tours pour laquelle on utilise $2^{n/2}$ messages. En adaptant les conditions sur les variables d'entrés et en faisant les calculs variances, on construit des attaques CPA-1 jusqu'au tour k^2+k-1. Les résultats sont donnés dans le tableau 1.7. D'autres attaques et calculs de variance sont décrites dans [36].

TABLE 1.6 – Attaque CPA-1 sur $3k-2$ pour les schémas de Feistel de type 1

round	**0**	**0**	**0**	...	**0**	Δ_k^0
1	0	0	0	...	Δ_k^0	0
2				...	Δ_k^0	
$\vdots$						
$k-2$	0	Δ_k^0		...		
$k-1$	Δ_k^0			...		0
k	Δ_1^k	0	0	...	0	Δ_k^0
$\vdots$						
$2k-2$	**0**	Δ_k^0		...		
$2k-1$	Δ_k^0			...		0
$2k$				...	0	Δ_k^0
$\vdots$						
$3k-3$		0	Δ_k^0	...		
$3k-2$		Δ_k^0		...		

TABLE 1.7 – Complexité des attaques sur les schémas de Feistel de type 1

tours	KPA
$1 \to k-1$	1
$k \to 2k-1$	$2^{n/2}$
$2k \to 3k-2$	2^n
$\vdots$	
$rk-2$	$2^{(r-2)n}$
$rk-1$	$2^{(r-3/2)n}$
rk	
$\vdots$	$2^{(r-1)n}$
$(r+1)k-2$	
$\vdots$	
k^2+2k-2	2^{kn}

tours	CPA-1	*tours*	CPA-1
1			
$\vdots$	1	$\vdots$	
$k-1$			
k		$pk-(p-2)$	
$\vdots$	2	$\vdots$	$2^{(p-2)n}$
$2k-2$		$(p+1)k-p$	
$2k-1$			
$\vdots$	$2^{n/2}$	$\vdots$	
$3k-2$			
$3k-1$		k^2+1	
$\vdots$	2^n	$\vdots$	$2^{(k-1)n}$
$4k-3$		k^2+k-1	

Schémas de Feistel de type 2. Ils sont décrits dans la figure 1.4.

Pour construire les attaques sur ces schémas, nous utilisons des techniques semblables à aux attaques sur les schémas de type 1 et nous avons toujours besoin de calculer les variances. Nous donnons les résultats obtenus dans la table 1.8. Le détail des attaques et des calculs se trouve dans [36].

Schémas de Feistel de type 3. Ils sont décrits dans la figure 1.5.

A nouveau, pour construire les attaques sur ces schémas, nous utilisons des techniques semblables à aux attaques sur les schémas de type 1 et nous avons toujours besoin de calculer les variances. Nous donnons les résultats obtenus

FIGURE 1.4 – Un tour pour les schémas de Feistel de type 2

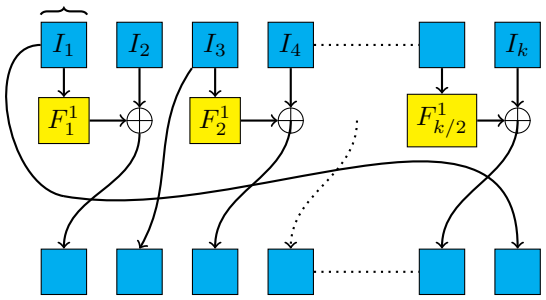


TABLE 1.8 – Complexité des attaques sur les schémas de Feistel de type 2

Tour	KPA	CPA-1
1	1	1
2	$2^{n/2}$	2
$\vdots$		
p	$2^{\frac{p-2}{2}n}$	2
$\vdots$		
k	$2^{\frac{k-2}{2}n}$	2
$k+1$	$2^{(k-1/2)n}$	$2^{n/2}$
$k+2$	$2^{\frac{k}{2}n}$	$2^{n/2}$
$\vdots$		
$k+v$	$2^{\frac{k+v-2}{2}n}$	$2^{(v-2)n}$
$\vdots$		
$2k$	$2^{(k-1)n}$	$2^{(k-2)n}$
$2k+1$	$2^{(k-1/2)n}$	$2^{(k-1)n}$
$2k+2$	2^{kn}	2^{kn}

dans les tables 1.9 et 1.10 . Le détail des attaques et des calculs se trouve dans [36].

Schémas Alternés

Ils sont décrits dans la figure 1.6.

Nous résumons les complexités des différentes attaques dans les tables 1.11 et 1.12

FIGURE 1.5 – Un tour pour les schémas de Feistel de type 3

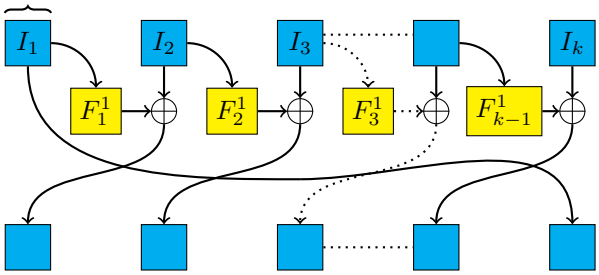


FIGURE 1.6 – Un tour pour les schémas alternés

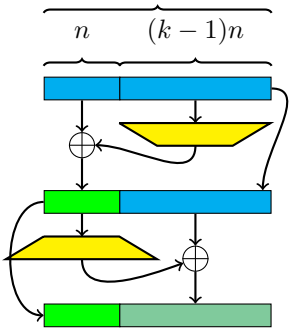


TABLE 1.9 – Complexité des attaques sur les schémas de Feistel de type 3, $k = 2\ell$

Tour	KPA	CPA-1
1	1	1
2	$2^{n/2}$	2
3	2^n	2
$\vdots$		
k	$2^{(k-1)n/2}$	2
$k+1$	$2^{\ell n}$	$2^{n/2}$
$k+2$	$2^{(\ell+1)n}$	$2^{(\ell+1)n}$
$\vdots$		
$k+t$	$2^{(\ell+t-1)n}$	$2^{(\ell+t-1)n}$
$\vdots$		
$k+\ell+1$	2^{kn}	2^{kn}

TABLE 1.10 – Complexité des attaques sur les schémas de Feistel de type 3, $k = 2\ell + 1$

Tour	KPA	CPA-1
1	1	1
2	$2^{n/2}$	2
3	2^n	2
$\vdots$		
k	$2^{(k-1)n/2}$	2
$k+1$	$2^{(\ell+1/2)n}$	$2^{n/2}$
$k+2$	$2^{(\ell+3/2)n}$	$2^{(\ell+3/2)n}$
$\vdots$		
$k+t$	$2^{(\ell+t-1/2)n}$	$2^{(\ell+t-1/2)n}$
$\vdots$		
$k+\ell+1$	$2^{(k-1/2)n}$	$2^{(k-1/2)n}$

1.0.2 Schémas avec attaques deux points et attaques rectangles

Schémas de Feistel expansifs

Description du schéma - Notations. Pour les schémas de Feistel expansifs, nous utilisons les attaques 2-points et les attaques rectangles. Pour les premiers tours, les attaques 2-points sont les plus performantes. Ensuite viennent les attaques rectangles avec différentes conditions selon le nombre de tours. Au-delà du nombre maximum de messages, c'est-à-dire 2^{kn}, lors de l'attaque de générateurs de permutations, les attaques 2-points sont de nouveau les meilleures. Les premières attaques sur les schémas de Feistel expansifs ont été introduites dans [24], puis étudiées de façon systématique dans [43] et [66]. Nous décrivons maintenant les schémas de Feistel expansifs dénotés F_k^d.
Il s'agit d'un schéma de Feistel à d tours qui produit une permutation kn bits

TABLE 1.11 – Complexités des attaques pour les schémas alternés, k pair

d	KPA	CPA-1
1	1	1
2	$2^{n/2}$	2
3	$2^{n/2}$	$2^{n/2}$
⋮		
$2p-1,\ 2 \leq p \leq k$	$2^{\frac{(p-1)n}{2}}$	$2^{\frac{(p-1)n}{2}}$
$2p,\ 2 \leq p \leq k$	$2^{\frac{pn}{2}}$	$2^{\frac{pn}{2}}$
⋮		
$2k$	$2^{\frac{kn}{2}}$	$2^{\frac{kn}{2}}$
⋮		
$2p-1,\ p \geq k$	$2^{(p-\frac{(k-1)}{2})n}$	$2^{(p-\frac{(k-1)}{2})n}$
$2p,\ p \geq k$	$2^{(p-\frac{k}{2})n}$	$2^{(p-\frac{k}{2})n}$
⋮		
$3k-3$	$2^{(k-3/2)n}$	$2^{(k-3/2)n}$
$3k-2$	$2^{(k-1)n}$	$2^{(k-1)n}$
$3k-1$	$2^{(k-1/2)n}$	$2^{(k-1/2)n}$
$3k$	2^{kn}	2^{kn}

vers kn bits. A chaque tour j, f^j représente la fonction interne de n bits vers $(k-1)n$ bits. On a $f_j = (f^j_{(1)}, f^j_{(2)}, \ldots, f^j_{(k-1)})$, où chaque fonction $f^j_{(i)}$ est définie de $\{0,1\}^n$ vers $\{0,1\}^n$. Avec une entrée $[I_1, I_2, \ldots, I_k]$, F^d_k produit une sortir $[S_1, S_2, \ldots, S_k]$ après d tours. Au tour j, les premiers n bits de la variable d'entrée sont notés X^{j-1}. Par définition, on a $I_1 = X^0$. Le calcul de $f^j(X^{j-1})$ donne $(k-1)n$ bits. On calcule ensuite le Xor de ces bits avec les $(k-1)n$ derniers bits de la variable d'entrée. Ensuite, on fait une rotation de n bits pour obtenir la variable de sortie du tour j.

Le premier tour est représenté dans la Figure 1.7

FIGURE 1.7 – Premier tour d'un schéma de Feistel expansif

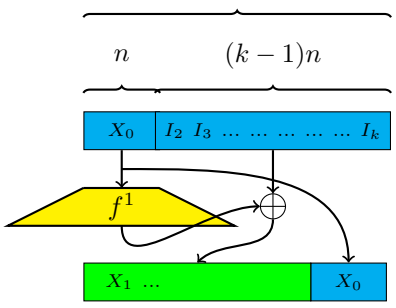


TABLE 1.12 – Complexités des attaques pour les schémas alternés, k impair

Tour	KPA	CPA-1
1	1	1
2	$2^{n/2}$	2
3	$2^{n/2}$	$2^{n/2}$
$\vdots$		
$2p-1,\ 2 \leq p \leq k$	$2^{\frac{(p-1)n}{2}}$	$2^{\frac{(p-1)n}{2}}$
$2p,\ 2 \leq p \leq k$	$2^{\frac{pn}{2}}$	$2^{\frac{pn}{2}}$
$\vdots$		
$2k$	$2^{\frac{kn}{2}}$	$2^{\frac{kn}{2}}$
$\vdots$		
$2p-1,\ p \geq k$	$2^{(p-\frac{(k-1)}{2})n}$	$2^{(p-\frac{(k-1)}{2})n}$
$2p,\ p \geq k$	$2^{(p-\frac{k}{2})n}$	$2^{(p-\frac{k}{2})n}$
$\vdots$		
$3k-3$	$2^{(k-1)n}$	$2^{(k-1)n}$
$3k-2$	$2^{(k-1/2)n}$	$2^{(k-1/2)n}$
$3k-1$	$2^{(k-1/2)n}$	$2^{(k-1/2)n}$

Nous avons

$$\begin{aligned} X^0 &= I_1 \\ X^1 &= I_2 \oplus f^1_{(1)}(I_1) \\ X^2 &= I_3 \oplus f^1_{(2)}(I_1) \oplus f^2_{(1)}(X^1) \\ X^3 &= I_4 \oplus f^1_{(3)}(I_1) \oplus f^2_{(2)}(X^1) \oplus f^3_{(1)}(X^2) \end{aligned}$$

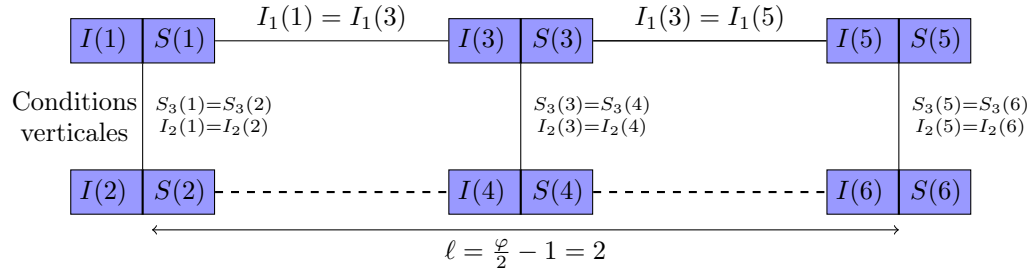
Plus généralement

$$\begin{aligned} \forall \xi < k,\ X^\xi &= I_{\xi+1} \bigoplus_{i=1}^{\xi} f^i_{(\xi-i+1)}(X^{i-1}) \\ \forall \xi \geq 0,\ X^{k+\xi} &= X^\xi \bigoplus_{i=2}^{k} f^{\xi+i}_{(k-i+1)}(X^{\xi+i-1}) \end{aligned}$$

Après d tours ($d \geq k+1$), nous pouvons exprimer la sortie $[S_1, S_2, \ldots, S_k]$ à l'aide des variables internes X^j :

$$\begin{aligned} S_k &= X^{d-1} \\ S_{k-1} &= X^{d-2} \oplus f^d_{(k-1)}(X^{d-1}) \\ S_{k-2} &= X^{d-3} \oplus f^{d-1}_{(k-1)}(X^{d-2}) \oplus f^d_{(k-2)}(X^{d-1}) \\ &\ldots \\ S_\xi &= X^{d-1-k+\xi} \bigoplus_{i=d-k+\xi}^{d-1} f^{i+1}_{(\xi+d-i-1)}(X^i) \\ &\ldots \\ S_1 &= X^{d-k} \bigoplus_{i=d-k+1}^{d-1} f^{i+1}_{(d-i)}(X^i) \end{aligned}$$

FIGURE 1.8 – Exemples d'égalités pour $\varphi = 6$



Nous notons également $[M_1^p, M_2^p, \ldots, M_k^p]$ la sortie après p tours. En particulier $M_1^p = X^p$, et pour tout i $i \in \{1, 2, \ldots, k\}$ $M_i^0 = I_i$ et $M_i^d = S_i$.

Dans les attaques, nous utilisons des ensembles de « points ». Un « point » est une paire de texte clair/chiffré. Le nombre total de points utilisés donne la complexité de l'attaque. Étant donné un ensemble de points, nous choisissons tous les φ-uplets de points distincts $P(1)$, $P(2)$ … $P(\varphi)$, et nous comptons combien d'entre eux vérifient des conditions données. Un exemple est donné dans la Figure 1.8.

Nous décrivons maintenant une attaque à partir d'un chemin différentiel. Nous expliquons ainsi pourquoi le nombre de φ qui satisfont les conditions données est plus important dans le cas d'un schéma F_k^d que dans le cas d'une permutation aléatoire. Dans les chemins différentiels, nous imposons des conditions pour garantir la propagation des égalités. Il y a différents types d'égalités qui sont définies de la façon suivante.

Définition 2. On pose $\ell = \frac{\varphi}{2} - 1$ et on considère le p-ième tour.

1. Les « égalités horizontales » sur la partie M_i de la sortie M sont définies par
$$M_i^p(1) = M_i^p(3) = \ldots = M_i^p(\varphi - 1)$$
et
$$M_i^p(2) = M_i^p(4) = \ldots = M_i^p(\varphi)$$
2. Les « égalités verticales » sur la partie M_i sont définies par
$$\forall j,\ 0 \leq j \leq \ell,\ M_i^p(2j+1) = M_i^p(2j+2)$$
3. Les « égalités différentielles » sur la partie M_i sont définies par
$$\forall j,\ 0 \leq j \leq \ell - 1,\ M_i^p(2j+1) \oplus M_i^p(2j+2) = M_i^p(2j+3) \oplus M_i^p(2j+4)$$

On remarque que lorsque l'on a des égalités différentielles, pour obtenir des égalités horizontales, il est suffisant d'avoir uniquement la première suite d'égalités. De même, pour les égalités verticales, il suffit d'avoir la première égalité. Lorsque nous imposons des égalités, nous parlons alors de **conditions** (elles sont vérifiée avec une probabilité d'environ $\frac{1}{2^n}$). Ces conditions vont permettre à certaines égalités d'être vérifiées avec une probabilité égale à 1. Sur

les variables d'entrés, nous imposerons toujours de conditions différentielles et soit des conditions horizontales, soit des conditions verticales. Sur les variables internes, nous obtiendrons des égalités horizontales ou verticales et nous pourrons imposer d'autres conditions horizontales ou verticales. Les nouvelles conditions permettront de garder des égalités différentielles. Cependant, quand nous ajoutons des conditions, il est important de ne pas ajouter trop de conditions, ce qui conduirait à des impossibilités. Finalement, le nombre de φ-uplets vérifiant les conditions données sera plus important dans le cas d'un schéma que dans le cas d'une permutation aléatoire. En effet, dans le cas d'une permutation aléatoire, les égalités apparaissent par hasard, alors que dans le cas d'un schéma, elles apparaissent pas hasard mais aussi grâce au chemin différentiel.

La table 1.13 illustre une attaque sur le schéma F_3^6.

TABLE 1.13 – Attaque sur F_3^6

i (round)	$M_1^i(2j+1) \oplus M_1^i(2j+2)$	$M_2^i(2j+1) \oplus M_2^i(2j+2)$	$M_3^i(2j+1) \oplus M_3^i(2j+2)$
0	**0**	**0**	Δ_1
1	0	Δ_1	0
2	$\bullet\Delta_1$	$\bullet 0$	0
3	$.\Delta_2$	Δ_3	$.\Delta_1$
4	**0**	$.\mathbf{0}$	$.\Delta_2$
5	0	Δ_2	0
6	Δ_2	0	0

La notation « . » (respectivement « • ») dans cette table signifie que nous avons des égalités (respectivement conditions) horizontales. La notation « 0 » (respectivement **0**) signifie que nous avons des égalités (respectivement conditions) verticales. Ces notations seront utilisés dans toutes les attaques.

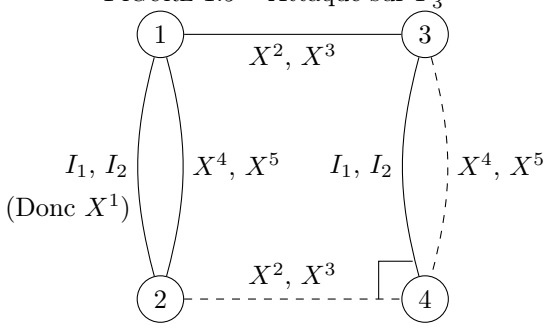
Nous définissons également les valeurs suivantes :

1. Le nombre de conditions imposées sur les variables d'entrée est noté n_I.
2. On doit ensuite imposer des des conditions sur les variables internes pour permettre la propagation des relations. On note n_X le nombre de telles conditions.
3. Le nombre de conditions imposées sur les variables d'entrée est noté n_S.

Pour cette attaque sur F_3^6, nous comptons le nombre de conditions sur les variables d'entrée. On a $n_I = 3\ell + 2$ (les trois conditions différentielles sur les trois parties du message et les 2 conditions verticales). Ensuite, selon les conditions imposés (cf table 1.13), nous comptons le nombre de conditions sur les variables internes et on obtient $n_x = 2\ell + 2$. Enfin nous regardons les conditions sur les variables de sortie, c'est-à-dire $n_S = 3\ell + 2$. Si les conditions variables d'entrée et les variables internes sont satisfaites, alors quand nous avons un schéma expansif, les égalités sur les variables de sortie seront automatiquement vérifiées. Bien sûr, les conditions sur les variables de sortie peuvent aussi être vérifiées de manière aléatoire. Donc nous pouvons prouver que le nombre de φ-uplets vérifiant les conditions données dans le cas du schéma F_3^6 est supérieur à celui obtenu dans le cas d'une permutation aléatoire. Si $n_X \leq n_S$, l'attaque réussit et dans ce cas, le nombre de messages sera $2^{\frac{n_X + n_I}{2\ell+2}} n$. Dans cette attaque, la condition

$n_X \leq n_S$ donne $\ell = 1$, d'où $\varphi = 4$ et la complexité de l'attaque est $2^{\frac{9}{4}n}$. Cette attaque est également représentée dans la figure 1.9. Cette figure illustre les

FIGURE 1.9 – Attaque sur F_3^6



définitions d'égalités horizontales et verticales. De plus, les conditions imposées sont représentées pas des traits pleins et les égalités qui sont automatiquement obtenues pas des pointillés.

L'attaque donnée plus haut est une attaque KPA. Nous expliquons maintenant comment obtenir une attaque CPA-1. On génère le nombre maximum de messages $[I_1, I_2, I_3]$ tels que $I_1 = 0$ et les $n/2$ premiers bits de I_2 soient égaux à 0. On obtient ainsi exactement $m = 2^{3n/2}$ messages. Nous cherchons maintenant le nombre de 4-uplets qui vérifient les conditions sur les variables d'entrée. Pour le premier message, il y a m possibilités. Pour le second message, il y a seulement 2^n possibilités puisque I_1 et I_2 sont imposés par le premier message. Pour le troisième point, nous avons à nouveau m possibilités, et le dernier point est alors complètement fixé. Donc il y a $m^2 \times 2^n = 2^{4n}$ 4-uplets qui satisfont les conditions d'entrée. Pour un schéma F_3^6, chaque 4-uplets va satisfaire aléatoirement les 4 conditions internes avec une probabilité égale à $1/2^{4n}$. Finalement, le nombre de 4-uplets qui vont aussi vérifier les conditions de sortie sera proche de 1. Puisqu'il y a 5 conditions de sortie, le nombre de 4-uplets vérifiant les conditions d'entrée et les conditions de sortie quand on a une permutation aléatoire sera beaucoup plus faible. Donc cette attaque CPA-1 va réussir avec une bonne probabilité. Nous avons obtenu une attaque avec une complexité en $O(2^{3n/2})$ et $O(2^{3n/2})$ messages. Cette attaque a été simulée avec un pourcentage de réussite de $57,5\%$ (cf [66]).

Génération des attaques pour $k \leq 7$. Nous avons généré toutes les attaques possibles pour $k \leq 7$. Nous expliquons le processus. Tout d'abord, on choisit une valeur de k, puis en commençant par $d = 1$, on augmente les valeurs de d jusqu'à ce qu'il n'y ait plus d'attaque possible. Toutes les attaques (ou seulement les meilleures attaques quand le nombre d'attaques est trop important) sont placées dans un fichier en fonction de k et de d. Pour trouver une attaque, il faut construire tous les chemins différentiels possibles. Nous devons tenir compte des contraintes suivantes :

- Durant un seul tour, il n'est pas possible d'avoir k conditions verticales, car cela produit une collision entre les points : $P(1) = P(3) = \cdots = P(\varphi - 1)$ et $P(2) = P(4) = \cdots = P(\varphi)$.
- Durant un seul tour, il n'est pas possible d'avoir k conditions horizontales

car cela conduit encore à une collision entre les points, c'est- à -dire $P(1) = P(2)$ et $P(3) = P(4)$ et ... $P(\varphi - 1) = P(\varphi)$.

Quand le chemin différentiel est construit, nous devons vérifier que l'attaque est acceptable. Nous devons satisfaire les cinq contraintes suivantes :

1. La complexité de l'attaque doit être inférieure au nombre total de messages possibles :
$$\frac{n_I + n_X}{2\ell + 2} \leq k$$
2. Nous devons avoir moins de conditions sur les variables internes que sur les variables de sortie : $n_X \leq n_S$
3. Si $n_X = n_S$ alors n_S doit être différent du nombre final de conditions verticales successives dans les conditions de sortie. Sinon, il est facile de montrer alors que les conditions de sortie sont équivalentes aux conditions internes. Ceci implique que les conditions de sortie n'apparaîtront pas plus souvent que pour une permutation aléatoire.
4. Le nombre d'égalités dans un chemin doit être inférieur au nombre de variables contenu dans ces égalités. De plus, nous ne gardons pas les égalités pour quand une variable apparaît une seule fois dans ces égalités.
Voici un exemple dans l'attaque sur F_3^6 donnée précédemment. Les équations sont : $f_3^{(1)}(X_2 \oplus \Delta_1) \oplus f_3^{(1)}(X_2) = \Delta_2$, $f_3^{(2)}(X_2 \oplus \Delta_1) \oplus f_3^{(2)}(X_2) = \Delta_3$,$f_4^{(1)}(X_3 \oplus \Delta_2) \oplus f_4^{(1)}(X_3) = \Delta_3$, $f_4^{(2)}(X_3 \oplus \Delta_2) \oplus f_4^{(2)}(X_3) = \Delta_1$. Nous avons 4 équations et 5 variables X_2, Δ_1, Δ_2, Δ_3, X_3. Toutes les variables sont utilisées au moins dans deux égalités donc nous ne pouvons pas simplifier.
5. Il n'y a pas de goulot d'étranglement dans les égalités, c'est-à -dire aucun sous-ensemble d'égalités ne contient plus de variables que d'égalités. Si ce n'est pas le cas, l'attaque ne marchera que pour des fonctions internes particulières (clés faibles). Ce dernier point est difficile à vérifier sans l'aide d'un ordinateur.

Finalement, les attaques sont triées en fonction de leur complexité (KPA ou CPA-1). Par exemple, il y a 71 attaques différentes sur le schéma F_3^6 et 20 attaques avec une complexité CPA-1 égale à $2^{3n/2}$.

Les différentes formes d'attaques : 2-points, $R1$, $R2$, $R3$ et $R4$

1. Les attaques « 2-points ». Ces attaques sont les meilleures attaques sur les $k+2$ premiers tours. Elles sont étudiées dans [43]. Les complexités sont données dans la table 1.19. Nous allons étudier de manière plus détaillée les autres types d'attaques.
2. Les attaques $R1$
Pour ces attaques, nous utilisons des conditions verticales sur les variables d'entrée et les variables de sortie. Les premières attaques $R1$ ont été étudiées pat Jutla [24]. Elles ont été ensuite reprises dans [43]. Enfin nous les avons généralisées dans [66] en autorisant un nombre plus important de conditions verticales sur les variables d'entrée et les variables de sortie, tout en respectant les contraintes sur les variables internes. En utilisant les notations différentielles, on a :

Entrée	I_1	...	I_r	I_{r+1}	...	I_k
Tour 0	$\mathbf{0}$	...	$\mathbf{0}$	Δ^0_{r+1}	...	Δ^0_k
Sortie	S_1	...	S_{k-v}	S_{k-v+1}	...	S_k
Tour d	Δ^d_1	...	Δ^d_{k-v}	0	...	0

On obtient $n_I = k\ell + r$, $n_X = t\ell + w$, $n_S = k\ell + v$. On a $\ell = \frac{\varphi}{2} - 1$. Le nombre de conditions verticales sur les variables d'entrée est r. On a t conditions horizontales et w conditions verticales sur les variables internes. Le nombre de conditions verticales sur les variables de sortie est donc v. On vérifie que le nombre de tours est donné par $r+t+w$. Quand $n_X \leq n_S$, on obtient facilement un condition suffisante de succès (sans calculer l'écart-type), puisque dans ce cas nous avons pour la plupart des permutations deux fois plus de solutions dans le cas d'un schéma F^d_k que dans le cas d'une permutation aléatoire. On obtient ainsi la condition : $(k-t)\ell \geq w - v$. Pour éviter les clés faibles, le nombre d'équations sur les variables internes doit être inférieur ou égal au nombre de variables internes. Il faut également éviter les goulots d'étranglement. Ceci n'était pas toujours vérifié dans [43] mais est satisfait dans [66]. Pour les attaques $R1$, il est facile de vérifier que le nombre d'équations est donné par $t(k-1)$ et le nombre de variables est $k(t+1)-r-w$. Nous obtenons la condition : $r + w \leq t + k$. La complexité d'une telle attaque est $2^{\frac{n_I+n_X}{\varphi}n}$. Ceci implique $\frac{n_I+n_X}{\varphi} \leq k$, i.e. $\frac{(k+t)\ell+r+w}{2\ell+2} \leq k$.

3. Les attaques $R2$
   Pour les attaques $R2$, nous avons des conditions horizontales sur les variables d'entrée et des conditions verticales sur les variables de sortie. De nouveau, ces attaques sont plus générales que celles du même nom introduites dans [43] puisqu'il y a plus de conditions horizontales sur les variables d'entrée et plus de conditions verticales sur les variables de sortie. Les attaques $R2$ sont décrites dans le tableau suivant :

Entrées	I_1	...	I_u	I_{u+1}	...	I_k
Tour 0	$\bullet\Delta^0_1$	...	$\bullet\Delta^0_u$	Δ^0_{u+1}	...	Δ^0_k
Sorties	S_1	...	S_{k-v}	S_{k-v+1}	...	S_k
Tour d	Δ^d_1	...	Δ^d_{k-v}	0	...	0

   On a $n_I = (k+u)\ell$, $n_X = t\ell + w$, $n_S = k\ell + v$. Le nombre de conditions horizontales sur les variables d'entrée est noté u. Le nombre de tours est donné par $u + t + w$. La condition $n_X \leq n_S$ est équivalente à $(k-t)\ell \geq w - v$. Pour les attaques $R2$, on vérifie facilement que le nombre de relations est égale à $(t+1)(k-1)$ et le nombre de variables est $k(t+2)-w$. On obtient alors la conditions : $w \leq t + k + 1$. La complexité d'une telle attaque est $2^{\frac{n_I+n_X}{\varphi}n}$. Ceci implique $\frac{n_I+n_X}{\varphi} \leq k$, c'est-à-dire $\frac{(k+t+u)\ell+w}{2\ell+2} \leq k$.

4. Les attaques $R3$ et $R4$
   Nous décrivons rapidement les attaques $R3$ et $R4$. On obtient comme précédemment le nombre de tours ; le nombre de de conditions sur les variables internes et le nombre de variables internes.
   Pour les attaques $R3$, on a des conditions verticales sur les variables d'entrés et des conditions verticales sur les variables de sortie :

Entrée	I_1	...	I_r	I_{r+1}	...	I_k
Tour 0	**0**	...	**0**	Δ^0_{r+1}	...	Δ^0_k
Sortie	S_1	...	S_{k-s}	S_{k-s+1}	...	S_k
Tour d	Δ^d_1	...	Δ^d_{k-s}	$.\Delta^d_{k-s+1}$	...	$.\Delta^d_k$

et $n_I = k\ell + r$, $n_X = t\ell + w$, $n_S = (k+s)\ell$.
Dans les attaques $R4$, nous avons des conditions horizontales que les variables d'entrée et les variables de sortie.

Entrée	I_1	...	I_u	I_{u+1}	...	I_k
Tour 0	$\bullet\Delta^0_1$	...	$\bullet\Delta^0_u$	Δ^0_{u+1}	...	Δ^0_k
Sortie	S_1	...	S_{k-s}	S_{k-s+1}	...	S_k
Tour d	Δ^d_1	...	Δ^d_{k-s}	$.\Delta^d_{k-s+1}$	...	$.\Delta^d_k$

et $n_I = (k+u)\ell$, $n_X = t\ell + w$, $n_S = (k+s)\ell$.

Les attaques KPA Nous avons généré toutes les attaques possibles (« 2-points », $R1$, $R2$, $R3$ et $R4$) pour $k \leq 7$. Nous avons ainsi obtenu les meilleures attaques pour $k \leq 7$ et nous avons ensuite généralisé les résultats. Pour $1 \leq d \leq k+2$, ce sont les attaques 2-points qui sont les meilleures. Ensuite, ce sont les attaques $R2$ qui donnent les meilleurs résultats. A chaque tour, il y a même plusieurs possibilités d'attaques $R2$ avec des performances identiques. Remarquons que, pour les attaques KPA, il y a une symétrie entre les attaques $R2$ et les attaques $R3$. Pour certains tours, il existe parfois des attaques $R1$ avec la même complexité mais demandant un nombre de points plus important. Enfin les attaques $R4$ sont généralement moins bonnes. Le tableau 1.14 donne les meilleures attaques KPA pour $d \geq k+3$. On peut vérifier que les conditions données précédemment sont satisfaites. Les attaques proposées sont celles qui demandent le moins de points.

TABLE 1.14 – Meilleures attaques KPA obtenues sur F^d_k, pour $d \geq k+3$

d	n_I	n_X	n_S	ℓ	Complexité
$k+2q \in [k{+}3, 2k{-}2]$	$(2k{-}1)\ell$	$q\ell + q + 1$	$k\ell{+}q{+}1$	1	$2^{\frac{k+d}{4}n}$
$k+2q+1 \in [k{+}3, 2k{-}2]$	$(2k{-}1)\ell$	$q\ell + q + 2$	$k\ell{+}q{+}2$	1	$2^{\frac{k+d}{4}n}$
$k+2q \in [2k{-}1, 3k{-}2]$	$(2k{-}1)\ell$	$(q - \lfloor\frac{k-1}{2}\rfloor)\ell + q + \lfloor\frac{k+1}{2}\rfloor$	$k\ell{+}k{-}1$	1	$2^{\frac{k+d}{4}n}$
$k+2q+1 \in [2k{-}1, 3k{-}2]$	$(2k{-}1)\ell$	$(q - \lfloor\frac{k-3}{2}\rfloor)\ell + q + \lfloor\frac{k+1}{2}\rfloor$	$k\ell{+}k{-}1$	1	$2^{\frac{k+d}{4}n}$
$3k-1$	$k\ell{+}\ell$	$k\ell + 2k - 1$	$k\ell{+}k{-}1$	k	$2^{(k-\frac{1}{2k+2})n}$

Remarque : il est aussi possible de construire des attaques avec des conditions verticales et horizontales sue les entrées et les sortie. Ces attaques ne sont généralement pas aussi performantes que les attaques 2 points, $R1$, $R2$, $R3$ ou $R4$.

Passage des attaques KPA aux attaques CPA-1 Toutes les attaques KPA peuvent être transformées en attaques CPA-1. Nous expliquons la méthode

pour les attaques dans lesquelles les variables d'entrée vérifient des conditions verticales (attaques $R1$ ou $R3$). On étudie de manière identique les attaques pour lesquelles nous avons de conditions horizontales en entrée (attaques $R2$ ou $R4$). Le cas où on acceptent des conditions verticales et horizontales est également étudié dans [66].

Dans le cas où l'on considère une attaque de type $R1$ ou $R3$, La meilleure façon pour construire les attaques est de garder une partie des bits d'entrée constants (par exemple égaux à 0), et de considérer toutes les possibilités pour les bits restants. On note r le nombre de conditions verticales sur les variables d'entrée, b le nombre de bits pouvant varier parmi les rn premiers bits et β le nombre de bits pouvant varier parmi les derniers $(k-r)n$ bits. On a donc $0 \leq b \leq rn$ et $0 \leq \beta \leq (k-r)n$, et ceci permet de générer $2^{b+\beta}$ points (paires de texte clair/chiffré). On compte ensuite combien de φ-uplets $M^0(1)$, ..., $M^0(\varphi)$ de points vérifient les conditions d'entrée. Pour $M^0(1)$ nous avons $2^{b+\beta}$ possibilités, pour $M^0(2)$ seulement $2^\beta - 1 \approx 2^\beta$ possibilités, puisque les rn premiers bits sont imposés par $M^0(1)$. Pour $M^0(3)$ nous avons à nouveau $2^{b+\beta} - 2 \approx 2^{b+\beta}$ possibilités. Pour $M^0(4)$ seulement une possibilité : $M^0(4) = M^0(3) \oplus (M^0(1) \oplus M^0(2))$. Nous continuons ainsi jusqu'à ce que nous arrivions aux deux derniers points. Pour $M^0(\varphi - 1)$ nous avons à nouveau environ $2^{b+\beta}$ possibilités, et pour $M^0(\varphi)$ seulement une possibilité. Donc le nombre total de φ-uplets est $\left(2^{b+\beta}\right)^{\varphi/2} \times 2^\beta = 2^{(b+\beta)(\ell+1)+\beta}$. La complexité de cette CPA-1 est égale à $2^{b+\beta}$. Ce nombre doit être le plus petit possible, et en même temps, on veut pouvoir générer le maximum de φ uplets satisfaisant les conditions d'entrée. β doit être aussi grand que possible. Pour chaque φ uplets, la probabilité que les conditions internes soient satisfaites est $1/2^{n_X \cdot n}$. Donc pour que ces conditions puissent raisonnablement être réalisées, nous devons avoir $(b+\beta)(\ell+1)+\beta = n_X \cdot n$. Si $b = 0$ on obtient $\beta = \frac{n_X}{\ell+2} n$. Mais ceci n'est possible que si l'on a $\beta \leq (k-r)n$, c'est- à-dire si $\frac{n_X}{\ell+2} \leq k-r$. Si $\frac{n_X}{\ell+2} > k-r$, on doit alors prendre la plus grande valeur possible pour β : $\beta = (k-r)n$ et cela donne une CPA-1 dont la complexité est égale à $2^{b+\beta} = 2^{\frac{n_X-k+r}{\ell+1} n}$. Le tableau 1.15 résume les différents cas.

TABLE 1.15 – Construction des attaques CPA-1 à partir des attaques KPA

Conditions		$\log_{2^n}(CPA)$
$u = 0$	$\frac{n_X}{\ell+2} \leq k-r$	$\frac{n_X}{\ell+2}$
r conditions verticales	$\frac{n_X}{\ell+2} > k-r$	$\frac{n_X - k + r}{\ell+1}$
$r = 0$	$\frac{n_X}{\ell+2} \leq k-u$	$\frac{n_X}{\ell+2}$
u conditions horizontales	$\frac{n_X}{\ell+2} > k-u$	$\frac{n_X - \ell(k-u)}{2}$

Les attaques CPA-1 pour $d \geq k+3$. Simulation

1. Description des attaques.

Pour $k \leq 7$, nous avons les meilleures attaques possibles. Nous avons ensuite généralisé ces attaques pour tout k. Si on compare les complexités obtenues dans [43] et [66], les complexités sont meilleures dans [66] sauf pour $3k-1$ tours mais l'attaque construite dans [66] évite les goulots d'étranglement. La plupart du temps, les meilleures attaques CPA-1 sont des attaques $R2$. Cependant, il peut exister des attaques $R1$ avec la même complexité. L'étude effectuée pour construire les attaques CPA-1 permet d'obtenir les meilleures CPA-1. Dans le tableau 1.16 nous donnons un exemple d'attaque pour chaque tour. Parfois, les mêmes conditions sur les entrées et les sorties peuvent mener à différentes attaques de même complexité, car les conditions sur les variables internes peuvent être réparties de plusieurs façons à l'intérieur de l'attaque, tant que l'on respecte les conditions entre le nombre de variables internes et le nombre d'équations à chaque étape. On remarque, à l'aide des tables 1.14 et 1.16, que les meilleures attaques CPA-1 ne proviennent pas toujours des meilleures attaques KPA.

TABLE 1.16 – Meilleures attaques CPA-1 sur les schémas F_k^d, pour tout $k \geq 3$

d	n_I	n_X	n_S	ℓ	Complexité
$k+3$	$k\ell+(k-1)\ell$	$\ell+3$	$k\ell+1$	1	$2^{\frac{3n}{2}}$
$k+4$	$k\ell+(k-2)\ell$	$2\ell+4$	$k\ell+2$	1	2^{2n}
$k+5$	$k\ell+(k-2)\ell$	$2\ell+5$	$k\ell+2$	1	$2^{5n/2}$
$k+2q \in [k+6, 3k-4]$	$k\ell+(k-q)\ell$	$(q-1)\ell+2q+1$	$k\ell+1$	$q-1$	$2^{\frac{q^2+2}{q+1}n}$
$k+2q+1 \in [k+7, 3k-5]$	$k\ell+(k-q)\ell$	$(q-1)\ell+2q+2$	$k\ell+1$	$q-1$	$2^{\frac{q^2+3}{q+1}n}$
$3k-3$	$k\ell+\ell$	$(k-2)\ell+2k-2$	$k\ell+1$	$k-1$	$2^{\frac{(k-1)k}{k+1}n}$
$3k-2$	$k\ell+(k-1)\ell$	$(k-\lfloor\frac{k}{2}\rfloor)\ell+2k-\lfloor\frac{k-1}{2}\rfloor$	$k\ell+k-1$	1	$2^{(k-1)n}$
$3k-1$	$k\ell+\ell$	$(k-1)\ell+2k-1$	$k\ell+k-1$	k	$2^{(k-\frac{1}{2})n}$

Remarque. Pour $d = k+3, k+4, k+5$ et $3k-2$, il existe des attaques $R1$ avec la même complexité et qui utilisent le même nombre de points.

2. L'attaque CPA-1 $R2$ sur F_k^{3k-1}.
   Nous avons effectué une simulation de notre meilleure attaque. Nous donnons ici les entrées et les sorties.

	I_1	I_2	...	I_k		S_1	S_2	...	S_k
Tour 0 0	$\bullet\Delta_1^0$	Δ_2^0	...	Δ_k^0	Tour d	Δ_1^d	**0**	...	**0**

   Nous avons plusieurs chemins différentiels possibles avec les mêmes conditions d'entrée et de sortie. Par exemple, pour le schéma F_3^8, il y a deux chemins possibles qui sont montrés ci-dessous.

   Pour $k \leq 7$, nous avons compté le nombre de chemin différentiels possibles dans l'attaque $R2$ sur F_k^{3k-1} :

k	3	4	5	6	7
# de chemins	2	8	27	89	296

3. Les résultats expérimentaux pour les attaques sur les schémas F_k^{3k-1}.

TABLE 1.17 – Chemins différentiels pour les attaques $R2$ sur F_3^8, $\varphi = 8$

	0	$\bullet\Delta_1$	Δ_2	Δ_3		0	$\bullet\Delta_1$	Δ_2	Δ_3
	1	**0**	**0**	$.\Delta_1$		1	**0**	Δ_4	$.\Delta_1$
	2	0	Δ_1	0		2	$\bullet\Delta_4$	Δ_1	0
	3	$\bullet\Delta_1$	0	0		3	**0**	**0**	$.\Delta_4$
Chemin 1 :	4	**0**	Δ_4	$.\Delta_1$	Chemin 2 :	4	0	Δ_4	0
	5	$\bullet\Delta_4$	Δ_1	0		5	$\bullet\Delta_4$	0	0
	6	**0**	**0**	$.\Delta_4$		6	**0**	**0**	$.\Delta_4$
	7	0	Δ_4	0		7	0	Δ_4	0
	8	Δ_4	0	0		8	Δ_4	0	0

Nous avons simulé ces attaques CPA-1. Pour chaque simulation, nous générons un schéma de Feistel aléatoire à 20 tours et un schéma F_k^{3k-1}. Pour les deux schémas, nous calculons $2^{(k-1/2)n}$ paires de texte clair/chiffré en faisant varier $(k-1/2)n$ derniers bits. Ensuite, nous sélectionnons tous les couples qui satisfont les conditions d'entrée et de sortie. Nous trions ces couples de points pour pouvoir compter combien de φ uplets de points satisfont les conditions d'entrée et de sortie. Si nous avons trouvé q couples de points satisfaisant les conditions avec $q \geq \varphi/2$, nous comptons $\frac{q!}{(q-\varphi/2)!}$ φ uplets, puisque c'est le nombre φ uplets que nous pouvons obtenir à partir des ces points en changeant la position des couples de points. Pour finir, nous comparons le nombre trouvé pour chaque permutation. La plupart du temps, nous sommes capables de distinguer les deux permutations considérées (Table 1.18).

TABLE 1.18 – Résultats expérimentaux pour F_k^{3k-1}

n	k	kn	% de succès	% de fausses alarme	# d'itérations
2	3	6	29,09%	0,35%	100000
2	4	8	61,6%	0,06%	10000
2	5	10	98,37%	0%	10000
2	6	12	99,99%	0%	10000
2	7	14	100%	0%	10000
2	8	16	100%	0%	1000
2	9	18	100%	0%	500
2	10	20	100%	0%	100
4	3	12	21,15%	1,12%	10000
4	4	16	42,5%	0%	1000
4	5	20	93%	0%	100
4	6	24	100%	0%	100
6	3	18	8%	1,2%	500
8	3	24	2%	0%	100

Table des complexités Nous donnons dans la table 1.19 les complexité de nos meilleures attaques KPA et CPA-1.

TABLE 1.19 – Meilleures attaques 2-points et rectangles sur les F_k^d, pour tout $k \geq 3$.

	KPA	CPA-1
F_k^1	1	1
F_k^2	$2^{\frac{n}{2}}$, 2-points	2
F_k^3	2^n, 2-points	2
F_k^d, $2 \leq d \leq k$	$2^{\frac{d-1}{2}n}$,2-points	2
F_k^{k+1}	$2^{\frac{k}{2}n}$, 2-points	$2^{\frac{n}{2}}$, 2-points
F_k^{k+2}	$2^{\frac{k+1}{2}n}$, 2-points	2^n, 2-points
F_k^{k+3}	$2^{\frac{2k+3}{4}n}$, R2, R3	$2^{3n/2}$, R2
F_k^{k+4}	$2^{\frac{k+2}{2}n}$, R1, R2, R3	2^{2n}, R2
F_k^{k+5}	$2^{\frac{2k+5}{4}n}$, R2, R3	$2^{5n/2}$, R2
$\vdots$	$\vdots$	$\vdots$
F_k^d, d= $k+2q$, $3 \leq q \leq k-2$	$2^{\frac{d+k}{4}n}$, R1, R2, R3	$2^{\frac{q^2+2}{q+1}n}$, R2
F_k^d, d= $k+2q+1$, $3 \leq q \leq k-3$	$2^{\frac{d+k}{4}n}$, R2, R3	$2^{\frac{q^2+3}{q+1}n}$, R2
$\vdots$	$\vdots$	$\vdots$
F_k^{3k-3}	$2^{(k-\frac{3}{4})n}$, R2, R3	$2^{\frac{(k-1)k}{k+1}n}$, R2
F_k^{3k-2}	$2^{(k-\frac{1}{2})n}$, R1, R2, R3	$2^{(k-1)n}$, R2
F_k^{3k-1}	$2^{(k-\frac{1}{2k+2})n}$, R2, R3	$2^{(k-\frac{1}{2})n}$, R2

Chapitre 2

Théorie du miroir

2.1 Introduction

La théorie du miroir est par définition (cf [46]) la théorie qui calcule ou qui donne un ordre de grandeur du nombre de solutions d'un système d'égalités linéaires et de non-égalités linéaires dans des groupes finis. Les calculs de variance vu dans le chapitre précédent sont des bons exemples de ce genre de systèmes.

Résoudre des systèmes linéaires est peut être la principale activité des ordinateurs aujourd'hui. On s'en sert pour les prévisions météorologiques, les neutroniques ainsi que d'autres expériences liées à la physique, les dernières étapes d'algorithmes de factorisation, etc. Ces résolutions sont en général effectuées grâce à des algorithmes améliorés de Gauss, et le nombre de solutions dans le cas où le groupe est fini est trouvé en analysant le nombre d'équations linéairement indépendantes. Tous ces algorithmes sont polynomiaux. Lorsque l'on passe à des systèmes de degré 2 ou plus le problème devient plus complexe, comme on peut le voir au chapitre 7.2. Par exemple, le problème quadratique à plusieurs variables (MQ) est NP-complet sur tous les corps finis (cf [15]).

Dans ce chapitre, on va considérer des systèmes où l'on garde le premier degré mais où l'on s'autorise à mettre des non-égalités. Prenons un exemple :

$$\begin{cases} P_1 \oplus P_2 &= \lambda_1 \\ P_2 \oplus P_3 &= \lambda_2 \\ P_1 &\neq P_3 \end{cases}$$

Puisque parler de la théorie qui traite du «nombre de solutions d'un système linéaire d'égalités et de non-égalités dans un groupe fini» est un titre un peu long, on utilise le raccourci imagé : « Théorie du miroir » (cf [46]).

Dans ce chapitre un peu court pour voir tout ce qui a été fait sur cette théorie, nous allons nous intéresser à deux cas particuliers fondamentaux. Dans les deux cas, chaque équation du système aura exactement 2 variables. Le premier s'appelle le théorème $P_i \oplus P_j$, et toutes les variables employées doivent être distinctes deux à deux. Une façon de fabriquer ce genre de système est de définir une fonction ainsi : $f(x) = g(x||0) \oplus g(x||1)$ où g est une permutation. Le deuxième cas particulier concerne la somme de deux bijections, $f(x) = g(x) \oplus h(x)$. C'est ce que nous utiliserons dans la partie suivante pour la fonction de hachage CRUNCH. Dans les deux cas il s'agit d'écrire des preuves de sécurité qui garantissent qu'en quelque sorte la fonction fabriquée est difficilement distinguable d'une fonction aléatoire. De nombreux travaux ont été faits dans le cas où l'on connaît qu'un nombre restreint de valeurs de la fonction f.

2.2 Notations, premier exemple

- $N \in \mathbb{N}^*$, $I_N = \{0,1\}^N$, et $M = 2^N$ (cardinal de I_N). I_N^* est I_N privé de l'élément nul.
- F_N est l'ensemble des applications de I_N dans lui même, $|F_N| = 2^{N \cdot 2^N} = M^M$.
- B_N est l'ensemble des bijections de I_N dans lui-même, $|B_N| = (2^N)! = M!$.
- $x \in_R A$ signifie que x est choisi aléatoirement dans A suivant une loi uniforme.

Nous commençons par donner un exemple d'un système combinant égalités et non-égalités. Cela concerne le Xor de deux bijections. Si $(f,g) \in {B_n}^2$, alors la fonction $h \in F_n$ définie par $h = f \oplus g$ vérifie $\oplus_{i \in I_n} h(i) = 0$. La réciproque est également vraie : si $h \in F_n$ vérifie $\bigoplus_{i \in I_n} h(i) = 0$, alors il existe $(f,g) \in {B_n}^2$ tels que $h = f \oplus g$. Ce résultat a été démontré indépendamment par Hall en 1952 ([23]), et Salzborn et Szekeres en 1979 ([13]. De plus on a la généralisation sur tout groupe fini G. Cependant, la démonstration nous donne l'existence et la construction d'une solution (f,g) mais il n'y a pas d'estimation du nombre de solutions.
Quand $h \in F_n$, on notera H_h le nombre de couples $(f,g) \in B_n^2$ tels que $h = f \oplus g$. Il est évident que la valeur maximum de $\{H_h,\ h \in F_n\}$ que l'on note H_{max} est obtenue lorsque h est constante et vaut $|B_n| = (2^N)!$, en effet lorsque f est fixée, g vaut forcément $g = h \oplus f$. La valeur minimum non nulle de H_h sera notée H_{min} (i.e. $H_{min} = \inf\{H_h,\ h \in F_n,\ \oplus_{i \in I_n} h(i) = 0\}$), nous avons alors les conjectures suivantes :

Conjecture 1. *H_{min} est obtenu pour les fonctions h qui sont des permutations.*

Conjecture 2. $H_{min} \geq \frac{|B_n|^2}{|F_n|} = \frac{(2^N)!^2}{2^{N \cdot 2^N}} = \frac{M!^2}{M^M}$.

Ici le système que l'on considère est donné par :

$$\begin{cases} P_1 \oplus P'_1 & = & \lambda_1 \\ P_2 \oplus P'_2 & = & \lambda_2 \\ \vdots & & \\ P_M \oplus P'_M & = & \lambda_M \end{cases}$$

avec les conditions suivantes :

$$\forall i \neq j,\ (P_i \neq P_j \text{ et } P'_i \neq P'_j) \text{ et } \bigoplus_i \lambda_i = 0$$

Nous voyons que ce système est un système de la «théorie du miroir», i.e. un ensemble d'équations linéaires et de non-équations linéaires.
Remarque 1. Ici $H^* = \frac{(2^N)!^2}{2^{N \cdot 2^N}} = \frac{M!^2}{M^M}$ peut être vu comme la valeur moyenne pour toutes les fonctions $h \in F_n$, c'est à dire en prenant en compte les fonctions h telles que $H_h = 0$.
Remarque 2. Si nous notons H^{**} la valeur moyenne pour les fonctions $h \in F_n$ telles que $H_h \neq 0$, alors $H^{**} = \frac{|B_n|^2}{|F_n|} \cdot 2^N = 2^N H^*$.
Remarque 3. Les conjectures 1 et 2 sont des problèmes ouverts (2012). Les simulations faites sur des petits groupes finis (ordre inférieur à 17) semblent confirmer ces conjectures. Nous indiquons plus loin comment ont été faites les simulations et nous donnons également quelques détails des calculs.

2.3 Les résultats obtenus dans la théorie du miroir

Nous présentons ici les résultats principaux (cf [46]).

2.3.1 Définitions

Définition 3. Soit (A) un ensemble d'équations $P_i \oplus P_j = \lambda_k$, avec $P_i, P_j \in I_N$. On dit que (A) « n'a pas de cercle en P » si on ne peut pas engendrer à partir des (A) et par linéarité une équations en les λ_k. On dit aussi que les équations de (A) sont « linéairement indépendantes en P ».

On note a le nombre d'équations dans (A), et q le nombre de variables P_i dans (A). On a donc les paramètres $\lambda_1, \lambda_2, \ldots, \lambda_a$ et $a + 1 \leq q \leq 2a$

Définition 4. On dit que deux indices i et j sont « dans le même bloc » si à partir des équations de (A), on peut exprimer $P_i \oplus P_j$ en fonction de certaines valeurs parmi $\lambda_1, \lambda_2, \ldots, \lambda_a$.

Définition 5. On note ξ_{max} le nombre maximum d'indices qui sont dans un même bloc.

Exemple. Si $A = \{P_1 \oplus P_2 = \lambda_1,\ P_1 \oplus P_3 = \lambda_2,\ P_4 \oplus P_5 = \lambda_3\}$, alors nous avons deux blocs d'indices $\{1, 2, 3\}$ et $\{4, 5\}$. Donc $\xi_{max} = 3$.

Définition 6. Soit (A) un système d'équations. Quand les valeurs $\lambda_1, \lambda_2, \ldots, \lambda_a$ sont fixés, on note $h_q(A)$ le nombre de $P_1, P_2, \ldots, P_q$ distincts 2 à 2 solutions de (A). On pose aussi $H_q(A) = 2^{Na} h_q(A)$. En général, on écrira H_q au lieu de $H_q(A)$ et h_q au lieu de $h_q(A)$ quand il n'y a pas d'ambiguïté sur (A). Ces notations sont plus concises mais il ne faut pas oublier que ces valeurs diffèrent selon (A).

On note J_q le nombre de $P_1, P_2, \ldots, P_q$ distincts 2 à 2. Donc $J_q = 2^N(2^N - 1)\ldots(2^N - q + 1)$.

2.3.2 Le Théorème $P_i \oplus P_j$

Nous donnons un premier énoncé pour $\xi_{max} = 2$, puis pour tout ξ_{max}.

Théorème 2. *(« Le Théorème $P_i \oplus P_j$ pour $\xi_{max} = 2$ »)*
Soit (A) un ensemble d'équations : $P_1 \oplus P_2 = \lambda_0$, $P_3 \oplus P_4 = \lambda_1, \ldots, P_{q-1} \oplus P_q = \lambda_{q/2-1}$, où toutes les valeurs λ_i sont $\neq 0$. Si $q \ll 2^N$ (de manière plus précise, cette condition peut être écrite avec la borne explicite $q \leq \frac{2^N}{67}$), alors le nombre h_q de solutions $(P_1, \ldots, P_q)$ de (A) telles que tous les P_i $(1 \leq i \leq q)$ soient des variables deux à deux distinctes de I_N satisfait : $h_q \geq \frac{2^N(2^N-1)\ldots(2^N-q+1)}{2^{Na}}$ où $a = q/2$ est le nombre d'équations de (A). On peut aussi écrire $H_q \geq J_q$.

On rappelle $\xi_{max} = 2$ signifie que si une variable P_i est fixée, alors au plus une autre variable P_j est fixée à partir des équations de (A). Le système est donné par :

$$\begin{cases} P_1 \oplus P_2 &= \lambda_0 \\ P_3 \oplus P_4 &= \lambda_1 \\ P_5 \oplus P_6 &= \lambda_2 \\ & \vdots \\ P_{q-1} \oplus P_q &= \lambda_{a-1} \end{cases}$$

où $q = 2a$
Le théorème 2 montre que si $q \ll 2^N$ alors $h_q \geq \frac{J_q}{2^{Na}}$. On remarque que $\frac{J_q}{2^{Na}}$

représente la moyenne du nombre de solutions quand on considère toutes les 2^{Na} valeurs pour $\lambda_0, \lambda_1, \ldots, \lambda_{q/2-1}$ (y compris les valeurs $\lambda_i = 0$ pour lesquelles $h_q = 0$). De même, $\frac{J_q}{(2^N-1)^a}$ représente la moyenne du nombre de solutions quand on considère toutes les $(2^N-1)^a$ valeurs non nulles $\lambda_0, \lambda_1, \ldots, \lambda_{q/2-1}$. D'après le théorème 2 lorsque $q \ll 2^N$, et lorsque les valeurs λ_i sont compatibles par linéarité avec la condition d'avoir les P_i deux à deux (i.e. $\lambda_i \neq 0$), le nombre de solutions est toujours supérieur à la moyenne et parfois nettement supérieur à la moyenne.

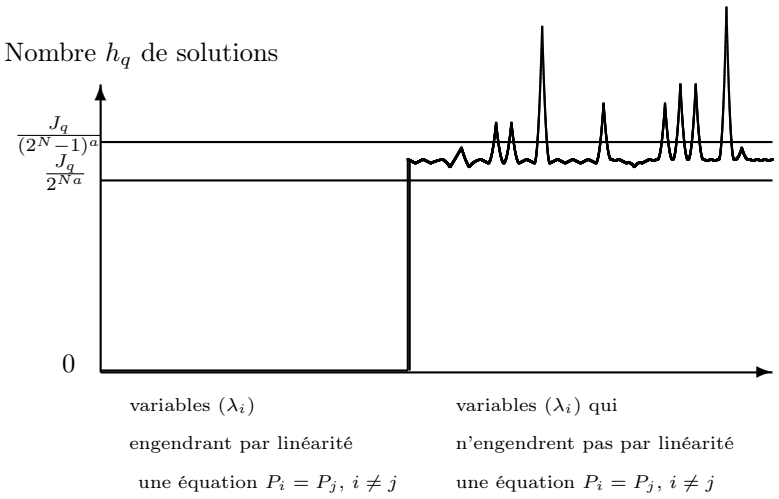


FIGURE 2.1 – Illustration de la théorie de miroir

Théorème 3. *(« Le théorème $P_i \oplus P_j$ » pour tout ξ_{max}).*
Soit (A) un ensemble de a équations à q variables $P_i \oplus P_j = \lambda_k$ tel que :

1. *(A) n' pas de cercle en P.*
2. *Il n'y a pas plus de ξ_{max} indices dans un même bloc.*
3. *Par linéarité, on ne peut pas engendrer une équation $P_i = P_j$ avec $i \neq j$ à partir de (A). (c'est-à-dire que si i et j sont dans le même bloc, alors l'expression de $P_i \oplus P_j$ à l'aide des valeurs $\lambda_1, \lambda_2, \ldots, \lambda_a$ est $\neq 0$).*

Alors : si $\xi_{max} q \ll 2^N$, on a $H_q \geq J_q$.

On peut remplacer la condition $\xi_{max} q \ll 2^N$ à l'aide de la borne explicite $(\xi_{max} - 1)q \leq \frac{2^N}{67}$.

2.3.3 Trois exemples de valeurs remarquables pour H

Exemple 1

Le théorème $P_i \oplus P_j$ affirme que lorsque $H(A) \neq 0$, avec ξ_{max} fixé, et $q \ll 2^N$, on a $H(A)$ toujours supérieur à J_q (où J_q peut être vu comme la valeur moyenne de $H(A)$ pour tous les λ_i).

Cependant, $H(A)$ est quelquefois beaucoup plus grand que cette valeur moyenne, même lorsque $\xi_{max} = 2$, comme nous allons le voir dans cet exemple.

Soit λ un élément non nul de I_n. Nous voulons calculer le nombre $h(A)$ d'éléments $P_1, P_2, \ldots, P_q$ deux à deux distincts tels que :

$$\begin{cases} P_2 &= & P_1 \oplus \lambda \\ P_4 &= & P_3 \oplus \lambda \\ P_6 &= & P_5 \oplus \lambda \\ \vdots & & \\ P_q &= & P_{q-1} \oplus \lambda \end{cases}$$

(Pour ce système on a $\xi_{max} = 2$ et tous les λ_i sont égaux).

Soit $x_1 = P_3 \oplus P_1, x_2 = P_5 \oplus P_1, \ldots, x_{q/2-1} = P_{q-1} \oplus P_1$.

Comme nous avons 2^N possibilités pour P_1, ici $h(A)$ est exactement 2^N fois le nombre de $x_1, \ldots, x_{q/2-1}$ tels que :

$x_1 \notin \{0, \lambda\}$
$x_2 \notin \{0, \lambda, x_1, x_1 \oplus \lambda\}$
$x_3 \notin \{0, \lambda, x_1, x_1 \oplus \lambda, x_2, x_2 \oplus \lambda\}$
$\vdots$
$x_{q/2-1} \notin \{0, \lambda, x_1, x_1 \oplus \lambda, x_2, x_2 \oplus \lambda, \ldots, x_{q/2-2}, x_{q/2-2} \oplus \lambda\}$

Pour x_1 nous avons exactement $2^N - 2$ possibilités. Puis, lorsque x_1 est fixé ainsi, nous avons exactement $2^N - 4$ solutions pour x_2, ensuite si (x_1, x_2) sont fixés, nous avons exactement $2^N - 6$ solutions pour x_3 etc.

Ainsi, puisque nous avons $q/2$ équations, $H(A) = 2^{(q/2)N} h(A)$ (parce que nous avons $q/2$ équations), donc nous avons :

$$H(A) = 2^{(q/2)N} 2^N (2^N - 2)(2^N - 4)(2^N - 6) \cdots (2^N - q + 2)$$

$$J(A) = 2^N (2^N - 1)(2^N - 2)(2^N - 3) \cdots (2^N - q + 1)$$

$$\frac{J(A)}{H(A)} = \left(1 - \frac{1}{2^N}\right)\left(1 - \frac{3}{2^N}\right)\left(1 - \frac{5}{2^N}\right) \cdots \left(1 - \frac{q-1}{2^N}\right)$$

Lorsque $q \gg 2^{N/2}$ (mais toujours $q \ll 2^N$) cette expression peut être arbitrairement petite. Donc $H(A)$ peut être beaucoup plus grand que J_q (même si $\xi_{max} = 2$ and $q \ll 2^N$).

Exemple 2

Cet exemple est une illustration du fait que si $q\xi_{max} \geq 2^N$, alors nous pouvons avoir $H(A) = 0$ et donc nous n'aurons plus $H(A) \geq J_q$. En effet $q < 2^N$ entraîne $J_q \neq 0$. Cela ne contredit pas le théorème $P_i \oplus P_j$, parce que dans ce théorème $\xi_{max} q << 2^N$.

Soit $a = q/2$. Soit $\lambda_2, \lambda_3, \lambda_4, \ldots \lambda_a \in I_N^*$ tels que $\lambda_i \oplus \lambda_j \neq 0$ pour tous i, j $1 \leq i < j \leq a$.

Soit $\lambda_{a+2}, \lambda_{a+3}, \ldots \lambda_q$ $q/2 - 1$ éléments de I_N^* tels que $\lambda_i \oplus \lambda_j \neq 0$ pour tous i, j $a + 1 \leq i < j \leq q$.

Dans cet exemple nous voulons évaluer le nombre h' de $(P_1, \ldots, P_q)$ tels que tous les P_i soient distincts deux à deux et vérifient :

$$\begin{cases} P_2 &= P_1 \oplus \lambda_2 \\ P_3 &= P_1 \oplus \lambda_3 \\ P_4 &= P_1 \oplus \lambda_4 \\ & \vdots \\ P_i &= P_1 \oplus \lambda_i \\ & \vdots \\ P_a &= P_1 \oplus \lambda_a \end{cases} \quad \text{et} \quad \begin{cases} P_{a+2} &= P_{a+1} \oplus \lambda_{a+2} \\ P_{a+3} &= P_{a+1} \oplus \lambda_{a+3} \\ & \vdots \\ P_j &= P_{a+1} \oplus \lambda_j \\ & \vdots \\ P_q &= P_{a+1} \oplus \lambda_q \end{cases}$$

(ici $\xi = \max(a, q-a) = q/2$ et nous avons deux blocs).

Pour $(P_1, \ldots, P_a)$ nous avons 2^N possibilités, fixons P_1 à n'importe quelle valeur. Pour P_{a+1} nous voulons que sa valeur soit différente de toutes les valeurs suivantes :

— $P_1 \oplus \lambda_i \oplus \lambda_j$, pour tout $0 \leq i \leq a$, $a+1 \leq j \leq q$ (parce que nous voulons $P_i \neq P_j$).

Puisque nous pouvons avoir $q^2 \gg 2^N$ avec $q \ll 2^N$, il peut arriver que $\lambda_i \oplus \lambda_j$ couvre toutes les valeurs de I_n, $0 \leq i \leq a$, $a+1 \leq j \leq q$. Donc nous aurons ici $H(A) = 0$ et $J_q \neq 0$.

Remarque Si $\xi = \xi_{max}$ est constant, nous avons ici, puisque $q = 2\xi$, q qui est constant et $H(A) \geq 2^{qN} \cdot \left(1 - \frac{\xi^2}{2^N}\right)$. Nous avons $J = 2^N(2^N - 1)\cdots(2^N - q + 1) = 2^N(2^N - 1)\cdots(2^N - 2\xi + 1)$. Puisque q dans cet exemple est constant, nous pouvons écrire $J = 2^{qN}\left(1 + \frac{-q^2+q}{2\cdot 2^N} + o\left(\frac{1}{2^N}\right)\right)$ et ainsi $J = 2^{qN}\left(1 + \frac{1}{2^N}(-2\xi^2 + \xi) + o\left(\frac{1}{2^N}\right)\right)$. Nous voyons que pour un N suffisamment grand, $H \geq J$, comme d'habitude.

Exemple 3

Cet exemple est une illustration du fait que, lorsque nous calculons H par induction, il est quelquefois plus simple d'ordonner les équations en mettant en premier les équations comportant des nouvelles valeurs de λ_i.

Soit λ et λ' deux éléments de I_N^* ; avec $\lambda \neq \lambda'$. Soit q entier. Dans cet exemple nous voulons évaluer le nombre h' de $(P_1, \ldots, P_q)$ deux à deux distincts tels que :

$$\begin{cases} P_2 = P_1 \oplus \lambda & , & P_4 = P_3 \oplus \lambda' \\ P_6 = P_5 \oplus \lambda & , & P_8 = P_7 \oplus \lambda' \\ \vdots & & \vdots \\ P_{q-2} = P_{q-3} \oplus \lambda & , & P_q = P_{q-1} \oplus \lambda' \\ P_{q+2} = P_{q+1} \oplus \lambda \oplus \lambda' & & \end{cases}$$

Ici la dernière équation est spéciale, nous avons $\lambda \oplus \lambda'$ au lieu de l'un des deux).

Lorsque $(P_1, \ldots, P_q)$ sont des solutions fixées de toutes ces équations, nous devons trouver une valeur pour P_{q+1} qui n'est pas parmi les suivantes :

— $P_1, P_1 \oplus \lambda, P_3, P_3 \oplus \lambda', P_5, P_5 \oplus \lambda, \ldots, P_{q-1}, P_{q-1} \oplus \lambda'$

— $P_1 \oplus \lambda \oplus \lambda', P_1 \oplus \lambda', P_3 \oplus \lambda \oplus \lambda', P_3 \oplus \lambda, \ldots, P_{q-1} \oplus \lambda \oplus \lambda', P_{q-1} \oplus \lambda$

Ici $P_1 \oplus \lambda$ est toujours différent de $P_1 \oplus \lambda'$, $P_3 \oplus \lambda$ est toujours différent de $P_3 \oplus \lambda \oplus \lambda'$, etc. Par conséquent, pour P_{q+1} nous aurons à peu près $2^N - 2q + \frac{q^2}{4\cdot 2^N}$

possibilités, au lieu de $2^N - 2q + \frac{q^2}{2 \cdot 2^N}$ possibilités pour le cas général (ici nous supposons $q \ll 2^N$ mais également $q^2 \gg 2^N$).

Cependant ce problème apparaît seulement pour P_{q+1} lorsque tous les $P_1, \ldots, P_q$ sont fixés. Si nous calculons la valeur totale pour h', nous obtiendrons $H \geq J$ comme d'habitude. En fait nous aurons ici moins de possibilités pour P_{q+2} que dans le cas général mais ce sera compensé par le fait que nous avons plus de possibilités pour $P_1, \ldots, P_q$. De plus, si l'équation $P_{q+2} = P_{q+1} \oplus \lambda \oplus \lambda'$ était rencontrée dans une première ou seconde position, l'étape d'induction aurait été passée sans aucun problème.

2.3.4 Généralisations

Généralisation 1. Le théorème $P_i \oplus P_j$ dans le cas d'un groupe non commutatif. En effet, ce théorème est encore vrai sur tout groupe commutatif G au lieu de I_N. Ceci est relativement facile à vérifier puisque dans les preuves, on utilise uniquement le nombre de variables et les égalités linéaires mais pas la nature particulière de I_N. Cependant quand G n'est pas commutatif, il faut faire une analyse différente.
Généralisation 2. Le théorème $P_i \oplus P_j$ lorsque l'on considère la condition $\xi_{average} \ll 2^N$ au lieu de la condition $\xi_{max} q \ll 2^N$, c'est-à-dire que l'on considère la moyenne au lieu de la valeur maximale.
Généralisation 3. Le théorème $P_i \oplus P_j$ avec des non-égalités plus générales de la forme $[P_{i_1}, P_{j_1}, P_{k_1}] \neq [P_{i_2}, P_{j_2}, P_{k_2}]$ par exemple.
Généralisation 4. Le théorème $P_i \oplus P_j$ avec équation $P_i \oplus P_j \oplus P_k = \lambda_{ijk}$ au lieu de $P_i \oplus P_j = \lambda_{ij}$. Plus généralement, on peut aussi considérer des équations de la forme $P_{i_1} \oplus P_{i_2} \oplus \ldots \oplus P_{i_k} = \lambda_l$ avec un nombre k quelconque de variables dans chaque égalité linéaire.
Généralisation 5. Le théorème $P_i \oplus P_j$ avec des non-égalités partielles, par exemple sur les m premiers bits, $m \leq N$, i.e. $\lceil P_i \rceil \neq \lceil P_j \rceil$ au lieu de $P_i \neq P_j$, où les m premiers bits de P_i sont représentés par la notation $\lceil P_i \rceil$.

2.4 Théorème $P_i \oplus P_j$: Exemples d'applications aux preuves de sécurité en cryptographie.

Dans cette section nous donnons des exemples où le problème de sécurité peut s'exprimer comme un problème de la théorie du miroir. Les théorèmes $P_i \oplus P_j$ peuvent alors s'appliquer.

1. **Le problème de distinguer $f(x\|0) \oplus f(x\|1)$ où f est une permutation aléatoire sur N bits, d'une fonction aléatoire.**
   Ce problème a été d'abord étudié par Bellare et Implagliazzo dans [2]. Le lien avec la théorie du miroir est étudié dans [46].
2. **Schémas de Feistel classiques.**
   Les résultats sont donnés dans [47].
3. **Schémas de Feistel déséquilibrés.**
   Les résultats sont donnés dans [47].

2.5 Conjectures

Dans cette section, nous proposons différentes conjectures dans le domaine des groupes, faisant référence à différentes versions du théorème $P_i \oplus P_j$ ou bien à propos de la somme des éléments d'un groupe. Nous donnons ces conjectures pour des groupes abéliens, mais il existe aussi des versions semblables pour les groupes non abéliens.

2.5.1 Conjectures relatives à la somme de deux bijections

Dans la section 2.2, nous avons vu que $h \in F_N$ vérifiait $\oplus_{i \in I_N} h(i) = 0$ ssi il existe deux bijections $(f, g) \in B_N^2$ telles que $h = f \oplus g$ (cf [13, 23]) et nous avons donné deux conjectures :
Conjecture 1 H_{min} est obtenue pour les permutations.
Conjecture 2 $H_{min} \geq \frac{|B_n|^2}{|F_n|} = \frac{(2^N)!^2}{2^{N \cdot 2^N}}$.

2.5.2 Conjectures relatives au théorème $P_i \oplus P_j$ lorsque $\xi_{max} = 2$ et $q = M$

Le problème initial.

Nous considérons le groupe fini $G = (\mathbb{Z}/2\mathbb{Z})^N$ dont le cardinal M est pair, et nous voulons calculer le nombre H de solutions deux à deux distinctes du système, lorsque les inconnues sont P_1, P_2 ... P_M :

$$\begin{cases} P_1 \oplus P_2 &= \lambda_0 \\ P_3 \oplus P_4 &= \lambda_1 \\ & \vdots \\ P_{M-1} \oplus P_M &= \lambda_a \end{cases}$$

avec $a = \frac{M}{2} - 1$ et les valeurs $\lambda_i \in G$ fixées.

Nous remarquons qu'ici il y a $M^{M/2}$ possibilités pour les valeurs λ, et $M!$ possibilités pour $(P_1, P_2, \ldots, P_M)$. La valeur moyenne du nombre de solutions est $\dfrac{M!}{M^{M/2}}$. Alors nous posons la conjecture suivante :

Conjecture 3. $H \neq 0 \Rightarrow H \geq \frac{M!}{M^{M/2}}$.

Proposition 1. *Pour $N \geq 2$,*

$$H \neq 0 \Rightarrow \left(\sum_{i=0}^{a} \lambda_i = \bar{0} \; et \, \forall i \in \{0, \ldots, a\}, \; \lambda_i \neq \bar{0} \right).$$

Conjecture 4. *Pour $N \geq 2$,*

$$H \neq 0 \iff \left(\sum_{i=0}^{a} \lambda_i = \bar{0} \; et \, \forall i \in \{0, \ldots, a\}, \; \lambda_i \neq \bar{0} \right)$$

Proposition 2. *H est un multiple de $M \times 2^a$*

Preuve : On peut dire que deux solutions P_1, ..., P_M et P'_1, ..., P'_M sont équivalentes lorsque $P_1 \oplus P'_1 = P_2 \oplus P'_2 = C$ et pour tout i entre 2 et $M/2$ (a possibilités) on a soit $P_{2i-1} \oplus P'_{2i-1} = P_{2i} \oplus P'_{2i} = C$, soit $P_{2i} \oplus P'_{2i-1} = P_{2i-1} \oplus P'_{2i} = C$.
On trouve alors $M \times 2^a$ éléments dans chaque classe d'équivalence.
Remarque : on peut voir les classes d'équivalence comme étant à une translation près et inversion des termes pairs/impairs à partir de la seconde équation.

Généralisation du théorème $P_i \oplus P_j$.

Il y a deux façons différentes de généraliser le problème. Nous considérons un groupe abélien G avec un cardinal pair et nous pouvons choisir soit la somme, soit la différence à la place de $\oplus$. Nous supposons également que G est ordonné avec $\bar{0} < \bar{1} < \cdots < \overline{M-1}$.

Cas de la somme $P_i + P_j$.

Nous voulons calculer le nombre H de solutions deux à deux distinctes du système d'inconnues P_1, P_2 ... P_M :

$$\begin{cases} P_1 + P_2 &= \lambda_0 \\ P_3 + P_4 &= \lambda_1 \\ & \vdots \\ P_{M-1} + P_M &= \lambda_a \end{cases}$$

avec $a = \frac{M}{2} - 1$ et toutes les valeurs $\lambda_i \in G$ fixées.

Conjecture 5. $H \neq 0 \Rightarrow H \geq \frac{M!}{M^{M/2}}$.

Proposition 3. $H \neq 0 \Rightarrow \left(\sum_{i=0}^{a} \lambda_i = \sum_{x \in G} x\right)$.

Conjecture 6. *Si il existe $g \in G$ tel que $g + g \neq 0$ alors :*

$$H \neq 0 \iff \left(\sum_{i=0}^{a} \lambda_i = \sum_{x \in G} x\right)$$

Proposition 4. *H est un multiple de $2^{a+1} = 2^{M/2}$*

Cas de la différence $P_i - P_j$.

Nous voulons calculer le nombre H de solutions deux à deux distinctes du système, où les inconnues sont P_1, P_2 ... P_M :

$$\begin{cases} P_1 - P_2 &= \lambda_0 \\ P_3 - P_4 &= \lambda_1 \\ & \vdots \\ P_{M-1} - P_M &= \lambda_a \end{cases}$$

avec $a = \frac{M}{2} - 1$ et tous les $\lambda_i \in G$ fixés.

Conjecture 7. $h \neq 0 \Rightarrow h \geq \frac{M!}{M^{M/2}}$.

Proposition 5. $h \neq 0 \Rightarrow (\forall i \in \{0, \ldots, a\}, \quad \lambda_i \neq \bar{0})$.

Conjecture 8. $h \neq 0 \iff (\forall i \in \{0, \ldots, a\}, \quad \lambda_i \neq \bar{0})$

Proposition 6. *h est un multiple de M*

2.6 Somme de deux bijections

Dans cette section, contrairement au premier exemple on considère la somme de deux bijections sur un groupe commutatif fini quelconque, donc pas forcément le groupe $(\mathbb{Z}/2\mathbb{Z})^N$. Cela nécessite donc l'introduction de nouvelles notations. Certaines sont motivées par les simulations qui ont été faites sur les premiers groupes finis. L'idée est de se fixer une fonction (de somme nulle) puis de compter le nombre de décomposition. Il est clair que certains changements dans la fonction ne vont pas affecter le calcul du nombre. On va donc regrouper les fonctions dans des classes, les plus grandes possibles, afin d'accélérer la vérification.

Notations

— $(G, +)$: groupe fini commutatif de cardinal M.
— $F = G^G$: ensemble des fonctions de G dans G.
— $(B, \circ)$: groupe des bijections de G dans G.
— F_0 : sous-ensemble des fonctions de F nulles en 0.
— B_0 : sous-ensemble de B des bijections nulles en 0.
— H_f : nombre de couples $(g, h) \in B^2$ tels que $f = g + h$ où $f \in F$
— $Aut(G)$: sous-groupe de B constitué des automorphismes de G.

Rappels sur la théorie des groupes

Rappel 1 : À isomorphisme près, il existe un unique groupe de cardinal p où p est premier, il s'agit du groupe cyclique $\mathbb{Z}/p\mathbb{Z}$.
Rappel 2 : À isomorphisme près, il existe un unique groupe commutatif de cardinal pq où p et q sont des nombres premiers distincts, il s'agit du groupe cyclique $\mathbb{Z}/pq\mathbb{Z}$.

Objectif

Il s'agit de vérifier une généralisation de la conjecture 2 :

Conjecture 9. *Lorsque H_f n'est pas nul, il est plus grand que la moyenne, c'est à dire que $H^* = \frac{(M!)^2}{M^M}$.*

Ci-dessous un tableau pour donner la partie entière de ce nombre pour N petit. En gras les valeurs pour M qui est une puissance de 2.

M	2	3	4	5	6	7	8	9	10	11	12
H^*	**1**	1	**2**	4	11	39	**96**	340	1320	5580	25700

N	13	14	15	16	17
H^*	128000	684000	3900000	**23700000**	153000000

On a une condition nécessaire et suffisante pour que H_f soit non nul, donnée par le théorème suivant :

Théorème 4.

$$\forall f \in F, \quad \sum_{x \in G} f(x) = 0 \quad \Longleftrightarrow \quad \exists (g,h) \in B^2, \quad f = g + h$$

Je rappelle qu'il y a deux démonstrations de ce théorème [23] et [13].

D'autre part une autre conjecture pourrait nous aider à obtenir ce résultat, il s'agit de la généralisation de la première conjecture :

Conjecture 10. *Le minimum pour H_f (sur les fonctions de somme nulle) est atteint lorsque f est une permutation (si elle est de somme nulle).*

À noter que dans les groupes de la forme $G = (\mathbb{Z}/2\mathbb{Z})^N$, les permutations sont de somme nulle.

Résultats élémentaires

Proposition 7. *$H_f \leq M!$ et on a l'égalité ssi f est une constante.*

Proposition 8. *Grâce à la formule de Stirling $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ on obtient :*

$$H^* \sim 2\pi \frac{M^{M+1}}{e^{2M}} = 2\pi e^2 \left(\frac{M}{e^2}\right)^{M+1}$$

2.6.1 Programmation

On va vérifier les conjectures 1 et 2 pour N allant jusqu'à 16.

Groupement des fonctions

Au lieu de chercher pour chaque fonction le nombre H, on va d'abord montrer que l'on peut grouper les fonctions dans des classes d'équivalence dans lesquelles ce nombre sera constant.

Proposition 9. *Soit $(f_1, f_2) \in F^2$. Si il existe $\psi \colon B^2 \to B^2$ bijective telle que :*

$$\forall (g,h) \in B^2, \quad f_1 = g + h \Leftrightarrow f_2 = g' + h' \quad \text{où} \quad (g', h') = \psi(g,h)$$

Alors $H_{f_1} = H_{f_2}$.

Corollaire 1. *Soient $(f_1, f_2) \in F^2$. $H_{f_1} = H_{f_2}$ lorsqu'une des conditions suivantes est vérifiée :*

1. *$f_1 - f_2$ est constante.*
2. *$\exists \sigma \in B, \quad f_1 \circ \sigma = f_2$*
3. *$\exists \phi \in Aut(G), \quad \phi \circ f_1 = f_2$*

Preuve :

1. Soit $k = f_1 - f_2$. On prend $\psi(g,h) = (g + k, h)$
2. On prend $\psi(g,h) = (g \circ \sigma, h \circ \sigma)$
3. On prend $\psi(g,h) = (\phi \circ g, \phi \circ h)$

Définition 7. Lorsque f_1 et f_2 vérifient une des trois relations précédentes on note $f_1 \sim f_2$.

Définition 8. On suppose que G est ordonné avec la relation $<$.
Sur F on définit un ordre total $\prec_{\text{lex}}$ de la façon suivante :

$$f \prec_{\text{lex}} g \iff \exists k \in G \quad \forall l < k \quad f(l) = g(l) \quad \text{et} \quad f(k) < g(k)$$

Définition 9. On définit sur F la relation suivante : $f_1 \mathcal{R} f_2$ lorsque $f_1 \sim f_2$ ou bien lorsqu'il existe une suite de fonctions f'_1, f'_2 ... f'_k telle que $f_1 \sim f'_1$, $f'_1 \sim f'_2$, ..., $f'_{k-1} \sim f'_k$ et $f'_k \sim f_2$.

Comptage

Enfin, voici des propriétés qui vont nous permettre de gagner beaucoup de temps dans le calcul de H. On va tenir compte de certaines symétries du problème.

Proposition 10. *Soit $f \in F_0$*

1. *Notons H'_f le nombre de couples $(g,h) \in {B_0}^2$ tels que $f = g + h$. Alors $H_f = N \times H'_f$.*
2. *Notons H'_1 le nombre de couples $(g,h) \in {B_0}^2$ tel que $f = g+h$ et $g \prec_{lex} h$ et notons H'_2 le nombre de fonctions $g \in B_0$ tel que $f = 2g$. Alors $H'_f = 2 \times H'_1 + H'_2$.*
3. *Notons H''_f le nombre de couples $(g,h) \in {B_0}^2$ tels que g soit strictement croissante sur chacun des sous-ensembles $A_0 = f^{-1}(\{0\})$, ..., $A_{n-1} = f^{-1}(\{n-1\})$ formant une partition de G. Alors*

$$H'_f = (card(A_0) - 1)! \times \prod_{i=1}^{n-1} card(A_i)! \times H''_f$$

Remarque. Le deuxième point n'a pas été exploité dans l'algorithme, car il est incompatible avec le troisième point.

Premier algorithme

Idée Le principe est de générer toutes les fonctions croissantes de somme nulle, et de regarder dans un premier temps si on peut trouver une fonction plus petite pour l'ordre lexical dans la même classe, si c'est le cas on passe au candidat suivant, sinon on compte les permutations dont la somme fait la fonction avec comme condition que la première permutation est nulle en 0 et strictement croissante sur chaque sous-ensemble où la fonction est constante.

Ainsi cet algorithme est sûr de trouver toutes les valeurs prises par H_f. Il ne reste plus qu'à trier les résultats.

Algorithme détaillé On commence par chercher un représentant pour chaque classe de fonction. Le représentant doit avoir une somme nulle, doit être plus petit que tous ses translatés et tous ses composés avec un automorphisme du groupe. Ainsi, on est sûr de ne pas étudier deux fois la même fonction.

$Aut(G) \leftarrow$ liste des automorphismes du groupe
$l_{crois} \leftarrow \{f : G \to G \mid$
f croissante,
$\sum_{i \in G} f(i) = 0$,
$\forall \phi \in Aut(G) \quad f \prec_{\text{lex}} \phi \circ f$,
$\forall i \in G, \quad f \prec_{\text{lex}} f + i\}$
Pour $f \in l_{crois}$ **faire**
| `NbPermutations(`f`)`
finpour

Maintenant je vais montrer en détail la procédure principale qui compte le nombre de permutations.

Pour $i = 0 \to 10$ **faire**
| $A[i] \leftarrow \text{card}(f^{-1}(i))$
finpour
$p \leftarrow (A[0]-1)! \prod_{i=1}^{n} A[i]!$
Renvoie $p \times$ `NbPermutations(`$\emptyset$`,`$\emptyset$`,`*0,1,0,A,0*`)`

Procédure NbPermutations(f : fonction croissante)

`// pg:éléments déjà pris pour` g`,ph:éléments déjà pris pour` h
`// prof:on génère la permutation au niveau de` $A[prof]$
`// indice:On traite le terme numéro indice+1 dans` $A[prof]$
`// DernierPris:dernier élément pris pour` g `dans` $A[prof]$
`// A:le tableau calculé précédemment`
`// nb: le nombre de solutions jusqu'à présent)`
Si $prof \geq N$ **alors**
| **Renvoie** *nb+1*
| `// on vient de trouver une solution`
finsi
Si $A[prof] = indice$ **alors**
| **Renvoie** `NbPermutations2(`*pg,ph,prof+1,0,-1,A,nb*`)`
| `// on passe à la profondeur suivante`
sinon
| **Pour** $i = DernierPris + 1 \to N - 1$ **faire**
| | **Si** $i \notin pg$ **alors**
| | | `// Calcul de la valeur prise par l'autre bijection` h
| | | $j \leftarrow \text{prof} - i$
| | | **Si** $j \notin ph$ **alors**
| | | | nb $\leftarrow$`NbPermutations2(`$pg \cup \{i\}$`,`$ph \cup \{j\}$*,prof,indice+1,i,A,nb*`)`
| | | **finsi**
| | **finsi**
| **finpour**
finsi
Renvoie *nb*

Procédure NbPermutations2(pg,ph,prof,indice,DernierPris,A,nb)

Résultats

En **gras** les valeurs de N correspondant à des puissances de deux.

N	4	5	6	7	8
H^*	**2**	4	11	39	**96**
H min	$(\mathbb{Z}/2\mathbb{Z})^2 : 8$ $\mathbb{Z}/4\mathbb{Z} : 8$	15	48	133	$(\mathbb{Z}/2\mathbb{Z})^3$ et $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z} : 384$ $\mathbb{Z}/8\mathbb{Z} : 512$

N	9	10	11	12
H^*	340	1320	5580	25700
H min	$\mathbb{Z}/9\mathbb{Z} : 2025$	9280	37851	$\mathbb{Z}/12\mathbb{Z} : 210432$

N	13	14	15	16	17
H^*	128000	684000	3900000	**23700000**	153000000
H min	1030367	6356224	36362925	$(\mathbb{Z}/2\mathbb{Z})^4$:244744192	1606008513

Remarque. À partir de $N = 16$ les calculs deviennent vraiment long (on compte en jours), si on peut quand même étudier $(\mathbb{Z}/2\mathbb{Z})^4$, c'est parce que ce groupe possède un grand nombre d'automorphismes.

Remarque. Pour $N = 17$ je n'ai pas vérifié les conjectures, j'ai juste calculé H_f pour f qui est une permutation.

Deuxième algorithme

Principe On regarde le problème sous un autre angle. On part cette fois-ci du cœur du groupe, à savoir sa table de loi, et on remarque que si on ne s'occupe pas de l'ordre dans lequel on somme deux bijections, sommer deux bijections revient à choisir dans la table du groupe un élément par ligne et par colonne.

Un moyen de programmer cela facilement et rapidement est d'utiliser MAPLE. On fabrique une table du groupe en nommant les éléments a_0, a_1, etc., puis on calcul un déterminant positif [14] de la table, en regroupant les termes comportant le même nombre de a_0, a_1 etc.

2.6.2 Généralisation

L'objectif de cette section est de regarder les résultats qui peuvent être généralisés aux groupes non commutatifs, voire aux tables régulières.

Enoncé des problèmes

Soit $G, *$ un ensemble fini de n éléments muni d'une loi interne $*$ telle que la table de G soit un carré latin (tous les éléments d'une même ligne ou d'une même colonne sont distincts).

Conjecture 11. *Pour tout $(a_1, a_2, \dots a_{n-1}) \in G^{n-1}$ il existe $n-1$ éléments distincts $(b_1, b_2, \dots, b_{n-1}) \in G^{n-1}$ tels que $a_1 * b_1$, $a_2 * b_2$, …, $a_{n-1} * b_{n-1}$ sont tous distincts.*

Remarque. Le théorème 1 montre cette propriété dans le cas où G est un groupe commutatif.

Définition 10. Pour $f : \{1; \dots; n-1\} \to G$ on note H_f le nombre de $n-1$ uplets $(b_1, \dots b_{n-1}) \in G^{n-1}$ tels que les éléments de $f(i) * b_i \in G$ pour i variant de 1 de $n-1$ sont tous distincts.

Remarque. La conjecture précédente équivaut à : $\forall f, \quad H_f > 0$

Conjecture 12. $H_f \geq H^*$ *où* $H^* = \frac{(n!)^2}{n^n}$.

Remarque. Il s'agit de la généralisation de la conjecture 1.

Remarque. La véracité de cette conjecture entraîne bien sûr la véracité de la conjecture précédente.

Conjecture 13. *Le minimum de* H_f *est pris pour* f *injective.*

Remarque. Encore une fois il s'agit d'une généralisation d'une conjecture précédente : la conjecture 2.

Quelques propriétés

Définition 11. On dit que $(G, *)$ est une boucle lorsque la loi $*$ possède un élément neutre.

Proposition 11. *Si on montre l'une des trois conjectures pour n'importe quelle boucle alors on montre la conjecture pour n'importe quel carré latin. Autrement dit, on peut limiter l'étude aux boucles.*

Preuve : Soit $(C, *)$ un carré latin. On pose $C = \{c_1, \dots c_n\}$. On définit alors :

$$\phi \colon c \in C \mapsto c_1 * c \in C$$

C étant un carré latin, ϕ est bijective. On définit alors une nouvelle loi sur C :

$$\forall (c, d) \in C^2, \quad c \star d = c * \phi^{-1}(d)$$

On a alors :

$$\forall c \in C, \quad c_1 \star c = c_1 * \phi^{-1}(c) = \phi(\phi^{-1}(c)) = c$$

De sorte que c_1 est un élément neutre à gauche pour la loi $\star$.
Maintenant remarquons que :

$$\psi \colon c \in C \mapsto c \star c_1$$

est une fonction bijective, on peut alors définir une dernière loi ∇ :

$$\forall (c, d) \in C^2, \quad c \nabla d = \psi^{-1}(c) \star d = \psi^{-1}(c) * \phi^{-1}(d)$$

Alors C, ∇ est une boucle dont l'élément neutre est c_1, en effet :

$$\forall c \in C, \quad c_1 \nabla c = \psi^{-1}(c_1) \star c = c_1 \star c = c$$

car $c_1 \star c_1 = c_1$ entraîne $\psi(c_1) = c_1$, et d'autre part :

$$\forall c \in C, \quad c \nabla c_1 = \psi^{-1}(c) \star c_1 = \psi(\psi^{-1}(c)) = c$$

On a alors une bijection de $C \times C$ dans lui même telle que

$$\forall (c, d) \in C^2, \quad \psi(c) \nabla \phi(d) = c * d$$

Résoudre le problème dans $(C, *)$ pour les éléments $a_1, \dots a_{n-1}$ revient à résoudre le problème dans (C, ∇) pour $\psi(a_1), \dots, \psi(a_{n-1})$.
On peut donc ramener le problème à l'étude des boucles.

Comptage Pour compter on est obligé de modifier la proposition 4 qui n'est plus valable. On ordonne notre ensemble G d'une façon quelconque, l'élément neutre étant le premier élément noté 0. Quitte à renommer les éléments de G, on les appelle $\{0; 1; \ldots; n-1\}$

Proposition 12. *Soit f une fonction prenant $n-1$ valeurs.*
Notons H'_f le nombre de couples $(g,h) \in B^2$ tels que g soit strictement croissante sur chacun des sous-ensembles $A_0 = f^{-1}(\{0\})$, ..., $A_{n-1} = f^{-1}(\{n-1\})$ et $f * g = h$. Alors

$$H_f = \prod_{i=0}^{n-1} card(A_i)! \times H'_f$$

Résultats pour les groupes non commutatifs J'ai trouvé la liste et la description des premiers groupes non commutatifs dans [9].
Voici les résultats pour les premiers groupes non commutatifs :

N	6	8	10	12
H^*	11	96	1320	25700
H min	S_3 :36	D_4 :512 H_8 :512	D_5 :8200	D_6 et $\langle 2,2,3\rangle$:207360 A_4 :200448

Toutes les conjectures sont vérifiées sur les petits groupes non commutatifs. Mais au vu de ces valeurs je me suis plus intéressé aux boucles.
Remarque. Si on montre qu'en prenant la moitié des valeurs ($k = n/2$) quelconques on arrive encore à compléter au moins une fois alors le théorème est vrai jusqu'à $N = 24$.

2.6.3 Détails des résultats pour les groupes commutatifs

On écrit chaque fonction de G dans G en notant les images des éléments, du plus petit au plus grand (suivant l'ordre le plus naturel).

Groupes d'ordre 8

— $(\mathbb{Z}/2\mathbb{Z})^3$:
Il y a 12 classes de fonctions. Il faut comprendre chaque chiffre en binaire, par exemple 4 correspond à $(1,0,0)$.

H	Classe
384	0 1 2 3 4 5 6 7
640	0 0 1 1 2 3 4 5 0 0 1 1 2 2 4 4
768	0 0 0 1 1 2 4 6
1152	0 0 0 1 1 1 2 3 0 0 0 0 1 2 4 7 0 0 0 0 1 1 2 2
1408	0 0 1 1 2 2 3 3
1920	0 0 0 0 0 1 2 3
3456	0 0 0 0 1 1 1 1
5760	0 0 0 0 0 0 1 1
40320	0 0 0 0 0 0 0 0

Le meilleur candidat est donc 0 1 2 3 4 5 6 7. Ci-dessous toutes les permutations de B_0 telle que cette fonction soit la somme de deux d'entre elles.

0	7	6	1	5	2	3	4
0	7	6	1	3	4	5	2
0	7	5	2	6	1	3	4
0	7	5	2	1	6	4	3
0	7	3	4	6	1	5	2
0	7	3	4	1	6	2	5
0	7	1	6	5	2	4	3
0	7	1	6	3	4	2	5
0	6	7	1	5	3	2	4
0	6	7	1	2	4	5	3
0	6	4	2	7	1	3	5
0	6	4	2	1	7	5	3
0	6	3	5	7	1	4	2
0	6	3	5	1	7	2	4
0	6	1	7	5	3	4	2
0	6	1	7	2	4	3	5
0	5	7	2	6	3	1	4
0	5	7	2	3	6	4	1
0	5	4	1	7	2	3	6
0	5	4	1	3	6	7	2
0	5	3	6	7	2	4	1
0	5	3	6	2	7	1	4
0	5	1	4	6	3	7	2
0	5	1	4	2	7	3	6

$+$

0	4	6	2	7	3	1	5
0	4	6	2	3	7	5	1
0	4	5	1	7	3	2	6
0	4	5	1	2	6	7	3
0	4	3	7	6	2	5	1
0	4	3	7	2	6	1	5
0	4	1	5	6	2	7	3
0	4	1	5	3	7	2	6
0	3	7	4	5	6	2	1
0	3	7	4	2	1	5	6
0	3	6	5	7	4	1	2
0	3	6	5	1	2	7	4
0	3	5	6	7	4	2	1
0	3	5	6	2	1	7	4
0	3	4	7	5	6	1	2
0	3	4	7	1	2	5	6
0	2	7	5	6	4	1	3
0	2	7	5	3	1	4	6
0	2	6	4	5	7	3	1
0	2	6	4	1	3	7	5
0	2	5	7	6	4	3	1
0	2	5	7	1	3	4	6
0	2	4	6	5	7	1	3
0	2	4	6	3	1	7	5

$= 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7$

— $\mathbb{Z}/8\mathbb{Z}$:

Remarque : dans $\mathbb{Z}/8\mathbb{Z}$ il n'y a pas de bijection de somme nulle.
Il y a 49 classes de fonctions.

H	classes
512	0 0 1 2 3 5 6 7
640	0 0 1 3 4 4 5 7
	0 0 1 2 4 5 6 6
	0 0 1 2 4 4 6 7
	0 0 1 2 2 3 4 4
	0 0 1 1 2 3 4 5
	0 0 1 1 2 2 5 5
	0 0 1 1 2 2 4 6
768	0 0 1 1 2 3 3 6
	0 0 1 1 2 2 3 7
	0 0 0 1 2 4 4 5
	0 0 0 1 2 3 4 6
	0 0 0 1 2 2 5 6
	0 0 0 1 2 2 4 7
	0 0 0 1 1 4 4 6
	0 0 0 1 1 3 5 6
	0 0 0 1 1 3 4 7
	0 0 0 1 1 2 6 6
	0 0 0 1 1 2 5 7
896	0 0 1 1 3 3 4 4
1152	0 0 2 2 4 4 6 6
	0 0 0 2 2 4 4 4
	0 0 0 2 2 2 4 6
	0 0 0 1 5 6 6 6
	0 0 0 1 3 4 4 4

H	classes
1152	0 0 0 1 1 4 5 5
	0 0 0 1 1 2 2 2
	0 0 0 1 1 1 2 3
	0 0 0 0 2 4 4 6
	0 0 0 0 2 2 2 2
	0 0 0 0 1 4 5 6
	0 0 0 0 1 4 4 7
	0 0 0 0 1 3 5 7
	0 0 0 0 1 2 6 7
	0 0 0 0 1 2 2 3
	0 0 0 0 1 1 3 3
	0 0 0 0 1 1 2 4
	0 0 0 0 1 1 1 5
1920	0 0 0 0 2 2 6 6
	0 0 0 0 1 1 7 7
	0 0 0 0 0 2 2 4
	0 0 0 0 0 1 3 4
	0 0 0 0 0 1 2 5
	0 0 0 0 0 1 1 6
3456	0 0 0 0 4 4 4 4
5760	0 0 0 0 0 0 4 4
	0 0 0 0 0 0 2 6
	0 0 0 0 0 0 1 7
40320	0 0 0 0 0 0 0 0

Le meilleur candidat est donc 0 0 1 2 3 5 6 7. Ci-dessous toutes les permutations de B_0 telle que cette fonction soit la somme de deux d'entre elles.

0 7 6 5 4 3 2 1		0 1 3 5 7 2 4 6
0 7 6 3 5 1 4 2		0 1 3 7 6 4 2 5
0 7 5 4 1 6 3 2		0 1 4 6 2 7 3 5
0 7 5 3 6 2 4 1		0 1 4 7 5 3 2 6
0 7 3 6 4 2 1 5		0 1 6 4 7 3 5 2
0 7 3 5 1 6 2 4		0 1 6 5 2 7 4 3
0 7 2 6 5 3 1 4		0 1 7 4 6 2 5 3
0 7 2 4 6 1 3 5		0 1 7 6 5 4 3 2
0 6 5 3 2 7 1 4		0 2 4 1 7 6 3 5
0 6 5 1 4 7 3 2		0 2 4 7 1 6 5 3
0 6 4 3 7 2 5 1		0 2 5 1 6 3 7 4
0 6 4 1 5 2 7 3		0 3 6 4 2 1 7 5
0 6 3 7 4 1 5 2		0 3 5 4 1 2 7 6
0 6 3 5 2 1 7 4		0 3 5 7 2 6 4 1
0 6 2 7 5 4 1 3		0 3 6 1 4 7 2 5
0 6 2 5 7 4 3 1		0 3 2 7 5 1 4 6
0 5 7 4 2 6 1 3	+	0 3 2 6 1 7 5 4
0 5 7 3 6 4 2 1		0 2 7 5 4 1 3 6
0 5 4 6 2 3 7 1		0 2 7 3 6 1 5 4
0 5 4 3 1 7 2 6		0 2 6 5 1 4 7 3
0 5 3 6 1 4 7 2		0 2 6 3 7 4 1 5
0 5 3 1 7 6 4 2		0 2 5 7 4 3 1 6
0 5 2 4 7 3 1 6		0 3 7 1 5 6 2 4
0 5 2 1 6 7 4 3		0 3 7 6 4 2 5 1
0 4 7 5 2 6 3 1		0 4 2 1 6 7 3 5
0 4 7 3 6 2 5 1		0 4 2 5 1 7 3 6
0 4 7 3 5 2 1 6		0 4 2 7 5 3 1 6
0 4 7 1 5 6 3 2		0 4 2 7 6 3 5 1
0 4 6 5 2 3 7 1		0 4 3 5 1 2 7 6
0 4 6 3 2 7 1 5		0 4 3 7 1 6 5 2
0 4 6 3 1 7 5 2		0 4 3 7 2 6 1 5
0 4 6 1 5 3 7 2		0 4 3 1 6 2 7 5

$= 0\ 0\ 1\ 2\ 3\ 5\ 6\ 7$

$\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$ Il y a 44 classes de fonctions.

384	0 1 2 3 4 5 6 7
640	0 0 1 3 4 4 5 7
	0 0 1 2 4 4 6 7
	0 0 1 2 2 3 5 7
	0 0 1 2 2 3 4 4
	0 0 1 1 4 5 6 7
	0 0 1 1 2 3 4 5
	0 0 1 1 2 2 5 5
	0 0 1 1 2 2 4 6
768	0 0 0 1 4 4 5 6
	0 0 0 1 2 3 4 6
	0 0 0 1 2 2 4 7
	0 0 0 1 1 3 4 7
	0 0 0 1 1 2 5 7
	0 0 0 1 1 2 4 4
896	0 0 1 1 4 4 7 7

1152	0 0 1 1 4 4 5 5
	0 0 1 1 2 2 3 3
	0 0 0 2 4 4 4 6
	0 0 0 2 2 2 4 6
	0 0 0 1 4 4 4 7
	0 0 0 1 1 2 2 2
	0 0 0 1 1 1 4 5
	0 0 0 1 1 1 2 3
	0 0 0 0 4 4 6 6
	0 0 0 0 2 2 4 4
	0 0 0 0 1 3 5 7
	0 0 0 0 1 3 4 4
	0 0 0 0 1 2 4 5
	0 0 0 0 1 2 2 3
	0 0 0 0 1 1 5 5
	0 0 0 0 1 1 4 6
	0 0 0 0 1 1 1 1

1408	0 0 2 2 4 4 6 6
1920	0 0 0 0 1 1 3 3
	0 0 0 0 0 2 4 6
	0 0 0 0 0 1 4 7
	0 0 0 0 0 1 1 2
3456	0 0 0 0 4 4 4 4
	0 0 0 0 2 2 2 2
5760	0 0 0 0 0 0 4 4
	0 0 0 0 0 0 2 2
	0 0 0 0 0 0 1 3
40320	0 0 0 0 0 0 0 0

2.7 Algorithme de calcul de valeurs de h et h'

2.7.1 Rappel du problème et notations

On travaille dans $(\mathbb{Z}/2\mathbb{Z})^n$, et on cherche le nombre h de solutions, distinctes deux à deux, du système d'inconnues P_1, P_2 ... P_α :

$$\begin{cases} P_1 \oplus P_2 &= \lambda_0 \\ P_3 \oplus P_4 &= \lambda_1 \\ P_5 \oplus P_6 &= \lambda_2 \\ & \vdots \\ P_{\alpha-1} \oplus P_\alpha &= \lambda_{a-1} \end{cases}$$

où on a bien sûr $\alpha = 2a$
On notera pour tout $i \in [1, \alpha]$, $(i) = \lfloor (i-1)/2 \rfloor$, de sorte que P_i intervient dans l'équation où il y a $\lambda_{(i)}$

2.7.2 Choix des λ

Le nombre h ne change pas si on change l'ordre des équations ou des inconnues, et il est également le même si on se donne un isomorphisme ϕ de $(\mathbb{Z}/2\mathbb{Z})^n$ et qu'on remplace chaque P_i par $\phi(P_i)$ et chaque λ_i par $\phi(\lambda_i)$. À noter qu'un isomorphisme de groupe est également un isomorphisme d'espace vectoriel sur $\mathbb{Z}/2\mathbb{Z}$.

Sachant cela, on peut supposer que les premiers λ sont classés dans l'ordre et linéairement indépendants. On peut par exemple imposer que $\lambda_0 = 1$ puis $\lambda_1 = \lambda_0$ ou bien λ_1 linéairement indépendant et donc $\lambda_1 = 2$ (10 en binaire). La prochaine valeur linéairement indépendante sera 4 (100 en binaire) etc. On peut également supposer que le groupe de λ valant 1 comporte plus d'éléments que le groupe de λ valant 2 etc. Enfin, les λ suivants sont combinaison linéaire des précédents, et, à taille de groupe égale, on choisit le groupe d'indice le plus

petit. Par exemple, si on a :

$$\lambda_0 = \lambda_1 = \lambda_2 = 1, \lambda_3 = \lambda_4 = 2, \lambda_5 = \lambda_6 = 4$$

on pourra choisir $\lambda_7 = \lambda_0 \oplus \lambda_3$ mais pas $\lambda_7 = \lambda_0 \oplus \lambda_5$.
En fait l'algorithme est un petit peu plus compliqué. Une fois que l'on a choisi les λ en utilisant toutes les combinaisons possibles, on vérifie qu'en permutant deux groupes de même taille, on ne diminue pas le nombre de fois que l'on utilise le premier bit (celui de droite), ou bien en cas d'égalité le deuxième bit etc.

2.7.3 Dénombrement des (P_i).

Une fois les λ fixés, on compte le nombre de bits utilisés pour écrire les λ en base 2. Soit u ce nombre de bit. Par exemple, si $\lambda_0 = \lambda_1 = 1$ et $\lambda_2 = 2$, alors on utilise en tout deux bits et $u = 2$. On transforme alors le système initial en deux systèmes de la façon suivante : on écrit pour tout i, $P_i = \bar{P}_i || \tilde{P}_i$ où $\tilde{P}_i$ est la partie sur u bits et $\bar{P}_i$ représente les bits les plus à gauche. Le système est alors équivalent à :

$$(1) \begin{cases} \tilde{P}_1 \oplus \tilde{P}_2 = \lambda_0 \\ \tilde{P}_3 \oplus \tilde{P}_4 = \lambda_1 \\ \vdots \\ \tilde{P}_{\alpha-1} \oplus \tilde{P}_\alpha = \lambda_{a-1} \end{cases} \quad \text{et} \quad (2) \begin{cases} \bar{P}_1 \oplus \bar{P}_2 = 0 \\ \bar{P}_3 \oplus \bar{P}_4 = 0 \\ \vdots \\ \bar{P}_{\alpha-1} \oplus \bar{P}_\alpha = 0 \end{cases}$$

avec comme condition supplémentaire que pour tout couple (i, j) distincts $\tilde{P}_i \neq \tilde{P}_j \quad \text{ou} \quad \bar{P}_i \neq \bar{P}_j$. Le deuxième système se résout facilement : $\bar{P}_1 = \bar{P}_2$, $\bar{P}_3 = \bar{P}_4$ etc.

On résout alors le premier système dans $(\mathbb{Z}/2\mathbb{Z})^u$, sans tenir compte du fait que les P_i doivent être distincts (remarque : comme les λ_i sont non nuls, P_{2i+1} et P_{2i+2} seront tout de même toujours distincts).

On essaye alors (presque) toutes les possibilités. En fait on impose $\tilde{P}_1 = 0$ car on peut translater tous les P_i d'une même quantité constante, et on s'arrange la plupart du temps pour que $\tilde{P}_{2i+1} < \tilde{P}_{2i+2}$. Ceci n'est hélas pas toujours possible car lorsqu'on calcule h' cela peut nous empêcher de trouver des solutions. Donc on impose cette inégalité qu'à partir de la première équation où il n'y aura pas d'interaction avec de nouvelles égalités. Etant donné que le choix de $\tilde{P}_1$, $\tilde{P}_3$, $\ldots$ $\tilde{P}_{\alpha-1}$ entraîne celui des autres valeurs, le calcul va aboutir en au plus 2^{au} calculs.

Exemple. Reprenons l'exemple avec $\alpha = 6$ et $\lambda_0 = \lambda_1 = 1$, $\lambda_2 = 2$. On doit résoudre dans $\mathbb{Z}/2\mathbb{Z}^2$ le système :

$$(1) \begin{cases} \tilde{P}_1 \oplus \tilde{P}_2 = 1 \\ \tilde{P}_3 \oplus \tilde{P}_4 = 1 \\ \tilde{P}_5 \oplus \tilde{P}_6 = 2 \end{cases}$$

Les solutions sont $(0,1,0,1,0,2)$; $(0,1,0,1,1,3)$; $(0,1,2,3,0,2)$; $(0,1,2,3,1,3)$, à symétrie (pair/impair à partir de la deuxième équation) et translation près. Il y a 4 translations possibles et pour chacune $2^{a-1} = 4$ symétries. Chaque solution en représente en fait un ensemble de 16 solutions équivalentes.

Pour chaque solution trouvée, on va alors construire un graphe qui a autant de sommets qu'il y a d'équations dans le système. Appelons S_0, S_1, ... S_a les sommets. Pour $i \neq j$, il y a une arrête de S_i vers S_j si et seulement si les deux équations ont au moins une solution commune, c'est à dire $\tilde{P}_{2i+1} = \tilde{P}_{2j+1}$ ou $\tilde{P}_{2i+2} = \tilde{P}_{2j+2}$ ou $\tilde{P}_{2i+2} = \tilde{P}_{2j+1}$ ou $\tilde{P}_{2i+1} = \tilde{P}_{2j+2}$. Entre deux sommets S_i et S_j reliés, on doit avoir $\bar{P}_{2i+1} \neq \bar{P}_{2j+1}$. On est donc ramené à un problème de coloriage d'un graphe avec 2^{n-u} couleurs, deux sommets reliés devant avoir des couleurs différentes.

Exemple. Dans l'exemple précédent on obtient les graphes suivants :


pour les deux premières solutions et


pour les deux dernières.
Si on dispose de X couleurs, on peut colorier le premier de $X(X-1)(X-2)$ façons et le second de $X(X-1)^2$ façons. Posons $N = 2^n$, le nombre de solutions du système initial pour ces valeurs de λ est donc :

$$2 \times 16 \times \frac{N}{4}\left(\frac{N}{4}-1\right)\left(\frac{N}{4}-2\right) + 2 \times 16 \times \frac{N}{4}\left(\frac{N}{4}-1\right)^2 = N^3 - 10N^2 + 24N$$

J'ai utilisé différents algorithmes pour compter le nombre de coloriage d'un graphe en X couleurs. Voici le plus simple.

Entrées : un graphe G
Sortie : Un polynôme P tel que si on colore le graphe avec X couleurs, il y a $P(X)$ possibilités
Si G *est vide* **alors**
  Renvoie 1
sinon
  $S \leftarrow$ sommet de G
  Si `NbArrete(`S`,`G`)`$= 0$ **alors**
    Renvoie $X \times$ `NbColoriage(`G *privé de* S`)`
  sinon
    $S' \leftarrow$ sommet relié à S
    $G' \leftarrow G$ privé de l'arrête $S - S'$
    $G'' \leftarrow G$ avec fusion des sommets S et S'
    Renvoie `NbColoriage(`G'`)-NbColoriage(`G"`)`
  finsi
finsi

Algorithme 1 : NbColoriage(G)

2.7.4 Résultats

On note $N = 2^n$.

$\alpha = 2$

$h = N$

$\alpha = 4$

Lin. ind.	Lin. dép.	h	h'	$h' - h/N$
λ_0, λ_1		$-4N + N^2$	N	4
$\lambda_0(2)$		$-2N + N^2$	N	2

$\alpha = 6$

Lin. ind.	Lin. dép.	h	h'	$h' - h/N$
$\lambda_0, \lambda_1, \lambda_2$		$40N - 12N^2 + N^3$		
	$\lambda' = P_1 \oplus P_3 = \lambda_2$		$-6N + N^2$	$-40 + 6N$
$\lambda' = P_1 \oplus P_3$			$-8N + N^2$	$-40 + 4N$
λ_0, λ_1	$\lambda_2 = \lambda_0 \oplus \lambda_1$	$32N - 12N^2 + N^3$		
$\lambda' = P_1 \oplus P_3$			$-8N + N^2$	$-32 + 4N$
$\lambda_0 = \lambda_1, \lambda_2$		$24N - 10N^2 + N^3$		
	$\lambda' = P_1 \oplus P_3 = \lambda_2$		$-4N + N^2$	$-24 + 6N$
$\lambda' = P_1 \oplus P_3$			$-8N + N^2$	$-24 + 2N$
$\lambda_0 = \lambda_1 = \lambda_2$		$8N - 6N^2 + N^3$		
$\lambda' = P_1 \oplus P_3$			$-4N + N^2$	$-8 + 2N$

$\alpha = 8$

1. λ_0, λ_1, λ_2, λ_3 linéairement indépendants.

$$h = -672N + 208N^2 - 24N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$88N - 18N^2 + N^3$	$672 - 120N + 6N^2$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$104N - 20N^2 + N^3$	$672 - 104N + 4N^2$

2. $\lambda_0, \lambda_1, \lambda_2$ linéairement indépendants et $\lambda_3 = \lambda_0 \oplus \lambda_1$

$$h = -576N + 200N^2 - 24N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$80N - 18N^2 + N^3$	$576 - 120N + 6N^2$
$P_1 \oplus P_3 = ind.$	$112N - 20N^2 + N^3$	$576 - 88N + 4N^2$
$P_1 \oplus P_5 = ind.$	$104N - 20N^2 + N^3$	$576 - 96N + 4N^2$

3. $\lambda_0, \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_1 \oplus \lambda_2$

$$h = -624N + 208N^2 - 24N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$68N - 16N^2 + N^3$	$624 - 140N + 8N^2$
$P_1 \oplus P_3 = ind.$	$112N - 20N^2 + N^3$	$624 - 96N + 4N^2$

4. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3$

$$h = -448N + 168N^2 - 22N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$64N - 16N^2 + N^3$	$448 - 104N + 6N^2$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$96N - 20N^2 + N^3$	$448 - 72N + 2N^2$

5. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_2$

$$h = -320N + 152N^2 - 22N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$32N - 12N^2 + N^3$	$320 - 120N + 10N^2$
$P_1 \oplus P_3 = ind.$	$96N - 20N^2 + N^3$	$320 - 56N + 2N^2$

6. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3$

$$h = -272N + 132N^2 - 20N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$24N - 10N^2 + N^3$	$272 - 108N + 10N^2$
$P_1 \oplus P_3 = ind.$	$80N - 18N^2 + N^3$	$272 - 52N + 2N^2$

7. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3$

$$h = -192N + 104N^2 - 18N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_3$	$32N - 12N^2 + N^3$	$192 - 72N + 6N^2$
$P_1 \oplus P_3 = ind.$	$64N - 16N^2 + N^3$	$192 - 40N + 2N^2$

8. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3$

$$h = -48N + 44N^2 - 12N^3 + N^4$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = ind.$	$24N - 10N^2 + N^3$	$48 - 20N + 2N^2$

$\alpha = 10$

1. $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4$

$$h = 16128N - 4960N^2 + 640N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1872N + 416N^2 - 34N^3 + N^4$	$-16128 + 3088N - 224N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$-2176N + 464N^2 - 36N^3 + N^4$	$-16128 + 2784N - 176N^2 + 4N^3$

2. $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_1$

$$h = 14400N - 4736N^2 + 632N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1744N + 408N^2 - 34N^3 + N^4$	$-14400 + 2992N - 224N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$-2128N + 464N^2 - 36N^3 + N^4$	$-14400 + 2608N - 168N^2 + 4N^3$
$P_1 \oplus P_5 = \lambda_1 \oplus \lambda_3$	$-1936N + 448N^2 - 36N^3 + N^4$	$-14400 + 2800N - 184N^2 + 4N^3$

3. $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_1 \oplus \lambda_2$

$$h = 15360N - 4912N^2 + 640N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1504N + 364N^2 - 32N^3 + N^4$	$-15360 + 3408N - 276N^2 + 8N^3$
$P_1 \oplus P_3 = \lambda_3$	$-1824N + 416N^2 - 34N^3 + N^4$	$-15360 + 3088N - 224N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$-2048N + 456N^2 - 36N^3 + N^4$	$-15360 + 2864N - 184N^2 + 4N^3$

4. $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_1 \oplus \lambda_2 \oplus \lambda_3$

$$h = 15744N - 4960N^2 + 640N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1760N + 408N^2 - 34N^3 + N^4$	$-15744 + 3200N - 232N^2 + 6N^3$
$P_1 \oplus P_3 = ind.$	$-2304N + 472N^2 - 36N^3 + N^4$	$-15744 + 2656N - 168N^2 + 4N^3$

5. $\lambda_0, \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_1, \lambda_4 = \lambda_0 \oplus \lambda_2$

$$h = 12160N - 4464N^2 + 624N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1216N + 344N^2 - 32N^3 + N^4$	$-12160 + 3248N - 280N^2 + 8N^3$
$P_1 \oplus P_3 = ind.$	$-2128N + 464N^2 - 36N^3 + N^4$	$-12160 + 2336N - 160N^2 + 4N^3$
$P_3 \oplus P_5 = ind.$	$-2048N + 456N^2 - 36N^3 + N^4$	$-12160 + 2416N - 168N^2 + 4N^3$

6. $\lambda_0, \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_1, \lambda_4 = \lambda_0 \oplus \lambda_1 \oplus \lambda_2$

$$h = 12160N - 4464N^2 + 624N^3 - 40N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1184N + 340N^2 - 32N^3 + N^4$	$-12160 + 3280N - 284N^2 + 8N^3$
$P_1 \oplus P_3 = ind.$	$-2112N + 464N^2 - 36N^3 + N^4$	$-12160 + 2352N - 160N^2 + 4N^3$
$P_1 \oplus P_5 = ind.$	$-2048N + 456N^2 - 36N^3 + N^4$	$-12160 + 2416N - 168N^2 + 4N^3$

7. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4$

$$h = 11520N - 4016N^2 + 568N^3 - 38N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1440N + 360N^2 - 32N^3 + N^4$	$-11520 + 2576N - 208N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$-2048N + 456N^2 - 36N^3 + N^4$	$-11520 + 1968N - 112N^2 + 2N^3$

8. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2$

$$h = 9088N - 3632N^2 + 552N^3 - 38N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-832N + 264N^2 - 28N^3 + N^4$	$-9088 + 2800N - 288N^2 + 10N^3$
$P_1 \oplus P_3 = \lambda_3$	$-1216N + 344N^2 - 32N^3 + N^4$	$-9088 + 2416N - 208N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$-1600N + 424N^2 - 36N^3 + N^4$	$-9088 + 2032N - 128N^2 + 2N^3$

9. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_2 \oplus \lambda_3$

$$h = 9984N - 3808N^2 + 560N^3 - 38N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1152N + 336N^2 - 32N^3 + N^4$	$-9984 + 2656N - 224N^2 + 6N^3$
$P_1 \oplus P_3 = ind.$	$-2112N + 464N^2 - 36N^3 + N^4$	$-9984 + 1696N - 96N^2 + 2N^3$

10. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2 \oplus \lambda_3$

$$h = 10368N - 3920N^2 + 568N^3 - 38N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-1216N + 344N^2 - 32N^3 + N^4$	$-10368 + 2704N - 224N^2 + 6N^3$
$P_1 \oplus P_3 = ind.$	$-2208N + 472N^2 - 36N^3 + N^4$	$-10368 + 1712N - 96N^2 + 2N^3$

11. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4$

$$h = 7808N - 3184N^2 + 500N^3 - 36N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-640N + 224N^2 - 26N^3 + N^4$	$-7808 + 2544N - 276N^2 + 10N^3$
$P_1 \oplus P_3 = \lambda_4$	$-1024N + 304N^2 - 30N^3 + N^4$	$-7808 + 2160N - 196N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_2 \oplus \lambda_4$	$-1344N + 376N^2 - 34N^3 + N^4$	$-7808 + 1840N - 124N^2 + 2N^3$

12. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2$

$$h = 4736N - 2544N^2 + 468N^3 - 36N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_2$	$-320N + 152N^2 - 22N^3 + N^4$	$-4736 + 2224N - 316N^2 + 14N^3$
$P_1 \oplus P_3 = ind.$	$-1344N + 376N^2 - 34N^3 + N^4$	$-4736 + 1200N - 92N^2 + 2N^3$

13. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3, \lambda_4$

$$h = 5760N - 2576N^2 + 440N^3 - 34N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_3$	$-832N + 264N^2 - 28N^3 + N^4$	$-5760 + 1744N - 176N^2 + 6N^3$
$P_1 \oplus P_3 = \lambda_3 \oplus \lambda_4$	$-1216N + 344N^2 - 32N^3 + N^4$	$-5760 + 1360N - 96N^2 + 2N^3$

14. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_3$

$$h = 3840N - 2144N^2 + 416N^3 - 34N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_3$	$-384N + 176N^2 - 24N^3 + N^4$	$-3840 + 1760N - 240N^2 + 10N^3$
$P_1 \oplus P_3 = ind.$	$-1152N + 336N^2 - 32N^3 + N^4$	$-3840 + 992N - 80N^2 + 2N^3$

15. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3 = \lambda_4$

$$h = 3456N - 1936N^2 + 380N^3 - 32N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_3$	$-320N + 152N^2 - 22N^3 + N^4$	$-3456 + 1616N - 228N^2 + 10N^3$
$P_1 \oplus P_3 = ind.$	$-1024N + 304N^2 - 30N^3 + N^4$	$-3456 + 912N - 76N^2 + 2N^3$
$P_7 \oplus P_9 = ind.$	$-960N + 296N^2 - 30N^3 + N^4$	$-3456 + 976N - 84N^2 + 2N^3$

16. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3, \lambda_4$

$$h = 1920N - 1232N^2 + 284N^3 - 28N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = \lambda_4$	$-320N + 152N^2 - 22N^3 + N^4$	$-1920 + 912N - 132N^2 + 6N^3$
$P_1 \oplus P_3 = ind.$	$-640N + 224N^2 - 26N^3 + N^4$	$-1920 + 592N - 60N^2 + 2N^3$

17. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4$

$$h = 384N - 400N^2 + 140N^3 - 20N^4 + N^5$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = ind.$	$-192N + 104N^2 - 18N^3 + N^4$	$-384 + 208N - 36N^2 + 2N^3$

$\alpha = 12$

dimension 1 $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5$

$$h = -3840N + 4384N^2 - 1800N^3 + 340N^4 - 30N^5 + N^6$$

λ'	h'	$h' - h/N$
$P_1 \oplus P_3 = ind.$	$1920N - 1232N^2 + 284N^3 - 28N^4 + N^5$	$3840 - 2464N + 568N^2 - 56N^3 + 2N^4$

dimension 2

1. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2 = \lambda_5$

$$h = -90368N + 52960N^2 - 11320N^3 + 1132N^4 - 54N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 4736N - 2544N^2 + 468N^3 - 36N^4 + N^5$ $h' - h/N = 90368 - 48224N + 8776N^2 - 664N^3 + 18N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 24960N - 8272N^2 + 996N^3 - 52N^4 + N^5$ $h' - h/N = 90368 - 28000N + 3048N^2 - 136N^3 + 2N^4$

2. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3 = \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_3$

$$h = -78336N + 46656N^2 - 10208N^3 + 1052N^4 - 52N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 5376N - 2848N^2 + 512N^3 - 38N^4 + N^5$ $h' - h/N = 78336 - 41280N + 7360N^2 - 540N^3 + 14N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 22272N - 7520N^2 + 928N^3 - 50N^4 + N^5$ $h' - h/N = 78336 - 24384N + 2688N^2 - 124N^3 + 2N^4$
$\lambda' = P_7 \oplus P_9 = ind.$	$h' = 21504N - 7360N^2 + 920N^3 - 50N^4 + N^5$ $h' - h/N = 78336 - 25152N + 2848N^2 - 132N^3 + 2N^4$

3. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3 = \lambda_4 = \lambda_5$

$$h = -62976N + 38080N^2 - 8544N^3 + 916N^4 - 48N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 3840N - 2144N^2 + 416N^3 - 34N^4 + N^5$ $h' - h/N = 62976 - 34240N + 6400N^2 - 500N^3 + 14N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 17664N - 6176N^2 + 800N^3 - 46N^4 + N^5$ $h' - h/N = 62976 - 20416N + 2368N^2 - 116N^3 + 2N^4$

4. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_4$

$$h = -53760N + 33856N^2 - 7968N^3 + 892N^4 - 48N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 5376N - 2848N^2 + 512N^3 - 38N^4 + N^5$ $h' - h/N = 53760 - 28480N + 5120N^2 - 380N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 16128N - 5856N^2 + 784N^3 - 46N^4 + N^5$ $h' - h/N = 53760 - 17728N + 2112N^2 - 108N^3 + 2N^4$

5. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3, \lambda_4 = \lambda_5$

$$h = -49920N + 31392N^2 - 7400N^3 + 836N^4 - 46N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 4736N - 2544N^2 + 468N^3 - 36N^4 + N^5$ $h' - h/N = 49920 - 26656N + 4856N^2 - 368N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 14848N - 5408N^2 + 732N^3 - 44N^4 + N^5$ $h' - h/N = 49920 - 16544N + 1992N^2 - 104N^3 + 2N^4$
$\lambda' = P_9 \oplus P_1 1 = ind.$	$h' = 13440N - 5104N^2 + 716N^3 - 44N^4 + N^5$ $h' - h/N = 49920 - 17952N + 2296N^2 - 120N^3 + 2N^4$

6. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4, \lambda_5$

$$h = -23040N + 16704N^2 - 4640N^3 + 620N^4 - 40N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_5$	$h' = 3840N - 2144N^2 + 416N^3 - 34N^4 + N^5$ $h' - h/N = 23040 - 12864N + 2496N^2 - 204N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 7680N - 3328N^2 + 536N^3 - 38N^4 + N^5$ $h' - h/N = 23040 - 9024N + 1312N^2 - 84N^3 + 2N^4$

Dimension 3

1. $\lambda_0, \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_1, \lambda_4 = \lambda_0 \oplus \lambda_2, \lambda_5 = \lambda_1 \oplus \lambda_2$

$$h = -317952N + 122816N^2 - 18960N^3 + 1488N^4 - 60N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 24704N - 7984N^2 + 948N^3 - 50N^4 + N^5$ $h' - h/N = 317952 - 98112N + 10976N^2 - 540N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 52096N - 12832N^2 + 1240N^3 - 56N^4 + N^5$ $h' - h/N = 317952 - 70720N + 6128N^2 - 248N^3 + 4N^4$
$\lambda' = P_1 \oplus P_1 1 = ind.$	$h' = 51072N - 12656N^2 + 1232N^3 - 56N^4 + N^5$ $h' - h/N = 317952 - 71744N + 6304N^2 - 256N^3 + 4N^4$

2. $\lambda_0, \lambda_1, \lambda_2, \lambda_3 = \lambda_0 \oplus \lambda_1, \lambda_4 = \lambda_0 \oplus \lambda_2, \lambda_5 = \lambda_0 \oplus \lambda_1 \oplus \lambda_2$

$$h = -317952N + 122816N^2 - 18960N^3 + 1488N^4 - 60N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 24704N - 7984N^2 + 948N^3 - 50N^4 + N^5$ $h' - h/N = 317952 - 98112N + 10976N^2 - 540N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 52096N - 12832N^2 + 1240N^3 - 56N^4 + N^5$ $h' - h/N = 317952 - 70720N + 6128N^2 - 248N^3 + 4N^4$
$\lambda' = P_1 \oplus P_1 1 = ind.$	$h' = 51072N - 12656N^2 + 1232N^3 - 56N^4 + N^5$ $h' - h/N = 317952 - 71744N + 6304N^2 - 256N^3 + 4N^4$

3. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2, \lambda_5 = \lambda_0 \oplus \lambda_3$

$$h = -252672N + 102368N^2 - 16656N^3 + 1376N^4 - 58N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 21888N - 7216N^2 + 880N^3 - 48N^4 + N^5$ $h' - h/N = 252672 - 80480N + 9440N^2 - 496N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_2 \oplus \lambda_3$	$h' = 34688N - 10608N^2 + 1168N^3 - 56N^4 + N^5$ $h' - h/N = 252672 - 67680N + 6048N^2 - 208N^3 + 2N^4$

4. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2, \lambda_5 = \lambda_2 \oplus \lambda_3$

$$h = -264704N + 105664N^2 - 16944N^3 + 1384N^4 - 58N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 21248N - 7072N^2 + 872N^3 - 48N^4 + N^5$ $h' - h/N = 264704 - 84416N + 9872N^2 - 512N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 27392N - 8736N^2 + 1016N^3 - 52N^4 + N^5$ $h' - h/N = 264704 - 78272N + 8208N^2 - 368N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 52096N - 12832N^2 + 1240N^3 - 56N^4 + N^5$ $h' - h/N = 264704 - 53568N + 4112N^2 - 144N^3 + 2N^4$

5. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_0 \oplus \lambda_2, \lambda_5 = \lambda_0 \oplus \lambda_2 \oplus \lambda_3$

$$h = -264704N + 105664N^2 - 16944N^3 + 1384N^4 - 58N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 21248N - 7072N^2 + 872N^3 - 48N^4 + N^5$ $h' - h/N = 264704 - 84416N + 9872N^2 - 512N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 27392N - 8736N^2 + 1016N^3 - 52N^4 + N^5$ $h' - h/N = 264704 - 78272N + 8208N^2 - 368N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 52096N - 12832N^2 + 1240N^3 - 56N^4 + N^5$ $h' - h/N = 264704 - 53568N + 4112N^2 - 144N^3 + 2N^4$

6. $\lambda_0 = \lambda_1, \lambda_2, \lambda_3, \lambda_4 = \lambda_2 \oplus \lambda_3, \lambda_5 = \lambda_0 \oplus \lambda_2 \oplus \lambda_3$

$$h = -264704N + 105664N^2 - 16944N^3 + 1384N^4 - 58N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 27392N - 8736N^2 + 1016N^3 - 52N^4 + N^5$ $h' - h/N = 264704 - 78272N + 8208N^2 - 368N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 52096N - 12832N^2 + 1240N^3 - 56N^4 + N^5$ $h' - h/N = 264704 - 53568N + 4112N^2 - 144N^3 + 2N^4$

7. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_2$

$$h = -192000N + 83392N^2 - 14496N^3 + 1268N^4 - 56N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 12288N - 4672N^2 + 664N^3 - 42N^4 + N^5$ $h' - h/N = 192000 - 71104N + 9824N^2 - 604N^3 + 14N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 24320N - 7904N^2 + 944N^3 - 50N^4 + N^5$ $h' - h/N = 192000 - 59072N + 6592N^2 - 324N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_2 \oplus \lambda_4$	$h' = 29696N - 9408N^2 + 1080N^3 - 54N^4 + N^5$ $h' - h/N = 192000 - 53696N + 5088N^2 - 188N^3 + 2N^4$

8. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_4$

$$h = -221696N + 90944N^2 - 15104N^3 + 1284N^4 - 56N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 17664N - 6176N^2 + 800N^3 - 46N^4 + N^5$ $h' - h/N = 221696 - 73280N + 8928N^2 - 484N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_2 \oplus \lambda_4$	$h' = 29696N - 9408N^2 + 1080N^3 - 54N^4 + N^5$ $h' - h/N = 221696 - 61248N + 5696N^2 - 204N^3 + 2N^4$

9. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_2 \oplus \lambda_4$

$$h = -237056N + 95680N^2 - 15584N^3 + 1300N^4 - 56N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 17664N - 6176N^2 + 800N^3 - 46N^4 + N^5$ $h' - h/N = 237056 - 78016N + 9408N^2 - 500N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 24320N - 7904N^2 + 944N^3 - 50N^4 + N^5$ $h' - h/N = 237056 - 71360N + 7680N^2 - 356N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 46208N - 11680N^2 + 1160N^3 - 54N^4 + N^5$ $h' - h/N = 237056 - 49472N + 3904N^2 - 140N^3 + 2N^4$

10. $\lambda_0 = \lambda_1, \lambda_2 = \lambda_3, \lambda_4 = \lambda_5$

$$h = -190720N + 79840N^2 - 13624N^3 + 1196N^4 - 54N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_2$	$h' = 14848N - 5408N^2 + 732N^3 - 44N^4 + N^5$ $h' - h/N = 190720 - 64992N + 8216N^2 - 464N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_2 \oplus \lambda_4$	$h' = 24960N - 8272N^2 + 996N^3 - 52N^4 + N^5$ $h' - h/N = 190720 - 54880N + 5352N^2 - 200N^3 + 2N^4$

11. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_3$

$$h = -158208N + 71360N^2 - 12912N^3 + 1176N^4 - 54N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 14592N - 5344N^2 + 728N^3 - 44N^4 + N^5$ $h' - h/N = 158208 - 56768N + 7568N^2 - 448N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 21248N - 7072N^2 + 872N^3 - 48N^4 + N^5$ $h' - h/N = 158208 - 50112N + 5840N^2 - 304N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_3 \oplus \lambda_4$	$h' = 27392N - 8736N^2 + 1016N^3 - 52N^4 + N^5$ $h' - h/N = 158208 - 43968N + 4176N^2 - 160N^3 + 2N^4$

12. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3, \lambda_4, \lambda_5 = \lambda_3 \oplus \lambda_4$

$$h = -182784N + 78016N^2 - 13488N^3 + 1192N^4 - 54N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 21248N - 7072N^2 + 872N^3 - 48N^4 + N^5$ $h' - h/N = 182784 - 56768N + 6416N^2 - 320N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 38784N - 10208N^2 + 1064N^3 - 52N^4 + N^5$ $h' - h/N = 182784 - 39232N + 3280N^2 - 128N^3 + 2N^4$

13. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3, \lambda_4, \lambda_5 = \lambda_0 \oplus \lambda_3 \oplus \lambda_4$

$$h = -186624N + 79520N^2 - 13680N^3 + 1200N^4 - 54N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 21888N - 7216N^2 + 880N^3 - 48N^4 + N^5$ $h' - h/N = 186624 - 57632N + 6464N^2 - 320N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = ind.$	$h' = 39744N - 10384N^2 + 1072N^3 - 52N^4 + N^5$ $h' - h/N = 186624 - 39776N + 3296N^2 - 128N^3 + 2N^4$

14. $\lambda_0 = \lambda_1 = \lambda_2, \lambda_3 = \lambda_4, \lambda_5$

$$h = -142848N + 64704N^2 - 11840N^3 + 1100N^4 - 52N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_3$	$h' = 12288N - 4672N^2 + 664N^3 - 42N^4 + N^5$ $h' - h/N = 142848 - 52416N + 7168N^2 - 436N^3 + 10N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_5$	$h' = 18944N - 6400N^2 + 808N^3 - 46N^4 + N^5$ $h' - h/N = 142848 - 45760N + 5440N^2 - 292N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_3 \oplus \lambda_5$	$h' = 24320N - 7904N^2 + 944N^3 - 50N^4 + N^5$ $h' - h/N = 142848 - 40384N + 3936N^2 - 156N^3 + 2N^4$
$\lambda' = P_7 \oplus P_9 = \lambda_0 \oplus \lambda_5$	$h' = 21504N - 7360N^2 + 920N^3 - 50N^4 + N^5$ $h' - h/N = 142848 - 43200N + 4480N^2 - 180N^3 + 2N^4$

15. $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3, \lambda_4, \lambda_5$

$$h = -84480N + 43328N^2 - 8928N^3 + 924N^4 - 48N^5 + N^6$$

$\lambda' = P_1 \oplus P_3 = \lambda_4$	$h' = 12288N - 4672N^2 + 664N^3 - 42N^4 + N^5$ $h' - h/N = 84480 - 31040N + 4256N^2 - 260N^3 + 6N^4$
$\lambda' = P_1 \oplus P_3 = \lambda_4 \oplus \lambda_5$	$h' = 17664N - 6176N^2 + 800N^3 - 46N^4 + N^5$ $h' - h/N = 84480 - 25664N + 2752N^2 - 124N^3 + 2N^4$

Deuxième partie

Hachage

Chapitre 3

Les fonctions de hachage

3.1 Introduction

3.1.1 Qu'est-ce qu'une fonction de hachage ?

Avant de voir une définition précise et mathématique d'une fonction de hachage, nous allons en donner un avant-goût à l'aide de métaphores culinaires.

"Hacher", littéralement, c'est réduire en petits morceaux en mélangeant. Imaginons deux plats légèrement différents que l'on hache et dont on extrait une petite cuiller. Juste en goûtant cette cuiller, on doit pouvoir savoir de quel plat cela provient. On authentifie le plat. En revanche, il nous est impossible ou extrêmement difficile d'élaborer un plat qui donnera le même "haché".

En informatique, les objets que l'on va hacher sont des fichiers, autrement dit des suites finies de 0 et de 1, ce que l'on peut également assimiler à des nombres entiers. L'idée est donc de construire une fonction, de préférence rapidement calculable, qui va de l'ensemble des entiers naturels vers un ensemble fini $\mathcal{H}$ qui contient tous les hachés – ou empreintes numériques – possibles. Dans la pratique $\mathcal{H}$ est l'ensemble de tous les mots de n bits, avec n qui vaut au moins 224 actuellement. Comme les nombres sont un peu long à écrire en binaire, on va grouper les bits par 16 et écrire une empreinte en hexadécimal. Prenons un exemple : la célèbre fonction de hachage MD5, inventée par Rivest en 1991, est fournie avec la plupart des système Unix (Linux, MacOs ...). Si on calcule le haché d'un fichier vide on trouve d41d8cd98f00b204e9800998ecf8427e, chaque lettre ou chiffre représente 4 bits, il y a donc $4 \times 32 = 128$ bits. Ainsi chaque empreinte prend très peu de place en mémoire, 32 caractères sur notre exemple.

La fonction de hachage idéale doit en fait se comporter comme une fonction aléatoire. C'est à dire que la connaissance de certaines valeurs de la fonction ne doit en aucun cas nous donner des indices pour le calcul d'une nouvelle empreinte. En cuisine, rajouter un grain de sel à un plat ne va peut être pas changer son haché. Mais, pour notre fonction idéale, changer un seul bit, ou un seul chiffre, va nous donner une empreinte radicalement différente. En effet, le nombre d'empreintes possibles est énorme : $2^{224} \approx 10^{67}$, donc obtenir la même valeur ou une valeur proche est très improbable. On a, par exemple, une chance sur 20 millions de gagner au loto. Donc trouver une valeur proche serait comme gagner 9 fois de suite au loto ! Regardons maintenant le haché MD5 d'un fichier contenant juste la lettre "a" : 60b725f10c9c85c70d97880dfe8191b3 !

3.1.2 À quoi ça sert ?

L'intégrité des données

On sait que lorsque l'on stocke longtemps des fichiers, ils peuvent se trouver altérés. De même, des données transmises via un réseau peuvent être légèrement modifiées. Pour vérifier l'intégrité des données, on a besoin qu'un calcul de l'empreinte ait été fait préalablement et qu'elle soit stockée ou transmise de manière sûre. On calcule alors de nouveau l'empreinte et on compare les deux valeurs. Si les empreintes diffèrent, il y a eu des modifications des données. Si elles sont semblables, on est pratiquement certain que les données n'ont pas été modifiées.

C'est aussi un moyen de vérifier qu'un fichier n'a pas été échangé avec un autre contenant des virus par exemple, et ceci en gardant uniquement en mé-

moire l'empreinte numérique. On peut imaginer plusieurs scénarii où l'échange d'un programme par un autre peut s'avérer très dommageable.

L'engagement

Il y a des situations dans la vie où l'on aimerait s'engager sur une réponse sans la révéler. On peut bien sûr marquer la réponse sur un bout de papier, mais n'importe quel magicien pourrait aisément tromper son monde en échangeant deux bouts de papier. Grâce aux fonctions de hachage, on a un moyen simple de le faire. On met sa réponse suivie d'un nombre aléatoire (112 bits par exemple) dans un fichier, on calcule l'empreinte et l'engagement consiste à rendre public le résultat. Pour la vérification du résultat, on révèle le fichier avec la réponse, et tout le monde peut vérifier que l'empreinte correspond avec la valeur donnée lors de l'engagement.

Les mots de passe

Il est dangereux de stocker des mots de passe dans des serveurs. Si jamais un pirate accède au fichier des mots de passe, il pourra faire énormément de dégâts, y compris sur d'autres ordinateurs où les mêmes mots de passe sont utilisés. Un moyen simple de corriger cette faille est de stocker les empreintes des mots de passe. Ainsi, lorsqu'un utilisateur rentre son mot de passe, on le hache pour le comparer avec la valeur stockée. Même si une personne prend connaissance des empreintes, il ne pourra retrouver les mots de passe qu'en les essayant un par un. Cette façon de faire comporte malgré tout quelques problèmes de sécurité, comme l'existence de tables inverses qui révèlent le mot de passe en fonction de l'empreinte. Sur internet ce genre de tables fleurissent, par exemple pour la célèbre fonction de hachage $md5$. Elles donnent ainsi tous les empreintes de mots du dictionnaire, à un ou deux caractères près. Un moyen d'éviter cette attaque est de bien choisir son mot de passe, et là encore on peut utiliser une fonction de hachage. Le mieux est d'utiliser une fonction de hachage privée, ce qui peut s'obtenir assez facilement avec une fonction de hachage publique. On lui demande l'empreinte d'un mot ou d'une phrase simple, et on se sert de l'empreinte, ou d'une partie de l'empreinte, comme mot de passe. L'inconvénient de cette méthode est qu'elle nécessite d'avoir toujours sa fonction de hachage privée avec soi, ou bien de retenir un mot de passe très bizarre.

La signature

Les protocoles de signatures sont assez difficiles à décrire. Mais il est facile de comprendre qu'il est plus efficace de signer l'empreinte d'un message plutôt que le message tout entier. En effet, les algorithmes de signature utilisent souvent de la cryptographie asymétrique qui est beaucoup plus lente sur des objets de grande taille.

Les structures de données

Historiquement, les premières fonctions de hachage servaient à stocker des données à l'aide de tables de hachage. Les fonctions d'alors étaient beaucoup moins robustes qu'aujourd'hui mais en revanche beaucoup plus rapides et simples à décrire. Prenons par exemple la fonction hashcode() uti-

lisée par le langage Java. Pour hacher la chaîne de caractères "$a_1 a_2 \dots a_n$", hascode() transforme chaque caractères en nombre à l'aide du code ASCII ('a' donne 97), 'b' donne 98 etc.) puis calcule par récurrence $Ascii(a_n) + 31 \times hashcode(\text{“}a_1 a_2 \dots a_{n-1}\text{”})$ en ne gardant que les 32 derniers bits. Ainsi $hashcode(\text{“}ac\text{”}) = 97 \times 31 + 99 = 3136$.

3.2 Définitions

Définition 12. Une fonction de hachage h est une fonction

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

Nous donnons maintenant les propriétés requises sur les fonctions de hachage. Dans les définitions suivantes, il faut comprendre l'expression assez vague «il est difficile» comme «il est calculatoirement difficile», c'est à dire que l'on ne peut pas trouver d'algorithme avec une complexité raisonnable pour résoudre chaque problème.

Définition 13. «Pre image resistance» = à sens unique
Etant donné une sortie y, il est difficile de trouver x tel que $y = h(x)$.

Définition 14. «2nd image resistance» = faible résistance aux collisions
Etant donné une entrée x, il est difficile de trouver $x' \neq x$ tel que $h(x) = h(x')$.

Définition 15. «Collision resistance» = résistance forte aux collisions
Il est difficile de trouver x et x' distincts tels que $h(x) = h(x')$.

On calcule donc une version condensée y d'un message M. Le condensé doit être spécifique du message.
La construction de fonction de hachage nécessite l'utilisation de fonction de compression.

Définition 16. Une fonction $f : \{0,1\}^m \to \{0,1\}^n$ où $n < m$ est appelée une fonction de compression.

3.3 Paradoxe des anniversaires

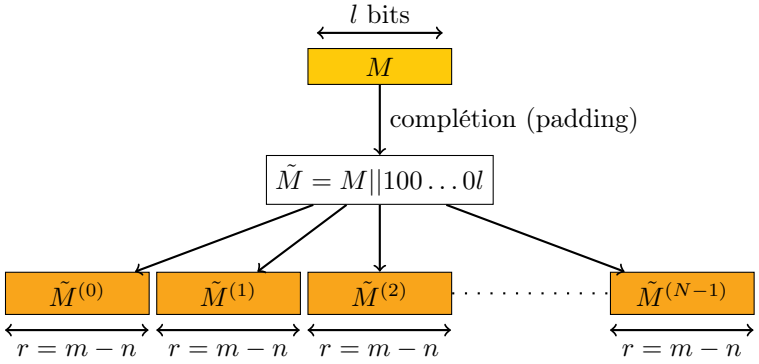
En supposant que l'on dispose d'une fonction de hachage idéale, c'est-à-dire qui se comporte comme une fonction pseudo-aléatoire, on peut trouver une collision avec un environ $O(\sqrt{n})$ calculs. En effet, supposons que l'on ait calculé k valeurs de la fonction de hachage. Appelons h_1, h_2, … h_k ces valeurs. Alors pour tout $i, j \in \{1; \dots; k\}$ avec $i < j$, on appelle δ_{ij} la variable aléatoire de Bernoulli qui vaut 1 si $h_i = h_j$ et 0 sinon. Le nombre de collisions que l'on trouvera dans la suite $(h_i)_{i \in \{1;\dots;k\}}$ est alors la variable aléatoire $X = \sum_{i<j} \delta_{ij}$. Et on a $E(X) = \frac{k(k+1)}{2} \frac{1}{n}$. Donc lorsque $k = O(\sqrt{n})$, la probabilité de trouver des collisions devient non négligeable.

Ainsi, pour une sécurité classique de résistance à 2^{80} calculs, il faut choisir $n = 160$ au minimum. Les standards actuels de sécurité préconisent même au moins 224 bits en sortie.

3.4 Construction de fonctions de hachage par le procédé de Merkle-Damgård

Soient M le message et f une fonction de compression. Posons $r = m - n$.

Complétion du message (padding) On construit à partir de M un message paddé $\tilde{M}$. On ajoute à la fin du message le bit "1" suivi de p zéros et de la représentation binaire de la longueur du message. On choisit p pour que le nombre total de bits du message paddé $\tilde{M}$ soit un multiple de r, soit Nr. On écrit $\tilde{M} = \tilde{M}^0\tilde{M}^1 \dots \tilde{M}^{N-1}$. Chaque $\tilde{M}^i$ est de taille r.

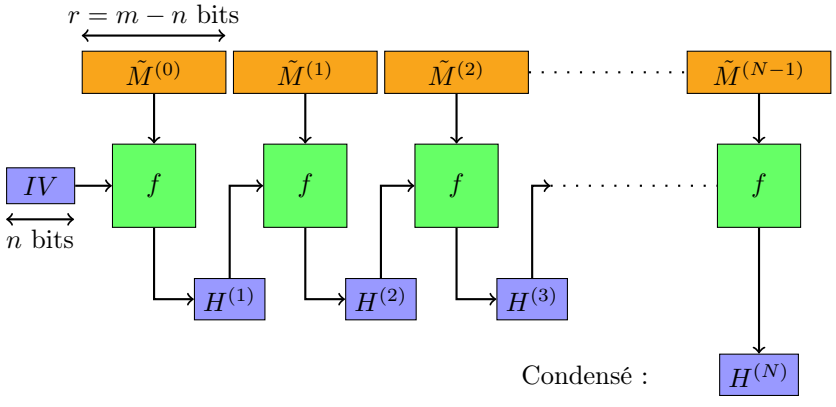


Algorithme de Merkle-Damgård Nous donnons maintenant la construction de la fonction de hachage. Soit IV un vecteur d'initialisation. $||$ représente la concaténation. On utilise l'algorithme suivant :

$I_0 \leftarrow IV||\tilde{M}^0$
$H^{(1)} \leftarrow f(I_0)$
Pour $i = 1$ *à* $N - 1$ **faire**
  $I_i \leftarrow H^{(i)}\|\widetilde{M}^{(i)}$
  $H^{(i+1)} \leftarrow f(I_i)$
finpour

Algorithme 2 : Merkle-Damgård

$H^{(N)}$ est le condensé du message.



Théorème 5. *Si la fonction de compression est résistante aux collisions, alors la fonction de hachage ainsi construite est résistante aux collisions.*

Démonstration. Par contraposée, si on trouve une collision pour notre fonction de hachage, il est facile de construire une collision pour la fonction de compression : soit la collision se produit à la dernière utilisation de f, soit elle a lieu en amont. □

Attaque par «length extension» Je vais maintenant décrire une attaque que l'on peut utiliser contre les schémas du type Merckle-Damgård, lorsque l'on arrive à fabriquer des collisions sur des condensés élémentaires.

L'idée est la suivante : si on trouve une collision entre deux messages M et M' sur un des condensés intermédiaires, alors on peut remplacer la fin des messages par ce que l'on veut, dès l'instant que la taille est la même, et on aura alors le même condensé final. On peut même raffiner le procédé en trouvant des collisions à différents niveaux, c'est à dire entre des condensés d'étapes différentes. Il faut alors combiner habilement différentes collisions pour se ramener à une collision sur deux messages de même longueur que l'on peut compléter à sa guise.

Évidement, cette attaque nécessite de trouver au moins une collision sur la fonction de compression. Lorsque c'est le cas, on sait qu'il faut rapidement abandonner la fonction de hachage. Pour ceux qui persistent à l'utiliser, cette attaque accentue les faiblesses de la fonction de hachage.

Un moyen d'éviter ce genre d'attaque est de garder une partie des variables internes cachées. Nous verrons que pour CRUNCH, une solution a été proposée en doublant les informations transmises à chaque étape.

3.5 Une histoire sanglante

Un site internet, appelé le zoo des fonctions de hachage (rajouter une référence) a répertorié 45 fonctions de hachage. La plus ancienne est la fonction md2 (message digest 2) crée par Rivest en 1989. Depuis, Rivest a sorti la fonction md4 et la célèbre fonction md5, largement implémentée dans tous les langages. Sur un terminal Unix par exemple, on peut calculer le haché d'un fichier en tapant *md5 NomDeFichier*. Sur ce site également, il est signalé les fonctions considérées comme «cassées», soit parce qu'une collision effective a été trouvée, soit parce qu'une attaque théorique a été publiée. C'est le cas par exemple pour la fonction SHA-1, qui a été la fonction standard de hachage préconisée par le NIST (National Institute of Standard and Technology). Malgré ces faiblesses, les fonctions md5 et SHA-1 continuent à être largement utilisées. Sur les 45 fonctions listées dans ce zoo, 25 ont été officiellement «cassées». Le NIST a réagi en proposant un nouveau standard SHA-2, qui se décline dans plusieurs versions suivant la sécurité désirée. Mais un des défauts du SHA-2 est sa lenteur. De plus, au vu de l'hécatombe récente, il n'est pas exclu que des chercheurs trouvent une faille également dans la fonction SHA-2, et le NIST se retrouverait sans remplaçant, c'est pourquoi un concours pour un nouveau standard, le SHA-3, a été ouvert en 2007. On connait maintenant le vainqueur de ce concours qui a duré 5 ans : Keccak, algorithme proposé par une équipe de cryptographes belges. C'est un algorithme de type «éponge», c'est à dire avec un état interne caché, qui évolue au cours du hachage, en absorbant certaines informations.

Chapitre 4

Un sha-3 candidat : CRUNCH

Le NIST a annoncé le 2 novembre 2007 le début officiel de la compétition pour une nouvelle fonction de hachage : SHA-3. La date limite pour les propositions d'algorithmes était le 31 octobre 2008. La famille de fonctions proposée devait pouvoir produire des condensés d'un message qui auront 224, 256, 384, ou 512 bits de sorte que l'intégrité du message soit préservée : tout changement dans le message produira un changement dans le condensé différent. Cette propriété est très utile pour engendrer et vérifier des signatures digitales et des codes d'authentification de messages (MAC), et aussi pour générer des chaînes de bits aléatoires.
Nous avons proposé une nouvelle fonction de hachage nommée CRUNCH, qui a été accepté pour le premier tour, et qui reste encore solide même si elle n'a pas été retenue pour les tours suivants. Un bon tiers des fonctions proposées ont été « cassées ».

4.0.1 La fonction de hachage construite pour la compétition du NIST : CRUNCH

Nous décrivons maintenant l'algorithme CRUNCH.

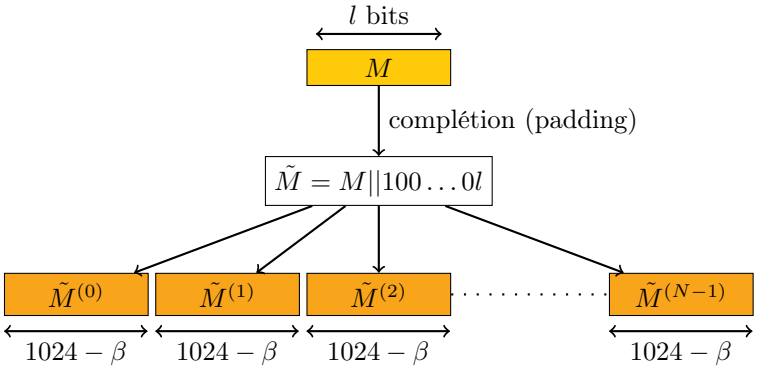
Constantes aléatoires Crunch utilise des nombres aléatoires pour les vecteurs d'initialisation ainsi que pour les permutations de chiffrement de sa fonction de compression. Pour générer et fixer ces valeurs nous utilisons des décimales de la fonction sinus. En tout il y a 1Mo de nombres aléatoires.

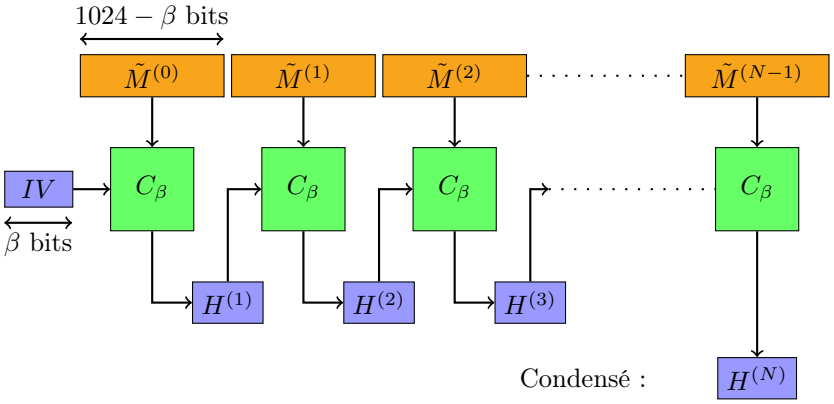
Notations $\beta \in \{224, 256, 384, 512\}$ est la taille du condensé voulu.

M est le message de départ. $\tilde{M}$ est le message complété qui contient N blocs de $1024 - \beta$ bits.

IV est le vecteur d'initialisation défini par des constantes aléatoires.

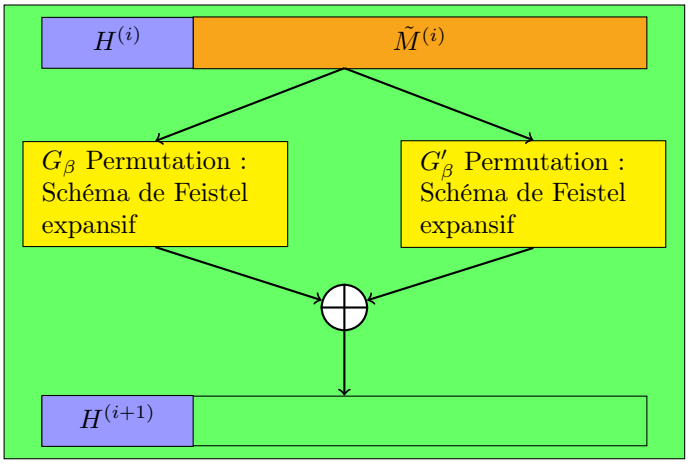
Complétion et mode d'entraînement (Merkle-Damgård)



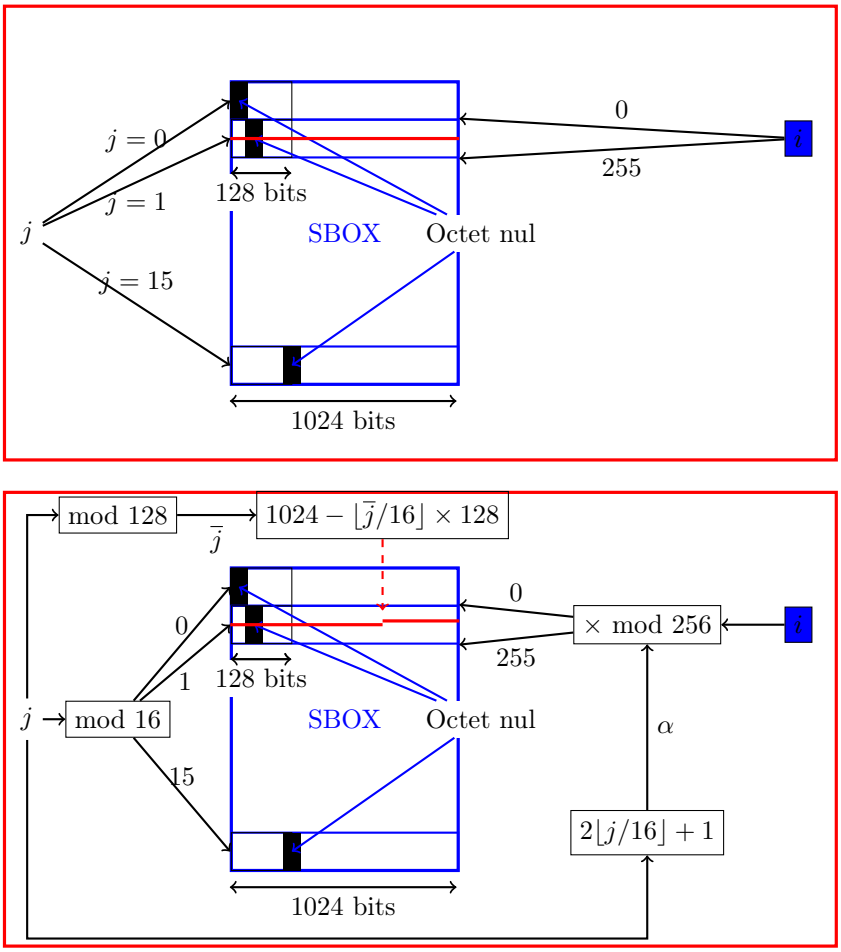


Fonctions de compression On appelle C_β les fonctions de compression. Elles sont définies comme le Xor de deux permutations appelées G_β et G'_β.

Le fait de prendre le Xor de deux permutations permet d'accroître la sécurité. Dans [41] et [45], il est montré que pour le Xor de deux permutations secrètes envoyant L vers L bits, la borne de sécurité pour le nombre de messages est 2^L.



Permutations de chiffrement Les permutations de chiffrement sont des schémas de Feistel expansifs que l'on a étudié précédemment. Le choix a été fait de prendre $n = 8$ et $k = 128$, pour des raisons d'implémentation informatique. Pour les fonctions expansives internes ont utilise les nombres aléatoires générés au début. On s'arrange pour que G et G' n'utilisent jamais les mêmes nombres, et pour que les 16 premières fonctions expansives de chacune des permutations utilise des zones distinctes de la partie de la mémoire où se trouvent les nombres aléatoires.



Choix du nombre de tours dans les schémas de Feistel Quand les fonctions internes sont secrètes, les attaques génériques sur les schéma de Feistel expansifs sont décrites dans [43, 66]. Dans le cas présent, les fonctions $g^j = (j_0^j, g_1^j, \ldots, g_{127}^j)$ sont publiques et ne sont pas complètement aléatoires puisque nous avons des contraintes liées à la mémoire. Donc les bornes de sécurité ne sont pas les mêmes que lorsque les fonctions internes sont publiques. Le nombre de tours choisis dépend de la taille du condensé. Il faut prendre en compte deux critères. Tout d'abord, il faut augmenter le nombre de tours donné par rapport au cas où les fonctions internes sont secrètes (cf [43, 66]). Ensuite, on a une meilleure sécurité si l'on s'assure que les octets sont tous utilisés le même nombre de fois. On va donc choisir un nombre de tours qui sera un multiple de 128, sauf pour $\beta = 224$. Finalement, si β est la longueur du condensé, le nombre de tours sera $d_\beta = \beta$. La table 4.0.1 donne le nombre de tours utilisés pour avoir le sécurité dans le cas d'un schéma de Feistel expansif et le nombre de tours choisis pour la permutation de chiffrement que l'on aura en fonction de la taille du condensé.

FIGURE 4.1 – Nombre de tours pour la sécurité.

Condensé= β (bits)	Sécurité= $\beta/2$ (bits)	F_d^{128} (Nombre de tours)	Permutation de chiffrement= d_β (Nombre de tours)
224	112	158	224
256	128	162	256
384	192	178	384
512	256	194	512

TABLE 4.1 – Propriétés de CRUNCH

Algorithme	Message (bits)	Blocs (bits)	Mots (bits)	Condensé (bits)	Sécurité [a] (bits)
CRUNCH-224	$< 2^{64}$	1024	32	224	112
CRUNCH-256	$< 2^{64}$	1024	32 or 64	256	128
CRUNCH-384	$< 2^{128}$	1024	64	384	192
CRUNCH-512	$< 2^{128}$	1024	64	512	256

a. Ici par, "sécurité" on utilise le fait qu'une attaque des anniversaires [HAC] sur un message de taille n produit une collision avec un facteur d'environ $2^{n/2}$.

4.0.2 Caractéristiques de CRUNCH

L'algorithme a été implémenté sur différentes machines. Nous donnons ici un exemple. Les détails se trouvent dans [18].

Processeur de 64 bits

Processeur : Intel Core 2 Duo E6400 @2.13GHz (cache L1 data 32KB, cache L2 2MB)
RAM : 4GB DDR2 dual channel
OS : kubuntu 8.04.1 64bits with KDE 3.5
compilateur : icc v10.1
Option de compilation : -fast

Condensé (bits)	Taille du message (MB)	Nombre de cycles	Vitesse (MB/s)
256	100	$16,95 * 10^9$	12,59
384	100	$29,62 * 10^9$	7,24
512	100	$46,97 * 10^9$	4,55

Les propriétés de CRUNCH sont données dans le tableau 4.1 :

4.0.3 Résistance aux attaques

Généralités. L'algorithme CRUNCH est basé sur le Xor de deux permutations. L'utilisation d'un chiffrement par blocs, pour la construction de fonctions de hachage, remonte à Preenel, Govaerts et Vandewalle [51] et les analyses faites

par Black, Rogaway et Shrimpton [5] qui ont montré que parmi les 24 constructions possibles, seules 20 parmi elles résistaient aux collisions jusqu'à la borne des anniversaires dans le modèle de la boîte noire. Cependant dans toutes ces constructions, la clé est changée à chaque tour et c'est un désavantage au niveau de l'efficacité. D'où l'idée de construire des fonctions de hachage à partir de chiffrements par blocs dont les clés sont fixées. Black, Cochran et Shrimpton [4] ont étudié la possibilité de construire une fonction de hachage sûre dont la fonction de compression fait appel exactement une fois à un chiffrement par blocs dont la clé est fixée. Ils ont obtenu une collision en faisant $O(n)$ appels à un oracle ayant accès à un schéma de chiffrement par blocs. Il n'est donc pas possible d'obtenir une preuve de sécurité envers un adversaire ayant une capacité de calculs illimitée. Donc il faut construire une fonction de compression faisant deux appels à une clé fixée pour le chiffrement par blocs.

Attaques par collision. Rogaway et Steinberger ont étudié dans [53] le cas des fonctions de hachage pour lesquelles la fonction de compression utilise deux appels à une clé fixé. Ils ont trouvé une attaque par collision qui donne une borne supérieure pour la sécurité. La meilleure attaque avec $O(2^{n/2})$ accès aux permutations et la complexité du temps de calcul est en $O(n2^{n/2})$. Donc il sera impossible d'avoir une sécurité contre les collisions meilleure que $O(2^{n/2})$ pour une construction basée sur deux permutations. Pour CRUNCH-β, $n = 1024$ et la meilleure attaque connue est l'attaque des anniversaires, dont la complexité est $O(2^{\beta})$.

Attaque pré-images Dans [53], Rogaway et Steinberger ont également étudié les attaques pré-images sur les fonctions de hachage faisant deux appels à une clé fixée. Pour les meilleures attaques, il faut $O(2^{n/2})$ accès aux permutations (et la complexité en temps est $> 2^n$). On ne peut donc pas avoir une sécurité en pré-image meilleure que $O(2^{n/2})$. Pour CRUNCH-β, on a $n = 1024$ et la meilleure sécurité en pré-image est en $O(2^{\beta})$.

Troisième partie

Authentification à clé publique

Chapitre 5

Généralités sur le Zero-Knowledge

5.1 Définitions utiles

5.1.1 Problèmes de la classe P, NP et NPC

Définition 17 (Calculabilité)**.** On dit qu'une fonction est calculable lorsqu'on peut la modéliser à l'aide d'une machine de Turing.

Proposition 13. *Toute fonction calculable a un inverse calculable.*

Démonstration. Il suffit en effet de calculer un à un l'image de tous les éléments jusqu'à ce qu'on tombe sur l'élément choisi. □

En théorie de la complexité, au lieu de fonction on préfère parler de problème. Un exemple de problème est :

> 3-coloriage
> ENTRÉE : Un graphe, c'est à dire un ensemble de sommets et d'arêtes (S, A).
> SORTIE : Un coloriage en 3 couleurs des sommets de telle sorte que deux sommets reliés par une arrête aient deux couleurs distinctes.

On appelle instance d'un problème la donnée d'une entrée du problème. La solution du problème est un algorithme qui, à partir d'une instance du problème, permet de donner la sortie.

5.1.2 Fonctions à sens unique, prédicats sûr

Définition 18 (Fonction à sens unique)**.** On dit que fonction $f\colon E \to F$ est à sens unique lorsque pour tout $x \in E$ il existe un algorithme efficace pour calculer $f(x)$ mais qu'en revanche la probabilité qu'un algorithme, même probabiliste, arrive à trouver un antécédent de l'image élément quelconque $y = f(x) \in F$ est négligeable.

Remarque. Dans l'état actuel de la cryptologie, on ne sait pas montrer si il existe ou non des fonctions à sens unique, même en supposant que $P \neq NP$! Pourtant ce genre de fonctions est constamment utilisé ...

Définition 19 (Prédicat sûr)**.** On dit qu'une fonction $P\colon E \to \{0, 1\}$ est un prédicat sûr pour la fonction $f\colon E \to F$, lorsque pour tout $x \in E$ il existe un algorithme efficace pour calculer $P(x)$, mais qu'en revanche il n'existe pas d'algorithme efficace qui à partir de la valeur de $f(x)$ peut deviner la valeur de $P(x)$ avec une probabilité meilleure que $1/2$ plus une quantité négligeable.

5.1.3 Les schémas de mise en gage

Les schémas de mise en gage (commitment schemes) sont des protocoles en deux phases entre deux personnes A et B, où A est la personne qui veut s'engager auprès de B sur un message m. Pendant la première phase, A prend une clé K au hasard avec laquelle elle chiffre le message d'une certaine manière et envoie à B son gage c. Dans la deuxième phase, A envoie la clé K à B qui peut vérifier que le gage correspondant à m est bien c.

Un schéma de mise en gage doit vérifier deux propriétés essentielles :

Dissimulation. La connaissance de c ne doit donner aucune information sur m.

Engagement. A ne peut pas tricher et envoyer une autre clé K' à B de sorte que l'ouverture du gage c donne un autre message m'.

Malheureusement, il est impossible de vérifier la première propriété contre un adversaire qui a une puissance de calcul illimité. Il lui suffit en effet d'essayer toutes les clés possibles pour retrouver le message m. C'est pourquoi nous allons plutôt parler de dissimulation calculatoire.

Définition 20 (Dissimulation calculatoire, engagement parfait)**.** Un schéma de mise en gage pour des messages de longueur ℓ avec une dissimulation calculatoire (t, ϵ) et un engagement parfait est un couple d'algorithmes (Com, O) tels que :

Justesse. Pour tout message m et toute clé K,

$$O(K, Com(K, m)) = m$$

Dissimulation calculatoire (t, ϵ). Pour tout couple de messages distincts m, $m' \in \{0,1\}^\ell$, les distributions $Com(K, m)$ et $Com(K, m')$ sont (t, ϵ) indistingable lorsque K est une clé aléatoire, c'est à dire, pour tout algorithme A de complexité inférieure ou égale à t,

$$|\mathbb{P}[A(Com(K, m)) = 1] - \mathbb{P}[A(Com(K, m') = 1]| \leq \epsilon$$

Engagement parfait. Pour tout message m et tout couple de clés K, K',

$$O(K', Com(K, m)) \in \{m, \text{Echec}\}$$

Dans ce qui suit nous parlerons simplement de schéma de mise en gage (t, ϵ) sécurisé.

Exemple. Soit $f\colon \{0,1\}^n \to \{0,1\}^n$ une bijection à sens unique, c'est à dire calculatoirement difficile à inverser, et $P\colon \{0,1\}^* \to \{0,1\}^\ell$ un prédicat sûr de la fonction f. On peut poser :

— $Com(K, m) = (f(K), m \oplus h(K))$.

— $O(K, (c_1, c_2)) =$ Echec si $f(K) \neq c_1$ et $h(K) \oplus c_2$ sinon.

Remarque. On peut proposer une fonction de mise en gage plus simple dont la sécurité repose sur la difficulté de trouver une collision d'une fonction de hachage h. Voici cette fonction :

— $Com(K, m) = h(K||m)$.

— $O(K, c) = m$ lorsque $h(K||m) = c$

Cela nécessite donc d'essayer toutes les valeurs de m telles que $h(K||m) = c$, mais en général il y a peu de valeurs de m possibles (souvent ℓ vaut 1 ou 2), ou bien on peut se débrouiller autrement en renvoyant également m avec K pour que B vérifie la justesse du gage.

5.2 Les schémas à divulgation nulle de connaissance

5.2.1 Introduction

Une preuve à divulgation nulle de connaissance (Zero-knowledge Proof) est un concept utilisé en cryptologie dans le cadre de l'authentification et de l'identification. Cette expression désigne un protocole sécurisé dans lequel une entité

nommée « fournisseur de preuve », prouve mathématiquement à une autre entité, le « vérificateur », qu'une proposition est vraie sans toutefois révéler une autre information que la véracité de la proposition.

En pratique, ces schémas se présentent souvent sous la forme d'un protocole de type « stimulation/réponse » (challenge/response). Le vérificateur et le fournisseur de preuve s'échangent des informations et le vérificateur accepte ou rejette la preuve.

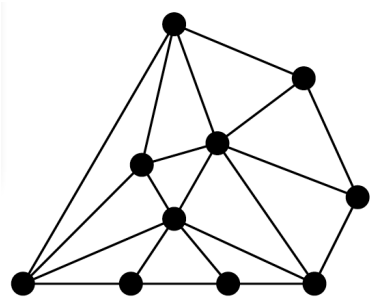
Les premiers schémas Zero-knowlegde ont été basés sur le problème de la factorisation (voir par exemple Fischer-Micali-Rackoff en 1984, ou Fiat-Shamir en 1986), ou encore sur le problème d'isomorphisme de graphes. Puis, en 1991, O. Goldreich, S. Micali et A. Widgerson ont montré que tout problème NP-complet admet une preuve Zero-Knowledge([17]). Cependant le problème de la factorisation n'est pas supposé NP-complet (puisqu'il est dans NP et dans Co NP) et on a un algorithme sous-exponentiel (NFS par exemple) et même il existe un algorithme polynômial avec des ordinateurs quantiques (algorithme de Shor). De plus, la construction générale d'un algorithme Zero-Knowledge à partir de tout problème NP-complet n'est pas très efficace (cf [17]). C'est pourquoi, différents schémas Zero-Knowledge ont été construits à partir de problèmes NP-complets bien choisis et basés sur des problèmes combinatoires supposés de difficulté exponentielle, PKP d' Adi Shamir [59], PP de David Pointcheval [50] ou CLE ou SD [61] de Jacques Stern par exemple.

On va maintenant illustrer cette notion sur un exemple.

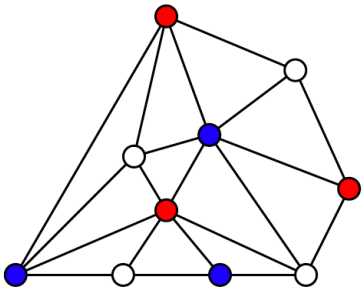
5.2.2 Un exemple : le problème des 3 couleurs

Le coloriage d'un graphe en trois couleurs, en s'imposant de colorier différemment deux sommets qui sont reliés par une arrête, est un problème NP-complet. Pourtant, pour un graphe pris au hasard (on s'impose un nombre de sommets par exemple) il est en général facile de savoir si on peut ou non le colorier, puisque la réponse est rapidement négative. Mais pour les graphes qui sont effectivement coloriables, ou bien presque coloriables la résolution de ce problème est prouvée difficile, alors que sa vérification reste triviale.

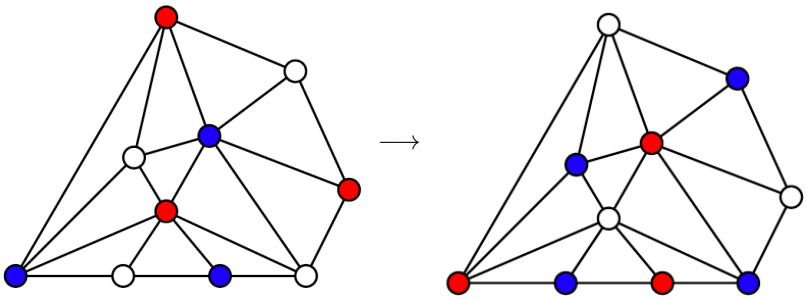
On suppose par exemple qu'Alice sait colorier ce graphe qu'elle rend public :



En fait elle n'a pas de mérite parce qu'elle a commencé par dessiner des sommets en nombre suffisant (sur l'exemple ce n'est évidemment pas le cas) puis elle les a coloriés un peu au hasard et a relié des sommets dont les couleurs étaient différentes. Voilà par exemple la solution d'Alice :

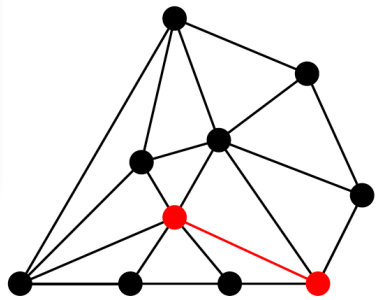


Alice veut alors prouver à Bob qu'elle sait le colorier sans révéler sa solution. Elle commence alors par brouiller les pistes en permutant au hasard les couleurs, par exemple :



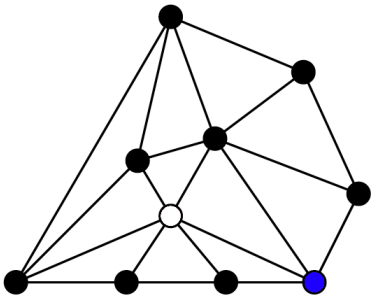
Puis, pour chacun des sommets, elle se donne une clé aléatoire (10 clés dans l'exemple) et s'engage vis à vis de Bob sur chacune des couleurs. Elle lui envoie donc 10 gages c_1, c_2, ... c_{10}. C'est la première passe.

Bob va alors la questionner et demander qu'on lui révèle la couleur de deux des sommets qui sont reliés par une arrête. Par exemple :



Il lui pose la question, c'est la deuxième passe.

La troisième passe consiste alors à l'envoi par Alice des deux clés qui correspondent aux gages sur les deux sommets. Bob vérifie que les couleurs des deux sommets sont bien distinctes.



On peut classer en trois catégories les propriétés vérifiées par un tel schéma. On verra dans la section suivante comment cela peut se formaliser. Les trois points essentiels sont :

1. La consistance : une personne qui connaît une solution répondra toujours bien.
2. La solidité : une personne qui est capable de répondre à toutes les questions peut trouver une solution.
3. Divulgation nulle de connaissance (ZK) : on ne révèle rien du secret.

Dans notre exemple, le premier point est vérifié sans problème. Pour le second, si Alice peut répondre correctement à toutes les questions possibles de Bob, c'est qu'Alice a vraiment réussi à colorier le graphe. Alice ne peut pas tricher car elle s'engage sur chacune des couleurs des sommets. Enfin, le troisième point est lui aussi vérifié car les couleurs sont mélangées au départ. Lorsqu'Alice répond à Bob, les couleurs ont la même probabilité d'avoir toutes les combinaisons possibles.

Évidemment un tricheur peut essayer de se faire passer pour Alice. On peut par exemple supposer qu'au mieux il arrive à colorier tout le graphe mais qu'une seule arrête lui cause problème. La probabilité qu'il soit découvert est donc supérieure ou égale à $1/A$ où A est le nombre total d'arrêtes. Maintenant, si on répète le protocole depuis le début n fois de façon indépendante, la probabilité que le tricheur ne soit pas découvert est $\left(1 - \frac{1}{A}\right)^n$. Il suffit alors de choisir n suffisamment grand pour que cette probabilité soit négligeable.

Nous allons maintenant définir rigoureusement ces notions.

5.2.3 Notations - Définitions

Dans un protocole interactif, il y a deux entités : le fournisseur de preuve et le vérificateur. Le fournisseur de preuve veut convaincre le vérificateur qu'il connaît une secret. Ils vont interagir et à la fin, le vérificateur accepte ou refuse. Dans les protocoles Zero-knowlegde, il y a une probabilité de tricherie. Un tricheur sera capable de répondre à certaines questions mais pas à toutes les questions. Il faut donc construire le protocole pour que la réponse à une des questions ne fournit aucune indication sur le secret, mais si une personne est capable de répondre à toutes les questions, alors le secret sera révélé. Nous donnons maintenant les définitions permettant de décrire les propriétés devant être satisfaites dans les protocoles.

Consistance : Un fournisseur de preuve honnête est toujours accepté.

Solidité calculatoire : Un fournisseur de preuve malhonnête est rejeté avec une probabilité fixe. On montre généralement que le fournisseur de preuve qui répond à toutes les questions peut être utilisé pour extraire le secret.

Divulgation nulle de connaissance : On veut qu'aucune stratégie du vérificateur ne puisse obtenir aucune information du fournisseur de preuve. Pour cela, les preuves sont faites à l'aide de simulation en utilisant le vérificateur comme un oracle ou une boîte noire. On a la définition suivante [17] :

Black box zero-knowledge : une stratégie P pour le fournisseur de preuve est black box zero-knowledge sur les entrées S s'il existe un simulateur efficace U tel que pour toute stratégie V du vérificateur, les deux distributions de probabilité sont calculatoirement indistingables :

— $\{(P, V)(x)\}_{x \in S}=$ toutes les sorties de V quand il interagit avec P sur $x \in S$.

— $\{U(V)(x)\}_{x \in S}=$ les sorties de U utilisant V comme une boîte noire sur $x \in S$.

Chapitre 6

ZK avec un Rubik's cube

Depuis l'invention du Rubik's cube par Ernö Rubik en 1974, d'autres puzzles de ce type ont été conçus avec un nombre de faces ou des formes variés. Le problème principal lié à ce casse-tête consiste, à partir d'une position aléatoire, de retrouver la position initiale du cube. Ce problème peut sembler difficile à première vue, mais il existe maintenant des algorithmes efficaces pour résoudre ce problème, même pour des Rubik's cubes avec un nombre supérieur de facettes [10]. Néanmoins, à l'aide de ce jouet, nous allons définir des problèmes difficiles d'un point de vue informatique. Par exemple, comment passer d'une position à une autre avec un nombre de mouvements égal ou inférieur à une valeur fixée d, valeur qui rend justement la solution unique ou presque. Dans [10] il est montré que trouver la solution optimale (avec le minimum de mouvements) est NP-difficile si on ignore certaines facettes du cube $n \times n \times n$. De plus la taille du groupe lié au Rubik's cube augmente de façon exponentielle avec le nombre de facettes. Nous verrons également dans la section 6.5 les relations entre ces problèmes et certains problèmes NP-complet ou NP-space.

Par conséquent, on peut essayer de construire des schémas Zero-Knowledge avec une sécurité basée sur la difficulté de résoudre ces problèmes et également sur une fonction de hachage utilisée pour les mises en gage. On pourra ensuite utiliser ce protocole comme un moyen de s'identifier. Comme le dit Colmez [7], le Rubik's cube est un des rares groupes avec lequel on peut se promener (un autre groupe vérifiant cette propriété est le groupe des noeuds). C'est donc assez satisfaisant de penser que l'on peut avoir sa clé d'identification en poche !

Bien sûr, il existe des algorithmes qui transforment automatiquement tout problème de la classe NP en un protocole d'authentification Zéro-Knowledge (cf [17]). La manière de le faire est polynomiale mais néanmoins cela produit des protocoles généralement peu efficaces. C'est pourquoi nous allons définir de nouveaux protocoles, plus astucieux, que l'on pourra de surcroit utiliser de façon plus générale, notamment avec des problèmes de factorisation dans des groupes non-commutatifs.

On peut imaginer une façon pratique de s'authentifier, avec notre cube en poche. Il suffit d'apprendre par cœur la suite des coups, puis de la reproduire à l'aveugle, sous un foulard par exemple. Le problème de cette méthode est qu'il est assez compliqué de vérifier le nombre de mouvements effectués, et, de plus, certaines personnes particulièrement douées et entraînées arrivent à résoudre de cette façon n'importe quelle position du Rubik's cube ! Ce n'est donc pas un moyen sûr de s'authentifier mais on peut retenir le procédé pour des jeux d'enfants par exemple.

Nous allons décrire notre protocole, en se plaçant dans le cas où le nombre de mouvements est imposé (égal à d et non inférieur ou égal). Trois procédés différents vont servir à cacher la solution. D'abord le cube est tourné de façon aléatoire (24 possibilités), ce qui a pour effet de mélanger les différents mouvements autorisés. Par exemple, un quart de tour positif de la face du haut peut devenir un quart de tour positif de la face de droite. Ainsi, à chaque étape on peut toujours voir que l'on a effectué un des mouvements du cube. Évidement, ce masquage ne suffit pas puisqu'il est facile de deviner comment le cube a été tourné, en tout cas aux étapes proches du début et de la fin. Le deuxième procédé consiste donc à cacher les mouvements successifs à l'aide d'une unique permutation aléatoire qui va préserver la composition. Enfin, le dernier procédé, classique, consiste à s'engager sur la plupart de ces opérations. Il va s'ensuivre trois passes entre la personne qui veut s'identifier, i.e. celle qui veut prouver

qu'elle connaît le secret, et la personne qui veut vérifier que le fournisseur de preuve ne triche pas. La première passe consiste à l'envoi des engagements sur la rotation du cube et les mouvements cachés, la deuxième est une question du vérificateur, et la troisième est la réponse à la question (certains engagements sont révélés).

Il est important de noter que l'ordre des deux premiers procédés ne peut être changé. Pour le premier procédé consistant à tourner le cube, nous allons introduire une notion nouvelle sur les groupes, que nous avons appelé le groupe de repositionnement. Grâce à cette notion, nous pouvons généraliser le protocole pour n'importe quel groupe fini non commutatif.

Après avoir décrit le protocole dans le cas du Rubik's cube classique $3\times3\times3$, nous verrons comment l'adapter à son grand frère le $5\times5\times5$. Pour celui-ci nous avons du surmonter une difficulté : en tournant le cube, on ne mélange pas vraiment tous les mouvements puisqu'un mouvement d'une couronne extérieure restera un mouvement d'une couronne extérieure. En terme mathématique, cela se traduit par la non-existence d'un groupe de repositionnement attaché au générateurs. L'intérêt de ce nouveau casse-tête est la taille de son groupe qui est enfin largement suffisante pour avoir une sécurité standard.

Enfin, nous verrons brièvement comment utiliser le schéma lorsque le nombre de mouvements est juste majoré par une certaine valeur. Ce qui équivaut donc à un problème de factorisation. Il y a des articles récents [52] qui nous aide à donner une approximation de cette valeur pour quelques groupes ou quelques ensembles particuliers de générateurs. Ces travaux essayent de répondre à la conjecture de Babai sur le diamètre des groupes simples. Dans [10] il est montré que le «nombre de Dieu», i.e. le nombre minimum de mouvements pour résoudre le Rubik's cube généralisé $n\times n\times n$ est $\Theta(n^2/log(n))$.

6.1 Notations et définitions

6.1.1 Notations mathématiques standards et définitions

La plupart de ces notations peuvent se trouver dans l'excellent livre de Joyner [22], qui est une façon originale et ludique d'aborder l'algèbre.

Soit X un ensemble fini, S_X est le groupe symétrique de X. En particulier quand $X=\{1;2;\ldots;n\}$ où $n\in\mathbb{N}^*$, ce groupe est noté S_n. Soient $(\sigma,\sigma')\in {S_X}^2$, alors $\sigma\sigma'$ représente la composée $\sigma'\circ\sigma$.
Pour tout groupe G, si $(g_1,g_2,\ldots,g_\alpha)\in G^\alpha$, alors $\langle g_1,g_2,\ldots,g_\alpha\rangle$ désigne le sous-groupe de G engendré par $g_1,g_2,\ldots g_\alpha$.
$\mathcal{F}=\{g_1,\ldots,g_\alpha\}$ est un **ensemble de générateurs** de G lorsque $\langle g_1,g_2,\ldots,g_\alpha\rangle=G$. Cet ensemble est **symétrique** lorsque pour tout $\sigma\in\mathcal{F}$ on a $\sigma^{-1}\in\mathcal{F}$.
Lorsque l'on a un ensemble de générateurs $\mathcal{F}$ d'un groupe G, on dit que deux éléments de G sont en relation ssi $g^{-1}g'\in\mathcal{F}$. Le graphe correspondant est appelé le **graphe de Cayley graph** du groupe.
Soit G un groupe, la **conjugaison** sur G est définie par :

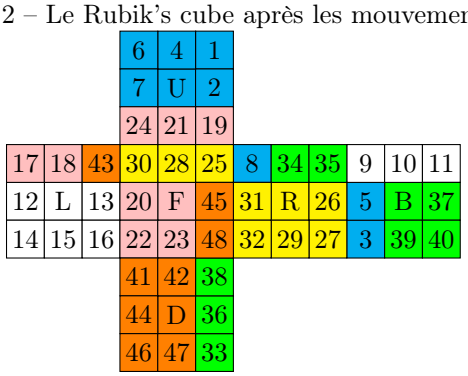
$$\forall(\sigma,\tau)\in G^2,\quad \sigma^\tau=\tau^{-1}\sigma\tau$$

De plus nous avons :

$$\forall(\sigma,\sigma',\tau,\tau')\in G^4,\quad (\sigma^\tau)^{\tau'}=\sigma^{\tau\tau'},\quad \sigma^\tau\sigma'^\tau=(\sigma\sigma')^\tau$$

FIGURE 6.1 – Rubik's cube creux

FIGURE 6.2 – Le Rubik's cube après les mouvements R et U

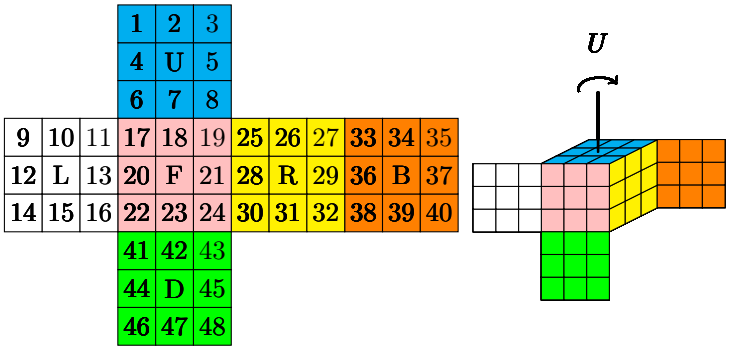


On utilisera aussi la notation généralisée pour les ensembles : $\boxed{\sigma^G = \{\sigma^g | g \in G\}}$.

Soit X un ensemble fini, on écrit $x \in_R X$ quand x est choisi aléatoirement de manière uniforme dans X.

6.1.2 Représentation mathématique du Rubik's cube

Nous écrivons un nombre sur chaque facette excepté les milieux. Dans nos protocoles, nous supposerons toujours que les centres des faces sont blanches ou vides (il existe d'ailleurs un Rubik's cube «creux» composé de 22 petits cubes cf figure 6.1). Le fait de ne pas considérer les centres ne va pas changer sensiblement les complexités des problèmes que l'on va étudier.



On définit ensuite les 6 permutations de S_{48} qui sont des quarts de tour des faces dans le sens des aiguilles d'une montre :

F = (17,19,24,22)(18,21,23,20)(6,25,43,16)(7,28,42,13)(8,30,41,11)
B = (33,35,40,38)(34,37,39,36)(3,9,46,32)(2,12,47,29)(1,14,48,27)
L = (9,11,16,14)(10,13,15,12)(1,17,41,40)(4,20,44,37)(6,22,46,35)
R = (25,27,32,30)(26,29,31,28)(3,38,43,19)(5,36,45,21)(8,33,48,24)
U = (1,3,8,6)(2,5,7,4)(9,33,25,17)(10,34,26,18)(11,35,27,19)
D = (41,43,48,46)(42,45,47,44)(14,22,30,38)(15,23,31,39)(16,24,32,40)

Si nous appliquons la permutation σ, alors la face $i \in \{1, 2, \ldots, 48\}$ sera dans la position $\sigma(i)$.
Le groupe des permutations du Rubik's cube est $< F, B, L, R, U, D > \subset S_{48}$. On peut simuler le Rubik's cube, en utilisant SAGE [60].

6.1.3 Le groupe de repositionnement

Dans cette sous-section, nous allons définir précisément le groupe de repositionnement pour un ensemble donné de générateurs $\mathcal{F}$. Nous allons voir que l'existence de ce genre de groupe est essentiel pour la construction de nos protocoles.

Définition 21. Soit $\mathcal{F} = \{f_1, \ldots, f_\alpha\} \subset G$, où G est un groupe. Si il existe un sous-groupe $H \subset G$ tel que ${f_1}^H = \{h^{-1} f_1 h \mid h \in H\} = \mathcal{F}$ alors H est appelé groupe de repositionnement de $\mathcal{F}$.

Remarque. Dans le cas du Rubik's cube, avec $\mathcal{F} = \{F, B, L, R, U, D\}$, il est facile de voire que l'on transforme une permutation de base en une autre en faisant rouler le cube sur lui même comme un dé.

Proposition 14. *On suppose que $\mathcal{F}$ a un groupe de repositionnement H. Si nous choisissons $\tau \in_R H$, $P({f_i}^\tau = f_j) = \frac{1}{\alpha}$ pour tout $(i, j) \in \{1; \ldots; \alpha\}^2$.*

Démonstration. Puisque ${f_1}^H = \mathcal{F}$, pour tout $i \in \{1, \ldots; \alpha\}$ il existe $\tau_i \in H$ tel que ${f_1}^{\tau_i} = f_i$. Alors, pour tout $j \in \{1; \ldots; \alpha\}$, ${f_i}^{{\tau_i}^{-1}\tau_j} = f_j$. Notons $\tau_{ij} = {\tau_i}^{-1}\tau_j$. Nous avons alors l'équivalence suivante :

$$ {f_i}^\tau = f_j \iff {f_k}^{\tau_{ki}\tau\tau_{j\ell}} = f_\ell $$

pour tout $k, \ell \in \{1; \ldots; \alpha\}$. Donc $\{\tau \in H \mid {f_i}^\tau = f_j\}$ and $\{\tau \in H \mid {f_k}^\tau = f_\ell\}$ sont en bijection et ont le même cardinal. □

Remarque. Il n'est pas facile de trouver un groupe de repositionnement dans le cas général. Lorsque les éléments de $\mathcal{F}$ ne sont pas conjugués, c'est même impossible. Nous verrons qu'il y a tout de même un moyen de contourner ce problème, quelle que soit la compositon de $\mathcal{F}$, en utilisant une extension d'ensemble (on va plonger G dans un groupe plus gros). Néanmoins la construction générale n'est pas souvent la construction optimale. Par «optimale», nous voulons dire avec un groupe de repositionnement le plus petit possible. Par exemple, dans le cas du Rubik's cube $5 \times 5 \times 5$, il suffit de plonger G dans le groupe ${S_{144}}^2$ au lieu de ${S_{144}}^{12}$, comme le suggérerait le cas général.

6.2 Différents problèmes de factorisation

Pour tous les problèmes suivants, nous considérons un groupe fini G avec un ensemble de générateurs $\mathcal{F} = \{f_1; f_2; \ldots; f_\alpha\}$ contenant toutes les permutations autorisées. On a également $f_i \neq f_j$ pour tout $i \neq j$.
$id \in X$ représente l'état initial.

Problème 1: *Résoudre le puzzle.*
Étant donné $x_0 \in G$, trouver $d \in \mathbb{N}$ et $i_1, i_2, \ldots, i_d \in \{1; 2; \ldots; \alpha\}$ tels que

$$x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = id$$

Remarque. Ce problème est équivalent au problème de factorisation dans G avec des éléments de $\mathcal{F}$ parce que :

$$x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = id \quad \Longleftrightarrow \quad x_0^{-1} = f_{i_1} f_{i_2} \ldots f_{i_d}$$

Problème 2: *résoudre le puzzle en un nombre fixé de coups.*
Étant donné $x_0 \in G$ et $d \in \mathbb{N}^*$, trouver $i_1, i_2, \ldots, i_d \in \{1; 2; \ldots; \alpha\}$ tels que

$$x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = id$$

Proposition 15. *Nous pouvons trouver une solution du problème 2 avec $O(2\alpha^{d/2})$ calculs lorsque d est pair.*

Démonstration. Il s'agit ici d'une attaque par le milieu. $f_{i_1} f_{i_2} \ldots f_{i_d} = x_0$ est équivalent à $x_0 f_{i_1} \ldots f_{i_{d/2}} = (f_{i_d})^{-1} \ldots (f_{i_{d/2+1}})^{-1}$.
Donc pour tout $i_1, i_2, \ldots, i_{d/2} \in \{1, \ldots, \alpha\}$ on calcule

$$\begin{aligned} & Y_{i_1 i_2 \ldots i_{d/2}} &=& \; f_{i_1} f_{i_2} \ldots f_{i_{d/2}} * x_0 \\ \text{et} \quad & Z_{i_1 i_2 \ldots i_{d/2}} &=& \; (f_{i_1})^{-1} (f_{i_2})^{-1} \ldots (f_{i_{d/2}})^{-1} \end{aligned}$$

on cherche ensuite des collisions entre Y et Z. □

Remarque. Nous donnons ici (α, d) quand la complexité de l'attaque est égale à 2^{80}

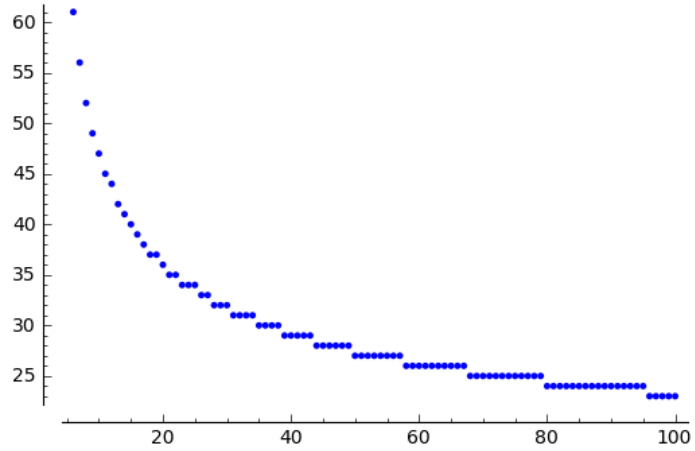
α	2	3	4	5	6	8	10	12
d	158	99	79	68	61	52	47	44

La Figure. 6.3 donne les résultats pour des grandes valeurs de α.

Remarque. Il y a d'autres techniques de factorisation [48] utilisant des chaînes de groupes. Cependant aucune de ces techniques ne nous permet de trouver la solution minimale.

Dans cette section, nous allons étudier comment transformer ces problèmes difficiles en des schémas d'identification avec divulgation nulle de connaissances. En d'autres mots, nous allons donner une méthode pour prouver que nous possédons une solution sans en révéler une quelconque partie.

FIGURE 6.3 – (α, d) pour $\alpha \geq 6$ et une complexité en 2^{80} calculs



6.3 Protocole à partir du Rubik's cube $3 \times 3 \times 3$

6.3.1 Introduction

Nous allons d'abord décrire notre schéma d'authentification dans le cas d'un Rubik's cube classique $3 \times 3 \times 3$. Nous procédons ainsi afin d'introduire les principales idées, cependant avec ce Rubik's cube la complexité du problème 2 est beaucoup plus petite que 2^{80} et donc il ne peut pas être utilisé pour des applications cryptographiques. Pour ces dernières, nous pourrons utiliser le Cube $5 \times 5 \times 5$ comme nous le verrons plus tard.

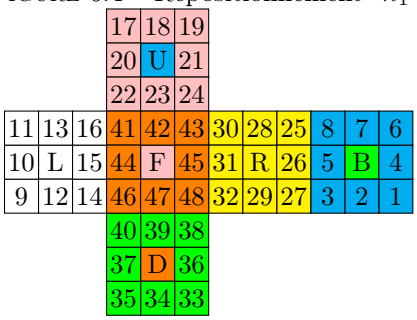
Nous avons en fait à peu près $43{,}2 \times 10^{18}$ combinaisons différentes du cube, c'est à dire environ 2^{61} ou 6^{25}. Si on considère qu'un demi-tour compte seulement pour un mouvement, alors on sait que le « nombre de Dieu », c'est-à-dire le nombre minimum de coups qui permet de résoudre n'importe quelle position du cube, est 20 [54]. Cependant dans notre cas nous n'autorisons pas les mouvements dans le sens contraire des aiguilles d'une montre, ni les demi-tours. Donc il parait raisonnable de prendre $d = 24$ pour le problème 2 , et on aura une sécurité de l'ordre de $6^{24/2} = 6^{12} \approx 2^{30}$ calculs.

6.3.2 Dissimuler le secret

D'abord nous devons cacher chaque permutation que l'on fait pour passer d'une position à une autre, tout en gardant l'information que l'on utilise une des permutations autorisées, c'est à dire un des éléments de $\mathcal{F}$. Avec le cube $3 \times 3 \times 3$, on peut le faire facilement en roulant le cube comme un dé (je rappelle que l'on ne s'occupe pas des centres des faces).

Soit H le groupe des déplacements de l'espace laissant invariant le cube. H est engendré par deux éléments : $H = \langle h_1, h_2 \rangle$ avec h_1 qui correspond au fait de tourner le cube en le posant sur sa face arrière (voir figure 6.4) , et h_2 la rotation du cube en le laissant sur la table et en faisant un quart de tour dans

FIGURE 6.4 – Repositionnement "h_1"



le sens des aiguilles d'une montre (vu du haut). De façon explicite nous avons :

$$\begin{aligned} h_1 &= RL^{-1}(2,39,42,18)(7,34,47,23) \\ h_2 &= UD^{-1}(13,37,29,21)(12,36,28,20) \end{aligned}$$

Il est facile de vérifier que card$H = 24$, parce que pour chaque face mise au dessus, on a quatre choix pour la face de devant. De plus, on a $U^H = \mathcal{F}$.

Proposition 16. *Si $f \in_R \mathcal{F}$ et $\tau \in_R H$, alors f^τ est une variable aléatoire avec une loi uniforme à valeur dans $\mathcal{F}$.*

Démonstration. C'est une conséquence directe de la proposition 17. □

Illustration. Soit $x_0 \in G_R = \langle \mathcal{F} \rangle$ une position du cube et $x_1 = x_0 f$, le diagramme suivant est commutatif (i.e. $f\tau = \tau f^\tau$) :

$$\begin{array}{ccc} x_0 & \xrightarrow{f} & x_1 \\ {\scriptstyle\tau}\downarrow & & {\scriptstyle\tau}\downarrow \\ x_0\tau & \xrightarrow{f^\tau} & x_1\tau \end{array}$$

Pour continuer à dissimuler le secret, nous devons maintenant cacher les conjugués des mouvements autorisés secrets. Nous allons le réaliser à l'aide d'un simple masquage. Le masque sera une permutation σ_0 tirée au hasard parmi les permutations de G. Si $f_{i_1}, f_{i_2}, \ldots f_{i_d}$ sont les permutations secrètes, nous cachons leurs conjugués de cette façon (on définit σ_j pour tout $j \in \{1;2;\ldots;d\}$) : ${f_{i_1}}^\tau = \sigma_0\sigma_1^{-1}$, puis ${f_{i_2}}^\tau = \sigma_1\sigma_2^{-1}$, ..., et ${f_{i_d}}^\tau = \sigma_{d-1}\sigma_d^{-1}$. Donc on a $f_{i_1} \ldots f_{i_d} = \sigma_0{\sigma_d}^{-1}$.

6.3.3 Le protocole Zero-Knowledge

Dans cette sous-section, nous allons donner le protocole général, valable pour n'importe quel groupe G et n'importe quel ensemble de générateurs $\mathcal{F}$ d'un sous-groupe G_R de G, dans la mesure où l'on dispose d'un groupe de repositionnement $H \subset G$. On prouvera dans la sous-section suivante que ce schéma est zéro knowledge.

On peut également utiliser ce protocole pour le puzzle que nous avons appelé $S41$ et qui est décrit dans la section 6.6.

Public :

— Un groupe G.
— Un ensemble $\mathcal{F} = \{f_1, \ldots, f_\alpha\} \subset G$ de générateurs de G_R
— Un groupe de repositionnement $H \subset G$ tel que ${f_1}^H = \mathcal{F}$.
— $d \in \mathbb{N}$, $d \geq 3$
— G' sous-groupe de G engendré par $\mathcal{F}$ et H. $G' = \langle \mathcal{F}, H \rangle$.
— K un ensemble de clé, $|K| \geq 2^{80}$.

Clé secrète : $i_1, i_2, \ldots, i_d \in \{1, 2, \ldots, \alpha\}$.
Clé publique : $x_0 = (f_{i_1} f_{i_2} \ldots f_{i_d})^{-1}$
Schéma (un tour) :

Prouveur		**Vérifieur**
Prend $\tau \in_R H$, $\sigma_0 \in_R G'$, $k_*, k_0, k_1, \ldots, k_d \in_R K$ **Calcule** $\forall j \in \{1, \ldots, d\}$, $\sigma_j = ({f_{i_j}}^\tau)^{-1} \sigma_{j-1}$ $c_0 = Com_{k_*}(\tau)$ $\forall i \in \{0, \ldots d\}$, $s_i = Com_{k_i}(\sigma_i)$	$c_0, s_0, \ldots, s_d$ $\longrightarrow$	
	$\longleftarrow$ q	**Choisit** $q \in_R \{0, \ldots d\}$
Cas $q = 0$	τ, σ_0 $\longrightarrow$ $\longleftarrow$ k_*, k_0, k_d	**Calcule** $\sigma_d = \tau^{-1} x_0 \tau \sigma_0$ **Vérifie** $\tau \in H$, $Com_{k_*}(\tau) = c_0$, $Com_{k_0}(\sigma_0) = s_0$, $Com_{k_d}(\sigma_d) = s_d$ **Si** tous les tests sont ok **alors accepte sinon rejette.**
Cas $q \neq 0$	${f_{i_q}}^\tau$, σ_q $\longrightarrow$ $\longleftarrow$ k_{q-1}, k_q	**Calcule** $\sigma_{q-1} = {f_{i_q}}^\tau \sigma_q$ **Vérifie** ${f_{i_q}}^\tau \in \mathcal{F}$, $s_{q-1} = Com_{k_{q-1}}(\sigma_{q-1})$ $s_q = Com_{k_q}(\sigma_q)$ **Si** Tous les tests ok **alors accepte sinon rejette.**

Remarque. Puisque H est un petit groupe, nous pouvons modifier le protocole en n'envoyant pas c_0 dans la première phase, et en n'envoyant que σ_0 dans le premier cas de la troisième phase. Le vérifieur devra ensuite essayer toutes les valeurs possibles de τ. Cela met d'ailleurs en évidence que l'on n'utilise pas la taille de H comme argument de sécurité.

Illustration. Si on définit pour tout $k \in \{1; \ldots; d\}$, $x_k = x_{k-1} f_{i_k}$, nous avons le diagramme commutatif suivant :

$$\begin{array}{ccccccc}
x_0 & \xrightarrow{f_{i_1}} & x_1 & \xrightarrow{f_{i_2}} & \dots & x_{d-1} & \xrightarrow{f_{i_d}} & x_d = id \\
\tau\downarrow & & \tau\downarrow & & & \tau\downarrow & & \tau\downarrow \\
x_0\tau & \xrightarrow[\sigma_0{\sigma_1}^{-1}]{{f_{i_1}}^{\tau}} & x_1\tau & \xrightarrow[\sigma_1{\sigma_2}^{-1}]{{f_{i_2}}^{\tau}} & \dots & x_{d-1}\tau & \xrightarrow[\sigma_{d-1}{\sigma_d}^{-1}]{{f_{i_d}}^{\tau}} & \tau
\end{array}$$

Lorsque $q = 0$, le Vérifieur regarde si la composition par le chemin extérieur est correcte :

$$\begin{array}{ccc}
x_0 & & x_d = id \\
\tau\downarrow & & {\tau^{-1}}\uparrow \\
x_0\tau & \xrightarrow[\sigma_0{\sigma_d}^{-1}]{} & \tau
\end{array}$$

Lorsque $q \neq 0$, le Vérifieur s'occupe d'une des mailles :

$$\begin{array}{ccc}
 & \xrightarrow{f_{i_q}} & \\
\tau\downarrow & & \tau\downarrow \\
 & \xrightarrow[\sigma_{q-1}{\sigma_q}^{-1}]{{f_{i_q}}^{\tau}} &
\end{array}$$

Ici τ n'est pas révélé, nous avons seulement un élément aléatoire σ_{q-1} de G et un élément aléatoire de $\mathcal{F}$, donc nous ne donnons aucune information sur le secret.

6.3.4 Preuves du protocole

Consistance.

Il est assez clair qu'un véritable Prouveur sera toujours accepté.

Solidité.

On commence par supposer qu'un Prouveur peut répondre correctement pour toutes les valeurs de q (c'est à dire que ses réponses sont toujours acceptées par le Vérifieur). Si nous ne mettons pas en évidence une collision pour la fonction de hachage utilisée pour les mises en gage (voir [19]) alors :

— σ_0 révélé lorsque $q = 0$ est le même que celui révélé pour $q = 1$.

— σ_i pour $i \in \{1; \dots; d-1\}$ révélé lorsque $q = i$ est le même que celui qui est calculé lorsque $q = i + 1$.

— σ_d révélé lorsque $q = d$ est le même que celui qui est calculé lorsque $q = 0$.

Pour tout $i \in \{1; \dots; d\}$, le Vérifieur s'est assuré que $\sigma_{i-1}{\sigma_i}^{-1} \in \mathcal{F}$, donc posons $u_i \in \{1; \dots; \alpha\}$ tel que $f_{u_i} = \sigma_{i-1}{\sigma_i}^{-1}$.

Lorsque $q = 0$, le vérifieur a établi $id = x_0\tau\sigma_0{\sigma_d}^{-1}\tau^{-1}$, donc

$$\begin{aligned}
id &= x_0\tau(\sigma_0{\sigma_1}^{-1})(\sigma_1{\sigma_2}^{-1})\dots(\sigma_{d-1}{\sigma_d}^{-1})\tau^{-1} \\
&= x_0\tau f_{u_1}f_{u_2}\dots f_{u_d}\tau^{-1} = \tau f_{u_1}\tau^{-1}\tau f_{u_2}\tau^{-1}\dots\tau f_{u_d}\tau^{-1} \\
&= x_0{f_{u_1}}^{\tau^{-1}}{f_{u_2}}^{\tau^{-1}}\dots{f_{u_d}}^{\tau^{-1}}
\end{aligned}$$

Le vérifieur peut donc construire une solution au problème initial.

Ainsi, si on considère un Prouveur malhonnête, c'est à dire une personne qui ne connait pas de solution au problème initial, il y a au moins l'une des demandes du Vérifieur qui ne sera pas satisfaite, et donc sera rejetée, donc la probabilité qu'un Prouveur malhonnête puisse convaincre un vérifieur honnête est plus petite ou égale à $\frac{d}{d+1}$.

Zero knowledge

D'abord, nous commençons par montrer que pour un Prouveur honnête, chaque réponse est donné comme une variable aléatoire uniforme.

- $q = 0$.
  Le Prouveur donne $(\tau, \sigma_0, k_*, k_0, k_d) \in H \times G' \times K^3$ qui sont toutes des permutations aléatoires dans leur ensemble.
- $1 \leq q \leq d$.
  Le Prouveur donne $({f_{i_q}}^\tau, \sigma_q, k_{q-1}, k_q) \in F \times G' \times K^2$. Puisque nous avons $f_{i_1} \dots f_{i_q} = \tau\sigma_0\sigma_q^{-1}\tau^{-1}$, si on pose $h = \tau^{-1}{f_{i_q}}^{-1} \dots {f_{i_1}}^{-1}\tau$, alors $\sigma_q = h\sigma_0$ avec σ_0 pris au hasard dans G', donc σ_q est une permutation aléatoire indépendante de ${f_{i_q}}^\tau$. De plus, puisque $\tau \in_R H$, ${f_{i_q}}^\tau$ est pris aléatoirement et de façon uniforme dans F (voir section 6.3.2, ou bien la proposition 17 qui généralise la propriété). Donc nous avons à nouveau une probabilité uniforme sur $F \times G' \times K^2$.

Ensuite, nous pouvons construire un simulateur qui prend x_0 en entrée, sans rien connaître du secret, et qui exécute le protocole avec un Vérifieur malhonnête. Nous allons montrer que le simulateur peut se faire passer pour un honnête Prouveur avec une probabilité $\frac{1}{d+1}$. Le simulateur choisit au hasard une valeur $q^* \in_R \{0; 1; \dots; d\}$, comme étant une valeur que le Vérifieur malhonnête ne va pas choisir. On considère deux cas :

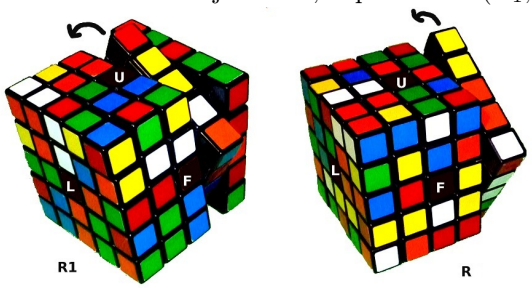
- $q^* = 0$
  Le simulateur prend $\tau \in_R H$, $f'_1, \dots, f'_d \in_R F$ et $\sigma_0 \in_R G'$. Puis il calcule pour tout $k \in \{1; \dots; d\}$ $\sigma_k = {f'_k}^{-1}\sigma_{k-1}$.
- $1 \leq q^* \leq d$
  Il prend $f'_1, f'_2, \dots, f'_{q^*-1}, f'_{q^*+1}, \dots f'_d \in_R \mathcal{F}$. Puis il prend $\tau \in_R H$ et $\sigma_0 \in_R G'$. Il calcule $f'_{q^*} \in G'$ (pas nécessairement dans $\mathcal{F}$) tel que $x_0 f'_1 \dots f'_d = id$, et pour tout $k \in \{1; \dots; d\}$ $\sigma_k = {f'_k}^{\tau^{-1}}\sigma_{k-1}$

Il est facile de vérifier que, à part le cas $q = q^*$, toutes les demandes du Vérifieur malhonnête auront une réponse qui sera acceptée. Donc la probabilité qu'une réponse soit rejetée est $\frac{1}{d+1}$. De plus, lorsque l'on reporte seulement les échanges sans problème, on ne pourra pas distinguer les échanges de ce que l'on aurait obtenu avec un Prouveur honnête.

6.3.5 Choix du nombre de tours pour le Rubik's cube $3 \times 3 \times 3$

Nous revenons au cas du Rubik's cube classique, et nous allons choisir le nombre de fois que l'on doit exécuter le protocole pour prouver avec une probabilité raisonnable que l'on connait le secret. On appelle r ce nombre (r comme «round») qui représente le nombre de tours. Par exemple, si on veut que cette probabilité soit égale à $1 - 2^{-m}$, on doit avoir $\left(\frac{d}{d+1}\right)^r \leq 2^{-m}$, autrement dit

FIGURE 6.5 – Cubes jumeaux, déplacement (R_1, R)



$r \approx md\ln(2)$. Par exemple avec $m = 30$ et $d = 24$, il suffit de faire 500 tours. Nous avons simulé le schéma avec un processeur tournant à 3GHz computer, et en utilisant une fonction de hachage plutôt lente par rapport au standard, et cela a pris moins d'une seconde pour simuler 100 fois les 500 tours.

6.4 Rubik's cube $5 \times 5 \times 5$

Pour une authentification cryptographique avec les standards de sécurité actuels, on a besoin d'avoir au moins 2^{160} positions différentes du cube. Le Rubik's cube $4 \times 4 \times 4$ n'a que 2^{152} positions. Donc, nous avons choisi le membre suivant de la famille, i.e. le cube $5 \times 5 \times 5$. On dénombre en effet 2^{247} positions (calcul effectué avec SAGE).

6.4.1 Représentation mathématiques

Nous écrivons un nombre sur chaque facette, à l'exception des centres. Pour la manipulation du cube, on se restreint aux 12 permutations de base suivantes : les 6 quarts de tour de la couronne extérieure de chaque face dans le sens des aiguilles d'une montre (U, D, F, B, R, L), et les 6 quarts de tour de la couronne intermédiaire de chaque face, dans le sens des aiguilles d'une montre $(U_1, D_1, F_1, B_1, R_1, L_1)$. Mais nous pourrions faire d'autres choix. Nous avons :

$$G_R = \langle U, D, \ldots, L, U_1, D_1, \ldots, L_1 \rangle \subset S_{144}$$

6.4.2 Dissimuler le secret

On remarque qu'il est insuffisant de faire rouler le cube pour brouiller les pistes. En effet, cela mélangera de façon uniforme toutes les rotations des couronnes supérieures et toutes les rotations des couronnes intermédiaires. Nous avons besoin d'une nouvelle idée. On voit d'ailleurs, en observant les cycles des permutations supérieures et intermédiaires, que l'on ne peut pas espérer trouver un groupe de repositionnement puisqu'elles ne sont pas conjuguées. Une façon de contourner ce problème est de dupliquer le cube. Au lieu de considérer un cube, on va donc en considérer deux, et chaque fois que l'on tournera une couronne extérieure sur le premier cube, on tournera la couronne intermédiaire de la même face sur l'autre cube (voir figure 6.5). On va donc considérer le groupe $G_R \times G_R$.

On appellera $\boldsymbol{e}$ l'échange des deux cubes. $\boldsymbol{e}$ est donc un nouvel élément qui échange les coordonnées :

$$e^2 = id \quad \text{et} \quad \forall (a,b) \in G_R \times G_R, \quad \boldsymbol{e}(a,b)\boldsymbol{e} = (b,a).$$

Pour prouver l'existence d'un tel élément, on peut utiliser une morphisme injectif de groupe de G^2 dansS_{288}, en supposant que les facettes du nouveau cube sont numérotées de 145 à 288.
Alors, $e = (1{,}145)(2{,}146)\dots(144{,}188)$ vérifie la propriété demandée. Cependant, pour des raisons de clarté, on va garder la notation dans G^2.
On définit $\boldsymbol{F} = \{(U,U_1),\dots,(L,L_1),(U_1,U),\dots,(L_1,L)\}$ et $\boldsymbol{G}_R = \langle \boldsymbol{F} \rangle \subset G_R \times G_R$. La taille de $\boldsymbol{G}_R$ est à peu près la même que G_R. Avec Sage, on trouve explicitement card$G_R \approx 2^{300}$ et card$\boldsymbol{G}_R \approx 2^{364}$. On cachera donc le mouvement effectué sur le premier cube en roulant les deux cubes comme dans la section précédente, et en échangeant (ou non !) les deux cubes. Donc nous avons $\boldsymbol{H} = \langle (h_1,h_1),(h_2,h_2),\boldsymbol{e} \rangle$. Cette fois-ci, notre groupe de repositionnement $\boldsymbol{H}$ a 48 éléments : les 24 rotations du cubes, et toutes ces rotations combinées avec un échange du cube.

6.4.3 Protocole Zero knowledge

Le protocole décrit dans la section précédente fonctionne lorsque l'ensemble de générateurs dispose d'un groupe de repositionnement. Nous venons de voir que dans le cas du Rubik's cube $5\times5\times5$, nous pouvons construire un groupe de repositionnement en se plaçant dans G^2 au lieu de G. On généralise d'ailleurs facilement ce procédé pour les Rubik's cubes suivants : pour les $6\times6\times6$ et $7\times7\times7$, on se placera dans G^3, et dans G^n pour les Rubik's cubes $(2n)^3$ et $(2n+1)^3$). Nous verrons également dans la section suivante comment construire ces groupes dans tous les cas, mais la construction générale impose de se placer dans G^α où α est le nombre de générateurs (ou déplacements autorisés), ce qui, dans le cas du cube $5\times5\times5$, nous donnerait un schéma beaucoup plus lourd puisque $\alpha = 12$.

Nous donnons ci-dessous le protocole pour un entier $n \geq 2$ quelconque, donc adaptable pour tous les Rubik's cubes généralisés.
Publique :

- Un groupe G^n.
- Un ensemble $\mathcal{F} = \{f_1,\dots,f_\alpha\}$ de generateurs de $G_R \subset G$
- Un ensemble $\boldsymbol{F} = \{\boldsymbol{f}^1,\dots,\boldsymbol{f}^\alpha\} \subset G^n$ avec $\boldsymbol{f}^i = (f_1^i = f_i, f_2^i, \dots f_n^i)$ pour tout $i \in \{1,\dots\alpha\}$.
- Un groupe de repositionnement $\boldsymbol{H}$ tel que ${\boldsymbol{f}^1}^{\boldsymbol{H}} = \boldsymbol{F}$.
- $d \in \mathbb{N}$, $d \geq 3$
- Un groupe $\boldsymbol{G}_R$ engendré par $\boldsymbol{F}$ et $\boldsymbol{H}$. $\boldsymbol{G}_R = \langle \boldsymbol{F}, \boldsymbol{H} \rangle$.
- K un ensemble de clé, $|K| \geq 2^{80}$.

Clé privée : $i_1, i_2, \dots, i_d \in \{1,2,\dots,\alpha\}$.
Clé publique : $x_0 = (f_{i_1} f_{i_2} \dots f_{i_d})^{-1}$ (or $\boldsymbol{X}^0 = (x_0, id, \dots, id)$)
Schéma (un tour) :

Prouveur		**Vérifieur**
Prend $\boldsymbol{\tau} \in_R \boldsymbol{H}$, $\boldsymbol{\sigma}_0 \in_R \boldsymbol{G}_R$, $k_*, k_0, k_1, \ldots, k_d \in_R K$ **Calcule** $\forall j \in \{1, \ldots, d\}$, $\boldsymbol{\sigma}_j = (\boldsymbol{f}^{i_j{}^{\boldsymbol{\tau}}})^{-1}\boldsymbol{\sigma}_{j-1}$ $c_0 = Com_{k_*}(\boldsymbol{\tau})$ $\forall i \in \{0, \ldots, d\}$, $s_i = Com_{k_i}(\boldsymbol{\sigma}_i)$	$c_0, s_0, \ldots, s_d$ $\longrightarrow$	
	$\longleftarrow$ q	**Prend** $q \in_R \{0, \ldots d\}$
Cas $q = 0$	$\boldsymbol{\tau}$, $\boldsymbol{\sigma}_0$, $\boldsymbol{\sigma}_d$ $\longrightarrow$ k_*, k_0, k_d	**Calcule** $\boldsymbol{X}^d = \boldsymbol{X}^0{}^{\boldsymbol{\tau}\boldsymbol{\sigma}_0\boldsymbol{\sigma_d}^{-1}\boldsymbol{\tau}^{-1}}$ **Vérifie** $\boldsymbol{X}_1^d = id$ $\boldsymbol{\tau} \in \boldsymbol{H}$, $Com_{k_*}(\boldsymbol{\tau}) = c_0$, $Com_{k_0}(\boldsymbol{\sigma}_0) = s_0$, $Com_{k_d}(\boldsymbol{\sigma}_d) = s_d$ **Si** tous les tests sont bons **alors accepte sinon rejette.**
Cas $q \neq 0$	$\boldsymbol{f}_{i_q}{}^{\boldsymbol{\tau}}$, $\boldsymbol{\sigma}_q$ $\longrightarrow$ k_{q-1}, k_q	**Calcule** $\boldsymbol{\sigma}_{q-1} = \boldsymbol{f}_{i_q}{}^{\boldsymbol{\tau}}\boldsymbol{\sigma}_q$ **Vérifie** $\boldsymbol{f}_{i_q}{}^{\boldsymbol{\tau}} \in \boldsymbol{F}$, $s_{q-1} = Com_{k_{q-1}}(\boldsymbol{\sigma}_{q-1})$ $s_q = Com_{k_q}(\boldsymbol{\sigma}_q)$ **Si** tous les tests sont bons **alors accepte sinon rejette.**

La preuve du protocole est la même que pour le protocole précédent.

6.4.4 Choix de d et du nombre de tours pour le cube $5 \times 5 \times 5$.

Si nous suivons l'attaque générique donnée dans la proposition 15, nous devons choisir $d = 46$. On a en fait $12^{46} \approx 2^{165}$ ce qui est beaucoup plus petit que le nombre total de positions. Néanmoins, lorsque l'on choisit i_1, i_2 ..., i_d la plupart du temps la solution n'est pas unique car on peut inverser toutes les permutations qui commutent. On peut alors imposer que deux permutations consécutives doivent être égale ou bien ne doivent pas commuter. Il y a alors $12 \times 9^{d-1}$ combinaisons possibles. En prenant $\boxed{d = 52}$ nous obtenons $12 \times 9^{49} \approx 2^{165}$.

Le **nombre de tours nécessaires** est $\boxed{1081}$, puisque $\left(\frac{51}{52}\right)^{1081} \leq 2^{-30}$.

Généralisation pour S_n. Pour appliquer le protocole précédent dans le cas général, nous avons besoin d'un groupe de repositionnement.

Définition 22. Soit $\mathcal{F} = \{f_1, \ldots, f_\alpha\} \subset S_n$. Un sous-groupe $H \subset S_n$ est appelé un groupe de repositionnement de $\mathcal{F}$, si ${f_1}^H = \mathcal{F}$.

Proposition 17. *Supposons que $\mathcal{F}$ admette un groupe de repositionnement H. Soit S_1 le H-stabilisateur de f_1 i.e. $S_1 = \{\tau \in H \mid {f_1}^\tau = f_1\}$, alors $|H| = |\mathcal{F}| \times |S_1|$. Donc si $\tau \in_R H$, alors pour tout (i,j), $P({f_i}^\tau = f_j) = \frac{1}{\alpha}$.*

Nous avons vu précédemment qu'il n'est pas facile de trouver un groupe de repositionnement. Donc dans le cas général, si les fonctions de $\mathcal{F}$ ne sont pas conjuguées c'est même impossible. On va élargir les ensembles. Nous donnons ici un algorithme gńéral pour construire H. Mais il est possible parfois d'obtenir de meilleures solutions comme nous l'avons fait dans le cas du Rubik's cube $5 \times 5 \times 5$.
Méthode générale : Soient $G = S_n$, $I_n = \{1; 2; \ldots n\}$, $\alpha \in \mathbb{N}$ ($\alpha \geq 2$), $\mathcal{F} = \{f_1, \ldots, f_\alpha\}$. Pour tout $k \in \{0, \ldots \alpha - 1\}$, on pose $I_n^{(k)} = \{1 + kn; \ldots; (k+1)n\}$. On définit $f_k^{(i)}$ comme étant la même permutation que f_k mais agissant sur $I_n^{(i)}$ au lieu I_n. Pour tout $k \in \{1; 2; \ldots; \alpha\}$, on pose :

$$\tilde{f}_k = f_k^{(0)} f_{k+1}^{(1)} \ldots f_\alpha^{(\alpha-k)} f_1^{(\alpha-k+1)} \ldots f_{k-1}^{(\alpha-1)}$$

considérées comme des permutations sur $I_{\alpha n}$. Soient $\tilde{G} = S_{\alpha n}$ et $\sigma \in \tilde{G}$ définis par :

$$\forall i \in \{1; \ldots; \alpha n\}, \quad \sigma(i) = \begin{cases} i + n & \text{si} \quad i \leq (\alpha - 1)n \\ i - (\alpha - 1)n & \text{si} \quad i > (\alpha - 1)n \end{cases}$$

Alors$H =< \sigma >$ est le groupe de repositionnement de $\tilde{\mathcal{F}} = \{\tilde{f}_1; \ldots; \tilde{f}_\alpha\}$.
On utilise ensuite le protocole précédent.

6.5 Les connections entre les problèmes liés au Rubik's cube et les problèmes P-space complets et NP Complets

Il y a donc plusieurs problèmes liés au Rubik's cube et à ses généralisations. La façon dont évolue la complexité lorsque n augmente n'est pas connue par exemple. Nous ne savons pas si certains de ces problèmes sont NP-complets (voir [25] p. 27). Il est même vraisemblable que ces problèmes ne soient pas NP-complets car leur description est trop limitée pour pouvoir décrire tous les problèmes de la classe NP.
Cependant, certains problèmes NP-complets ou P-space complets présentent des similarités avec ceux du Rubik's cube. On peut donc considérer que ces problèmes se trouvent dans une classe voisine de celle des problèmes NP-complets ou P-space complets. Ceci n'est pas une preuve de leur difficulté mais c'est un argument indirect la justifier.
Exemple 1 D'après [15] p. 280 et [28] nous savons que le problème "Finite Function Generation" est P-space complet.

Finite Function Generation
DONNEES : Un ensemble fini A, une collection F de fonctions $f\colon A \to A$ et une fonction particulière $h\colon A \to A$.
QUESTION : h est-elle la composée de fonctions de F ?

Remarque. On remarque que dans ce problème, le nombre de fonctions entrant dans la composition n'est pas spécifié, ce qui n'était pas le cas dans les problèmes que nous avons considérés avec le Rubik's cube et cette valeur de d est primordiale pour le complexité du problème.

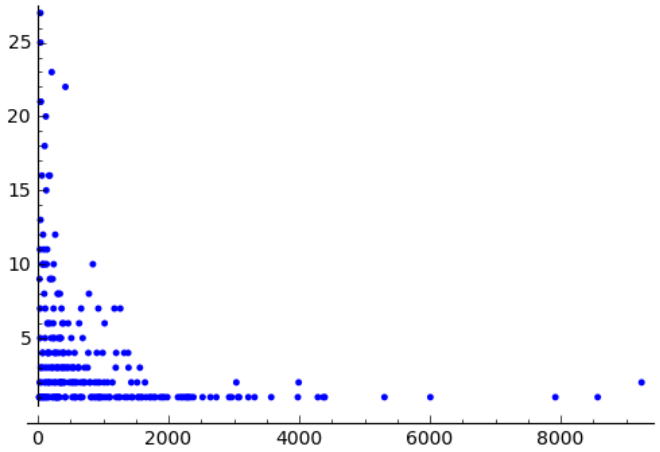
Exemple 2 D'après [15] p. 213, nous savons que le problème "Longest path" est NP complet.
LONGEST PATH
DONNEES : Graphe $G = (V, E)$, pour chaque $e \in E$ une longueur length $l(e) \in \mathbb{Z}^+$, un entier positif K, deux sommets $s, t \in V$
QUESTION : Existe-t-il un chemin simple dans de longueur supérieure ou égale à K et allant de s à t ?

Remarque. Ce problème est encore NP-complet si $l(e) = 1$ pour tout $e \in E$. Donc ce problème présente des similarités avec le problème du Rubik's cube quand on va d'une position à une autre. Cependant, ce problème devient polynômial quand la condition sur la longueur devient "inférieure ou égale" au lieu de "supérieure ou égale" (voir [15] p. 79). Cependant, si nous modélisons le graphe G tel que chaque sommet représente une position du Rubik's cube $n \times n \times n$, alors le nombre de sommets va croître de manière exponentielle en n.

6.6 Un nouveau puzzle appelé $S41$

Nous allons vous présenter un nouveau puzzle dont les performances semblent intéressantes. Nous l'avons appelé $S41$ parce qu'il est construit dans le groupe symétrique S_{41}, qui est le premier groupe symétrique avec un cardinal supérieur à 2^{160}. Plus précisément, le cardinal de ce groupe avoisine 2^{165}.

Á l'aide du logiciel SAGE, nous avons tiré au hasard deux permutations du groupe h et f_1, jusqu'à ce que le groupe engendré par ces deux éléments soit S_{41} tout entier. On calcule alors $f_1^{\langle h \rangle} = \alpha$. En répétant l'opération 1000 fois on obtient le graphe suivant (α en abscisse, et en ordonnée le nombre d'essais correspondant à cette valeur de α).



Pour avoir la plus petite valeur possible pour d, on choisit la valeur de α la plus grande possible. On obtient ainsi un schéma d'authentification avec un nombre réduit de tours. Voici les valeurs que nous avons trouvées :

$$\begin{aligned} h &= (1,14,39,19,31,18,37)(3,36,4,23,20,34,16,25,17,26,35) \\ &\quad (5,13,30,33)(6,7,10)(8,24,15,38,41,27,11,9) \\ &\quad (12,40,32,21,28)(22,29), \end{aligned}$$

and

$$\begin{aligned} f_1 &= (1,11,31,6,17,34,25,24,22,12,4,28,3,14,5,27,32,13,26,8,23,2, \\ &\quad 20,41,19,10,40,15,38,16,37,39,35,21,18)(7,29,36)(9,30). \end{aligned}$$

On pose alors $H = \langle h \rangle$ et $\mathcal{F} = {f_1}^H$. On vérifie avec SAGE que $\mathcal{F}$ est un ensemble de générateurs de S_{41} et on a $|H| = |\mathcal{F}| = \alpha = 9240$. On peut donc fixer $d = 12$ pour une sécurité de 79 bits. Et pour avoir une authentification avec un risque d'erreur inférieur à 2^{-30}, seulement $r = 260$ tours sont nécessaires.

Chapitre 7

ZK avec des polynômes à plusieurs variables

Un schéma zero-knowledge basé sur une solution d'un système d'équations quadratiques sur un corps fini a été proposé dans [56]. Nous proposons deux méthodes pour généraliser cette construction en utilisant des systèmes de polynômes multivariés de degré d quelconque. Nous construisons ainsi deux schémas notés $ZK(d)$ et $\widetilde{ZK}(d)$. Le schéma $\widetilde{ZK}(d)$ est optimal si on considère le nombre de calculs à faire durant les échanges et le schéma $ZK(d)$ est optimal en terme de bits envoyés. De plus, ces résultats sont satisfaits pour tous types de polynômes : lacunaires ou denses. Nous présentons ensuite deux applications : l'une avec les équations de Brent et l'autre avec les morphismes de polynômes. Nous exposons ici le cas $d = 3$ pour simplifier la présentation. C'est aussi ce cas qui sera utilisé pour les applications. La généralisation se trouve dans [35].

7.1 Le schéma $ZK(3)$

Soient les fonctions cubiques suivantes : $F(x) = (f_1(x), f_2(x), \ldots, f_m(x))$ où $\forall \ell,\ 1 \leq \ell \leq m$ et $x = (x_1, \ldots, x_n)$

$$f_\ell(x) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma_{ijk}^\ell x_i x_j x_k + \sum_{1 \leq i \leq j \leq n} \gamma_{ij}^\ell x_i x_j + \sum_{1 \leq i \leq n} \gamma_i^\ell x_i$$

Il n'y a pas de terme constant. Soit

$$G(x, y, z) = F(x+y+z) - F(x+y) - F(x+z) - F(y+z) + F(x) + F(y) + F(z)$$

On a : $G(x, y, z) = (g_1(x, y, z), g_2(x, y, z), \ldots, g_m(x, y, z))$
où $\forall \ell,\ 1 \leq \ell \leq m$, $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ et $z = (z_1, \ldots, z_n)$

$$g_\ell(x, y, z) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma_{ijk}^\ell (x_i y_j z_k + x_i y_k z_j + x_j y_i z_k + x_j y_k z_i + x_k y_i z_j + x_k y_j z_i)$$

La fonction G ainsi définie est trilinéaire.
Nous étudions le problème suivant : Etant donnés F et $v \in \mathbb{F}_q^m$ trouver $s \in \mathbb{F}_q^n$ tel que $F(s) = v$.
La clé publique est (F, v). Le secret est s tel que $F(s) = v$. Le fournisseur de preuve veut convaincre le vérificateur qu'il connaît ce secret.

Description du schéma à 3 échanges. Pour simplifier le présentation, la chaîne de bits aléatoire de la fonction Com ne sera pas écrite de manière explicite. Si X représente un ensemble, on utilise la notation $x \in_R E$ pour signifier que x est choisi aléatoirement dans X avec une distribution de probabilité uniforme.

1. Le fournisseur de preuve choisit $r_0, r_1, t_0 \in_R \mathbb{F}_q^n$ et $e_0, f_0, h_0 \in_R \mathbb{F}_q^m$. Ensuite, il calcule

   $$r_2 = s - r_1 - r_0,\ \ t_1 = r_0 - t_0$$
   $$e_1 = F(r_0) - e_0,\ \ f_1 = F(r_0 + r_1) - f_0,\ \ h_1 = F(r_0 + r_2) - h_0$$

   Le fournisseur de preuve envoie alors au vérificateur

   $$c_0 = Com(r_1, r_2, G(t_0, r_1, r_2) - e_0 + f_0 + h_0)$$
   $$c_1 = Com(r_1, t_0, e_0, f_0),\ \ c_2 = Com(r_1, t_1, e_1, f_1)$$
   $$c_3 = Com(r_2, t_0, e_0, h_0),\ \ c_4 = Com(r_2, t_1, e_1, h_1)$$

2. Le vérificateur choisit une stimulation $\mathcal{Q} \in_R \{0,1,2,3\}$ et envoie $\mathcal{Q}$ au fournisseur de preuve.
3. (a) Si $\mathcal{Q} = 0$, alors le fournisseur de preuve envoie $(r_0, r_1, t_1, e_1, f_1)$. Le vérificateur contrôle si $c_1 = Com(r_1, r_0 - t_1, F(r_0) - e_1, F(r_0 + r_1) - f_1)$, $c_2 = Com(r_1, t_1, e_1, f_1)$

   (b) Si $\mathcal{Q} = 1$ alors le fournisseur de preuve envoie $(r_0, r_2, t_1, e_1, h_1)$. Le vérificateur contrôle si $c_3 = Com(r_2, r_0 - t_1, F(r_0) - e_1, F(r_0 + r_2) - h_1)$, $c_4 = Com(r_2, t_1, e_1, h_1)$

   (c) Si $\mathcal{Q} = 2$ alors le fournisseur de preuve envoie $(r_1, r_2, t_1, e_1, f_1, h_1)$. Le vérificateur contrôle si $c_0 = Com(r_1, r_2, v - G(t_1, r_1, r_2) + e_1 - f_1 - h_1 - F(r_1 + r_2) + F(r_1) + F(r_2))$, $c_2 = Com(r_1, t_1, e_1, f_1)$, $c_4 = Com(r_2, t_1, e_1, h_1)$

   (d) Si $\mathcal{Q} = 3$ alors le fournisseur de preuve envoie $(r_1, r_2, t_0, e_0, f_0, h_0)$. Le vérificateur contrôle si $c_0 = Com(r_1, r_2, G(t_0, r_1, r_2) - e_0 + f_0 + h_0)$, $c_1 = Com(r_1, t_0, e_0, f_0)$, $c_3 = Com(r_2, t_0, e_0, h_0)$

Le vérificateur répond 1 s'il obtient des valeurs correctes par la fonction Com, 0 sinon.

Consitance. On vérifie facilement que le vérificateur accepte toujours dans le cas où il interagit avec un fournisseur de preuve honnête. Le schéma est donc parfaitement consistant.

Black box zero knowledge On construit un simulateur $\mathcal{S}$ qui utilise comme oracle un vérificateur malhonnête $\mathcal{CV}$, prend en entrée F et v, et produit des valeurs simulés avec probabilité $3/4$ de la façon suivante. Le simulateur choisit au hasard $\mathcal{Q}^* \in_R \{0,1,2,3\}$ $s', r'_0, r'_1, t'_0 \in_R \mathbb{F}_q^n$ et $e'_0, f'_0, h'_0 \in_R \mathbb{F}_q^m$. En choisissant $\mathcal{Q}^*$, le simulateur prédit que le vérificateur malhonnête $\mathcal{CV}$ ne choisira pas cette valeur $\mathcal{Q}^*$. Ensuite il calcule

$$r'_2 = s' - (r'_0 + r'_1),\ t'_1 \leftarrow r'_0 - t'_0,\ e'_1 \leftarrow F(r'_0) - e'_0$$

Ensuite, il pose :

1. Si $\mathcal{Q}^* = 0$, $f'_1 = v - F(s') + F(r'_0 + r'_1) - f'_0$, sinon $f'_1 = F(r'_0 + r'_1) - f'_0$.
2. Si $\mathcal{Q}^* = 1$, $h'_1 = v - F(s') + F(r'_0 + r'_2) - f'_0$, sinon $h'_1 = F(r'_0 + r'_2) - h'_0$.
3. Si $\mathcal{Q}^* = 3$, $c'_0 = Com(r'_1, r'_2, v - G(t'_1, r'_1, r'_2) - f'_1 - h'_1 + e'_1 - F(r'_1 + r'_2) + F(r'_1) + F(r'_2))$, sinon $c'_0 = Com(r'_1, r'_2, G(t'_0, r'_1, r'_2) - f'_0 - h_0 + e'_0)$

Il calcule également :

$$c'_1 = Com(r'_1, t'_0, e'_0, f'_0),\ \ c'_2 = Com(r'_1, t'_1, e'_1, f'_1)$$
$$c'_3 = Com(r'_2, t'_0, e'_0, h'_0),\ \ c'_4 = Com(r'_2, t'_1, e'_1, h'_1)$$

et envoie $(c'_0, c'_1, c'_2, c'_3, c'_4)$ à $\mathcal{CV}$.
Quand il reçoit une demande $\mathcal{Q}$ de la part de $\mathcal{CV}$ le simulateur émet $\perp$ si $\mathcal{Q} = \mathcal{Q}^*$ et s'arrête. Si $\mathcal{S}$ n'émet pas $\perp$, il produit les valeurs suivantes :

— Si $\mathcal{Q} = 0$, il produit $((c'_0, c'_1, c'_2, c'_3, c'_4), 0, (r'_0, r'_1, t'_1, e'_1, f'_1))$
— Si $\mathcal{Q} = 1$, il produit $((c'_0, c'_1, c'_2, c'_3, c'_4), 1, (r'_0, r'_2, t'_1, e'_1, h'_1))$
— Si $\mathcal{Q} = 2$, il produit $((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (r'_1, r'_2, t'_1, e'_1, f'_1, h'_1))$
— Si $\mathcal{Q} = 3$, il produit $((c'_0, c'_1, c'_2, c'_3, c'_4), 3, (r'_1, r'_2, t'_0, e'_0, f'_0, h'_0))$

On peut vérifier que lorsque $\mathcal{S}$ n'émet pas $\perp$, les valeurs sont acceptées. Considérons par exemple, le cas pour lequel on a $\mathcal{Q}^* = 0$ et $\mathcal{Q} = 2$. La sortie produite est $((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (r'_1, r'_2, t'_1, e'_1, f'_1, h'_1))$. Donc on a des valeurs correctes pour c'_2 et c'_4. Ensuite, le calcul de c'_0 est donné par : $c'_0 = Com(r'_1, r'_2, v - G(t'_1, r'_1, r'_2) + e'_1 - f'_1 - h'_1 - F(r'_1 + r'_2) + F(r'_1) + f(r'_2))$. Dans ce cas, $f'_1 = v - F(s') + F(r'_0 + r'_1) - f'_0$. Donc on obtient $c'_0 = Com(r'_1, r'_2, G(t'_0, r'_1, r'_2) - f'_0 - h_0 + e'_0)$ et la valeur est acceptée. Les autres cas sont vérifiés de la même manière.

Montrons que la distribution des sorties simulées est statistiquement proche de celle des sorties réelles. On note $\langle P(s), \mathcal{CV}\rangle(F, v)$, les valeurs produites par un fournisseur de preuve honnête P et un vérificateur malhonnête $\mathcal{CV}$ avec les données initiales (F, v, s). Les valeurs de sortie du simulateur $\mathcal{S}$ sont notées $\langle \mathcal{S}, \mathcal{CV}\rangle(F, v)$. Nous analysons maintenant la distribution do sortie.
Considérons d'abord le cas où $\mathcal{Q} = 0$. Alors

$$\langle P(s), \mathcal{CV}\rangle(F, v) = ((c_0, c_1, c_2, c_3, c_4), 0, (r_0, r_1, t_1, e_1, f_1))$$

$$\langle \mathcal{S}, \mathcal{CV}\rangle(F, v) = ((c'_0, c'_1, c'_2, c'_3, c'_4), 0, (r'_0, r'_1, t'_1, e'_1, f'_1))$$

Supposons que l'on ait $(r'_0, r'_1, t'_0, e'_0, f'_0) = (r_0, r_1, t_0, e_0, f_0)$. Alors on obtient $t'_1 = t_1$, $e'_1 = e_1$, $f'_1 = f_1$ et $c'_1 = c_1$, $c'_2 = c_2$ dans les cas $\mathcal{Q}^* = 1, 2, 3$.
On considère ensuite le cas où $\mathcal{Q} = 1$. Alors

$$\langle P(s), \mathcal{CV}\rangle(F, v) = ((c_0, c_1, c_2, c_3, c_4), 1, (r_0, r_2, t_1, e_1, h_1))$$

$$\langle \mathcal{S}, \mathcal{CV}\rangle(F, v) = ((c'_0, c'_1, c'_1, c'_3, c'_4), 1, (r'_0, r'_2, t'_1, e'_1, h'_1))$$

Ce cas est semblable au précédent. On obtient $t'_1 = t_1$, $e'_1 = e_1$, $h'_1 = h_1$ et $c'_3 = c_3$, $c'_4 = c_4$ dans les cas $\mathcal{Q}^* = 0, 2, 3$.
Le troisième cas est donné par $\mathcal{Q} = 2$. Alors

$$\langle P(s), \mathcal{CV}\rangle(F, v) = ((c_0, c_1, c_2, c_3, c_4), 2, (r_1, r_2, t_1, e_1, f_1, h_1))$$

$$\langle \mathcal{S}, \mathcal{CV}\rangle(F, v) = ((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (r'_1, r'_2, t'_1, e'_1, f'_1, h'_1))$$

Quand $\mathcal{Q}^* = 0$, on suppose que $(r'_0, r'_1, t'_0, e'_0, f'_0, h'_0) = (r_0 + s' - s, r_1, t_0 + s' - s, e_0 - F(r_0) + F(r_0 + s' - s), f_0 - F(r_0 + r_1) + v - F(s') + F(r_0 + r_1 + s' - s), h_0 - F(r_0 + r_2) + F(r_0 + r_2 + s' - s))$. Quand $\mathcal{Q}^* = 1$, on suppose que $(r'_0, r'_1, t'_0, e'_0, f'_0, h'_0) = (r_0 + s' - s, r_1, t_0 + s' - s, e_0 - F(r_0) + F(r_0 + s' - s), f_0 - F(r_0 + r_1) + F(r_0 + r_1 + s' - s), h_0 - F(r_0 + r_2) + v - F(s') + F(r_0 + r_2 + s' - s))$. Quand $\mathcal{Q}^* = 1$, on suppose que $(r'_0, r'_1, t'_0, e'_0, f'_0, h'_0) = (r_0 + s' - s, r_1, t_0 + s' - s, e_0 - F(r_0) + F(r_0 + s' - s), f_0 - F(r_0 + r_1) + F(r_0 + r_1 + s' - s), h_0 - F(r_0 + r_2) + F(r_0 + r_2 + s' - s))$. On peut vérifier que dans tous ces cas, on obtient : $r'_2 = r_2$, $t_1 = t'_1$, $e'_1 = e_1$, $f'_1 = f_1$, $h'_1 = h_1$ et alors $c'_0 = c_0$, $c'_2 = c_2$, $c'_4 = c_4$.
Le dernier cas est donné par $\mathcal{Q} = 2$. Alors

$$\langle P(s), \mathcal{CV}\rangle(F, v) = ((c_0, c_1, c_2, c_3, c_4), 3, (r_1, r_2, t_0, e_0, f_0, h_3))$$

$$\langle \mathcal{S}, \mathcal{CV}\rangle(F, v) = ((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (r'_1, r'_2, t'_0, e'_0, f'_0, h'_0))$$

Supposons que $(r'_0, r'_1, t'_0, e'_0, f'_0, h'_0) = (r_0 s' - s, r_1, t_0, e_0, f_0, h_0)$, nous avons alors $r'_2 = r_2$, $c'_0 = c_0$, $c'_1 = c_1$ et $c'_3 = c_3$. Puisque la mise en gage est statistiquement dissimulatrice, quand $\mathcal{S}$ n'émet pas $\perp$, la distribution de sortie de $\mathcal{S}$ est statistiquement proche de celle d'une vraie sortie.

Solidité. Supposons qu'il existe un faux fournisseur de preuve C qui peut répondre à toutes les questions. Alors soit C calcule une collision pour la fonction Com, soit C va produire une solution pour (F, v). Soient $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_0, Rsp_0)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_1, Rsp_1)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_2, Rsp_2)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_3, Rsp_3)$, 4 sorties telles que $\mathcal{Q}_i = i$ et toutes les réponses sont acceptées. Supposons que les réponses soient analysées de la façon suivante : $Rsp_0 = (r_0^{(0)}, r_1^{(0)}, t_1^{(0)}, e_1^{(0)}, f_1^{(0)})$, $Rsp_1 = (r_0^{(1)}, r_2^{(1)}, t_1^{(1)}, e_1^{(1)}, h_1^{(1)})$, $Rsp_2 = (r_1^{(2)}, r_2^{(2)}, t_1^{(2)}, e_1^{(2)}, f_1^{(2)}, h_1^{(2)})$, $Rsp_3 = (r_1^{(3)}, r_2^{(3)}, t_0, e_0, f_0, h_0)$.
ON obtient :

$$\begin{aligned} c_0 &= Com(\ r_1^{(2)},\ r_2^{(2)},\ v - G(t_1^{(2)}, r_1^{(2)}, r_2^{(2)}) - f_1^{(2)} - h_1^{(2)} + e_1^{(2)} \\ &\qquad -F(r_1^{(2)} + r_2^{(0)}) + F(r_1^{(2)}) + F(r_2^{(2)}) \ ) \\ &= Com(\ r_1^{(3)},\ r_2^{(3)},\ G(t_0, r_1^{(3)}, r_2^{(3)}) + f_0 + h_0 - e_0 \ ) \qquad (1) \end{aligned}$$

$$\begin{aligned} c_1 &= Com(\ r_1^{(0)},\ r_0^{(0)} - t_1^{(0)},\ F(r_0^{(0)}) - e_1^{(0)},\ F(r_0^{(0)} + r_1^{(0)}) - f_1^{(0)} \ ) \\ &= Com(\ r_1^{(3)},\ t_0,\ e_0,\ f_0 \ ) \qquad (2) \end{aligned}$$

$$c_2 = Com(r_1^{(0)}, t_1^{(0)}, e_1^{(0)}, f_1^{(0)}) = Com(r_1^{(2)}, t_1^{(2)}, e_1^{(2)}, f_1^{(2)}) \qquad (3)$$

$$\begin{aligned} c_3 &= Com(\ r_2^{(1)},\ r_0^{(1)} - t_1^{(1)},\ F(r_0^{(1)}) - e_1^{(1)},\ F(r_0^{(1)} + r_2^{(1)}) - h_1^{(1)} \ ) \\ &= Com(\ r_2^{(3)},\ t_0,\ e_0,\ h_0 \ ) \qquad (4) \end{aligned}$$

$$c_4 = Com(r_2^{(1)}, t_1^{(1)}, e_1^{(1)}, h_1^{(1)}) = Com(r_2^{(2)}, t_1^{(2)}, e_1^{(2)}, h_1^{(2)}) \qquad (5)$$

Si deux arguments distincts de la fonction Com satisfont Com l'une des ces équations, alors nous avons une collision pour la fonction Com. Sinon, ces égalités permettent d'obtenir :

$$h_1^{(1)} \overset{(5)}{=} h_1^{(2)}, r_1^{(0)} \overset{(2)}{=} r_1^{(3)} \overset{(1)}{=} r_1^{(2)}, t_1^{(0)} \overset{(3)}{=} t_1^{(1)} \overset{(5)}{=} t_1^{(2)}, r_0^{(0)} \overset{(3,4)}{=} r_0^{(1)}$$

$$\text{and} \quad r_2^{(2)} \overset{(1)}{=} r_2^{(3)} \overset{(4)}{=} r_2^{(1)}, e_1^{(2)} \overset{(3)}{=} e_1^{(0)} \overset{(2,4)}{=} e_1^{(1)}, f_1^{(0)} \overset{(3)}{=} f_1^{(2)}$$

Donc on peut supprimer les indices supérieurs et (1) donne :

$$v = G(t_1 + t_0, r_1, r_2) + f_1 + h_1 - e_1 + F(r_1 + r_2) - F(r_1) - F(r_2) + f_0 + h_0 - e_0$$

Ensuite d'après (2) et (4) on ae $r_0 = t_0 + t_1$, $F(r_0) = e_0 + e_1$, $F(r_0 + r_1) = f_0 + f_1$ et $F(r_0 + r_2) = h_0 + h_1$. Si on remplace ces valeurs dans l'égalité précédente, on obtient : $v = G(r_0, r_1, r_2) + F(r_0 + r_1) + F(r_0 + r_2) + F(r_1 + r_2) - F(r_0) - F(r_1) - F(r_2) = F(r_0 + r_1 + r_2)$ On a donc trouvé une solution $r_0 + r_1 + r_2$ pour v.
Un fournisseur de preuve malhonnête pourra répondre à 3 questions sur 4. Is sera donc rejeté avec une probabilité égale à 3/4.

Calculs exécutés durant le protocole Nous donnons le nombre maximum de calculs exécutés soit par le fournisseur de preuve, soit par le vérificateur dans le cas du corps fini $\mathbb{F}_2$. Nous donnons le nombre de calculs pour F et pour G. De plus, F est calculée au plus 3 fois et G une fois. Nous comptons uniquement les multiplications. Dans $\mathbb{F}_2$, on a : $x_i^3 = x_i^2 = x_i$. On peut écrire :

$$f_\ell(x) = \sum_{i=1}^{n} x_i \big[\gamma_i^\ell + \gamma_{ii}^\ell + \gamma_{iii}^\ell + \sum_{j=i+1}^{n} x_j \big(\gamma_{ij}^\ell + \gamma_{ijj}^\ell + \sum_{k=j+1}^{n} \gamma_{ijk}^\ell x_k\big)\big]$$

Soit M le nombre de calculs nécessaires pour calculer F. En utilisant l'expression précédente pour F, on obtient $M \simeq \frac{n^3}{6}m$. La table 7.1 donne les caractéristiques du schéma $ZK(3)$ et les valeurs que l'on obtient quand on choisit $n = 84$, $m = 80$, pour avoir 80 bits de sécurité (cf. [3]). De plus, sinon veut que la probabilité d'imitation soit inférieur à 2^{-30}, nous devons exécuter au moins 73 tours. R représente le nombre de tours et C le nombre maximum de calculs exécutés soit pas le fournisseur de preuve, soit par le vérificateur. Les valeurs sont données dans la table 7.1.

TABLE 7.1 – Le schéma $ZK(3)$

	Formules	Sécurité en 2^{80}
Clé publique (bit)	m	80
Clé privée (bit)	n	84
M	$\frac{n^3}{6} \times m$	7902720
Communication (bit)	$(4 \times 160 + 2 + 3n + 2m) \times R$ ou $(3 \times 160 + 2 + 3n + 3m) \times R$	76942 71102
Nombre de multiplications	C=9MR	2^{33}

7.2 Le schéma $\widetilde{ZK}(3)$

Le schéma suivant est également une généralisation du schéma proposé dans [56]. Au lieu d'adapter le schéma décrit dans [56] au cas cubique, on transforme le système d'équations cubiques en un système quadratique et on utilise directement le schéma donné dans [56]. Pour ce nouveau schéma zero-knowledge, que nous noterons $\widetilde{ZK}(3)$, le nombre de calculs exécutés soit par le fournisseur de preuve, soit par le vérificateur est inférieur à celui obtenu pour le schéma $ZK(3)$, mais le nombre de bits de communication est plus important.
Nous décrivons la transformation d'un système d'équations cubiques en un système d'équations quadratiques. Nous introduisons de nouvelles variables. On a : $F(x) = (f_1(x), f_2(x), \ldots, f_m(x))$ avec $\forall \ell,\ 1 \leq \ell \leq m$,

$$f_\ell(x) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma_{ijk}^\ell x_i x_j x_k + \sum_{1 \leq i \leq j \leq n} \gamma_{ij}^\ell x_i x_j + \sum_{1 \leq i \leq n} \gamma_i^\ell x_i$$

et $x = (x_1, \ldots, x_n)$. Les nouvelles variables sont définies par $\forall i, j,\ 1 \leq i \leq j \leq n,\ X_{ij} = x_i x_j$. Le nombre de nouvelles variables est $\frac{n(n-1)}{2}$, si $q = 2$, et $\frac{n(n+1)}{2}$, si $q \neq 2$. Dans le nouveau système, on a :

$$\tilde{F}(\tilde{x}) = (\tilde{f}_1, \ldots, \tilde{f}_m, (\tilde{f}_{ij})_{1 \leq i \leq j \leq n})$$

où $\tilde{x} = (x_1, \ldots, x_n, (X_{ij})_{1 \leq i \leq j \leq n})$. Si $1 \leq \ell \leq m$, on a

$$f_\ell(\tilde{x}) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma_{ijk}^\ell X_{ij} x_k + \sum_{1 \leq i \leq j \leq n} \gamma_{ij}^\ell X_{ij} + \sum_{1 \leq i \leq n} \gamma_i^\ell x_i$$

et si $1 \leq i \leq j \leq n$, alors $f_{ij}(\tilde{x}) = X_{ij} - x_i x_j$. On note $\tilde{n}$ le nombre de total de variables dans le nouveau système. On a $\tilde{n} \simeq n + \frac{n^2}{2}$. Le nombre total

d'équations est noté $\tilde{m}$. On a alors $\tilde{m} = m + \frac{n^2}{2}$. Comme précédemment , M représente le nombre de multiplications nécessaires pour calculer F. De même $\tilde{M}$ représente le nombre de multiplications nécessaires pour calculer $\tilde{F}$. On prend $q = 2$. Alors $\tilde{M} = M + \frac{n(n-1)}{2}$. Donc $\tilde{M} \simeq M \simeq \frac{n^3}{6}$. $\tilde{C}$ représente le nombre maximum de multiplications exécutées soit par le fournisseur de preuve, soit par le vérificateur. Le nombre de tours nécessaire est noté $\tilde{R}$. D'après [56], on prend $\tilde{R} = 52$. La table 7.2 donne les caractéristiques du schéma et les valeurs pour $n = 84$ et $m = 80$.

TABLE 7.2 – Le schéma $\widetilde{ZK}(3)$

	Formules	Sécurité en 2^{80}
Clé publique (bit)	$\tilde{m}$	3483
Clé privée (bit)	n	84
M	$\frac{n^3}{6} \times m$	7902720
Communication (bit)	$(2 \times 160 + 2 + 2\tilde{n} + \tilde{m}) \times \tilde{R}$	560508
Nombre de multiplications	$\tilde{C} = 3\tilde{M}\tilde{R}$	2^{31}

Quatrième partie

Conclusion

Chapitre 8

Contributions essentielles et perspectives

8.1 Contributions essentielles

Je vais ici plus utiliser la première personne pour détailler mon travail personnel.

1. Candidat pour un nouveau standard de fonction de hachage : CRUNCH. Lorsque nous avons décidé de nous lancer dans l'aventure d'un nouveau standard de fonction de hachage, il ne nous restait plus que quelques mois pour tout finaliser. J'ai alors en grande partie coordonné les différentes personnes travaillant sur ce projet, sur la programmation, sur la présentation et les études de sécurité. J'ai finalisé le rapport pour l'envoyer au NIST. Puis, lors de la première conférence SHA-3, je suis allé présenter seul le projet à l'Université Catholique de Leuvens. CRUNCH a été critiqué pour son mode d'enchaînement, j'ai alors présenté lors de cette conférence un patch pour contrer les attaques connues sur ce système, en utilisant un double-pipe. On peut bien sûr utiliser le même principe que CRUNCH. Deux chercheurs russes, Sergey Panasenko et Sergey Smagin, ont repris ce principe en utilisant d'autres fonctions de chiffrement pour faire de la cryptographie embarquée (projet KATAN).

   Pour la programmation elle même, j'ai été à l'origine du choix des fonctions aléatoires pour réduire la place en mémoire. Certains nombres aléatoires sont utilisés plusieurs fois mais à différents endroits. La probabilité pour qu'un même nombre soit utilisé lors d'un appel à un des deux schémas de Feistel est tout de même très faible. De plus, les tables de nombres aléatoires pour les deux schémas de Feistel sont complètement séparées.
2. Théorie du miroir. Vérification des conjectures jusqu'à un certain ordre et énoncé de la plupart des conjectures.
3. Zero knowledge dans les groupes symétriques et application au Rubik's cube.
4. Automatisation de la recherche d'attaques génériques dans les schémas de Feistel expansifs grâce à laquelle une nouvelle attaque plus efficace a été trouvée.

8.2 Perspectives

1. Automatisations du calcul de variances dans des schémas de Feistel. Une des applications serait de trouver un schéma de Feistel optimal par rapport à ce type d'attaque générique.
2. Vers une résolution des équations de Brent.
3. Théorie du miroir.

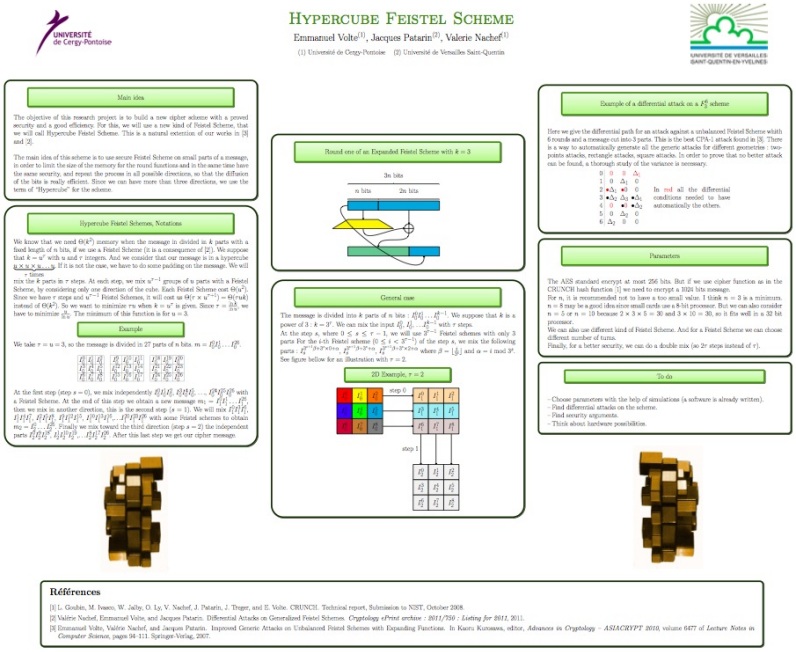
Cinquième partie

Annexes

Annexe A

Schémas de Feistel hypercubique

FIGURE A.1 – Poster présenté à Latincrypt 2012



A.1 Introduction

Les schémas de Feistel Hypercubiques ont fait l'objet d'une présentation sous forme de poster à Latincrypt 2012, voir figure A.1. Nous allons voir que ces schémas reprennent des études de cryptanalyse sur les schémas de Feistel et nous sommes presque en mesure d'étudier la complexité des attaques génériques sur ces schémas.

Suite à l'étude de différents types de schémas de Feistel, nous nous sommes rendu compte que certains schémas résistaient mieux que d'autres aux attaques génériques, pour une efficacité comparable. Nous avons donc cherché à écrire un nouveau type de schéma Feistel dont la diffusion serait exceptionnelle, et qui utiliserait une quantité plus raisonnable de mémoire. Si on prend l'exemple du schéma de Feistel utilisé pour CRUNCH, le message est divisé en 128 parties de 8 bits. Une modification du dernier bit sera donc prise en compte qu'au bout du 128^{e} tour. Au contraire, les schémas de Feistel étudiés dans le premier chapitre, comme celui de type 2 (voir figure 1.4) et celui de type 3 (voir figure 1.5) ont une diffusion plus rapide puisque le changement d'un bit et pris en compte au bout d'un ou deux tour, cependant il ne sera vraiment diffusé à l'ensemble du message qu'au bout d'un nombre de tours beaucoup plus important.

Afin d'illustrer géométriquement ce nouveau schéma, on peut prendre l'exemple d'un cube divisé en u^3 parties de n bits. On va tour à tour utiliser des schémas de Feistel expansifs dans chacune des directions du cube. On voit bien ainsi l'avantage en terme de diffusion, en revanche il va falloir aller chercher des bouts de mémoires un peu partout et cela risque d'être moins efficace de façon logicielle. Rien n'empêche d'imaginer une fabrication hardware

qui accélèrerait le processus.

Pour aller plus loin, on peut imaginer faire la même chose avec une dimension quelconque, en généralisant ce que l'on a vu sur le cube. C'est ce que nous allons décrire dans la section suivante. Nous proposerons ensuite un schéma de chiffrement particulier qui a d'ores et déjà été programmé en C, et qui chiffre des mots de 162 bits. En utilisant le même principe que CRUNCH, c'est à dire le XOR de deux permutations, on peut ainsi se servir de cette fonction pour faire du hachage avec une sécurité supérieure à 2^{80}.

A.2 Hypercube Feistel Schemes, Notations

D'après l'étude faire au premier chapitre, nous avons constaté que l'on avait besoin de $\Theta(k^2)$ place en mémoire lorsque le message était divisé en k parties de n bits, pour un schéma de Feistel (c'est une conséquence de [36]). On suppose maintenant que $k = u^\tau$ avec u et τ entiers, τ étant en quelque sorte la dimension du schéma. On considère donc que notre message est un hypercube $\underbrace{u \times u \times u \dots u}_{\tau \text{ fois}}$. Si ce n'est pas le cas, on complète le message pour arriver à cette taille particulière. On va ensuite mélanger les k parties en τ étapes. À chaque étape, on mélange $u^{\tau-1}$ groupes de u parties avec un schéma de Feistel en considérant seulement une des directions de l'hypercube. Chaque schéma de Feistel coûte en mémoire $\Theta(u^2)$. Comme il y a τ étapes et $u^{\tau-1}$ schémas de Feistel, cela nous coûtera au total $\Theta(\tau \times u^{\tau+1}) = \Theta(\tau u k)$ au lieu $\Theta(k^2)$. Nous voulons donc minimiser τu lorsque $k = u^\tau$ est donné. Comme $\tau = \frac{\ln k}{\ln u}$, cela revient à minimiser $\frac{u}{\ln u}$. Le minimum de cette fonction est obtenu pour $u = 3$.

Exemple : On prend $\tau = u = 3$, donc le message est divisé en 27 parties de n bits. $m = I_0^0 I_0^1 \dots I_0^{26}$.

I_0^0	I_0^1	I_0^2	I_0^9	I_0^{10}	I_0^{11}	I_0^{18}	I_0^{19}	I_0^{20}
I_0^3	I_0^4	I_0^5	I_0^{12}	I_0^{13}	I_0^{14}	I_0^{21}	I_0^{22}	I_0^{23}
I_0^6	I_0^7	I_0^8	I_0^{15}	I_0^{16}	I_0^{17}	I_0^{24}	I_0^{25}	I_0^{26}

À la première étape (étape $s = 0$), on mélange indépendamment $I_0^0 I_0^1 I_0^2$, $I_0^3 I_0^4 I_0^5$, …, $I_0^{24} I_0^{25} I_0^{26}$ avec un schéma de Feistel. À la fin de cette étape, on obtient un nouveau message $m_1 = I_1^0 I_1^1 \dots I_1^{26}$, puis on mélange dans une autre direction, c'est la deuxième étape ($s = 1$). On va mélanger $I_1^0 I_1^3 I_1^6$, $I_1^1 I_1^4 I_1^7$, $I_1^2 I_1^5 I_1^8$, $I_1^9 I_1^{12} I_1^{15}$, $I_1^{10} I_1^{13} I_1^{16}$,… $I_1^{20} I_1^{23} I_1^{26}$ avec un nouveau schéma de Feistel pour obtenir $m_2 = I_2^0 \dots I_2^{26}$. Finalement, on mélange suivant la troisième direction (étape $s = 2$) les parties $I_2^0 I_2^9 I_2^{18}$, $I_2^1 I_2^{10} I_2^{19}$,… $I_2^8 I_2^{17} I_2^{26}$. Après cette dernière étape, la diffusion est totale. C'est à dire qu'un seul bit modifié aura complètement modifié les autres bits. On refait alors un nouveau le même mélange en 3 étapes pour obtenir notre message chiffré.

Cas général : Le message est divisé en k parties de n bits : $I_0^0 I_0^1 \dots I_0^{k-1}$. On suppose que k est une puissance de 3 : $k = 3^\tau$. On peut mélanger l'entrée I_0^0, I_0^1, … I_0^{k-1} en τ étapes.

À l'étape s, avec $0 \leq s \leq \tau - 1$, nous allons utiliser $3^{\tau-1}$ schémas de Feistel avec seulement 3 parties (voir figure A.2).

Pour le i^{e} schéma de Feistel ($0 \leq i < 3^{\tau-1}$) de l'étape s, nous mélangeons les parties suivantes : $I_s^{3^{s+1}\beta+3^s\times 0+\alpha}$, $I_s^{3^{s+1}\beta+3^s+\alpha}$, $I_s^{3^{s+1}\beta+3^s\times 2+\alpha}$ où $\beta = \lfloor \frac{i}{3^s} \rfloor$

FIGURE A.2 – Premier tour d'un schéma de Feistel expansif avec $k = 3$

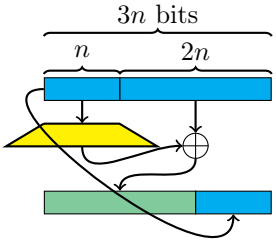
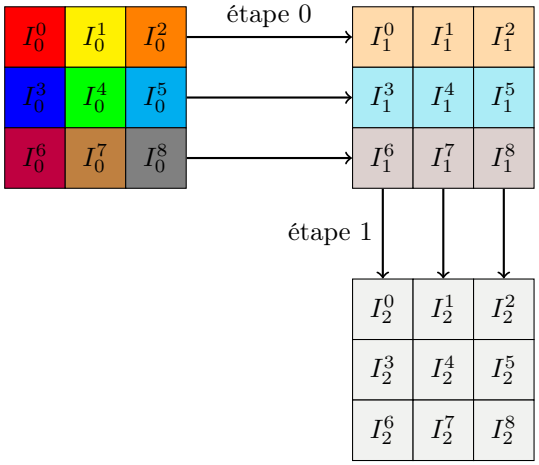


FIGURE A.3 – Exemple en dimension 2 ($\tau = 2$)



et $\alpha = i \bmod 3^s$. Voir figure A.3 pour une illustration avec $\tau = 2$.

A.3 Paramètres

Le standard de chiffrement AES chiffre au plus 256 bits. Mais si on veut utiliser une fonction de chiffrement comme dans CRUNCH, on a besoin de chiffrer un message de 1024 bits.

Pour n, nous recommandons de ne pas avoir une valeur trop petite. $n = 3$ est sans doute un minimum. $n = 8$ est peut être une bonne idée puisque les cartes à puces utilisent des micro-processeurs fonctionnant en 8 bits. Pour des raisons d'efficacité on peut aussi considérer $n = 5$ ou $n = 10$ parce que $2 \times 3 \times 5 = 30$ et $3 \times 10 = 30$, donc cela peut se mettre plus facilement dans un mot de 32 bits. On verra pourtant que pour la fonction choisie, nous avons pris $n = 6$, en trichant un peu sur les fonctions aléatoires, en effet on a fait tenir 3 fois 12 bits sur un mot de 32 bits !

On peut aussi envisager d'autre types de schémas de Feistel, et on peut moduler le nombre de tours utilisés pour ces schémas. Par exemple, pour un schéma expansif, il faut au minimum autant de tour qu'il y a de parties (à condition cependant d'un double mélange) et au plus 3 fois le nombre de parties

TABLE A.1 – Exemples de paramètres possibles ($u = 3$)

Longueur du message	Occupation mémoire (ko)	n	τ
72	55	8	2
81	3	3	3
108	8	4	3
135	19	5	3
162	47	6	3
189	109	7	3
216	249	8	3
243	12	3	4
324	31	4	4
405	78	5	4
486	187	6	4
567	435	7	4
648	995	8	4
729	44	3	5
972	117	4	5
1215	292	5	5
1458	700	6	5

d'après les études faites sur les attaques génériques [66].

Finalement, nous recommandons fortement l'utilisation d'un double mélange, donc 2τ étapes plutôt que τ). Dans le cas de schémas de Feistel expansifs, on peut donner la formule suivante pour la place en mémoire occupée par les fonctions aléatoires, dans le cas où on chiffre des messages de longueur nu^τ :

$$\text{Occupation mémoire} = 2\tau u^{\tau+1}(u-1)n2^n$$

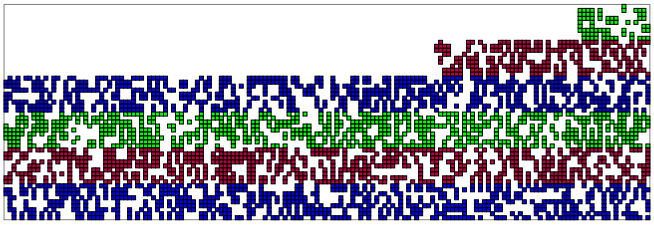
A.4 Application : la fonction de chiffrement HFS-162

Nous avons choisi pour cette fonction de chiffrement $n = 6$, $k = \tau = 3$, de sorte que notre message est divisé en $3^3 = 27$ morceaux de 6 bits. Ainsi on chiffre des messages de 162 bits, ce qui est une valeur inhabituelle mais assez pratique pour la plupart des utilisations. Par exemple, pour le hachage, si on utilise la même méthode que pour CRUNCH, on dispose de 160 bits pour la sécurité, et les deux bits qui restent nous permettent de n'utiliser qu'une seule permutation au lieu de deux, ce qui est un gain de place considérable. En effet, on peut considérer la fonction f définie sur des mots de 160 bits par

$$f(x) = \text{HFS-162}(x||00) \oplus \text{HFS-162}(x||01).$$

Pour la mémoire, on utilise une table d'environ $45ko$, ce qui nous donne un programme de $60ko$, beaucoup plus léger que CRUNCH, cependant cette dernière fonction utilisait des chiffrements sur 1024 bits.

FIGURE A.4 – Analyse différentielle de la fonction HFS-162 avec un seul bit changé.



En vert l'étape 0, en rouge l'étape 1 et en bleu l'étape 2.

Annexe B

Recherche d'une solution des équations de Brent.

B.1 Introduction

Soient K un anneau commutatif et n un entier naturel non nul.

La multiplication naïve de deux matrices $n \times n$ à coefficients dans K nécessite n^3 multiplications et $(n-1)n^2$ additions dans le corps. L'opération la plus coûteuse d'un point de vue calculatoire est la multiplication, ce qui donne une complexité en $O(n^3)$ opérations pour la multiplication de deux matrices. D. Coppersmith et S. Winograd [8] ont montré l'existence d'un algorithme qui ne nécessite que $O(n^{2.376})$ opérations arithmétiques ; toutefois, celui-ci n'est pas utilisable en pratique à cause de constantes trop élevées qui sont cachées dans la notation O. Une autre méthode pour diminuer la complexité de cette opération est de trouver un algorithme pour effectuer la multiplication de deux matrices de petite taille avec un nombre plus faible de produits que dans la méthode naïve, puis de l'appliquer récursivement. V. Strassen [62] a découvert un algorithme qui permet de multiplier des matrices 2×2 en 7 multiplications. Il a ensuite été prouvé que ce résultat était optimal pour $n = 2$ [68]. L'algorithme récursif obtenu effectue $O(n^{2.808})$ opérations arithmétiques. Pour $n = 3$, le problème de trouver le nombre optimal de multiplications à effectuer est toujours ouvert. Il a été montré que toute solution à ce problème devait effectuer au moins 21 multiplications. D'autre part, J. Ladermann [29], puis R.W. Johnson et A.M. McLoughlin [55] et N. Courtois [39] ont exhibé deux solutions, non équivalentes, qui utilisent 23 multiplications.

Le sujet de cette section est la conception d'un programme dont le but est de rechercher une solution à ce problème qui n'utiliserait que 21 ou 22 multiplications. La recherche exhaustive est cependant impossible à réaliser. Il est nécessaire de faire des conjectures sur la forme de la solution afin de la rendre possible. Nous allons détailler l'algorithme utilisé ainsi que différentes conjectures autour de ce problème.

B.2 Description de l'algorithme

B.2.1 Notations

Soit $s \in \mathbb{N}^*$ et $E = (\mathbb{Z}/2\mathbb{Z})^s$. $(E, +, \times)$ est une algèbre sur $\mathbb{Z}/2\mathbb{Z}$. Un élément $x \in E$ sera parfois noté $x = x_1 \dots x_s$. Nous définissons une forme bilinéaire sur E comme suit :

$$\forall x = x_1 \dots x_s \in E, \qquad \forall y = y_1, \dots, y_s, \qquad \langle x, y \rangle = \sum_{i=1}^{s} x_i y_i.$$

Si F est un sous-espace vectoriel de E, $F^\perp = \{x \in E \,|\, \forall y \in F, \langle x, y \rangle = 0\}$ est un sous-espace vectoriel de E et $\dim F^\perp = s - \dim F$. En utilisant les mêmes notations, nous pouvons définir une forme trilinéaire :

$$\forall (x, y, z) \in E^3, \qquad \langle x, y, z \rangle = \sum_{i=1}^{s} x_i y_i z_i.$$

Nous avons :

$$\forall (x, y, z) \in E^3, \qquad \langle x, y, z \rangle = \langle x \times y, z \rangle = \langle x \times y \times z, 111 \dots 111_s \rangle.$$

Quand le contexte est clair, pour $x \in E$, notons $\lceil x$ le vecteur $x_1 \dots x_u 0 \dots 0$ et $x\rceil = 0 \dots 0 x_{u+1} \dots x_s$. Ainsi $x = \lceil x + x \rceil$.

Pour $x \in E$, $|x|$ est le poids de Hamming de x, c'est-à-dire le nombre de bits de x égaux à 1.

Nous désignerons par $>$ l'ordre naturel sur E (en voyant ses éléments comme des entiers) qui sera étendu aux tuples d'éléments de E de la manière suivante :

$$(a^1, \dots, a^n) > (b^1, \dots, b^n) \iff \exists i \in \{0, \dots, n-1\}, \begin{cases} a^1 = b^1, \\ \vdots \\ a^i = b^i, \\ a^{i+1} > b^{i+1}. \end{cases}$$

Proposition 18. *Soit*

$$\begin{aligned} \xi : \{1, \dots, n^2\} &\to \{1, \dots, n\} \times \{1, \dots, n\} \\ u &\longmapsto \left(1 + \left\lfloor \frac{u-1}{n} \right\rfloor, u - n \left\lfloor \frac{u-1}{n} \right\rfloor\right) = (u_1, u_2). \end{aligned}$$

Alors ξ est une bijection.

Définition 23. Pour tout $u \in \{1, \dots, n^2\}$, $\hat{u} = \xi^{-1}(u_2, u_1)$. Cette notation est généralisée pour tout $a \in E^{n^2}$ par $\hat{a} = (a^{\hat{1}}, \dots, a^{\hat{n^2}})$.

Cette opération correspond à la transposition de la matrice

$$\begin{pmatrix} a^1 & \dots & a^n \\ \vdots & & \vdots \\ a^{n^2-n+1} & \dots & a^{n^2} \end{pmatrix}$$

B.2.2 Les équations de Brent

Définition 24. Les équations de Brent sur $\mathbb{Z}/2\mathbb{Z}$ sont définies comme suit :

$$\sum_{t=1}^{s} \gamma_{ijt} \alpha_{abt} \beta_{cdt} = \delta_{bc} \delta_{ia} \delta_{jd}$$

où $a, b, c, d, i, j \in \{1, \dots, n\}$.

Il convient d'expliquer par un exemple le rôle des coefficients α, β, γ dans la multiplication de deux matrices A et B, où $C = AB$. Prenons $t = 1, n = 2$. Supposons que :

$$\alpha_1 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \beta_1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \qquad \gamma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Ceci signifie qu'à l'étape 1, on effectue le produit $(a_{1,1} + a_{1,2})(b_{1,1} + b_{2,1})$ et que celui-ci intervient dans les coefficients $c_{1,1}$ et $c_{2,2}$.

Trouver une solution aux équations de Brent revient à trouver un algorithme qui effectue la multiplication de deux matrices $n \times n$ en s multiplications dans le corps de base. En effet, grâce à la bilinéarité du produit de matrices, il faut et il suffit que l'algorithme multiplie correctement entre eux tous les éléments de la base canonique de l'espace vectoriel des matrices $n \times n$. Cette condition est exprimée les équations de Brent.

Proposition 19. *Trouver une solution des équations de Brent équivaut à trouver* $a = (a^1, \ldots, a^{n^2})$, $b = (b^1, \ldots, b^{n^2})$ *et* $c = (c^1, \ldots, c^{n^2})$ *dans* E^{n^2} *tels que :*

$$\forall (i,j,k) \in \{1,\ldots,n^2\},\ \langle a^i, b^j, c^k \rangle = \delta_{i_1 j_1}\delta_{j_2 k_2}\delta_{k_1 i_2}. \tag{B.1}$$

Nous disposons ainsi d'un système de n^6 équations et de $3n^2$ inconnues sur E.

Proposition 20. (a,b,c) *est solution de (B.1) si et seulement si un de ces triplets est solution :*

$$(b,a,\hat{c}),\ (\hat{a},c,b),\ (\hat{c},\hat{b},\hat{a}),\ (\hat{b},\hat{c},a),\ (c,\hat{a},\hat{b}).$$

Ainsi, lors de la recherche d'une solution, nous pouvons supposer que

$$(a^1, a^2, a^3) \geq (b^1, b^2, b^3).$$

B.2.3 Quelques espaces vectoriels particuliers

Définition 25. Après avoir choisi $a = (a^1, \ldots, a^{n^2})$ et $b = (b^1, \ldots, b^{n^2})$, nous définissons n^2 sous-espaces vectoriels de E :

$$\forall u, v \in \{1,\ldots,n\},\ F_{uv}(a,b) = Vect(a^i \times b^j)_{i,j \in 1,\ldots,n^2, i_1 = u, j_1 = v}.$$

Lorsque le contexte est clair, $F_{uv}(a,b)$ sera simplement noté F_{uv}. Si $u = v$, on utilisera la notation $D_u := F_{uu}$. De plus, $F_{\neq} := \sum_{i \neq j} F_{ij}$, $F := \sum_{i,j} F_{ij}$.

Ces espaces vectoriels ont des propriétés spéciales qui permettront d'identifier plus rapidement les vecteurs qui ne permettent pas d'obtenir une solution. Ces propriétés sont résumées dans la proposition suivante :

Proposition 21. *Supposons que nous avons trouvé une solution* (a,b,c) *de (B.1), alors nous avons :*

1. *Chaque famille* a, b, c *de vecteurs est linéairement indépendante. En particulier* $\dim(Vect(a^1, \ldots, a^{n^2})) = n^2$.
2. $\forall u \in \{1,\ldots,n\}$, $\dim(D_u) = n^2$.
3. $c^1, \ldots, c^{n^2} \in F_{\neq}^{\perp}$.
4. $\dim\left(F_{\neq}^{\perp}\right) \geq n^2$, *d'où* $\dim(F_{\neq}) \leq s - n^2$.
5. $\forall u \in \{1,\ldots,n\}$, $D_u \cap F_{\neq} = \{0\}$.
6. *Si* s *est le nombre minimal de bits pour l'existence d'une solution, alors* $\dim(F) = s$ *et* $E = F$.
7. $\forall i \in \{1,\ldots,n\}$, $|a^i| \geq n$ *et* $|b^i| \geq n$.

Preuve.

1. Si $x = \sum_{i=1}^{n^2} \lambda_i a^i = 0$, alors pour tout $j,k \in \{1,\ldots,n^2\}$, $\langle x, b^j, c^k \rangle = 0$. Soit $i \in \{1,\ldots,n^2\}$. En prenant $j_1 = i_1$, $k_1 = i_2$ et $j_2 = k_2$, alors on obtient $\lambda_i = 0$.

2. Soit $x = \sum_{i,j\in\{1,\dots,n^2\},i_1=j_1=u} \lambda_{i,j} a^i \times b^j = 0$. Pour tout $i, j \in \{1,\dots,n^2\}$ tels que $i_1 = j_1 = u$, on choisit k tel que $k_1 = i_2$ et $k_2 = j_2$ et on a $\langle x, c^k\rangle = \lambda_{ij} = 0$.
3. ÉL'vident par définition de $F_{\neq}^{\perp}$.
4. Conséquence du point précédent, puisque c est une famille libre de vecteurs.
5. Si $x \in F_{\neq}$, alors pour tout k, $\langle x, c^k\rangle = 0$. Si $x \in D_u$,

$$x = \sum_{i,j\in\{1,\dots,n^2\},i_1=j_1=u} \lambda_{i,j} a^i \times b^j.$$

   En prenant $k_1 = i_2$ et $k_2 = j_2$, on obtient $\lambda_{i,j} = 0$, d'où $x = 0$.
6. Supposons que $\dim(F) < s$. Alors il existe un élément non nul $x \in F^{\perp}$. Soit $\alpha \in \{1,\dots,s\}$ tel que $x_\alpha = 1$. Soit $c' \in E^{n^2}$ tel que : pour tout $i \in \{1,\dots,n^2\}$, $c'^i = c^i + x1|_{c^i_\alpha \neq 0}$. Ainsi $c'^i_\alpha = 0$ pour tout i. Pour tout $i, j, k \in \{1,\dots,n^2\}$, nous avons $\langle a^i \times b^j, c'^k\rangle = \langle a^i \times b^j, c^k\rangle$ (car $x \in F^{\perp}$). Donc (a, b, c') est une solution de B.1.
   Définissons $y = 0\dots010\dots0 \in E$ pour lequel l'unique bit égal à 1 est situé en position α. Nous pouvons changer a et b en a' et b' de la façon suivante : pour tout $i \in \{1,\dots,n^2\}$, le bit en position α est mis à 0. Nous avons ainsi construit une solution (a', b', c') de (B.1) sans utiliser le bit en position α. Ainsi s n'est pas minimal.
7. Le résultat ne sera montré que pour a^1. Les autres cas sont similaires. Supposons que $|a^1| < n$. Alors $\dim(Vect(a^1 \times b^1, \dots, a^1 \times b^n) < n)$ ce qui entraîne que $\dim(D_1) < n^2$.

□

B.2.4 Étude des symétries

Les résultats précédents permettent d'accélérer la recherche de solutions, toutefois celle-ci reste irréaliste. Il faut donc se restreindre en émettant des conjectures sur la forme de la solution. Dans cette section, des conjectures seront exposées.

Définition 26. Soient $a \in E^{n^2}$ et $(\sigma_1, \sigma_2) \in S_n$ où S_n désigne l'ensemble des permutations de $\{1,\dots,n\}$. Alors

$$a^{\sigma_1,\sigma_2} = (a^{\xi^{-1}(\sigma_1(i_1)),\xi^{-1}(\sigma_2(i_2))})_{i\in\{1,\dots,n^2\}}.$$

Proposition 22. *Soient $\sigma_1, \sigma_2, \sigma_3$ 3 permutations de $\{1,\dots,n\}$ et (a, b, c) une solution de (B.1). Alors $(a^{\sigma_1,\sigma_2}, b^{\sigma_1,\sigma_3}, c^{\sigma_2,\sigma_3})$ en est une aussi.*

Lors de la recherche de solutions, nous pouvons ainsi supposer que $a^1 > a^2 > a^3$ et $b^1 > b^2 > b^3$.

Il est aussi possible de faire agir S_s sur les vecteurs de E comme suit :

Définition 27. Soit $\sigma \in S_s$. Pour tout $x = x_1 \dots x_s$, on définit $x_\sigma = x_{\sigma(1),\dots,\sigma(s)}$ et plus généralement, pour tout sous-ensemble $F \subset E$, $F_\sigma = \{x_\sigma \,|\, x \in F\}$.

Proposition 23. *Nous avons :*

$$\forall \sigma \in S_s, \forall x, y \in E,\ \langle x_\sigma, y\rangle = \langle x, y_{\sigma^{-1}}\rangle \text{ et } \langle x, y\rangle = \langle x_\sigma, y_\sigma\rangle.$$

De plus, $\forall \sigma \in S_s$, $\forall x, y \in E$, $(x \times y)_\sigma = x_\sigma \times y_\sigma$.

Pour appuyer les conjectures, il faut commencer par étudier la solution de V. Strassen. Soit σ la permutation de $\{1, \ldots, 7\}$ définie par $\sigma = \tau_{23} \circ \tau_{45} \circ \tau_{67}$ où τ_{ij} échange i et j. Nous avons évidemment que $\sigma^2 = \mathrm{Id}$ et

$$a^1_\sigma = a^4,\ a^2_\sigma = a^3,\ b^1_\sigma = b^4,\ b^2_\sigma = b^3.$$

Pour le cas $n = 3$ et $s = 21$, supposons qu'il existe une permutation σ telle que :

$$\begin{gathered}\sigma^3 = \mathrm{Id},\\ \{a^1, a^2, a^3\}_\sigma = \{a^4, a^5, a^6\}, \{a^4, a^5, a^6\}_\sigma = \{a^7, a^8, a^9\},\\ \{b^1, b^2, b^3\}_\sigma = \{b^4, b^5, b^6\}, \{b^4, b^5, b^6\}_\sigma = \{b^7, b^8, b^9\}.\end{gathered}$$

Alors nous avons $(F_{12})_\sigma = F_{23}$ et $(F_{12})_{\sigma^2} = F_{31}$. De plus, $(F_{21})_\sigma = F_{32}$ et $(F_{21})_{\sigma^2} = F_{13}$. Donc $(F_{\neq})_\sigma = F_{\neq}$. Rappelons que nous devons avoir $F_{\neq} \cap D_i = \{0\}$ pour tout $i \in \{1, 2, 3\}$. Dans ces conditions, il suffit de vérifier $F_{\neq} \cap D_1 = \{0\}$ pour avoir les deux autres conditions.

Supposons que la permutation σ ait ζ cycles de n bits. La partie gauche du vecteur reste fixe. Notons i_b le nombre de bits invariants. Nous avons $i_b + \zeta n = s$. Si $x = x_1 \ldots x_s$, alors

$$x_\sigma = x_1 \ldots x_{i_b} x_{s-\zeta+1} \ldots x_s x_{i_b+1} \ldots x_{i_b+\zeta} \ldots x_{s-2\zeta+1} \ldots x_{s-\zeta}.$$

Posons $\lceil x = x_1 \ldots x_{i_b} 0 \ldots 0$ et $x\rceil = 0 \ldots 0 x_{i_b+1} \ldots x_s$. Nous pouvons alors exposer les deux conjectures :

Conjecture 14. *Il existe σ de la forme précédente et (a, b, c) solution de (B.1) tels que, pour tout $i \in \{1, \ldots, n-1\}$,*

$$\{a^1, \ldots, a^n\}_{\sigma^i} = \{a^{ni+1}, \ldots, a^{ni+n}\} \text{ et } \{b^1, \ldots, b^n\}_{\sigma^i} = \{b^{ni+1}, \ldots, b^{ni+n}\}.$$

Conjecture 15. *Pour la catégorie de solutions précédentes, il en existe une dont le nombre de bits égaux à 1 dans chaque cycle de $a^1, \ldots, a^n, b^1, \ldots, b^n$ est inférieur à 1.*

Proposition 24. *Supposons que (a, b, c) soit une solution vérifiant les deux conjectures. Alors :*

$$\forall i \in \{1, \ldots, 9\},\ n \leq |a^i| \leq s + (1-n)\zeta$$

et

$$\forall i \in \{1, \ldots, 9\},\ n \leq |b^i| \leq s + (1-n)\zeta$$

Preuve. D'après la seconde conjecture, nous avons :

$$|a^i| \leq |\lceil a^i| + \zeta \leq s - n\zeta + \zeta.$$

□

Proposition 25. *Supposons qu'il existe une solution qui satisfait la première conjecture. Alors :*

$$\frac{s}{n} - n \leq \zeta \leq \frac{s}{n}.$$

Preuve. Pour tout $k \in \{1, \ldots, n^2\}$, on note $p^k = a^{k_1} \times a^{k_2}$. Alors

$$Vect(p^1, \ldots, p^{n^2}) = D_1.$$

De plus,

$$Vect(p^1, \ldots, p^{n^2}) \subset Vect(\lceil p^1, \ldots, \lceil p^{n^2}) + Vect(p^1\rceil, \ldots, p^{n^2}\rceil).$$

Soit $G = Vect(\lceil p^1, \ldots, \lceil p^{n^2})$. Nous avons

$$\forall i, j \in \{1, \ldots, n\},\ F_{i,j} \subset G \oplus F_{i,j}\rceil.$$

En effet, il est évident que ces deux sous-espaces vectoriels de E ont une intersection nulle puisque $\lceil$ et $\rceil$ sont des projections sur des coordonnées différentes. Comme $E = F$,

$$E = \sum_{i,j} F_{i,j} = G \oplus \sum_{i,j} F_{i,j}\rceil.$$

Ainsi $\dim(G) = i_b$ et $\dim(\sum_{i,j} F_{i,j}\rceil) = s - i_b = n\xi$. Puisque G est une projection de D_1, $\dim(G) \leq \dim(D_1) = n^2$. D'où $i_b \leq n^2$ et $s - n\xi \leq n^2$. □

Proposition 26. *Nous avons :*

$$i_b \leq \dim(Vect(\lceil a^1, \ldots, \lceil a^n)) \times \dim(Vect(\lceil b^1, \ldots, \lceil b^n)).$$

B.2.5 L'algorithme

L'algorithme central de cette recherche est récursif. Il procède par étape, de 0 à $2n-1$: à l'étape 0, on construit a_1, puis b_1 à l'étape 1, a_2 à l'étape 2,... Ces éléments sont stockés dans des tableaux successifs nommés a_ou_b$_i$.

L'entrée poids_restant correspond au nombre de 1 qu'il reste à distribuer dans le vecteur en construction v. L'entier nbbits_restant représente le nombre de bits qu'il reste à déterminer dans v.

Entrées : un entier poids_restant, un entier nbbits_restants, un vecteur $v \in L$, un entier étape.
Variables globales : Des sous-espaces vectoriels de E : $(D_i)_{0\leq i\leq 2n-1}$, $(F_{\neq i})_{0\leq i\leq 2n-1}$, F, des listes de 18 vecteurs $(\text{a\_ou\_b}_i)_{0\leq i\leq 2n-1}$, un entier ζ.
Début
  Si $a_1 < b_1$ **alors**
    Terminer
  finsi
  Si $a_1 < a_2$ *ou* $a_2 < a_3$ *ou* $b_1 < b_2$ *ou* $b_2 < b_3$ **alors**
    Terminer
  finsi
  Si *poids_ restant*$= 0$ **alors**
    On recopie les vecteurs et espaces vectoriels de l'étape précédente
    Placer v et ses symétriques dans $\text{a\_ou\_b}_{\text{étape}}$
    Mettre à jour $D_{\text{étape}}$
    Si $D_{étape}$ *n'est pas de dimension maximale* **alors**
      Terminer
    finsi
    Mettre à jour $F_{\neq \text{étape}}$
    Si $F_{\neq étape} \cap D_{étape} = \{0\}$ **alors**
      Terminer
    finsi
    Si *étape*$= 2n-1$ *et* (a_1, a_2, a_3) *et* (b_1, b_2, b_3) *forment un système libre et* $E = F_{étape}$ **alors**
      Rechercher c
    finsi
    poids$\leftarrow s + (1-n)\zeta$
    Tant que *poids* $\geq n$ **faire**
      Génère Liste de vecteurs(poids,s,0,étape+1)
    fintantque
  sinon
    Pour *chaque distribution w de bits restants* **faire**
      Génère Liste de Vecteurs(poids_restant-nombre de bits mis à 1,nbbits_restants-nombre de bits utilisés,w,étape)
    finpour
  finsi
Fin

Algorithme 3 : Génère Liste de vecteurs

La recherche de c utilise les équations de Brent en remplaçant a et b par les vecteurs ainsi construits. Si une solution est découverte, elle est affichée.

B.2.6 Résultats obtenus et prolongements possibles

Les calculs pour $n = 3$ sont effectués sur le serveur de calcul Osaka de l'université de Cergy-Pontoise. Le jour de la rédaction de cette thèse, les calculs dans le cas de 7 et 6 cycles ont été effectués. Par exemple, le cas 6 cycles a duré 17 jours sur 9 coeurs. Il n'y a pas eu de résultats positifs.

Une fois les calculs terminés, en cas d'absence de solutions, il est envisagé de les reprendre dans le cas $n = 3$ et $s = 22$, puis de travailler sur d'autres

symétries dans les solutions.

Bibliographie

[1] Ross J. Anderson and Eli Biham. Two Practical and Provably Secure Block Ciphers : BEAR and LION. In Dieter Gollman, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120. Springer-Verlag, 1996.

[2] Mihir Bellare and Russell Impagliazzo. A Tool for Obtaining Tighter Security Analyses of Pseudorandom Function Based Constructions, with Applications to PRP to PRF Conversion. ePrint Archive 1999/024 : Listing for 1999.

[3] Come Berbain, Henri Gilbert, and Jacques Patarin. QUAD : A Practical Cipher with Provable Security . In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4404 of *Lecture Notes in Computer Science*, pages 109–128. Springer-Verlag, 2006.

[4] J. Black, M. Cochran, and T. Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 526–541. Springer-Verlag, 2005.

[5] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 2002.

[6] Andrey Bogdanov and Vincent Rijmen. Zero-Correlation Linear Cryptanalysis on Block Cipher. *Cryptology ePrint archive : 2011/123 : Listing for 2011*, 2011.

[7] Pierre Colmez. Le Rubik's cube, groupe de poche. ENS Ulm, may 2010.

[8] S. Winograd D. Coppersmith. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3) :251–281, 1990.

[9] Jean Delcourt. *Théorie des groupes*. Dunod, 2001.

[10] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for Solving Rubik's Cubes. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 689–700. Springer Berlin Heidelberg, 2011.

[11] ETSI, editor. *Specification of the 3GPP Confidentiality and Integrity Algorithm KASUMI.*, Document available on http ://www.etsi.org/, 1999.

[12] Shimon Even and Oded Goldreich. Des-like functions can generate the alternating group. *IEEE Transactions on Information Theory*, 29(6) :863–865, 1983.

[13] G. Szekeres F. Salzborn. A problem in combinatorial group theory. *Ars Combinatoria*, 7 :3–5, 1979.

[14] Ferber. Sur un symbole analogue aux déterminants, 1899.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completness*. W.H Freeman and Co, 2nd, 1991 edition, 1979.

[16] Henri Gilbert and Marine Minier. New Results on the Pseudorandomness of Some Blockcipher Constructions. In Mitsuru Matsui, editor, *Fast Software Encrytion – FSE '01*, volume 2355 of *Lecture Notes in Computer Science*, pages 248–266. Springer-Verlag, 2001.

[17] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38 :690–728, July 1991.

[18] L. Goubin, M. Ivasco, W. Jalby, O. Ly, V. Nachef, J. Patarin, J. Treger, and E. Volte. CRUNCH. Technical report, Submission to NIST, October 2008.

[19] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing . In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 1996.

[20] Viet Tung Hoang and Phillip Rogaway. On Generalized Feistel Networks. In Tel Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 613–630. Springer-Verlag, 2110.

[21] Subariah Ibrahim and Mohd Aizaini Mararof. Diffusion Analysis of Scalable Feistel Networks. *World Academy of Science, Engineering and Technology*, 5 :98–101, 2005.

[22] David Joyner. *Adventures with Group Theory : Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*. The Johns Hopkins University Press, 2nd edition, 2008.

[23] Marshall Hall Jr. A combinatorial problem on abelian groups. *Proceedings of the American Society*, 3, august 1952.

[24] Charanjit S. Jutla. Generalized Birthday Arracks on Unbalanced Feistel Networks. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 186–199. Springer-Verlag, 1998.

[25] Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of np-complete puzzles. *ICGA Journal*, 31(1) :13–34, 2008.

[26] Lars Knudsen and David Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encrytion – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, 2002.

[27] Lars R. Knudsen. DEAL - A 128-bit Block Cipher. Technical Report 151, University of Bergen, Department of Informatics, Norway, february 1998.

[28] Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977.

[29] J. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82(1) :126–128, 1976.

[30] Xuejia. Lai and James.L. Massey. A Proposal for a New Block Encrytption Standard. In Ivan Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1991.

[31] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2) :373–386, 1988.

[32] Mitsuru Matsui. New Block Encrytpion Algorithm. In Eli Biham, editor, *Fast Software Encrytion – FSE '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 1997.

[33] Valérie Nachef. Generic Attacks on Alternating Unbalanced Feistel Schemes. *Cryptology ePrint archive : 2009/287 : Listing for 2009*, 2009.

[34] Valérie Nachef, Jacques Patarin, and Joana Treger. Generic Attacks on Misty Schemes. In Michel Abdalla and Paulo S.L.M. Barretol, editors, *Progress in Cryptology – LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 222–240. Springer-Verlag, 2010.

[35] Valérie Nachef, Jacques Patarin, and Emmanuel Volte. Zero-knowledge for multivariate polynomials. In *LATINCRYPT*, pages 194–213, 2012.

[36] Valérie Nachef, Emmanuel Volte, and Jacques Patarin. Differential Attacks on Generalized Feistel Schemes. *Cryptology ePrint archive : 2011/750 : Listing for 2011*, 2011.

[37] Valérie Nachef, Emmanuel Volte, and Jacques Patarin. Differential Attacks on Generalized Feistel Schemes. pages 1–19, 2013.

[38] Moni Naor and Omer Reingold. On the Construction of Pseudorandom Permutations : Luby-Rackoff Revisited. *J. Cryptology*, 12(1) :29–66, 1999.

[39] D. Hulme N.T. Courtois, G.V. Bard. A new general-purpose method to multiply 3×3 matrices using only 23 multiplications. http ://arxiv.org/abs/1108.2830 (preprint), 20011.

[40] Kaisa. Nyberg. Linear Approximation of Block Ciphers. In Alfredo de Santis, editor, *Advances in Cryptology– EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 439–444. Springer-Verlag, 1995.

[41] J. Patarin. A Proof of Security in $O(2^n)$ for the Xor of Two Random Permutations. In R. Safavi, editor, *ICITS '2008*, volume 5155 of *Lecture Notes in Computer Science*, pages 232–248. Springer-Verlag, 2008.

[42] J. Patarin, V. Nachef, and C. Berbain. Generic Attacks on Unbalanced Feistel Schemes with Contracting Functions. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 396–411. Springer-Verlag, 2006.

[43] J. Patarin, V. Nachef, and C. Berbain. Generic Attacks on Unbalanced Feistel Schemes with Expanding Functions. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 325–341. Springer-Verlag, 2007.

[44] Jacques Patarin. Generic Attacks on Feistel Schemes. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 222–238. Springer-Verlag, 2001.

[45] Jacques Patarin. Generic attacks for the xor of k random permutations, 2008. Cryptology ePrint Archive, Report 2008/009.

[46] Jacques Patarin. Introduction to Mirror Theory : Analysis of Systems of Linear Equalities and Linear Non Equalities for Cryptology. *Cryptology ePrint archive : 2010/287 : Listing for 2010*, 2010.

[47] Jacques Patarin. Security of balanced and unbalanced Feistel Schemes with Linear Non Equalities. *Cryptology ePrint archive : 2010/293 : Listing for 2010*, 2010.

[48] Christophe Petit and Jean-Jacques Quisquater. Rubik's for cryptographers. *IACR Cryptology ePrint Archive*, 2011 :638, 2011.

[49] Gilles Piret and Jean-Jacques Quisquater. Security of the MISTY Structure in the Luby-Rackoff Model : Improved results. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography– SAC '04*, volume 3357 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 2005.

[50] David Poincheval. A New Identification Scheme based on the Perceptrons Problem. In Alfredo de Santis, editor, *Advances in Cryptology – EUROCRYPT 1995*, volume 950 of *Lecture Notes in Computer Science*, pages 319–328. Springer-Verlag, 1995.

[51] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers : A synthetic approach. In D. Stinson, editor, *CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer-Verlag, 1994.

[52] L. Pyber and E. Szabó. Growth in finite simple groups of Lie type of bounded rank. *ArXiv e-prints*, May 2010.

[53] P. Rogaway and J. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 433–450. Springer-Verlag, 2008.

[54] Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethrige. God's number is 20. http ://cube20.org.

[55] A.M. McLoughlin R.W. Johnson. Noncommutative bilinear algorithm for 3x3 matrix multiplications. *SIAM J. Comput. 15 (2)*, 15(2) :595–603, 1986.

[56] Koichi Sakumuto, Taizo Shirai, and Harunaga Hiwatari. Public-Key Identification Schemes based on Multivariate Quadratic Polynomials. In Philip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 706–723. Springer-Verlag, 2011.

[57] Kouichi Sakurai and Yuliang Zheng. On Non-Pseudorandomness from Block Ciphers with Provable Immunity Against Linear Cryptanalysis. In *IEICE Trans. Fundamentals*, volume E80-A,n.1, January 1997.

[58] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Gollmann, editor, *Fast Software Encrytion – FSE '96*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer-Verlag, 1996.

[59] Adi Shamir. An Efficient Identification Scheme Based on Permuted Kernels (Extended Abstract). In Gilles Brassard, editor, *Advances in Cryptology*

– *CRYPTO 1989*, volume 435 of *LNCS*, pages 606–609. Springer-Verlag, 1989.

[60] W. A. Stein et al. *Sage Mathematics Software (Version 4.7-OSX-32bit-10.5)*. The Sage Development Team, 2011. `http://www.sagemath.org`.

[61] Jacques Stern. A New Identification Scheme based on Syndrome Decoding. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993.

[62] V. Strassen. Gaussian Elimination is not optimal. *Numerische Mathematik*, 13(4) :354–356, 1969.

[63] M. Sugita. Pseudorandomness of a Block Cipher MISTY. Technical report, Technical Report of IEIECE, ISEC 96-9, 1996.

[64] M. Sugita. Pseudorandomness of a Block Cipher with Recursive Strictures. Technical report, Technical Report of IEIECE, ISEC 97-9, 1997.

[65] Joana Treger and Jacques Patarin. Generic Attacks on Feistel Networks with Internal Permutations. In Bart Preenel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 41–59. Springer-Verlag, 2009.

[66] Emmanuel Volte, Valérie Nachef, and Jacques Patarin. Improved Generic Attacks on Unbalanced Feistel Schemes with Expanding Functions. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 94–111. Springer-Verlag, 2007.

[67] Emmanuel Volte, Jacques Patarin, and Valérie Nachef. Zero Knowledge with Rubik's Cubes and Non-abelian Groups. pages 74–91, 2013.

[68] S. Winograd. On Multiplication of 2x2 Matrices. *Linear Algebra and its application*, 4 :381–388, 1971.

[69] Y. Zhen, T. Matsumoto, and H. Imai. On the Construction of Block Ciphers provably secure and not relying on any unproved Hypotheses. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 461–480. Springer-Verlag, 1990.