Mise en oeuvre de cryptosystèmes basés sur les codes correcteurs d'erreurs et de leurs cryptanalyses

Gregory Landais

UPMC
SORBONNE UNIVERSITÉS

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Grégory Landais

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Mise en œuvre de cryptosystèmes basés sur les codes correcteurs d'erreurs et de leurs cryptanalyses

soutenue le 18 Septembre 2014

devant le jury composé de :

M. Nicolas Sendrier	Directeur de thèse
M. Pierre Loidreau	Rapporteur
M. Philippe Gaborit	Rapporteur
M. Jean-Claude Bajard	Examinateur
M. Matthieu Finiasz	Examinateur
M^{me} Caroline Fontaine	Examinateur
M. Antoine Joux	Examinateur
M. Jean-Pierre Tillich	Examinateur

Remerciements

Cette thèse a été effectuée dans l'équipe-projet SECRET de l'Inria Paris-Rocquencourt. Je vais tenter dans ces lignes de remercier tous ceux qui ont rendu celle-ci possible ainsi que tous ceux qui ont rendu ces années des plus agréables. Je vais commencer par Nicolas Sendrier, mon directeur de thèse qui a pris sous son aile le béotien que j'étais. Merci d'avoir de m'avoir fait partager tes connaissances et ta vision de la recherche ainsi que d'avoir su apprécier mon sens de la concision.

Je suis également très reconnaissant envers mes rapporteurs, Pierre Loidreau et Philippe Gaborit qui ont donné de leur temps pour la relecture de ce manuscrit ainsi qu'envers Jean-Claude Bajard, Matthieu Finiasz, Caroline Fontaine, Antoine Joux et Jean-Pierre Tillich qui ont accepté de faire parti de mon jury.

Cette thèse n'aurait pas été la même sans la bonne ambiance du projet SECRET. À ce titre, je remercie les permanents de l'équipe Anne Canteaut, André Chailloux, Pascale Charpin, Gaëtan Leurent, Anthony Leverrier, María Naya-Plasencia, Nicolas Sendrier, Jean-Pierre Tillich et je tiens à saluer la bienveillance qu'ils ont envers leurs étudiants. Un grand merci également à Christelle, championne souriante des tâches administratives. Merci également aux étudiants et invités que j'ai pu côtoyer : Adrien, Alexander, Andrea, Antonia, Audrey, Ayoub, Baudoin, Benoît, Bhaskar, Christina, Chrysanthi, Céline, Denise, Dimitris, Joëlle, Julia, Mamdouh, Marion B., Marion V., Mathieu, Matthieu, Maxime, Nicky, Rafael, Sébastien, Stéphane J., Stéphane M., Valentin, Valérie, Vincent, Virginie, Yann H., Yann L.-C.. Un autre grand merci aux teams baby-foot et mots croisés pour tous ces bons moments.

Mention spéciale au meilleur bureau du monde, j'ai nommé le bureau 1, où la bonne ambiance règne et où l'on peut toujours trouver quelqu'un à qui demander de l'aide ou avec qui pratiquer la méthode du canard en plastique. À Valentin, pour ses nombreux cours de maths et son humour trop souvent incompris, à Virginie, et ses amis imaginaires, à Marion, et ses pantalons aux couleurs improbables, à Christina, pour sa bienveillance et ses petits plats, à Céline et Ayoub pour leur écoute. Votre page de man préférée vous remercie. Deuxième mention spéciale cette fois pour le bureau 2, empli de

personnes sûrement jalouses de ne pas être dans le bureau 1 ; à Joëlle, qui doit prendre conscience que l'ordinateur a plus peur d'elle que l'inverse, à Audrey, enfin une personne ayant de bons goûts culinaires, à María, désolé d'être aussi râleur, et enfin à Benoît pour son aide et son soutien. Dernière mention spéciale (elles commencent à perdre leur coté « spécial » à force) à Mamdouh pour nos discussions scientifiques, à Matthieu pour tout ce qu'il m'a appris, ces séances de pair programming et les cours à l'ENSTA ; à Rafael pour nos partages de chambre et sa ponctualité, ainsi qu'à Vincent pour tous les problèmes que personne n'a jamais rencontré (et ne rencontrera jamais) qu'il m'a demandé de résoudre.

Merci également aux membres de l'équipe PEQUAN qui m'ont si bien accueilli, Anastasiia, Benoît, Jean-Claude, Julien, Olga, et Valérie.

Je termine par remercier ma famille et mes amis ainsi que la plus importante à mes yeux, ma future épouse, Stéphanie.

Table des matières

1 | Introduction

Les travaux présentés dans ce manuscrit sont le résultat de mes quatre années de thèse effectuées à l'Inria Paris-Rocquencourt sous la direction de Nicolas Sendrier d'octobre 2010 à 2014.

Dans ce premier chapitre introductif, je présente les notions de base ainsi que le contexte et les enjeux de la cryptographie basée sur les codes correcteurs d'erreurs. Le manuscrit est ensuite divisé en deux parties. La première partie décrit le schéma de signature CFS ainsi que les meilleurs décisions à prendre lors de la mise en œuvre de ce schéma. Les résultats relatés dans cette partie ont fait l'objet d'une publication lors de la conférence Indocrypt 2012 [43] ainsi qu'à la publication d'un logiciel [42] montrant que le schéma, malgré ses inconvénients, est peut-être utilisé en pratique. La deuxième partie traite de la cryptanalyse *Information Set Decoding* et des divers compromis faisables lors de la mise en œuvre des variantes de cet algorithme. Ces travaux ont donné lieu à un logiciel [41] qui a été évalué face à des challenges cryptographiques et qui a été utilisé lors de la cryptanalyse [44] d'un système proposé l'année passée.

1.1 Cryptographie

Depuis l'antiquité, l'homme a cherché à communiquer des informations de façon confidentielle malgré l'exposition potentielle à des regards indiscrets. L'essor des télécommunications a accru le besoin d'outils assurant la confidentialité, l'authenticité et l'intégrité des informations. Des secrets d'États à la protection de la vie privé, en passant par la sécurité des transactions commerciales, la cryptographie a aujourd'hui de nombreux usages. Jusqu'aux années 1970, les systèmes de chiffrement se basaient sur une information secrète, partagée entre les deux interlocuteurs. Ces systèmes, dits à clé secrète, ont pour avantage un débit élevé mais leur utilisation implique un partage antérieur de cette information secrète. Ce scénario est envisageable à petite échelle pour des besoins ponctuels mais ne l'est pas dans le monde actuel où chacun communique quotidiennement avec des centaines d'entités distinctes potentiellement inconnues (via courrier électronique, navigateur web, téléphonie mobile, matériel réseau, ...).

En 1976, Diffie et Hellman [25] publient ce qui deviendra la base de la cryptographie à clé publique. Ils énoncent les propriétés nécessaires à de tels systèmes et donnent un protocole permettant à deux interlocuteurs de se partager une information secrète uniquement à partir de données publiques. En pratique, les systèmes respectant ce protocole, dits à clé publique, ont souvent des débits faibles ; ils sont donc souvent utilisé afin de démarrer une communication protégée par un chiffrement à clé secrète. Ce procédé est connu sous le nom de cryptographie hybride. En 1977 naît l'algorithme RSA de Rivest, Shamir et Adelman [63], le premier cryptosystème à clé publique.

Depuis ce jour, la recherche sur ce sujet n'a eu de cesse de proposer de nouveaux systèmes et d'affaiblir les existants. Cryptographes et cryptanalystes s'affrontent afin de concevoir et d'évaluer des systèmes à la fois rapides et dignes de confiance.

Ces systèmes sont constitués des éléments suivants :

- Une fonction de génération de clé qui génère un couple $(K_{\text{sec}}, K_{pub})$ aléatoirement.
- Une fonction de chiffrement Enc qui, en utilisant la clé publique K_{pub}, associe à un message clair m un message chiffré c.

$$c = \text{Enc}(K_{pub}, m)$$

- Une fonction de déchiffrement Dec qui, en utilisant la clé secrète K_{sec}, calcule le message clair m associé à un message chiffré c.

$$m = \text{Dec}(K_{sec}, c)$$

Il existe actuellement trois familles de cryptosystèmes à clé publique se basant sur trois domaines différents : la théorie des nombres, les réseaux euclidiens et les codes correcteurs d'erreurs. Les systèmes les plus répandus aujourd'hui sont basés sur la théorie des nombres et reposent sur deux problèmes supposés difficiles, le problème de la factorisation et celui du logarithme discret. Ce quasi-monopole est inquiétant car il n'existe aucune preuve mathématique de la réelle difficulté de ces problème si ce n'est la non-existence de preuve opposée. Autre faille de ces systèmes, Shor [66] a montré que ces deux problèmes pouvaient être résolus en temps polynomial dans le modèle de l'ordinateur quantique. Certes, celui ci est loin d'être opérationnel mais la menace est bien réelle et il faudra, le jour venu, disposé d'alternatives crédibles afin de ne pas se retrouver dépourvu. Voilà pourquoi depuis plusieurs années la recherche examine les systèmes basés sur les réseaux euclidiens et les codes correcteurs d'erreur. Cette thèse s'insère dans le contexte de l'évaluation des cryptosystèmes basés sur les codes correcteurs d'erreurs.

Les fonctions de chiffrement asymétriques se basent sur des fonctions à sens unique munis d'une trappe. Une fonction à sens unique doit être

évaluable efficacement pour tout message clair, et trouver la préimage d'un élément généré par cette fonction doit être une opération difficile. La trappe permet au destinataire légitime de simplifier l'inversion et donc de déchiffrer le message. Elle doit en conséquence rester secrète pour conserver le caractère à sens unique de la fonction.

Une opération sera considérée difficile lorsqu'il sera considéré déraisonnable, en termes de temps ou de moyens et tenant compte du bénéfice potentiel, par l'entité adverse de tenter d'effectuer cette opération.

Un système cryptographique dispose de b bits de sécurité si un ordinateur doit effectuer au moins 2^b opérations pour résoudre le plus simple des problèmes sur lequel se base le système.

Étant donné l'évolution perpétuelle de la technologie, il faut régulièrement réévaluer le nombre de bits de sécurité nécessaire pour considérer une opération difficile.

Il est considéré aujourd'hui qu'un minimum de 112 bits de sécurité est nécessaire pour protéger une information d'ordre gouvernementale [5].

1.2 Codes correcteurs d'erreurs

Les codes correcteurs d'erreurs ont pour objectif de permettre la transmission d'information malgré l'ajout éventuel d'erreurs lors de la transmission. Afin d'y parvenir, les codes ajoutent une redondance au message à transmettre qui, lorsqu'un nombre suffisament faible d'élément de ce message étendu est perdu ou altéré, permettra de reconstituer le message initialement envoyé. Cette reconstitution est appellé le décodage. Je m'interesserai principalement aux codes en blocs ; codes découpant le message en blocs de taille fixe, et les traitant indépendament l'un après l'autre et plus précisément aux codes linéaires.

1.2.1 Codes linéaires

Lors de l'envoi d'un message composé de lettres d'un alphabet $\mathcal{A}$, celui-ci est découpé en bloc de k lettres auxquels sont ajoutés une redondance via une application linéaire transformant un bloc de k lettres en un bloc de n lettres (où n sera évidement choisi supérieur à k). Ce nouveau bloc est transmis à travers un canal de communication, ce qui altérera potentiellement le bloc. Puisque n est supérieur à k, le message reçu ne fera peut-être par parti de l'image de l'application linéaire appliquée (on en déduit la présence d'au moins une erreur). Le destinataire devra donc trouver l'élément de l'image qui a vraisemblablement été envoyé à l'origine. Cependant, si le nombre d'erreurs ajoutées est trop important, il se peut qu'un autre élément de l'image apparaisse plus vraisemblablement comme étant le bloc d'origine ; voire que le bloc reçu devienne un autre élément de l'image, ce qui nous empêcherait de deviner la présence d'erreurs.

Dans ce cadre, un bon code correcteur d'erreur est un code qui disperse suffisamment les mots de codes (afin de pouvoir corriger plus d'erreurs) tout en ayant un rendement, le ratio $\frac{k}{n}$, le plus haut possible (afin de limiter le surcoût du codage).

Les blocs de k lettres avant transmission sont des vecteurs de k éléments de l'alphabet $\mathcal{A}$ et sont appelés des mots d'information. En pratique $\mathcal{A}$ sera un corps fini $\mathbf{F}$, ce qui permet de créer l'espace vectoriel, de dimension k, des mots d'information. L'application linéaire appliquée aux mots d'information associe à chacun de ces mots un élément d'un espace vectoriel de dimension n. Puisque n est supérieur à k, l'image de l'application linéaire est un sous-espace de dimension k de l'espace $\mathbf{F}^n$. Cette image est appelée un code linéaire. La dimension du sous-espace, k, est appelée dimension du code et la dimension de l'espace d'arrivée, n, est appelée longueur du code. Les éléments d'un code linéaire sont appelés mots de code.

Une matrice formée des vecteurs d'une base d'un code $\mathscr{C}$ est appelée matrice génératrice de $\mathscr{C}$. Un mot d'information peut être codé en le multipliant par une matrice génératrice puis le mot de code obtenu peut être décodé en le multipliant par l'inverse de cette même matrice.

La matrice génératrice d'un code dont les k premières colonnes [1] forment la matrice identité $k \times k$ est dite sous forme systématique. Il n'existe qu'une matrice génératrice sous forme systématique pour un code linéaire donné. Un mot d'information codé via une telle matrice est simple à décoder puisqu'il suffit d'extraire les k premières coordonnées du mot de code pour retrouver le mot d'information.

Le code *dual* $\mathscr{C}^\perp$ d'un code $\mathscr{C}$ de dimension k et de longueur n sur $\mathbf{F}$ est le sous-espace vectoriel orthogonal à $\mathscr{C}$ c'est-à-dire le sous-espace vectoriel défini par

$$\mathscr{C}^\perp = \left\{ c' \in \mathbf{F}^n \mid c \cdot c' = 0, c \in \mathscr{C} \right\}.$$

où l'opérateur $\cdot$ est le produit scalaire qui à $x = x_0 \dots x_{n-1}$ et $y = y_0 \dots y_{n-1}$ associe

$$x \cdot y = \sum_{i=0}^{n-1} x_i y_i.$$

Il s'agit d'un code de longueur n et de dimension $n - k$.

Les matrices génératrices de $\mathscr{C}^\perp$ sont dites matrices de parité de $\mathscr{C}$. Le produit Hm^t où H est un matrice de parité de $\mathscr{C}$ et m est un mot de $\mathbf{F}^n$ est appelé le syndrome de m. Le syndrome d'un mot de $\mathscr{C}$ est nul et tout mot de $\mathbf{F}^n$ ayant un syndrome nul appartient à $\mathscr{C}$, c'est-à-dire

$$c \in \mathscr{C} \iff Hc^t = 0.$$

1. N'importe quelles k colonnes pourraient faire l'affaire ; on peut contraindre la définition aux k premières sans perdre de généralités

1.2.2 Décodage

Le canal le plus utilisé est le canal binaire symétrique. Il s'agit d'un canal qui transmet des éléments binaires et qui, indépendamment les uns des autres, peut modifier la valeur de chaque bit avec probabilité p. Lors de la réception d'un mot de code bruité, il faut trouver le mot de code qui est vraisemblablement celui qui a été envoyé, c'est-à-dire celui qui a le plus de coordonnées en commun avec ce premier. Dans le cas de mots binaires la distance de Hamming apportent une notion de distance entre des mots et permet donc d'exprimer la notion de "mot le plus proche".

Soit $\mathbf{F}$ un corps fini et $x = x_0 \dots x_{n-1}$ un mot de $\mathbf{F}^n$, le poids de Hamming de x est défini par

$$\textsc{Poids}(x) = |\{x_i \mid x_i \neq 0\}|.$$

Il s'agit du nombre de coordonnées non-nulles de x.

Soit $y = y_0 \dots y_{n-1}$ un mot de $\mathbf{F}^n$, la distance de Hamming entre x et y est définie par

$$d(x, y) = |\{i \mid x_i \neq y_i\}|.$$

Il s'agit du nombre de coordonnées en lesquelles x et y diffèrent. Cette distance peut également s'écrire

$$d(x, y) = \textsc{Poids}(x - y).$$

Décoder un mot $x \in \mathbf{F}^n$ vis-à-vis d'un code $\mathscr{C}$ de longueur n consiste à trouver le mot de $\mathscr{C}$ le plus proche de x, c'est-à-dire trouver $c \in \mathscr{C}$ tel que $\nexists\ c' \in \mathscr{C}, d(c', x) < d(c, x)$. Il est à noter que selon cette définition, le décodage n'est pas forcément unique.

La distance minimale d'un code $\mathscr{C}$ est définie par

$$d(\mathscr{C}) = \min\{d(x, y) \mid x \in \mathscr{C}, y \in \mathscr{C}, x \neq y\}.$$

Il s'agit de la distance de Hamming séparant deux mots distincts du code $\mathscr{C}$. De par la linéarité du code, elle est également le poids du mot non-nul de $\mathscr{C}$ ayant le plus petit poids et s'écrit donc

$$d(\mathscr{C}) = \min\{\textsc{Poids}(x) \mid x \in \mathscr{C}\}.$$

Soient c un mot de $\mathscr{C}$ et e est un mot de poids $\lfloor \frac{d(\mathscr{C})-1}{2} \rfloor$, alors c est l'unique mot de code le plus proche de $c + e$.

On parlera de succès de décodage lorsque le décodage d'un mot de code bruité $c + e$ donne de façon unique le mot de code non-bruité c.

Un code linéaire $\mathscr{C}$ permet de décoder avec succès tout mot $m = c + e$ où $c \in \mathscr{C}$ et e est un mot de poids inférieur à $\lfloor \frac{d(\mathscr{C})-1}{2} \rfloor$.

Cependant l'existence de cette possibilité de décoder ne donne pas d'algorithme de décodage, si ce n'est un parcours exhaustif des mots de codes.

Puisqu'il est difficile de décoder un code aléatoire, des codes particuliers possédant des structures particulières ont été créés afin de générer des familles de codes munies d'algorithmes de décodage efficace.

Les trois codes suivants sont des codes sur $\mathbf{F}_2$ illustrant ces notions.

1.2.3 Exemples

Le code à répétition

Le code à répétition émet chaque bit a fois. Par exemple, pour $a = 4$, si l'on veut transmettre la chaîne 1011, la séquence codée correspondante est 1111000011111111. Ce code est de dimension 1 et de longueur a, chaque bit étant pris un par un et transformé en a bits. Le rendement est donc de $\frac{1}{a}$, ce qui est très faible. La distance minimale entre deux mots de code est a. En effet, il faut changer les a répétitions d'un bit d'un mot de code pour obtenir un autre mot de code. Ce code permet de corriger jusqu'à $\lfloor \frac{a}{2} \rfloor$ erreur et dispose d'un algorithme de décodage très simple : il suffit de prendre le bit majoritaire de chaque bloc de a bits. Si l'on reprend l'exemple $a = 4$ et que le mot reçu est 0100, le mot de code le plus proche est 0000 et le mot d'information qui a sûrement été envoyé est 0. On remarque que dans cet exemple si le mot reçu est 0110 alors il existe deux mots de code à distance 2 ; on détecte toujours la présence d'une erreur mais le décodage n'est plus unique, on ne sait pas si le mot envoyé est 0 ou 1. Si a est impair alors ce scénario ne peut se produire et tous les mots de l'espace peuvent être décodés de façon unique.

Le code de parité

Le code de parité adjoint à un mot d'information la somme (le Xor) de chacun de ses symboles. Ce code a pour dimension k et pour longueur $k + 1$ ce qui donne un rendement de $\frac{k}{k+1}$, et permet de détecter une erreur mais ne permet pas de la corriger. En effet la distance minimale de ce code est 2 ; il suffit de changer 2 bits d'un mot du code pour obtenir un autre mot du code. La matrice génératrice systématique de ce code est simple, elle consiste en un matrice identité $k \times k$ accolée à une colonne de taille k tout à 1. La matrice de parité est une matrice ligne de taille $k + 1$ tout à 1 puisqu'un mot appartient à ce code si et seulement si la somme de ces composantes est nulle.

Le code de Hamming

Le code de Hamming est un code, pour un entier r donné, de longueur $2^r - 1$ et de dimension $2^r - 1 - r$. Il est construit via sa matrice de parité

qui est constituée de toutes les colonnes distinctes et non nulles de r bits. Ces codes ont tous une distance minimale valant 3 et permettent donc de corriger une erreur. L'algorithme de décodage est simple : si le mot reçu est un mot de code bruité en une seule position $(c + e)$ alors le calcul de son syndrome donne le syndrome du motif d'erreur $((c + e)H = eH)$ et donc le syndrome d'un mot de poids 1. Trouver un motif d'erreur de poids 1 ayant un syndrome donné est simple puisque la matrice de parité du code est constituée de colonnes uniques, il suffit de rechercher l'indice de la colonne correspondante pour trouver l'indice du bit erroné.

1.2.4 Codes de Goppa

Les codes de Goppa sont une sous-classe des codes alternant. Ils peuvent être définis sur le corps fini $\mathbf{F}_{q^m}$ à partir d'un polynôme unitaire, dit de Goppa, $g(x) \in \mathbf{F}_{q^m}[x]$ de degré t et d'un ensemble $L = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$, sous-ensemble de $\mathbf{F}_{q^m}$ dont aucun élément n'est racine de g. Le code de Goppa $\Gamma(L, g)$ est alors défini par sa matrice de parité construite à partir de la matrice

$$H_{\text{aux}} = \begin{pmatrix} \frac{1}{g(\alpha_0)} & \cdots & \frac{1}{g(\alpha_{n-1})} \\ \vdots & & \vdots \\ \frac{\alpha_0^{t-1}}{g(\alpha_0)} & \cdots & \frac{\alpha_{n-1}^{t-1}}{g(\alpha_{n-1})} \end{pmatrix}$$

Chaque élément de cette matrice, qui est un élément de $\mathbf{F}_{q^m}$, est *déroulé* ; c'est-à-dire qu'il est projeté sur $\mathbf{F}_q^m$ puis écrit comme une colonne de m éléments de $\mathbf{F}_q$, pour finalement former une matrice H de taille $mt \times n$. Le code de Goppa $\Gamma(L, g)$ est alors le code de matrice de parité H, c'est-à-dire l'ensemble des mots c de $\mathbf{F}_q^n$ vérifiant $Hc^t = 0$.

Un code de Goppa dont le polynôme de Goppa est irréductible sera dit code de Goppa irréductible.

1.3 Cryptographie basée sur les codes correcteurs d'erreurs

1.3.1 L'ordinateur quantique

Un ordinateur quantique est un ordinateur qui repose sur les propriétés quantiques de la matière pour résoudre des problèmes hors de portée d'un ordinateur classique. Dans [66], Peter Shor montre que tous les cryptosystèmes basés sur la difficulté de la factorisation ou le calcul d'un logarithme discret peuvent être attaqués en temps polynomial sur un tel ordinateur (voir [14] pour un rapport détaillé). Cela menace la quasi-totalité des cryptosystèmes à clé publique déployés en pratique, tels que RSA [63] ou DSA [40]. D'un

autre côté, la cryptographie basée sur la difficulté de décoder un code linéaire est estimée résistante aux attaques quantiques et est donc considérée comme une alternative viable à ces schémas à l'avenir. Cependant, indépendamment de leur prétendue nature *post-quantique*, les cryptosystèmes basés sur les codes offres d'autres bénéfices même pour des applications actuelles grâce à leur excellente efficacité algorithmique, meilleure de plusieurs ordres de grandeurs en termes de complexité que les schémas traditionnels.

1.3.2 Historique

Le premier cryptosystème basé sur les codes est le cryptosystème de McEliece [50], qui proposait d'utiliser des codes de Goppa. Suite à cela, plusieurs familles de code ont été suggérées pour remplacer les codes de Goppa dans ce schéma : les codes de Reed–Solomon généralisés (GRS) [56] ou bien des sous-codes de ces derniers [10], des codes de Reed–Muller [67], des codes algébriques géométriques [36], des codes LDPC [2], des codes MDPC [54] ou plus récemment des codes convolutifs [47]. Certains de ces schémas permettent d'obtenir des clés publiques plus petites que celle du système original tout en vraisemblablement conservant le même niveau de sécurité contre les algorithmes de décodage génériques.

Cependant, pour plusieurs des schémas susmentionnés, il a été montré qu'une description du code sous-jacent aidant au décodage peut être obtenue, cassant par là-même le schéma. Cela s'est produit pour les codes de Reed–Solomon généralisés (GRS) dans [68] et pour leurs sous-codes dans [74]. Dans ce cas, l'attaque retrouve entièrement et en temps polynomial la structure du code à partir de la clé publique. Les codes de Reed–Muller ont également été attaqué, mais cette fois, l'algorithme trouvant la description du code permuté a une complexité sous-exponentielle [52], ce qui est suffisant pour casser les paramètres proposés dans [67] mais qui ne casse pas complètement le schéma. Les systèmes basés sur les codes de géométrie algébrique sont également cassé en temps polynomial mais uniquement pour les courbes hyperelliptiques de faible genre [31]. Un schéma basé sur des codes LDPC [3] a été attaqué dans [57] (le nouveau schéma présenté dans [2] semble insensible à ce genre d'attaque). Deux variantes [1] [73] du schéma basé sur les GRS supposées résister à l'attaque de [68] ont été cassées (respectivement [34] et [24]) par une approche liée au distingueur des codes de Goppa proposé dans [28].

Le cryptosystème de McEliece d'origine reste lui intact. Une modification a été apporté dans [9, 53], utilisant les versions quasi-cycliques ou quasi-dyadiques des codes de Goppa (ou plus généralement des codes alternant dans [9]) afin de réduire significativement la taille de la clé publique. Cependant, il a été montré dans [30, 70] que la structure de ces codes permet de réduire radicalement le nombre d'inconnus de l'attaque algébrique. La plupart des schémas proposés dans [9, 53] ont été cassés par cette approche.

Ce genre d'attaque a une complexité exponentielle et peut être contrecarré en choisissant de petits blocs cycliques ou dyadiques afin d'augmenter le nombre d'inconnues du système algébrique. Lorsque le rendement du code de Goppa est proche de 1 (tel que dans le schéma de signature CFS [22] (voir partie I)), il a été montré dans [29] qu'il était possible de distinguer la clé publique d'une clé aléatoire. Cela invalide les preuves de sécurité des schémas utilisant des codes de rendement proche de 1 puisque tous reposent sur l'hypothèse d'indistinguabilité des codes de Goppa.

1.3.3 McEliece et Niederreiter

En 1978, McEliece [50] propose une fonction à sens unique qui code le message clair puis bruite le mot de code obtenu. Si le code est aléatoire et a des paramètres non triviaux, cette fonction est évaluable efficacement pour tout message et trouver une préimage revient à décoder un code aléatoire, ce qui est difficile. Il ne reste qu'à introduire une trappe. Pour cela, McEliece propose d'utiliser la famille des codes de Goppa. En effet, une fois maquillée, la matrice de parité d'un code de Goppa est semblable à une matrice aléatoire pour une personne ignorant la structure algébrique qui y est cachée alors qu'une personne connaissant cette structure peut utiliser l'algorithme de décodage associé aux codes de Goppa.

Plus concrètement, le système se décrit de cette façon :

La génération de clé consiste à tirer un code de Goppa binaire aléatoire de longueur n et de dimension k capable de corriger t erreurs. La clé secrète est une matrice génératrice G_{sec} de ce code et la clé publique est la matrice $G_{pub} = SG_{sec}P$ où S est une matrice inversible aléatoire $k \times k$ et P est une matrice de permutation aléatoire $n \times n$.

Le chiffrement d'un message m de k bits consiste à calculer $c = mG_{pub}$ et à transmettre $c' = c + e$ où e est un vecteur d'erreur de poids de Hamming t.

Le déchiffrement d'un message chiffré c' consiste à calculer

$$c'P^{-1} = mG_{pub}P^{-1} + eP^{-1} = mSG_{sec} + eP^{-1}$$

puis d'utiliser l'algorithme de décodage pour éliminer le vecteur d'erreur et obtenir mS et en déduire m.

En 1986 [56], Niederreiter propose une variante de la fonction à sens unique basé sur le même problème mais utilisant une matrice de parité d'un code de Goppa.

La génération de clé est similaire à celle du système de McEliece si ce n'est que la clé publique est cette fois la matrice $H_{pub} = SH_{sec}P$ où H_{sec} est une matrice de parité du code de Goppa, S est une matrice inversible aléatoire $r \times r$ et P est une matrice de permutation aléatoire $n \times n$.

Le chiffrement d'un message commence par utiliser un codage en mot de poids constant pour transformer ce message en un motif d'erreur e de longueur n et de poids t. Le syndrome du mot obtenu $s' = eH_{pub}^t$ est le chiffré transmis.

Le déchiffrement d'un message $s'^t = SH_{sec}Pe^t$ est similaire à celui du système de McEliece [2]. On commence par calculer $s = S^{-1}s'^t = H_{sec}Pe^t$ puis, étant donné que Pe^t est un mot de poids t, on peut utiliser l'algorithme de décodage pour retrouver le motif d'erreur dont le syndrome est s. On finit par défaire la permutation et le codage en mot de poids constant pour retrouver le message clair.

1.3.4 Sécurité

Comme tout système à clé publique, il existe deux approches pour attaquer le système de McEliece :

- obtenir une partie du secret à partir des données publiques,
- ou parvenir à inverser la fonction à sens unique pour retrouver un message clair à partir de son chiffré.

Dans le cas du système de McEliece, la sécurité de la clé secrète est assurée par la taille exponentiellement grande de la famille des codes de Goppa. En effet, la meilleure attaque connue aujourd'hui est l'algorithme *Support Splitting Algorithm* [64] qui permet de décider si un code $\mathscr{C}_1$ peut être obtenu à partir d'un code $\mathscr{C}_2$ en permutant les coordonnées des mots de $\mathscr{C}_2$. Un attaquant doit alors tirer un code de Goppa binaire aléatoire et vérifier s'il est équivalent au code public. La grande taille de la famille des codes de Goppa rend cette attaque impraticable.

La sécurité des messages repose sur la difficulté de décoder un code binaire aléatoire. Ne disposant pas de l'algorithme de décodage algébrique fourni par le secret, l'attaquant doit se contenter d'une recherche proche de l'exhaustif (voir partie II).

2. la transposition permet l'alléger la notation

Première PARTIE

LE SCHÉMA DE SIGNATURE CFS

Cette partie traite de la mise en œuvre du schéma de signature CFS. Ces travaux ont menés à une publication lors de la conférence Indocrypt 2012 [43] ainsi qu'à un logiciel diffusé sous licence libre [42].

Un système de signature numérique est un système associant à un document numérique, c'est-à-dire une suite de nombres, une signature numérique, une autre suite de nombres. Cette signature est calculée en utilisant le message ainsi qu'un secret connu uniquement du signataire. Une signature doit ensuite pouvoir être vérifié grâce à une procédure publique.

Ces systèmes peuvent être utilisés pour garantir l'intégrité d'un document, c'est-à-dire le fait qu'il n'a pas été modifié depuis la création de la signature ou bien pour authentifier l'entité ayant apposé une signature à un document.

Pour réaliser ces objectifs, les systèmes de signatures numériques s'appuient sur la cryptographie asymétrique. L'émetteur calcule en utilisant sa clé secrète (connue de lui uniquement) et le message la signature associée au message. Les destinataires utilisent la clé publique (connue de tous) associée à la clé secrète, le message et la signature pour vérifier que la signature a été conçue par quelqu'un possédant la clé secrète.

Pour qu'un système de signature soit qualifié de sûr, il faut que les opérations suivantes soient *difficiles* :

- retrouver la clé secrète à partir de la clé publique et d'un nombre quelconque de documents signés,
- altérer le message de telle façon que la signature reste valide aux yeux de la procédure de vérification,
- créer une signature valide sans connaissance de la clé secrète.

Un système de signature classique repose sur une fonction de chiffrement asymétrique $\mathcal{E}$, de la fonction de déchiffrement associé $\mathcal{D}$ et d'une fonction de hachage h. Pour signer un message m, un signataire commencera par calculer $h(m)$ puis à déchiffrer cette empreinte comme s'il s'agissait d'un message chiffré. La signature de m est alors $s = \mathcal{D}(h(m))$. Pour vérifier une telle signature, il faut également calculer $h(m)$ puis vérifier que $h(m) = \mathcal{E}(s)$. Ce fait prouvera que le signataire dispose des secrets nécessaires au déchiffrement.

Dans le cas du schéma CFS, qui utilise le système de Niederreiter, toutes les empreintes ne peuvent être déchiffrées. Contourner ce problème se fait en ajustant l'empreinte jusqu'à en obtenir une déchiffrable.

2 | Introduction

CFS [23] est un schéma de signature numérique basé sur le cryptosystème de Niederreiter [50]. Il fut publié en 2001 et s'appuie sur la difficulté du problème du décodage par syndrome (voir section 9) et sur l'indistinguabilité des codes de Goppa binaires.

Le monde de la signature numérique est peu diversifiée ; il existe relativement peu de primitive de signature et beaucoup sont basées sur la théorie des nombres. CFS, étant basé sur la théorie des codes, ne sera pas vulnérable aux améliorations algorithmiques que l'ordinateur quantique apporterait s'il venait un jour à atteindre des performances raisonnables et offrirait une alternative le moment venu.

Cependant, les problèmes liés à la mise en œuvre de CFS ont reçus peu d'attention. Cela vient peut-être de l'aspect peu pratique apparent du système et des résultats de cryptanalyses qui ont affaibli le schéma, au moins d'un point de vue théorique.

L'apparence peu pratique du système viens de la grande taille de la clé publique et des longs temps de signature. Certes la taille de la clé publique peut être un problème pour certaines applications, mais certains scénarios d'utilisation peuvent s'accommoder d'un espace de stockage de quelques mégaoctets pour vérifier des signatures. L'impression de lenteur de la primitive de signature peut s'expliquer par les premiers temps donnés dans le papier d'origine [23] qui font mention d'une mise en œuvre logicielle générant une signature en une minute. Or il s'agissait là d'une démonstration de faisabilité. Une mise en œuvre sur circuit logique programmable (ou FPGA) décrite dans [19] annonce une signature en moins d'une seconde.

Il a été prouvé dans [28] que la clé publique de CFS pouvait être distinguée en temps polynomial d'une matrice binaire aléatoire. Cette propriété affaiblit la preuve de sécurité du système mais aucune attaque n'en a été déduite.

3 | Contexte

Nous considérerons uniquement les codes linéaires binaires. La plupart des faits énoncés ici pourraient se généraliser à un alphabet plus grand mais aucun schéma semblable à CFS utilisant des codes non binaires n'a été proposé jusqu'à présent.

3.1 Codes de Goppa binaires

Soit $\mathbf{F}_{2^m}$ le corps fini à 2^m éléments. Soit $n \leq 2^m$, le *support* $L = (\alpha_0, \ldots, \alpha_{n-1})$ un séquence ordonnée d'éléments distincts de $\mathbf{F}_{2^m}$ et le *polynôme générateur* $g(z) \in \mathbf{F}_{2^m}$ un polynôme irréductible unitaire de degré t. Le code de Goppa binaire de support L et de polynôme générateur g est défini par :

$$\Gamma(L, g) = \{(a_0, \ldots, a_{n-1}) \in \{0,1\}^n \mid \sum_{j=0}^{n-1} \frac{a_j}{z - \alpha_j} \bmod g(z) = 0\}.$$

Ce code a pour longueur $n \leq 2^m$ et une dimension $\geq n - mt$. Ce code bénéficie d'une procédure de décodage algébrique pouvant corriger jusqu'à t erreurs. Pour la signature, nous prendrons toujours $n = 2^m$, puisque choisir n plus petit ne ferait qu'accroître le coût de la signature. Pour les paramètres dignes d'intérêt, la dimension sera exactement $k = n - mt$. Nous noterons $r = mt$ la codimension du code.

Dans les cryptosystèmes basés sur les codes de Goppa, les paramètres sont m et t sont connus de tous, la clé secrète est la paire (L, g) et la clé publique est $H \in \{0,1\}^{r \times n}$ une matrice de parité du code.

Densité des syndromes décodables pour un code de Goppa : L'algorithme de décodage algébrique du code ayant pour matrice de parité H pourra décoder un syndrome s si et seulement s'il est de la forme $s = eH^t$, où e a un poids de Hamming inférieur ou égal à t. Il existe $\sum_{i=0}^{t} \binom{n}{i} \approx \binom{n}{t}$ syndromes vérifiant cette propriété. Le nombre de syndromes total étant de 2^r, la proportion des syndromes décodables par la procédure est proche de

$$\frac{\binom{n}{t}}{2^r} = \frac{\binom{2^m}{t}}{2^{mt}} = \frac{2^m(2^m - 1)\cdots(2^m - t + 1)}{t!2^{mt}} \approx \frac{1}{t!}. \tag{3.1}$$

Telle est donc la probabilité d'un syndrome tiré aléatoirement d'être décodable. Il est donc nécessaire de pouvoir associer à un message une grande famille de syndrome afin d'avoir une chance que l'un de ces syndromes soit décodable.

3.2 Décodage complet

Étant donné un code linéaire binaire de matrice de parité $H \in \{0,1\}^{r\times n}$, un décodeur complet est une procédure qui pour tout syndrome $s \in \{0,1\}^r$ retournera un motif d'erreur de poids minimal tel que $eH^T = s$. L'espérance du poids w de e sera immédiatement supérieure au rayon de Gilbert-Varshamov τ_{gv}, défini comme le nombre réel [1] tel que $\binom{n}{\tau_{\text{gv}}} = 2^r$. L'effet de seuil peut être observé dans deux exemples du tableau 3.1.

En pratique, nous définirons un décodeur complet comme un décodeur borné par w (avec $w \geq \tau_{\text{gv}}$), c'est-à-dire une procédure $\psi : s \in \{0,1\}^r \to \{0,1\}^n$ renvoyant un motif d'erreur de syndrome s et de poids $\leq w$ s'il en existe un.

Un tel décodeur peut échouer même si $w \geq \tau_{\text{gv}}$ (voir le tableau 3.1 pour $(m,t) = (20,8)$ et $w = 9 > \tau_{\text{gv}} = 8.91$ par exemple). Cela se produira si aucun des $\binom{n}{w}$ motifs d'erreurs de poids w n'a pour syndrome s. La probabilité d'échec de cet évènement correspond à la probabilité de ne pas tirer un élément donné lors de $\binom{n}{w}$ tirages avec remise dans un espace de taille 2^r ; c'est-à-dire

$$\left(1 - \frac{1}{2^r}\right)^{\binom{n}{w}}.$$

(m,t)	τ_{gv}	$w=8$	$w=9$	$w=10$	$w=11$
(20,8)	8.91	$1-2^{-15}$	0.055	$2^{-131583}$	$2^{-10^{10}}$
(18,9)	10.26	$1-2^{-33}$	$1-2^{-18}$	0.93	2^{-2484}

TABLE 3.1 – Probabilité d'échec d'un décodeur borné à w pour un code de longueur $n = 2^m$ et de codimension $r = mt$

3.3 CFS initial

Une instance de CFS est définie par un code de Goppa binaire Γ de longueur n capable de corriger jusqu'à t erreurs ; de matrice de parité H ; sur le corps fini $\mathbf{F}_2$. Nous appellerons *decode* la fonction de décodage de Γ. Cette fonction prend un syndrome binaire en entrée et renvoie un t-uplet

1. la bijection $x \mapsto \binom{n}{x}$ s'étend aux nombres réels, cela rend la définition de τ_{gv} valable

de positions d'erreur correspondant à un motif d'erreur ayant l'entrée pour syndrome ou échoue si un tel motif n'existe pas. La matrice H est publique et la procédure *decode* est secrète. Signer un document se fait de cette façon :

1. Calculer l'empreinte du document (via une fonction de hachage).
2. Supposer que cette empreinte est un syndrome et utiliser *decode* pour tenter de la décoder.
3. La signature est le motif d'erreur obtenu.

Puisque l'empreinte du document a très peu de chance d'être un syndrome décodable (c'est-à-dire le syndrome d'un mot à distance de Hamming t ou moins d'un mot du code Γ), l'étape 2 va très sûrement échouer. Deux solutions sont proposées pour contourner cette limitation :

- *Le décodage complet* [voir Algorithme 1] ajoute un certain nombre de colonnes de H au syndrome jusqu'à ce qu'il devienne décodable (cela revient à tenter de deviner quelques erreurs).
- *L'adjonction d'un compteur* modifie le message avec un compteur puis calcule l'empreinte jusqu'à ce le syndrome associé devienne décodable. Le compteur qui a rendu le syndrome décodable est adjoint à la signature.

Les deux méthodes nécessitent une moyenne de $t!$ tentatives de décodage avant succès (conséquence de (3.1), voir [23]). *L'adjonction d'un compteur* a pour inconvénient d'inclure la fonction de hachage dans la procédure de décodage ce qui oblige à la mettre en œuvre sur la plateforme cible, ce qui serait dérangeant pour un coprocesseur dédié. De plus, cette méthode rend la taille de la signature variable puisque celui-ci fait partie de la signature et qu'il a un écart type élevé. Pour finir, la contre-mesure Parallel-CFS (voir §3.5) n'est pas applicable si cette méthode est employée.

Algorithme 1 Signature utilisant le décodage complet

Entrée :
Un message M,
un entier $w > t$,
une fonction de hachage h.

Sortie :
La signature du message M, composée d'un w-uplet de positions.

Fonction SIGNER(M, w, h)
$s \longleftarrow h(M)$
Boucle
$(i_{t+1}, \ldots, i_w) \xleftarrow{R} \{0, \ldots, n-1\}^{w-t}$
$(i_1, \ldots, i_t) \longleftarrow \text{decode}(s + H_{i_{t+1}} + \ldots + H_{i_w}, t)$
Si $(i_1, \ldots, i_t) \neq$ **échec** **Retourner** $(i_1, \ldots, i_w)$

3.4 Attaques

Il existe des attaques permettant de distinguer efficacement une clé publique CFS (une matrice de parité d'un code de Goppa binaire) d'une matrice aléatoire de même taille [28]. Cependant n'a aujourd'hui pas encore affaibli la sécurité de la clé secrète. En pratique, les meilleurs techniques pour contrefaire une signature sont basés sur le décodage générique d'un code linéaire, c'est-à-dire la résolution du problème de décodage par syndrome (CSD).

Les deux principales techniques pour résoudre le problème CSD sont le décodage par ensemble d'information (ISD), décrit partie II, et le décodage utilisant le paradoxe des anniversaires généralisé (GBA).

3.4.1 Décoder un parmi plusieurs (DOOM) :

Dans un scénario de forge de signature (la création d'une signature valide sans connaissance du secret), un attaquant peut créer autant de messages lui convenant qu'il le souhaite et être satisfait par le fait d'obtenir une signature valide pour un de ces messages. Les bénéfices d'un accès à plusieurs syndromes ont été mentionné par Bleichenbacher pour l'algorithme GBA [2]. L'adaptation à l'algorithme ISD fut proposée dans [37] et fut ensuite généralisée et analysée dans [65] sous le nom *Decoding One Out of Many (DOOM)*. Ce dernier montre que si l'on dispose de N syndromes cibles et que le décodage d'un d'entre eux est suffisant, la complexité temporelle est réduite d'un facteur approchant $\sqrt{N}$ par rapport à la situation d'un unique syndrome à décoder. Le gain cesse de croître une fois que N a atteint une limite supérieure dépendante de l'algorithme utilisé (ISD ou GBA). En pratique, pour contrer cette attaque et récupérer 80 bits de sécurité, il faut multiplier la taille de la clé par 400 ; ou bien par 100 si l'on est prêt à multiplier le coût de la signature par 10. La contre-mesure Parallel-CFS offre un bien meilleur compromis.

3.5 Parallel-CFS : une contre-mesure à DOOM

Parallel-CFS est une contre-mesure proposée par M. Finiasz en 2010 [32], visant à annuler le bénéfice obtenu par un attaquant souhaitant signer un message parmi plusieurs. L'idée consiste à produire λ empreintes différentes du document à signer (entre 2 et 4 en pratique) et de signer chacune séparément. Finalement, la signature consistera en la collection des signatures de chacune de ces empreintes (voir Algorithme 2). De cette façon, si un attaquant parvient à créer une signature pour l'empreinte d'un de ses messages, il sera contraint de créer les autres signatures à partir de cette

2. Attaque présentée en 2004, jamais publiée mais décrite dans [60]

seule empreinte, ce qui le ramène au scénario initial, c'est-à-dire le décodage d'un unique syndrome cible. Comme mentionné dans [32], signer en utilisant la méthode de l'adjonction d'un compteur devient impossible puisque la contre-mesure impose de décoder plusieurs empreintes d'un unique message et que l'adjonction d'un compteur modifie le message. Cette contre-mesure augmente d'un facteur λ le coût de la signature, la taille de la signature et le coût de la vérification.

L'attaque DOOM fait passer la sécurité du schéma CFS de $\approx \frac{r}{2}$ bits de sécurité à $\approx \frac{r}{3}$. La contre-mesure Parallel-CFS fait remonter celle-ci à $\approx \frac{2^{\lambda-1}}{2^{\lambda}-1} r$ bits de sécurité. Cette valeur tendant vers $\frac{r}{2}$ lorsque λ croît, il est possible de se rapprocher autant que souhaité de la sécurité initiale.

Algorithme 2 Parallel-CFS avec un décodage complet

Entrée :
Un message M,
un entier $w > t$,
un entier $\lambda > 0$,
un ensemble de fonctions de hachage $H = \{H_i\}_{1 \leq i \leq \lambda}$.

Sortie :
La signature totale composée de λ signatures individuelles.

Fonction SIGN_MULT(M, w, λ, H)
  Pour $1 \leq i \leq \lambda$
    $s_i \longleftarrow$ SIGN(M, w, H_i)
  Retourner $(s_i)_{1 \leq i \leq \lambda}$

Dans [32], l'attaque de Bleichenbacher est généralisée pour attaquer plusieurs empreintes. Cette analyse montre que pour la plupart des paramètres, trois empreintes, parfois même deux, suffisent pour annuler les bénéfices de l'attaque. Pour ISD, il est montré dans [65] que les bénéfices de DOOM ne sont pas aussi important que pour GBA. Ce résultat n'a pas été généralisé au cas des empreintes multiples comme dans [32], mais cela n'aurait que peu de chance de changer la situation ; si le nombre d'empreintes est assez haut pour contrecarrer DOOM-GBA, il en sera très probablement de même pour DOOM-ISD.

3.6 Implémentation passée

Nous n'avons connaissance d'aucune mise en œuvre logicielle publique de CFS. Il existe une mise en œuvre sur FPGA décrite dans [19], pour les paramètres originaux, à savoir $n = 2^{16}$, $t = 9$, et $w = 11$. Celle-ci utilise l'algorithme de Berlekamp-Massey pour décoder et annonce la création d'une signature en 0.86 secondes sur un FPGA de petite taille.

4 | Sélection des paramètres

Pour décoder un unique syndrome, ISD est plus efficace que GBA, et pour plusieurs, DOOM-GBA (l'attaque de Bleichenbacher généralisée) est plus efficace que DOOM-ISD. Le tableau 4.1 donne le nombre d'opérations binaires requises par les attaques suivantes :

- ISD-MMT [49], une variante d'ISD, permet de décoder λ instances distinctes à un syndrome.
- ISD-Dumer [26], une variante précédant ISD-MMT ; les nombres sont extraits de [33].
- GBA-DOOM [32], l'attaque de Bleichenbacher généralisée, permet de décoder un Parallel-CFS de multiplicité λ.

					proba.	taille clé	bits de sécurité		
m	t	w	λ	τ_{gv}	d'échec	publique	(1)	(2)	(3)
16	9	11	3	10.46	~ 0	1 MB	77.4	78.7	74.9
18	9	11	3	10.26	~ 0	5 MB	87.1	87.1	83.4
18	9	11	4	10.26	~ 0	5 MB	87.5	87.5	87.0
20	8	10	3	8.91	~ 0	20 MB	82.6	85.7	82.5
20	8	9	5	8.91	5.5%	20 MB	87.9	91.0	87.3
24	10	12	3	11.05	~ 0	500 MB	126.4	126.9	120.4
26	9	10	4	9.82	10^{-8}	2 GB	125.4	127.5	122.0

(1) ISD-MMT (2) ISD-Dumer (3) GBA-DOOM

TABLE 4.1 – Jeux de paramètres pour Parallel-CFS utilisant un code de Goppa binaire de longueur 2^m. Les bits de sécurité sont le $\log_2$ du nombre d'opérations binaires requises par l'attaque.

Le tableau 4.1 donne les principales caractéristiques (dont la sécurité) pour quelques jeux de paramètres. Les paramètres d'origine sont donnés pour référence mais sont désormais sous la barre des 80 bits de sécurité. Nous proposons deux familles de codes de Goppa : les codes de longueur 2^{18} corrigeant jusqu'à 9 erreurs et les codes de longueur 2^{20} corrigeant jusqu'à 8 erreurs. Ces derniers permettent de signer plus rapidement (car le temps de signature dépend de $t!$) mais ont en contrepartie une clé publique plus grande. Ces deux familles permettent d'atteindre 80 bits de sécurité en uti-

lisant $\lambda = 3$ décodages parallèles. Les derniers paramètres sont donnés pour des critères de sécurité plus grand mais n'ont pas été implémentés.

Le tableau 4.1 ne mentionne pas la variante BJMM [7] d'ISD car la complexité non-asymptotique de cette variante est difficile à évaluer et car les paramètres considérés ne sont pas ceux où l'amélioration apportée par cette variante est la plus importante. Il sera nécessaire de considérer cette attaque pour dimensionner le schéma à la sécurité voulu.

5 | Décodage algébrique des codes de Goppa

Ce chapitre décrit le décodage algébrique non pas dans un contexte communication mais dans le contexte cryptographique du schéma CFS. Ici, le secret est un code de Goppa $\Gamma(L, g)$ binaire de longueur $n = 2^m$, de dimension $n - r$, de polynôme générateur g et de support L. Le polynôme g est de degré t, unitaire, irréductible et à coefficient dans $\mathbf{F}_{2^m}$. Le support $L = (\alpha_0, \ldots, \alpha_{n-1})$ consiste en tous les éléments de $\mathbf{F}_{2^m}$ dans un ordre spécifique. La clé publique H est une matrice de parité sous forme systématique de $\Gamma(L, g)$. Nous dénommons $L_S = (\beta_0, \ldots, \beta_{r-1})$ les éléments du support correspondants à la partie identité de H (les premières ou dernières r coordonnées de L par exemple). Un décodeur algébrique de codes de Goppa prend en entrée un syndrome binaire $s = (s_0, \ldots, s_{r-1}) \in \{0, 1\}^r$ et renvoie, si celui ci existe, un motif d'erreur $e \in \{0, 1\}^n$ de poids t tel que $eH^t = s$. Il existe plusieurs algorithmes (décrit plus tard dans cette section) réalisant ce décodage. Chacun ont en commun ces trois étapes :

1. Transformer le syndrome binaire s en un nouveau syndrome, polynôme à coefficient dans $\mathbf{F}_{2^m}$.
2. Résoudre une équation clé liant ce nouveau syndrome au polynôme localisateur d'erreur.
3. Calculer les racines du polynôme localisateur pour trouver les positions non nulles du vecteur d'erreurs.

5.1 Équation clé des codes de Goppa

Le syndrome algébrique $R(z) = \sum_{0 \leq j < r} s_j f_{\beta_j}(z)$ correspondant à s est calculé comme une somme de syndromes élémentaires $f_\beta(z)$ définie pour tout $\beta \in \mathbf{F}_{2^m}$ comme

$$f_\beta(z) = \frac{1}{z - \beta} \bmod g(z) = \frac{1}{g(\beta)} \frac{g(z) - g(\beta)}{z - \beta}. \tag{5.1}$$

Seuls les syndromes élémentaires des r éléments de L_S sont nécessaires. L'équation clé correspondante est

$$\sigma(z)R(z) = \frac{d}{dz}\sigma(z) \bmod g(z), \deg \sigma \leq t \tag{5.2}$$

qui a pour unique solution, à un scalaire multiplicatif près, $\sigma(z) \in \mathbf{F}_{2^m}[z]$. S'il existe un motif d'erreur $e \in \{0,1\}^n$ de poids $\leq t$ tel que $eH^t = s$, alors toute solution de (5.2) est un scalaire multiple de $\sigma(z) = \prod_{\beta \in \mathrm{supp}(e)}(z-\beta)$, le polynôme localisateur de e ($\mathrm{supp}(e)$ est le sous-ensemble de L correspondant aux coordonnées non nulles de e). L'équation (5.2) est résolue en utilisant l'algorithme de Patterson [61].

5.2 Équation clé des codes alternants

Un code de Goppa $\Gamma(L, g)$ peut également être vu comme un code alternant. Nous utilisons le fait que, pour les codes de Goppa binaires, $\Gamma(L, g) = \Gamma(L, g^2)$ lorsque g est sans facteur multiple. Nous avons toujours $R(z) = \sum_{0 \leq j < r} s_j f_{\beta_j}(z)$ mais le syndrome élémentaire $f_\beta(z)$ a pour degré $2t-1$ et non plus $t-1$. Il est maintenant défini pour tout $\beta \in \mathbf{F}_{2^m}$ par

$$f_\beta(z) = \frac{1}{g(\beta)^2}\frac{1}{1-\beta z} \bmod z^{2t} = \sum_{i=0}^{2t-1} \frac{\beta^i z^i}{g(\beta)^2}. \tag{5.3}$$

L'équation clé correspondante est

$$\sigma_{inv}(z)R(z) = \omega(z) \bmod z^{2t}, \deg \omega < t, \deg \sigma_{inv} \leq t, \tag{5.4}$$

qui a pour unique solution, à un scalaire multiplicatif près, $(\sigma_{inv}(z), \omega(z)) \in \mathbf{F}_{2^m}[z]^2$. S'il existe un motif d'erreur $e \in \{0,1\}^n$ de poids exactement égal à t tel que $eH^t = s$ et si $(\sigma_{inv}(z), \omega(z))$ est solution de (5.4) alors le polynôme localisateur d'erreur vaut $\sigma(z) = z^t \sigma_{inv}(z^{-1}) = \prod_{\beta \in \mathrm{supp}(e)}(z-\beta)$, à un scalaire multiplicatif près. Pour rester cohérent avec l'équation clé des codes de Goppa, nous parlerons de $\sigma(z) = z^t \sigma_{inv}(z^{-1})$ comme étant la solution de l'équation. La résolution de (5.4) peut être réalisée en utilisant l'algorithme de Berlekamp-Massey [48] ou avec l'algorithme d'Euclide étendu.

5.3 Recherche de racines

L'état de l'art concernant la recherche des racines d'un polynôme dans une extension de $\mathbf{F}_2$ est l'algorithme des traces de Berlekamp [11]. Sa complexité en $O((m+t)t^2)$ le rend supérieur aux techniques utilisant une recherche exhaustive telles que la recherche de Chien ou l'évaluation de polynôme de Horner dont la complexité est linéaire en n et donc exponentielle en m.

Cet algorithme utilise les propriétés de la fonction *Trace* qui à $z \in \mathbf{F}_{2^m}$ associe $Tr(z) \in \mathbf{F}_2$ où

$$Tr(z) = z^{2^{m-1}} + z^{2^{m-2}} + \ldots + z^2 + z$$

La fonction *Trace* permet en particulier de représenter de façon unique un élément $\alpha \in \mathbf{F}_{2^m}$ par le m-uplet $(Tr(\alpha b_0), Tr(\alpha b_1), \ldots, Tr(\alpha b_m))$ où $B = \{b_0, b_1, \ldots b_m\}$ forme une base de $\mathbf{F}_{2^m}$ sur $\mathbf{F}_2$. Pour calculer les racines d'un polynôme $\sigma(z)$ de degré t scindé (ayant t racines distinctes dans $\mathbf{F}_{2^m}$) l'algorithme des traces de Berlekamp factorise $\sigma(z)$ en $\sigma(z) = \sigma_0(z)\sigma_1(z)$ où

$$\sigma_0(z) = \mathrm{PGCD}(\sigma(z), Tr(b_0 z)),$$
$$\sigma_1(z) = \mathrm{PGCD}(\sigma(z), Tr(b_0 z) - 1) = \frac{\sigma(z)}{\sigma_0(z)}$$

L'algorithme est ensuite appelé récursivement pour factoriser $\sigma_0(z)$ et $\sigma_1(z)$. Chaque étage de l'arbre binaire d'appel récursif fait intervenir un $b_i \in B$ différent. Les propriétés de la fonction *Trace* garantissent que toutes les racines de $\sigma(z)$ seront séparées.

L'arbre d'appel peut être réduit en utilisant des techniques adaptées lorsque le degré du polynôme passe sous un certain seuil, 3 ou 4 en pratique [20].

6 | Mise en œuvre

6.1 Arithmétique des corps finis

Nous avons besoin de mettre en œuvre les extensions du corps binaire $\mathbf{F}_2$ de degré $m = 18$ et $m = 20$. Pour des corps de petite taille la meilleure approche consiste à mettre en table les logarithmes et exponentielles de chaque élément en base α, un élément primitif de $\mathbf{F}_{2^m}$. Cela reste efficace tant que les tables rentrent dans le cache du processeur. Ce n'est pas le cas ici ; nous avons donc choisi de mettre en œuvre ces corps comme des extensions de degré 2 de $\mathbf{F}_{2^{m/2}}$. Nous avons utilisé

$$\mathbf{F}_{2^{20}} = \mathbf{F}_{2^{10}}[x]/(x^2 + x + \alpha), \mathbf{F}_{2^{10}} = \mathbf{F}_2[x]/(x^{10} + x^9 + x^7 + x^6 + 1)$$

avec α un élément primitif de $\mathbf{F}_{2^{10}}$ tel que $\alpha^{10} + \alpha^9 + \alpha^7 + \alpha^6 + 1 = 0$, et

$$\mathbf{F}_{2^{18}} = \mathbf{F}_{2^9}[x]/(x^2 + x + 1), \mathbf{F}_{2^9} = \mathbf{F}_2[x]/(x^9 + x^5 + 1).$$

Les corps $\mathbf{F}_{2^9}$ et $\mathbf{F}_{2^{10}}$ sont suffisamment petits pour être mis en table (et quasiment chaque accès mémoire sera satisfait par le cache L1 sur notre plateforme cible) et, en utilisant la technique de Karatsuba, une multiplication dans le corps d'extension nécessite trois multiplications dans le corps de base. Pour des architectures ayant de plus fortes contraintes, des tours d'extensions plus hautes pourraient apporter un gain.

6.1.1 Bit-slicing

La technique précédente, nommée *bit-packing*, stocke la représentation binaire d'un élément de $\mathbf{F}_{2^m}$ dans un mot machine de taille adéquate (de 16 bits pour un élément de $\mathbf{F}_{2^{10}}$ par exemple). Une autre stratégie, nommée *bit-slicing*, consiste à stocker cette même représentation dans m mots, un bit par mot. Ainsi, si le plus grand mot d'une machine peut contenir b bits, il est possible de représenter b éléments de $\mathbf{F}_{2^m}$ dans m mots. Il n'est alors plus possible de mettre en table les opérations élémentaires et il faut donc réaliser ces opérations de façon calculatoire. Effectuer une opération se fait cependant sur b éléments en parallèle. Si la perte de performance créée

par ce changement de représentation est un facteur inférieur à b alors nous obtenons une accélération.

L'addition de deux éléments est peu affectée par le changement de représentation ; il suffit d'appliquer m opérations XOR pour calculer en parallèle b sommes. Pour la multiplication, il est nécessaire de multiplier les représentations polynomiales des éléments modulo le polynôme définissant l'extension du corps. La page [13] recense les algorithmes minimisant le nombre d'opérations nécessaires pour multiplier deux polynômes à coefficients dans $\mathbf{F}_2$. L'inversion d'un élément x est réalisée en calculant x^{2^m-2}.

6.2 Décodage

Lorsque l'on signe un message M, nous calculons une empreinte $s = h(M)$ que nous verrons comme un syndrome correspondant à la clé publique H. Le mot e de poids minimal tel que $s = eH^t$ a pour poids $w > t$ et donc s ne peut être décodé en utilisant le décodeur algébrique qui ne peut corriger que t erreurs. Si, comme décrit dans l'algorithme 1, nous devinons correctement $\delta = w - t$ positions de l'erreur, nous pourrons décoder avec succès en utilisant le décodeur algébrique le syndrome modifié. Il a été prouvé dans [23] que cela réussit après une moyenne de $t!$ essais. Nous présentons dans l'algorithme 3 une variante où le syndrome est modifié sous forme polynomiale. Cela permet de réduire le coût du calcul de l'équation clé en factorisant le calcul ne dépendant pas des δ positions choisis. De plus un crible est utilisé pour décider si le polynôme localisateur est scindé dans $\mathbf{F}_{2^m}$ (s'il possède t racines distinctes dans ce corps). Cela permet de vérifier pour un coût moindre qu'une recherche de racine si le décodage a réussi.

6.2.1 Mise sous forme polynomial du syndrome :

Le premier syndrome polynomial $R_0(z)$ est calculé une fois seulement à partir de s. Ensuite, pour chaque essai de décodage, $R_0(z)$ est mis à jour en y ajoutant $\delta = w - t$ syndromes élémentaires $f_\beta(z)$. Cette mise à jour a un coût proportionnel à δt opérations dans le corps fini, ce qui est négligeable en pratique.

6.2.2 Résolution de l'équation clé :

Comme mentionné précédemment, il existe plusieurs équations clés et parfois plusieurs façons de les résoudre. Dans tous les cas, cette résolution doit être effectuée complètement et produit le même polynôme localisateur $\sigma(z)$. Le coût de cette opération est proportionnel à t^2 opérations dans le corps fini.

Algorithme 3 Signer en utilisant des codes de Goppa binaires

Entrée :
Un message M,
une fonction de hachage h.

Sortie :
La signature composée d'un w-uplet de positions ou **échec**.

Fonction SIGNER(M, h)
$s \longleftarrow h(M)$
$R_0(z) \longleftarrow \sum_{0 \leq j < r} s_j f_{\beta_j}(z)$
Pour tout $B \subset L$ de cardinalité $\delta = w - t$
$R(z) \longleftarrow R_0(z) + \sum_{\beta \in B} f_\beta(z)$
$\sigma(z) \longleftarrow$ solve_key_eq$(R(z))$
Si $z^{2^m} = z \bmod \sigma(z)$
$A \longleftarrow$ racines_de$(\sigma(z))$
Retourner indices des éléments de $A \cup B$ dans L
Retourner échec

6.2.3 Recherche des racines :

L'équation clé possède toujours une solution $\sigma(z)$, qu'il existe un motif d'erreur de poids t approprié ou non. Le motif a pour poids t si et seulement si le polynôme $\sigma(z)$ a toutes ses racines dans le corps $\mathbf{F}_{2^m}$, c'est-à-dire si $\sigma(z) \mid z^{2^m} - z$. En pratique, nous cherchons à savoir si $z^{2^m} = z \bmod \sigma(z)$ et nous faisons une recherche de racines uniquement lorsqu'un tel $\sigma(z)$ a été trouvé car la recherche de racines est plus coûteuse (voir tableau 7.2). Cela requiert m élévations au carré modulo $\sigma(z)$ pour un coût proportionnel à mt^2 opérations dans le corps fini. Cette étape sera le coût dominant de la signature.

6.3 Rejet des instances de décodage dégénérées

Différents syndromes et équations clés peuvent être utilisés pour décoder en utilisant des codes de Goppa. Pour chacun, des structures de contrôle conditionnelles sont requises. À un certain point, la valeur d'un coefficient est vérifiée (le coefficient de tête dans l'algorithme d'Euclide étendu, ou la divergence dans l'algorithme de Berlekamp-Massey), et si elle vaut zéro, le séquencement des opérations est affecté. Ignorer ce test (c'est-à-dire supposer la valeur non nulle) permet une accélération logicielle significative (les boucles sont plus simples à dérouler) et simplifie la mise en œuvre sur une plateforme contrainte. Pour réaliser cela il est nécessaire qu'une division par l'élément 0 ne stoppe pas l'exécution du programme ; il suffit de renvoyer un élément quelconque du corps lorsque cela se produit. La contrepartie est

qu'une faible proportion des tentatives de décodage produit des résultats incohérents et échoue. Cela a peu d'impact dans le contexte de CFS puisque presque tous les décodages échouent quoi qu'il arrive. Cette remarque fut faite dans [19] pour simplifier le contrôle dans la mise en œuvre sur FPGA.

6.4 Gérer les échecs de décodage

Pour les paramètres $(m, t) = (20, 8)$ et $w = 9$, la signature peut échouer avec probabilité $\nu = 5.5\%$. Cela signifie que certains messages ne pourront être signés. L'utilisation de Parallel-CFS aggrave encore ce comportement : avec une multiplicité λ, la probabilité atteint $\mu = 1 - (1 - \nu)^{\lambda}$, c'est-à-dire presque 25% pour $\lambda = 5$, ce qui n'est pas acceptable. *Ajouter un compteur* aux empreintes permet de contourner cette limitation :

> Soit une famille $\mathcal{F} = \{f_i, 0 \leq i < 2^b\}$ de 2^b transformations injectives sur les syndromes (par exemple l'ajout de vecteurs constants, quelconques, prédéfinis et distincts). Soit $s_1, \ldots, s_\lambda$ les empreintes que doit traiter un Parallel-CFS de multiplicité λ. Les tentatives de décodages se font sur le λ-uplet $f(s_1), \ldots, f(s_\lambda)$ pour tout $f \in \mathcal{F}$ jusqu'au succès. Un vérificateur essayera également tout $f \in \mathcal{F}$, dans le même ordre que le signataire. Les b bits formant l'indice de la transformation ayant permis la signature, peuvent être ajouté à la signature pour accélérer la vérification.

La probabilité d'échec du décodage pour tout $f \in \mathcal{F}$ est μ^{2^b}. En reprenant l'exemple précédent, avec $b = 5$ la probabilité passe de 0.249 à 2^{-64}. De plus chaque incrément de b élève la probabilité au carré. La sécurité n'est pas affectée car une même transformation est appliquée à chaque empreinte. Remarque : il est nécessaire d'appliquer une seule transformation à toutes les empreintes car dans le cas contraire, le degré de liberté apporté par ces transformations permettrait à un attaquant d'utiliser l'attaque DOOM avec 2^b cibles sur chacune des empreintes (ce qui lui ferait gagner un facteur $\sqrt{2^b}$).

7 | Performances

7.1 Génération d'une signature

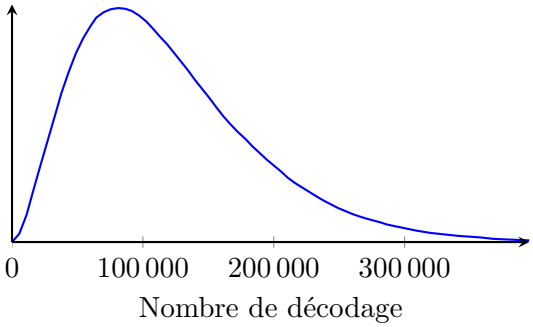
Deux décodeurs, Berlekamp-Massey et Patterson, sont comparés pour divers jeux de paramètres. Le nombre moyen de tentatives de décodage et le temps nécessaire pour générer une signature sur notre plateforme cible (un cœur d'un Intel Xeon W3670 à 3.20 GHz) sont reportés dans le tableau 7.1. Pour chaque jeu de paramètres, $\approx 400\,000$ signatures ont été générées.

	(m, t, w, λ)			
	(18,9,11,3)	(18,9,11,4)	(20,8,10,3)	(20,8,9,5)
nombre de décodages	1 117 008	1 489 344	121 262	360 216
temps (BM)	14.70 s	19.61 s	1.32 s	3.75 s
temps (Pat.)	15.26 s	20.34 s	1.55 s	4.26 s

TABLE 7.1 – Nombre moyen de décodage algébrique et temps nécessaire pour générer une signature (version non *bit-slicée*)

Les primitives de l'arithmétique dans le corps fini occupent 75% du temps de calcul, dont la majorité (66%) pour la multiplication. On remarque dans le tableau 7.1 que le nombre de décodages est très proche (mais légèrement supérieur) à la valeur attendue $\lambda t!$. La seule exception est pour $(m, t, w, \lambda) =$

FIGURE 7.1 – Distribution du nombre de décodage par signature $(m, t, w, \lambda) = (20, 8, 10, 3)$



$(20, 8, 9, 5)$. Dans ce cas un décodage échoue avec probabilité 5.5%, mais nous avons considéré qu'un décodage échouait lorsque le nombre de tentatives dépassait un seuil donné. Expérimentalement, le meilleur compromis lorsque $\lambda = 5$ est d'autoriser un maximum de 200 000 décodages par syndrome (au lieu des 2^{20} possibles dans cet exemple). Cela élève la probabilité d'échec à 8.4% et en adjoignant un compteur de 6 bits aux empreintes (voir section 6.4) la signature échoue avec probabilité 2^{-95}. En pratique, nous observons que la durée de signature est quasiment doublée.

Le schéma CFS se prête très bien à une représentation *bit-slicée* (voir sous-section 6.1.1) puisque l'algorithme réalise un très grand nombre de fois et de façon indépendante une même instruction. Une mise en œuvre de CFS *bit-slicée* réalisée avec Peter Schwabe parvient à générer une signature en utilisant les paramètres $(m, t, w, \lambda) = (20, 8, 10, 3)$ en 0.077 secondes en moyenne.

7.2 Comparaisons des décodeurs

La table 7.2 donne le nombre d'opérations dans le corps fini nécessaires pour réaliser un essai de décodage algébrique. Ces nombres ont été obtenu en comptant le nombre d'opérations dans le corps fini réalisées lors d'une exécution du logiciel. Les étapes dites *critiques* sont celles exécutées dans le corps de la boucle de l'algorithme 3. Les étapes *non critiques* sont celles réalisées en dehors de la boucle et sont donc réalisées une seule fois par décodage complet, c'est-à-dire $\lambda = 3$ fois par signature. Une opération dans le corps fini est une multiplication, une division, une élévation au carré, une inversion ou une racine carrée. Les additions ne sont pas comptées car elles sont réalisées à l'aide d'un simple Xor. En pratique, cela donne une estimation correcte de l'effort de calcul, permet de comparer simplement les décodeurs et donne un aperçu des coûts relatifs des diverses étapes. Tous les nombres sont constants à l'exception de l'étape de recherche de racines (l'algorithme des traces de Berlekamp).

Si l'on considère les parties non critiques, il apparaît que le calcul du syndrome initial et la recherche de racines sont les étapes dont le coût domine, ce qui fait de l'algorithme de Patterson un algorithme plus efficace que celui de Berlekamp-Massey (qui utilise un syndrome de taille double) pour effectuer un décodage algébrique. La situation se retrouve inversée lorsque l'on considère les étapes critiques (qui sont répétée un très grand nombre de fois dans CFS), car la résolution d'une équation clé est plus efficace dans l'algorithme de Berlekamp-Massey.

		critique				non critique	
(m,t)	type	(1)	(2)	(3)	(1)+(2)+(3)	(4)	(5)
(18,9)	BM	58	180	840	1078	2184	3079.1
(18,9)	Pat.	38	329	840	1207	1482	3079.1
(20,8)	BM	52	144	747	943	1950	3024.6
(20,8)	Pat.	34	258	747	1039	1326	3024.6

(1) mise à jour du syndrome
(2) résolution de l'équation clé
(3) test si $\sigma(z)$ est scindé
(4) calcul du syndrome initial
(5) recherche de racines

TABLE 7.2 – Nombre d'opérations dans le corps fini (hors additions) par décodage. Les opérations *critiques* sont effectuées $t!$ fois par signature. Les opérations *non critiques* une seule fois.

8 | Conclusion

Pour un choix de paramètres judicieux, nous avons montré que CFS, en fait Parallel-CFS, est utilisable en pratique malgré son aspect encombrant. La plus rapide de nos instances nécessite un peu plus d'une seconde, ce qui est lent mais acceptable. La clé publique correspondante a une taille de 20 mégaoctets, ce qui peut disqualifier le schéma pour certaines applications. On peut noter que la clé publique n'est pas nécessaire pour générer une signature, mais seulement la clé secrète qui consiste en un polynôme générateur occupant mt bits (160-162 ici) et d'un support, permutation de 2^m éléments (qu'il est possible de générer à la volée à partir d'une graine). La mise en œuvre de la primitive de signature présentée ici nécessite peu d'espace [1] et de mémoire, le rendant approprié pour des architectures massivement parallèles (telles que des GPUs) ou des dispositifs orientés circuit (tels que des FPGAs ou même des cartes à puces).

1. dont la majorité pour le corps fini pour lequel il existe plusieurs options

Deuxième PARTIE

INFORMATION SET DECODING

Cette partie décrit les compromis faisables lors de la mise en œuvre logicielle des variantes Stern/Dumer et MMT d'Information Set Decoding, l'une des meilleures attaques sur les cryptosystèmes basés sur les codes correcteurs d'erreur. Ces travaux sont illustrés par un logiciel publié sous licence libre [41].

9 | Le problème du décodage par syndrome

Le problème du décodage par syndrome est le suivant :
Entrée : $H_0 \in \{0,1\}^{r \times n}$, $s_0 \in \{0,1\}^r$ et $w \in \mathbb{N}$
Sortie : $e \in \{0,1\}^n, \textsc{Poids}(e) \leq w, eH_0^t = s_0$

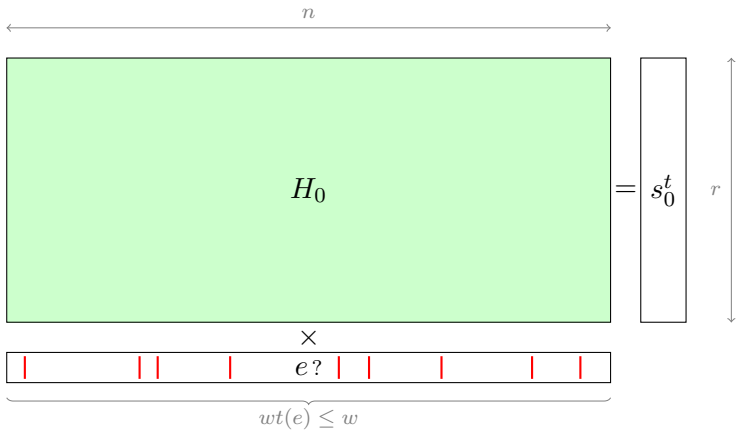


FIGURE 9.1 – Représentation du problème du décodage par syndrome.

Ce problème consiste à trouver w colonnes (ou moins) de H_0 qui, une fois sommées, donnent s_0.

La version décisionnelle, c'est-à-dire montrer l'existence d'un tel e pour une instance aléatoire du problème, est un problème NP-complet (pour des paramètres non triviaux) [12].

Une instance aléatoire de ce problème a $\binom{n}{w}2^{-r}$ solutions en moyenne (représenté figure 9.2). Si $\binom{n}{w} < 2^r$, on peut attendre d'une instance aléatoire de ce problème qu'elle n'ait pas de solution.

Dans le cadre de la cryptographie basée sur les codes, s_0 n'est pas choisi uniformément dans $\{0,1\}^r$ mais dans $\{eH_0^t \mid e \in \{0,1\}^n, \textsc{Poids}(e) \leq w\}$. Ce choix implique la présence d'au moins une solution. Le problème considéré devient le problème avec promesse suivant :

Entrée : $H_0 \in \{0,1\}^{r \times n}$, $s_0 \in \{0,1\}^r$ et $w \in \mathbb{N}$

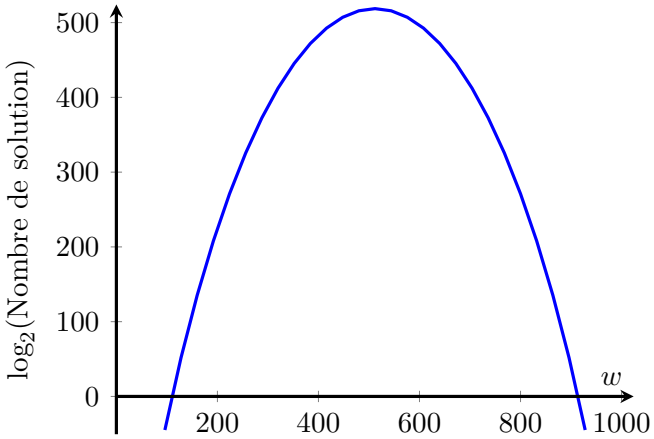


FIGURE 9.2 – Logarithme du nombre moyen de solutions d'une instance aléatoire du problème CSD en fonction du poids w pour $n = 1024$ et $r = 500$.

Promesse : $\exists\ e_0 \in \{0,1\}^n$ avec $\text{POIDS}(e) \leq w$ tel que $e_0 H_0^t = s_0$

Sortie : $e \in \{0,1\}^n, \text{POIDS}(e) \leq w, eH_0^t = s_0$

Lorsque $\binom{n}{w} \ll 2^r$, la probabilité de la présence d'une solution autre que celle injectée par le choix de s_0 est négligeable. De plus, le poids w est fixé et connu de tous et la probabilité d'une solution de poids $< w$ est encore plus négligeable. Le problème peut alors se simplifier en :

Entrée : $H_0 \in \{0,1\}^{r\times n}$, $s_0 \in \{0,1\}^r$ et $w \in \mathbb{N}$

Promesse : $\exists\ e_0 \in \{0,1\}^n$ avec $\text{POIDS}(e) = w$ tel que $e_0 H_0^t = s_0$

Sortie : e_0

Nous utiliserons la notation $\text{CSD}(H_0, s_0, w)$ pour représenter une instance de ce problème mais aussi pour représenter l'ensemble de ses solutions.

Il est important de remarquer qu'appliquer certaines transformations linéaires inversibles telles qu'une permutation des colonnes de H_0 ou une multiplication de H à droite par une matrice inversible ne modifie pas le problème :

Soit P une matrice de permutation $\in \{0,1\}^{n\times n}$ et S une matrice inversible $\in \{0,1\}^{r\times r}$, alors

$$e \in \text{CSD}(H, s, w) \iff eP \in \text{CSD}(S^t HP, sS, w).$$

En effet, $eH^t = s \iff ePP^{-1}HS = sS$ et $\text{POIDS}(e) = \text{POIDS}(eP^{-1})$.

Cela permet de mettre la matrice H_0, entrée d'une instance du problème, sous forme systématique mais également d'en permuter les colonnes.

Il existe deux familles d'algorithmes permettant de résoudre ces problèmes ; le décodage utilisant un paradoxe des anniversaires et le décodage par ensemble d'information (Information Set Decoding (ISD)).

9.1 Décodage par paradoxe des anniversaires

Le décodage par paradoxe des anniversaires calcule et stocke deux listes de sommes de $\frac{p}{2}$ colonnes de H_0. Le syndrome cible s_0 est ajouté à chaque élément d'une des listes. Si un élément commun à ces deux listes existe alors cet élément s s'écrit $s = eH_0^t = e'H_0^t + s_0$ où e et e' sont des mots de longueurs n et de poids $\frac{p}{2}$. Le vecteur $e + e'$ est de longueur n, de poids $\leq p$ et vérifie $(e + e')H_0^t = s_0$; il est donc solution du problème.

Le paradoxe des anniversaires nous dit que deux listes d'éléments tirés uniformément d'un ensemble à n éléments auront une intersection non-vide en moyenne lorsque leurs tailles sera de l'ordre de $\sqrt{n}$. Dans notre cas, il faut donc construire deux listes de taille $\sqrt{\binom{w}{n}}$ pour obtenir une solution en moyenne.

9.2 Décodage par ensemble d'information

Prange

L'idée du décodage par ensemble d'information fut introduite par Prange en 1962 [62]. Elle consiste en le choix d'un ensemble d'information, c'est-à-dire un ensemble de positions de taille $n - r$ tel que la matrice formée des colonnes de H_0 correspondantes à ces positions soit inversible. L'algorithme de décodage vérifie par algèbre linéaire si le vecteur s_0^t est obtenu en combinant w colonnes de H_0 parmi les r positions non choisies. Cela se produit si le support de l'erreur recherchée et l'ensemble d'information choisi sont disjoints. Si l'on note $\mathcal{P}$ la probabilité de succès de cette procédure, il suffit de la répéter, en tirant un nouvel ensemble d'information, $1/\mathcal{P}$ fois en moyenne pour obtenir une solution. Les améliorations apparues depuis relâchent les hypothèses de l'idée originale en augmentant la taille de l'ensemble d'information sélectionné et en autorisant une intersection entre l'ensemble d'information et le support de l'erreur, puis tentent de parcourir cette intersection de manière efficace.

Dans le papier original définissant son système de chiffrement de 1978 [51], McEliece utilise ce décodage pour évaluer la sécurité du système.

Lee et Brickell

La variante de Lee et Brickell parue en 1988 [45] introduit une intersection de taille p entre l'ensemble d'information et le support de l'erreur recherchée ; et vérifie s'il existe une de ces intersections telle que la somme des colonnes correspondantes et du syndrome donne un mot de poids $w - p$. L'optimisation du coût de l'algorithme (produit du coût d'une itération et du nombre moyen d'itérations avant terminaison) amène à un p optimal valant 2.

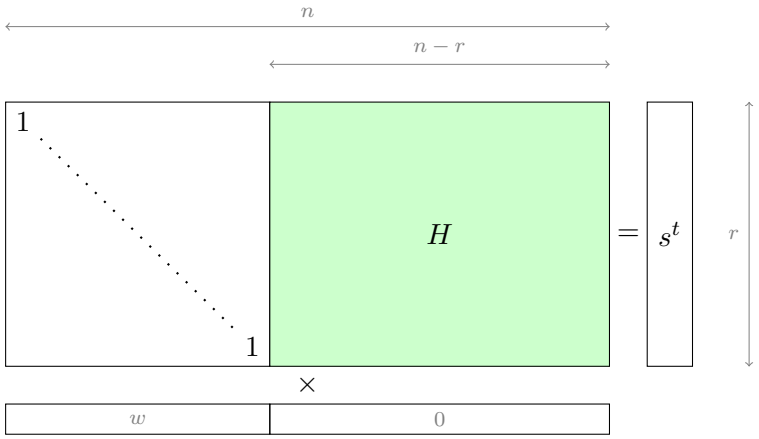


FIGURE 9.3 – L'algorithme de Prange vérifie si s est de poids w après pivot de Gauss sur les colonnes exclues de l'ensemble d'information

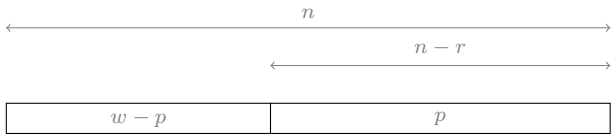


FIGURE 9.4 – Distribution des poids des motifs d'erreur considérés par l'algorithme de Lee et Brickell

Léon/Krouk

En 1988, Léon et Krouk [46] ajoutent une contrainte sur le support de l'erreur recherchée : une certaine quantité ℓ des bits du motif d'erreur choisis en dehors de l'ensemble d'information sont fixés à 0. Cela réduit la probabilité de tomber dans la configuration recherchée et donc augmente le nombre d'itérations nécessaires. Par contre, cette contrainte se répercute sur la somme des colonnes (elle doit valoir 0 sur les positions correspondantes) ce qui réduit fortement le coût de vérification de la configuration sélectionnée et donc le coût d'une itération.

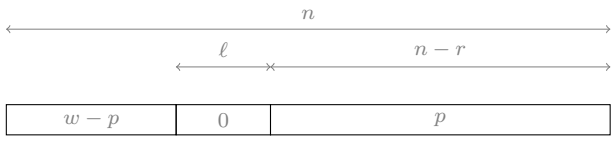


FIGURE 9.5 – Distribution des poids des motifs d'erreur considérés par l'algorithme de Leon et Krouk

Stern/Dumer

Stern introduit le paradoxe des anniversaires dans le décodage par ensemble d'information en 1989 [69]. Celui-ci est appliqué sur une matrice génératrice afin de trouver des mots de poids faibles. Une recherche de collisions sur une fenêtre de taille ℓ est effectuée sur les lignes de la matrice formée des colonnes indexées par l'ensemble d'information. L'algorithme de Stern est amélioré par Dumer en 1991 [27], qui applique cette idée sur la matrice de parité. Cette variante supprime la contrainte sur la fenêtre de taille ℓ sur les motifs recherchés et apporte un léger gain asymptotique. Ces deux variantes ne parcourent pas toutes les intersections possibles mais le font efficacement, ce qui cette fois encore, réduit le coût et la probabilité de réussite d'une itération. Elles sont étudiées plus en détail dans la section 10.0.3.

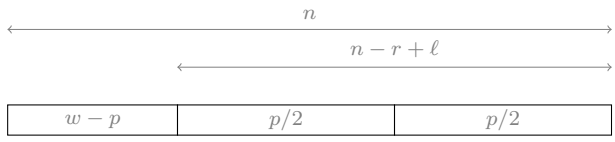


FIGURE 9.6 – Distribution des poids des motifs d'erreur considérés par l'algorithme de Dumer

Ball collision decoding

Ball collision decoding est présenté par Bernstein, Lange et Peters en 2011 [17]. Cet algorithme part de l'algorithme de Stern mais tolère un poids q sur les bits de la fenêtre de taille ℓ. Il est supérieur à l'algorithme de Stern mais est asymptotiquement équivalent à l'algorithme de Dumer.

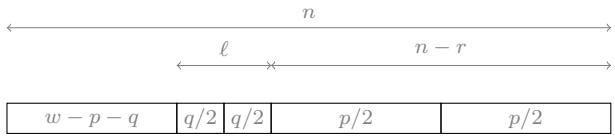


FIGURE 9.7 – Ball collision decoding

MMT

Présenté par May, Meurer et Thomae en 2011 [49], la variante MMT apporte une amélioration asymptotique en appliquant le paradoxe des anniversaires généralisé [71] et en utilisant la technique des représentants [35]. Cet algorithme est détaillé section 10.0.3.

BJMM

Introduite par Becker, Joux May et Meurer en 2012 [8], la variante BJMM est actuellement la variante de plus faible complexité asymptotique connue. Celle ci introduit un nouvel étage de recherche de collisions et ajoute une étape sélectionnant les motifs d'erreurs résultant d'une somme de deux motifs dont le support est joint en une position.

10 | Le décodage par ensemble d'information

Une partie des travaux couverts par cette thèse concerne l'écriture d'un programme mettant en œuvre certains algorithmes de la famille ISD. Ce programme utilise le cadre décrit dans [33] qui englobe les variantes Stern/Dumer, Ball collision decoding , MMT et BJMM. Les variantes Stern/Dumer et MMT sont couvertes par ce programme. Il pourra être étendu aux autres variantes.

10.0.1 Cadre

Le programme utilise le cadre défini dans [33].

La sélection de l'ensemble d'information est faite en permutant aléatoirement les colonnes de H_0. On notera P la matrice correspondante. Une élimination de Gauss partielle est ensuite appliquée sur H_0P ainsi qu'à s_0 pour former H, H', s et s'. Il s'agit d'une élimination de Gauss interrompue après traitement de $r - \ell$ colonnes. La figure 10.1 montre la forme de la matrice après cette opération.

Un algorithme SubISD est ensuite utilisé pour trouver des éléments candidats e de poids p tels que $eH^t = s$. Cet algorithme est différent selon la variante.

À tout candidat e généré par l'algorithme SubISD est appliqué un test final ; il est vérifié si le poids de Hamming de $e' = eH'^t + s'$ vaut $w - p$. Tout candidat passant ce test produit un vecteur $e_0 = (e' \mid e)P^{-1}$ solution du problème initial.

10.0.2 L'outil de base : la fusion de liste

Chaque variante d'Information Set Decoding utilise comme brique de base la fusion de liste. Cette fusion prend comme paramètres :

- deux listes de couples L_0 et L_1 où chaque élément de L_0 et L_1 est un couple (x, y) où $x \in \{0,1\}^b$ et $y \in \{0,1\}^c$,
- un vecteur $s \in \{0,1\}^c$.

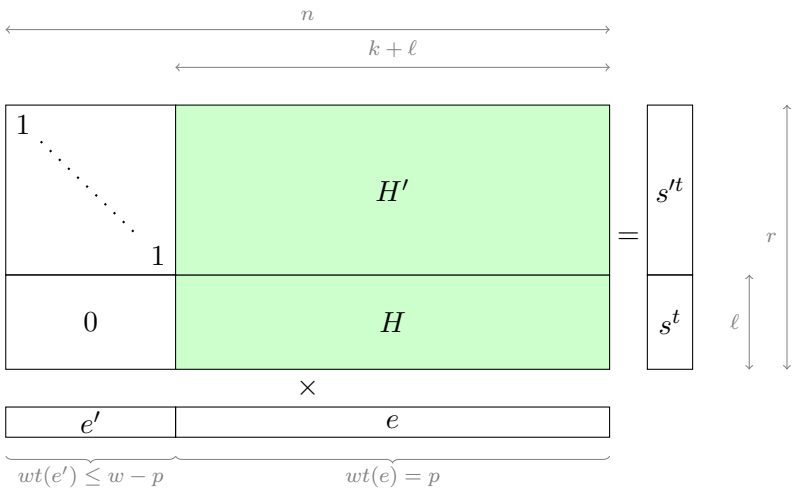


FIGURE 10.1 – H_0P et s_0 après élimination de Gauss partielle.

Algorithme 4 Information Set Decoding générique

Entrée :

Une matrice binaire H_0 de taille $r \times n$,
un vecteur binaire s_0 de longueur r,
un entier w,
un entier $p \leq w$,
un entier $0 \leq \ell \leq r$.

Sortie :

Un vecteur binaire $\hat{e}$ de longueur n et de poids de Hamming w tel que $\hat{e}H_0^t = s_0$.

Fonction ISD(H_0, s_0, w, p, ℓ)
  Boucle
    $P \leftarrow$ matrice de permutation aléatoire $n \times n$
    $(H, H', s, s') \leftarrow$ GAUSSPARTIEL(H_0P, ℓ, s_0)
    Pour $e \in$ SUBISD(H, s, p)
      Si POIDS$(eH'^t + s') = w - p$ **Retourner** $(eH'^t|e)P^{-1}$

Commentaires :

GAUSSPARTIEL(H) transforme H tel que montré figure 10.1.

Elle retourne l'ensemble des mots

$$\{x_0 + x_1 \mid (x_0, y_0) \in L_0, (x_1, y_1) \in L_1, y_0 + y_1 = s\}.$$

Si le deuxième élément de chaque couple de L_0 et de L_1 est tiré uniformément dans $\{0,1\}^c$, l'espérance du nombre d'éléments renvoyés par la fusion est $\frac{|L_0||L_1|}{2^c}$. En effet la liste $L_0 \times L_1$ possède $|L_0||L_1|$ éléments et chacun d'entre eux a probabilité $\frac{1}{2^c}$ de s'écrire $((x_0, y_0), (x_1, y_0 + s))$.

Cette fusion sera typiquement utilisée pour trouver un mot $e \in \{0,1\}^b$ de poids donné p vérifiant $eH^t = s$ où $H \in \{0,1\}^{c \times b}$ et $s \in \{0,1\}^c$ à partir de deux listes de couples de la forme $(e_0, e_0 H^t)$ où chaque $e_0 \in \{0,1\}^b$ sera de poids $\frac{p}{2}$.

10.0.3 Les algorithmes SubISD

Le problème que doit résoudre un algorithme SubISD est le suivant :

Étant donnés $H \in \{0,1\}^{\ell \times (k+\ell)}$, $s \in \{0,1\}^\ell$ et $p \in \mathbb{N}$, trouver $e \in \{0,1\}^{k+\ell}$ de poids de Hamming p tel que $eH^t = s$.

Les algorithmes SubISD de toutes les variantes qui rentrent dans le cadre décrit dans [33] suivent le même principe : des listes de sommes partielles de colonnes de H sont construites puis une ou plusieurs recherches de collision sont effectuées.

Stern/Dumer

Algorithme 5 SubISD : Stern/Dumer

Entrée :
Une matrice binaire H de taille $\ell \times (k + \ell)$,
un vecteur binaire s de longueur ℓ,
un entier p pair,
un ensemble S, sous-ensemble de $[\![0, k + \ell[\![$.

Sortie :
Un ensemble E de vecteurs binaires de poids p tel que $\forall e \in E, eH^t = s$.

Fonction SubISD_SD(H, s, p, S)
$W_0 \leftarrow \{e_0 \mid e_0 \in \{0,1\}^{k+\ell}, \text{Poids}(e_0) = p/2, \text{Support}(e_0) \subset S\}$
$W_1 \leftarrow \{e_1 \mid e_1 \in \{0,1\}^{k+\ell}, \text{Poids}(e_1) = p/2, \text{Support}(e_1) \subset \bar{S}\}$
$L_0 \leftarrow \emptyset, L_1 \leftarrow \emptyset$
Pour $e_0 \in W_0$
  $L_0 \leftarrow L_0 \cup (e_0, e_0 H^t)$
Pour $e_1 \in W_1$
  $L_1 \leftarrow L_1 \cup (e_1, e_1 H^t)$
Retourner Fusion(L_0, L_1, s)

L'algorithme 5 ne peut s'appliquer que lorsque p est multiple de 2.

L'algorithme est paramétré par le choix d'un sous-ensemble $S \subset [\![0, k+\ell[\![$ de cardinal $\lfloor \frac{k+\ell}{2} \rfloor$. Le complémentaire de S, c'est-à-dire $\{x \in [\![0, k+\ell[\![\mid x \notin S\}$, sera dénommé $\bar{S}$. Un découpage simple consiste à choisir $S = [\![0, \lfloor \frac{k+\ell}{2} \rfloor [\![$ et donc $\bar{S} = [\![\lfloor \frac{k+\ell}{2} \rfloor, k+\ell [\![$ (représenté figure 10.2).

Ces sous-ensembles permettent de définir W_0 et W_1, ensembles des mots binaires de longueur $k+\ell$ de poids $\frac{p}{2}$ et de support respectif S et $\bar{S}$.

Pour commencer, la liste L_0 des couples $(e_0, e_0 H^t)$ pour chaque mot $e_0 \in W_0$ est créée. De même, la liste L_1 des couples $(e_1, e_1 H^t)$ pour chaque mot $e_1 \in W_1$ est créée.

Finalement la procédure FUSION est appelée avec comme paramètre ces deux listes et le vecteur cible s. Le résultat de cette fusion est renvoyé.

Chaque élément de L_0 (respectivement L_1) est un couple (e, eH^t) où e est un élément de W_0 (respectivement W_1). Tout élément e retourné par la fusion de L_0 et L_1 s'écrit donc $e = e_0 + e_1$ où $e_0 \in W_0$, $e_1 \in W_1$ et vérifie $e_0 H^t + e_1 H^t = s$. Puisque e_0 et e_1 n'ont aucunes coordonnées en commun (car de supports complémentaires), e est de poids p et est solution de $eH^t = s$.

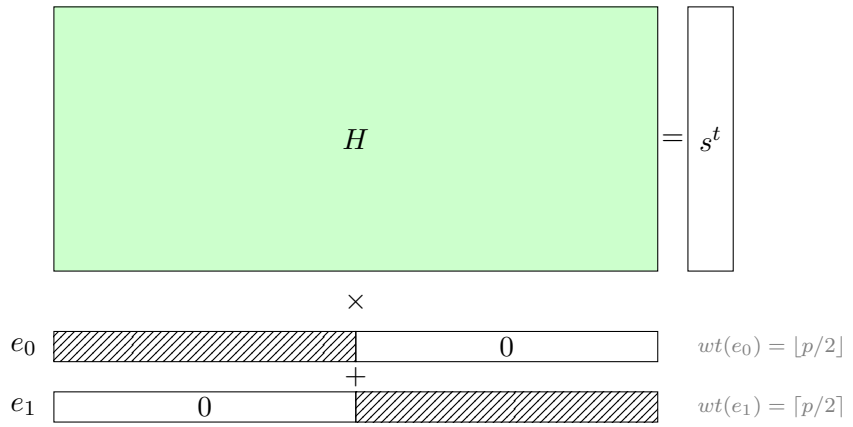


FIGURE 10.2 – Exemple de découpage du support pour l'algorithme Stern/Dumer.

MMT

L'algorithme 6 ne peut s'appliquer que lorsque p est multiple de 4.

L'algorithme est paramétré par un entier $0 \leq \ell_2 \leq \ell$, le choix d'un sous-ensemble $A \subset \{0,1\}^{\ell_2}$ ainsi que de deux sous-ensembles S_0 et $S_2 \subset [\![0, k+\ell[\![$ de cardinal $\lfloor \frac{k+\ell}{2} \rfloor$. Les complémentaires de S_0 et S_2 seront dénommés S_1 et S_3 respectivement.

Le paramètre ℓ_2 permet de définir la matrice H' en sélectionnant ℓ_2 lignes de H. Les $\ell - \ell_2$ lignes de H restantes forment la matrice H''. Les ℓ_2

Algorithme 6 SubISD : May, Meurer et Thomae

Entrée :

Une matrice binaire H de taille $\ell \times (k+\ell)$,
un vecteur binaire s de longueur ℓ,
un entier p multiple de 4,
un entier $0 \leq \ell_2 \leq \ell$,
deux ensembles S_0 et S_2, sous-ensembles de $[\![1, k+\ell]\!]$,
un ensemble A, sous-ensemble de $\{0,1\}^{\ell_2}$.

Sortie :

Un ensemble E de vecteurs binaires de poids p tel que $\forall e \in E, eH^t = s$.

Fonction SubISD_MMT$(H, s, p, \ell_2, S_0, S_2, A)$
  $(H', H'', s', s'') \leftarrow$ SelectionLignes(H, s, ℓ_2)
  $S_1 \leftarrow \bar{S_0}, S_3 \leftarrow \bar{S_2}$
  Pour $i \in \{0,1,2,3\}$
    $W_i \leftarrow \{e \mid e \in \{0,1\}^{k+\ell}, \text{Poids}(e) = p/4, \text{Support}(e) \subset S_i\}$
    $L_i \leftarrow \emptyset$
    Pour $e \in W_i$
      $L_i \leftarrow L_i \cup (e, eH'^t)$
  $L \leftarrow \emptyset$
  Pour $a \in A$
    $W_{01} \leftarrow$ Fusion(L_0, L_1, a)
    $W_{23} \leftarrow$ Fusion$(L_2, L_3, a + s')$
    $L_{01} \leftarrow \emptyset, L_{23} \leftarrow \emptyset$
    Pour $e \in W_{01}$
      $L_{01} \leftarrow L_{01} \cup (e, eH''^t)$
    Pour $e \in W_{23}$
      $L_{23} \leftarrow L_{23} \cup (e, eH''^t)$
    $L \leftarrow L \cup$ Fusion(L_{01}, L_{23}, s'')
  Retourner L

Commentaires :

SelectionLignes(H, s, ℓ_2) sélectionne ℓ_2 lignes de H pour former H', H'', s' et s'' (illustrés figure 10.3).

(respectivement $\ell - \ell_2$) coordonnées de s correspondantes sont sélectionnées pour former le vecteur s' (respectivement s''). Sans perte de généralité, on peut supposer que les ℓ_2 dernières lignes de H ont été sélectionnées pour former H' (tel que sur la figure 10.3).

Le sous-ensemble S_0 (respectivement S_1, S_2, S_3) permet de définir l'ensemble W_0 (respectivement W_1, W_2, W_3) formé des mots binaires de longueur $k + \ell$, de poids $\frac{p}{4}$ et de support S_0 (respectivement S_1, S_2, S_3).

Pour commencer, la liste L_0 (respectivement L_1, L_2, L_3) des couples (e, eH'^t) pour chaque mot $e \in W_0$ (respectivement W_1, W_2, W_3) est créée.

Deuxièmement, pour chaque $a \in A$, la fonction FUSION est appelée avec comme paramètre L_0, L_1 et le vecteur cible a. Le résultat de cette fusion sera dénommé $W_{01}(a)$.

Ainsi tout élément de $W_{01}(a)$ s'écrit $e = e_0 + e_1$ où $e_0 \in W_0$, $e_1 \in W_1$ et vérifie $e_0 H'^t + e_1 H'^t = a$. Chacun de ces éléments est donc un mot de poids p (puisque W_0 et W_1 sont complémentaires) vérifiant $eH'^t = a$

Pour cette même valeur de a, la fonction FUSION est appelée une seconde fois avec comme paramètre L_2, L_3 et le vecteur cible $a + s'$. Le résultat de cette fusion sera dénommé $W_{23}(a)$.

Par un raisonnement similaire au précédent, tout élément de $W_{23}(a)$ est un mot e de poids p vérifiant $eH'^t = a + s'$.

La liste $L_{01}(a)$ (respectivement $L_{23}(a)$) des couples (e, eH''^t) pour chaque mot $e \in W_{01}(a)$ (respectivement $W_{23}(a)$) et la fonction FUSION est appelée une dernière fois avec comme paramètre $L_{01}(a)$, $L_{23}(a)$ et le vecteur cible s'' formant la liste $W(a)$.

Finalement l'union des listes $W(a)$ pour chaque valeur de a est renvoyée.

Pour une valeur de a donnée, un élément e de $W(a)$ s'écrit $e = e_{01} + e_{23}$ où $e_{01} \in W_{01}(a)$, $e_{23} \in W_{23}(a)$ et vérifie $e_{01} H''^t + e_{23} H''^t = s''$. Chacun de ces éléments vérifie donc $eH''^t = s''$ mais aussi $eH'^t = a + a + s' = s'$. Par contre les mots de W_{01} et de W_{23} ont un support commun, donc les mots de $W(a)$ ne sont pas forcément de poids p. Seuls ceux de poids p sont solutions de $eH^t = s$.

Cet algorithme est une version améliorée de l'algorithme original [49]. En effet, dans la première version de cette variante, l'ensemble A est constitué d'un unique élément. Utiliser un ensemble A plus grand permet de rentabiliser la construction des listes L_i ainsi que l'élimination de Gauss. Plus important, cela permet de selectionner une plus grande valeur pour le paramètre ℓ_2 et donc de réduire la consommation mémoire de l'algorithme. Cela s'explique par l'équilibrage qui a lieu entre les tailles des deux premières listes, L_{01} et L_{23} (proportionelle à $2^{-\ell_2}$), et celle de la dernière liste L (proportionelle à $2^{\ell_2 - \ell}$).

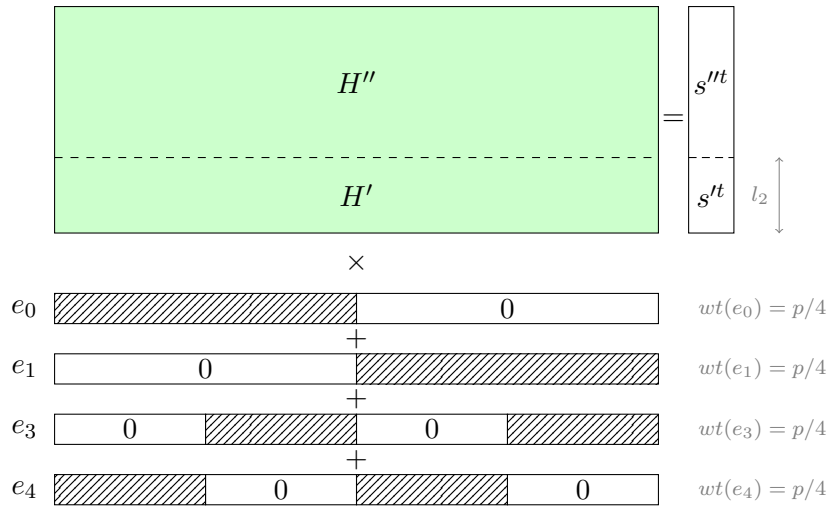


FIGURE 10.3 – Exemple de découpage de la matrice H et du support pour l'algorithme MMT.

11 | Mise en œuvre

11.1 Fusion de liste

Il existe plusieurs stratégies pour réaliser une fusion [6]. Les algorithmes de cette section sont inspirés des algorithmes présentés dans le chapitre 6 de [38]. Ces derniers dédiés à la recherche de collisions dans une liste sont ici adaptés à la recherche de collisions entre deux listes.

11.1.1 Fusion par tri

La fusion par tri consiste à ajouter s à chaque deuxième élément des couples d'une des listes puis à trier les deux listes selon le deuxième élément. Les listes sont ensuite parcourues conjointement jusqu'à trouver deux couples ayant un deuxième élément identique. Ces deux couples permettent de construire un élément à renvoyer. L'algorithme 7, réalise la fusion par tri.

11.1.2 Fusion par indexation

La fusion par indexation commence également par ajouter s à chaque deuxième élément des couples d'une des listes. Ensuite, une des listes est rangée dans une structure de donnée indexée qui à chaque élément $y \in \{0,1\}^c$ associera la liste (potentiellement vide) des éléments $x \in \{0,1\}^b$ tels que (x, y) appartient à cette liste. Cette structure sera interrogée pour chaque élément de l'autre liste. Chaque élément trouvé dans la structure permettra de construire un élément à renvoyer. L'algorithme 8, réalise la fusion par indexation.

11.1.3 Comparaison

La fusion par indexation (algorithme 8 a pour avantage une complexité temporelle en $O(N + M)$ (N pour construire la structure puis M interrogations) ou N et M sont les tailles respectives des listes. Cependant, les accès à la structure de données se font de façon aléatoire, ce qui ne permet pas de profiter des mécanismes de caches disponibles sur les architectures actuelles.

Algorithme 7 Recherche de collisions en triant les listes

Entrée :
Une liste L_0 de N couples $(x, y) \in \{0,1\}^b \times \{0,1\}^c$,
une liste L_1 de M couples $(x, y) \in \{0,1\}^b \times \{0,1\}^c$,
un vecteur $s \in \{0,1\}^c$.

Sortie :
L'ensemble $\{x_0 + x_1 \mid (x_0, y_0) \in L_0, (x_1, y_1) \in L_1, y_0 + y_1 = s\}$.

Fonction FUSION_TRI(L_0, L_1, s)
  $L \leftarrow \emptyset$
  $L_0 \leftarrow$ TRI(L_0), $L_1 \leftarrow$ TRI($L_1 + s$)
  $i \leftarrow 0$, $j \leftarrow 0$
  Tant que ($i < N$ **et** $j < M$)
    Si $L_0[i,1] < L_1[j,1]$
      $i \leftarrow i + 1$
    Sinon Si ($L_0[i,1] > L_1[j,1]$)
      $j \leftarrow j + 1$
    Sinon
      $i_0 \leftarrow i$, $j_0 \leftarrow j$
      Tant que $i < N$ **et** $L_0[i,1] = L_0[i_0,1]$
        $j \leftarrow j_0$
        Tant que $j < M$ **et** $L_1[j,1] = L_1[j_0,1]$
          $j \leftarrow j + 1$
          $L \leftarrow L \cup \{L_0[i,0] + L_1[j,0]\}$
  Retourner L

Commentaires :
$L_1 + s$ désigne la liste $[(x, y + s)\ \forall (x, y) \in L_1]$
La procédure TRI(L) organise les éléments de la liste L dans l'ordre croissant de leur deuxième élément.
$L[i,j]_{j \in \{0,1\}}$ désigne le j-ième élément du i-ième couple de la liste L.

Algorithme 8 Recherche de collisions en indexant une liste

Entrée :
Une liste L_0 de N couples $(x, y) \in \{0,1\}^b \times \{0,1\}^c$,
une liste L_1 de M couples $(x, y) \in \{0,1\}^b \times \{0,1\}^c$,
un vecteur $s \in \{0,1\}^c$.

Sortie :
L'ensemble $\{x_0 + x_1 \mid (x_0, y_0) \in L_0, (x_1, y_1) \in L_1, y_0 + y_1 = s\}$.

Fonction FUSION_INDEX(L_0, L_1, s)
$L \leftarrow \emptyset$
$L_1 \leftarrow L_1 + s$
Allouer une structure indexée T initialisée à $\emptyset$
Pour $(x, y) \in L_0$
$T[y] \leftarrow T[y] \cup \{x\}$
Pour $(x, y) \in L_1$
$L \leftarrow L \cup T[y]$
Retourner L

La fusion par tri (algorithme 7) en revanche a une complexité temporelle en $O(N \log(N) + M \log(M))$ (le coût des deux tris) mais qui, une fois les listes triées, effectue la plupart des accès mémoires de façon séquentielle, ce qui permet de rentabiliser le chargement d'une page mémoire ainsi que de profiter des mécanismes de préchargement de données. De plus, si l'on utilise un algorithme de tri tirant parti des mécanismes de cache (le *radix-sort* par exemple, voir l'algorithme 6.10 de [38]), l'algorithme peut s'exécuter quasi-entièrement en manipulant des données présentes dans les caches mémoires.

La figure 11.1 montre le pourcentage d'accès mémoire qui n'ont pas été satisfait par le cache L1 et le cache L2 lors de la fusion de deux listes de 2^ℓ éléments aléatoires de $[\![0, 2^\ell[\![$. Ces valeurs ont été mesurées en utilisant l'outil cachegrind de la suite valgrind [55] sur une machine disposant de 32Ko de cache L1 par cœur et de 256Ko de cache L2 par cœur.

La figure 11.2 montre le nombre de collisions par microsecondes (mesuré en utilisant gettimeofday()) générées par les algorithmes 7 et 8 lors de la fusion de deux listes de 2^ℓ éléments aléatoires de $[\![0, 2^\ell[\![$.

On remarque que les mécanismes de cache ne suffisent pas à compenser le facteur logarithmique de l'algorithme 7, ce qui fait de l'algorithme 8 le plus rapide.

Autre avantage de l'algorithme 8, il est possible de l'appliquer quasiment *à la volée* ; il peut démarrer dès que la liste L_0 est complète et se dérouler au fur et à mesure que la liste L_1 est construite. Cela permet d'économiser en mémoire puisqu'il n'y a pas besoin de conserver la liste L_1.

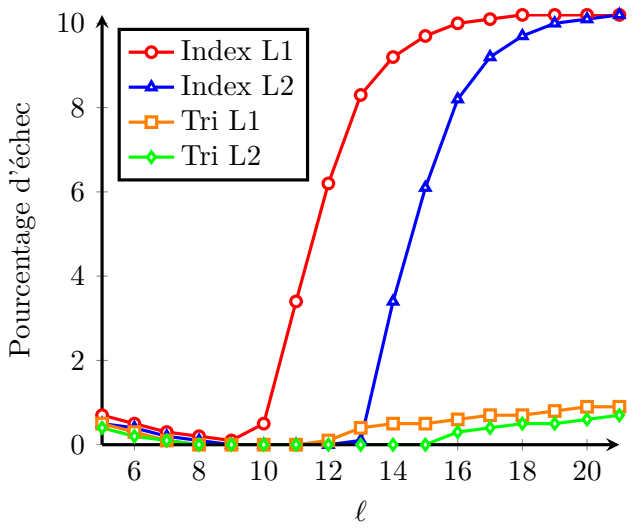


FIGURE 11.1 – Pourcentage d'échec d'accès mémoire dans les différentes mémoires caches lors de 100 fusions de deux listes de 2^ℓ éléments

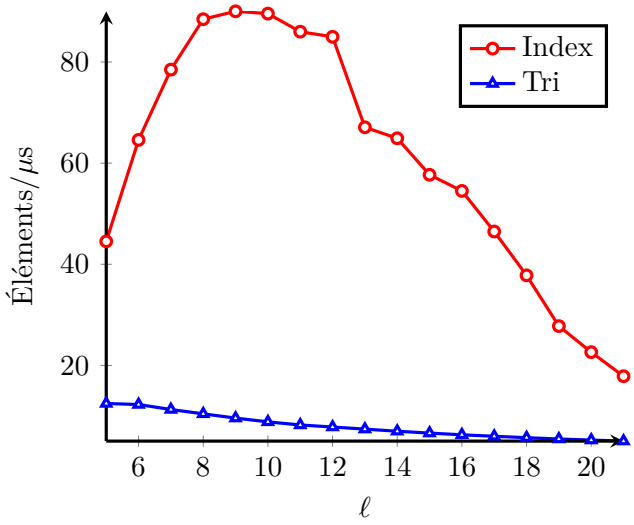


FIGURE 11.2 – Nombre de collisions générées par microsecondes lors de 100 fusions de deux listes de 2^ℓ éléments de ℓ bits.

11.2 Analyse de la complexité et estimation des paramètres

Pour cette analyse, nous ferons ces hypothèses :

Hypothèse 1. *Nous appliquons l'algorithme à un problème d'ordre cryptographique ; c'est-à-dire que la probabilité pour un mot de longueur n et de poids p tiré aléatoirement d'être solution du problème est très inférieur à 1.*

Hypothèse 2. *Le problème ne possède qu'une solution (ou bien on cherche une solution spécifique dans l'ensemble des solutions).*

Hypothèse 3. *La valeur $1-(1-p)^N$ où $p \ll 1$ et N est un entier positif peut être approximée par $\min(1, pN)$ (voir figure 11.3).*

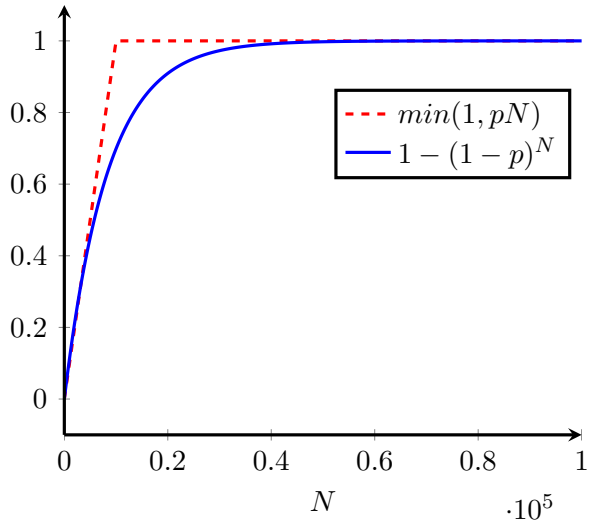


FIGURE 11.3 – Approximation de $1-(1-p)^N$ par $\min(1, pN)$

.

Le coût moyen de l'algorithme 4 est

$$\mathrm{WF} = \frac{1}{P}(K_G + K_S + \mu K_F)$$

où :

- P est la probabilité de succès d'une itération,
- K_{G} est le coût de la permutation et de l'élimination de Gauss,
- K_{S} est le coût d'une itération de l'algorithme SUBISD,
- K_{F} est le coût d'un test final,
- μ est le nombre moyen de candidats produits par l'algorithme SUBISD.

Afin de mesurer les coûts des diverses étapes nous utiliserons comme unité de base une opération de colonne, définie comme étant une lecture ou écriture d'une colonne en mémoire accompagnée d'une addition de deux colonnes ou d'un test de poids d'une colonne.

Lors du calcul de tous les éléments eM où M est une matrice binaire et où e prend successivement toutes les valeurs de l'ensemble des mots de support et de poids donné, le coût pour construire un de ces éléments sera ramené à celui d'une addition. En effet, en conservant les sommes partielles, il est possible de construire un nouvel élément eM en une seule addition ; la somme de p colonnes étant calculée en ajoutant une colonne à la somme de $p-1$ colonnes.

Il est possible de ramener le coût d'un test final à celui d'une lecture en mémoire et d'un test de poids (voir la section 11.3), c'est-à-dire à 1 selon notre unité de base.

Dans le cadre de l'hypothèse 1, le coût du pivot de Gauss partiel (coût $O(r^3)$) s'efface face au reste de l'algorithme (coût exponentiel en p). Le paramètre p grandissant avec la taille du problème considéré, nous pourrons ignorer ce coût lors des calculs. On constate en pratique que cette hypothèse se confirme rapidement lorsque la difficulté du problème croît.

Il est évident qu'une itération d'un algorithme SubISD pourrait se terminer dès que la liste des solutions à renvoyer est non vide. Cependant ce gain potentiel est négligeable vis-à-vis du nombre d'itérations nécessaires pour résoudre un problème d'ordre cryptographique. Nous ignorerons cet aspect pour l'analyse afin d'alléger les formules de complexité.

11.2.1 Probabilité de succès d'une itération

Lemme 1. *Soit X le nombre d'éléments distincts après n tirages uniformes avec remise dans un ensemble de taille M. L'espérance de X est :*

$$\mathbb{E}(X) = M\left(1 - \left(1 - \frac{1}{M}\right)^n\right)$$

Démonstration. Soient $A = \{a_1, \ldots, a_M\}$ et B l'ensemble des éléments distincts obtenus après n tirages uniformes avec remise dans A.

$$\text{Soit } y_i = \begin{cases} 1 \text{ si } a_i \in B \\ 0 \text{ sinon} \end{cases}$$

$$
\begin{aligned}
X &= \sum_{i=1}^{M} y_i \\
\mathbb{E}(X) &= \sum_{i=1}^{M} \mathbb{E}(y_i) \\
\mathbb{E}(y_i) &= \mathcal{P}(a_i \in B) = 1 - \mathcal{P}(a_i \notin B) = 1 - \left(\frac{M-1}{M}\right)^n \\
\mathbb{E}(X) &= M\left(1 - \left(1 - \frac{1}{M}\right)^n\right)
\end{aligned}
$$

□

La probabilité $\mathcal{P}$ de succès d'une itération dépend de la variante utilisée et du nombre de candidats distincts μ_d renvoyés par l'algorithme SubISD ainsi que de la probabilité qu'a un de ces candidats d'être solution du problème.

$$
\begin{aligned}
\mathcal{P} &= 1 - (1 - 2^\ell \mathcal{E}(p,\ell))^{\mu_d} \\
&\approx 2^\ell \mathcal{E}(p,\ell)\mu_d \\
\text{où } \mathcal{E}(p,\ell) &= \frac{\binom{r-\ell}{w-p}}{2^r\left(1 - (1 - \frac{1}{2^{r-\ell}})^{\binom{n}{w}}\right)} \\
&\approx \frac{\binom{r-\ell}{w-p}}{\min(2^r, 2^l\binom{n}{w})}
\end{aligned}
$$

Démonstration. Soit $W_{i,j}$ l'ensemble des mots binaires de longueur i et de poids j. Soient H, H', s et s' tels que représentés figure 10.1

La probabilité $\mathcal{P}$ est la probabilité qu'au moins un des μ_d candidats générés par l'algorithme SubISD passe le test final. Le vecteur s' est tiré dans $\mathcal{U} = \{e(I|H')^t \mid e \in W_{n,w}\}$ donc un candidat e passe le test final avec probabilité

$$
\mathcal{P}_f = \mathcal{P}(\text{Poids}(eH'^t + s') = w - p) = \frac{\binom{r-\ell}{w-p}}{\mathbb{E}(|\mathcal{U}|)}.
$$

En effet, tirer un vecteur e dans $W_{k+\ell,p}$ revient à tirer simultanément $\binom{r-\ell}{w-p}$ éléments de $\mathcal{U}$ car il existe $\binom{r-\ell}{w-p}$ vecteurs e' dans $W_{r-\ell,w-p}$ et que chacun d'eux nous permet de construire $(e'|e)(I|H')^t$, élément de $\mathcal{U}$.

L'ensemble $\mathcal{U}$ est l'ensemble des mots obtenus en multipliant chaque mot de $W_{n,w}$ par $(I|H')^t$. Si l'on suppose les mots $e(I|H')^t$ où $e \in W_{n,w}$ indépendants, chaque élément de $\mathcal{U}$ peut être vu comme le résultat d'un

tirage uniforme dans $\{0,1\}^{r-\ell}$. Le lemme 1 nous donne donc

$$\mathbb{E}(|\mathcal{U}|) = 2^{r-\ell}\left(1-(1-\frac{1}{2^{r-\ell}})^{\binom{n}{w}}\right)$$

Si l'on note $\mathcal{E}(p,\ell)$ la probabilité pour un mot e quelconque de longueur $k+\ell$ et de poids p d'être solution du problème, c'est-à-dire de vérifier $eH^t = s$ et de passer le test final. Ces deux évènements étant indépendants, nous avons donc

$$\mathcal{E}(p,\ell) = \mathcal{P}(eH^t = s)\mathcal{P}_f = \frac{1}{2^\ell}\mathcal{P}_f$$

Finalement

$$\begin{aligned}\mathcal{P} &= 1-(1-\mathcal{P}_f)^{\mu_d} = 1-(1-2^\ell\mathcal{E}(p,\ell))^{\mu_d}\\ &\approx 2^l\mathcal{E}(p,\ell)\mu_d\\ \text{et } \mathcal{E}(p,\ell) = \frac{\mathcal{P}_f}{2^\ell} &= \frac{\binom{r-\ell}{w-p}}{2^r\left(1-(1-\frac{1}{2^{r-\ell}})^{\binom{n}{w}}\right)}\\ &\approx \frac{\binom{r-\ell}{w-p}}{\min(2^r, 2^l\binom{n}{w})}\end{aligned}$$

□

11.2.2 Coût des algorithmes SubISD

Pour calculer le coût d'un algorithme SubISD, nous réécrivons ceux-ci en déroulant les appels à la fonction Fusion (en utilisant une fusion par indexation) puis nous comptons le nombre d'exécutions de chaque instruction effectuant une opération de colonne, notre opération de base qui, pour rappel, est définie comme étant une lecture ou écriture d'une colonne en mémoire accompagnée d'une addition de deux colonnes ou d'un test de poids d'une colonne.

Stern/Dumer

Pour calculer le coût de l'algorithme 9, nous allons compter le nombre d'exécutions des instructions (1), (2) et (3) puis sommer ces nombres. Le nombre d'éléments renvoyés par l'algorithme sera égal au nombre d'exécutions de l'instruction (3).

(1)	$\lvert W_0\rvert$
(2)	$\lvert W_1\rvert$
(3)	$\lvert W_1\rvert\mathbb{E}(\lvert\mathscr{C}\rvert)$

Algorithme 9 Stern/Dumer

Fonction SUBISD$_{\text{SD}}(H, s, p, S)$
  $(W_0, W_1) \leftarrow$ INITSD(p, S)
  $T \leftarrow$ INITSTRUCT()
  $L \leftarrow \emptyset$
  Pour $e_0 \in W_0$
    STOCKER$(T, e_0 H^t, e_0)$ (1)
  Pour $e_1 \in W_1$
    $\mathscr{E} \leftarrow$ LIRE$(T, s - e_1 H^t)$ (2)
    Pour $e_0 \in \mathscr{E}$
      $L \leftarrow L \cup \{e_0 + e_1\}$ (3)
  Retourner L

Commentaires :
INITSD(S, p) initialise la liste des mots de poids p et de support S et son complémentaire. Voir section 10.0.3.
INITSTRUCT() initialise une structure associant à un mot une liste de vecteur.
STOQUER(T, i, v) ajoute le vecteur v dans la liste associée au mot i dans la structure T.
LIRE(T, i) renvoie la liste de vecteurs associée au mot i dans la structure T.

Les instructions (1) et (2) effectuent chacune une opération de colonne. En effet, chaque produit eH^t est une somme de $p/2$ colonnes de H qui est presque tout le temps partiellement calculé par l'itération de la boucle précédente.

Le nombre d'exécutions de l'instruction (3) dépend du nombre d'éléments présents dans l'ensemble $\mathscr{E}$. Si l'on suppose que les éléments e_0, insérés dans T lors de l'instruction (1), y sont répartis uniformément, l'espérance du cardinal de $\mathscr{E}$ vaut $\frac{|W_0|}{2^\ell}$; il s'agit du nombre d'éléments insérés dans T multiplié par la probabilité pour qu'un élément donné se trouve à l'emplacement indexé $s - e_1 H^t$. Le nombre d'insertions dans T est donné par le nombre d'exécutions de l'instruction (1).

Si l'on se place dans le cas idéal et que l'on pose $L = |W_0| = |W_1| = \binom{(k+\ell)/2}{p/2}$, le coût moyen de l'algorithme est donc :

$$K_{SD} = |W_0| + |W_1| + \frac{|W_0||W_1|}{2^\ell} = 2L + \frac{L^2}{2^\ell}$$

Le nombre moyen d'éléments renvoyés est :

$$\mu = \frac{|W_0||W_1|}{2^\ell} = \frac{L^2}{2^\ell}$$

Certaines variantes peuvent renvoyer plusieurs fois un même élément mais cette variante considère des mots de supports disjoints. Nous avons donc $\mu_d = \mu$ et, si l'on néglige le coût de l'élimination de Gauss, le facteur de travail est

$$\mathrm{WF}_{\mathrm{SD}} = \frac{1}{\mathcal{P}}\left(2L + 2\frac{L^2}{2^\ell}\right)$$

En approximant que ce facteur de travail est minimum lorsque les deux termes de la somme sont égaux, nous obtenons un facteur de travail minimal lorsque $\ell \approx \log_2(L)$. Cette approximation peut faire perdre un facteur proche de 2, si les contraintes mémoires le permettent, il faudra choisir ℓ légèrement supérieur à cette valeur (voir figure 11.4).

Pour conclure, nous avons

$$\mathrm{WF}_{\mathrm{SD}}(p) \approx \frac{4L^2}{\mathcal{P}2^\ell} = \frac{c}{\mathcal{E}(p,\ell)2^\ell}$$

où c est une constante ; p sera choisi pour minimiser cette formule et ℓ sera choisi légèrement supérieur à $\log_2(\binom{(k+\ell)/2}{p/2})$.

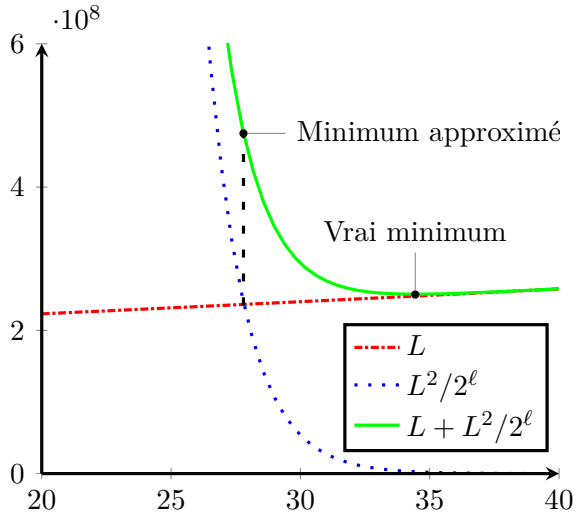


FIGURE 11.4 – Approximation de $\min(L + \frac{L^2}{2^\ell})$. Ici $k = 524, p = 8$

May, Meurer et Thomae

Comme précédemment, pour calculer le coût de l'algorithme 10, nous allons compter le nombre d'exécutions des instructions (1) à (7) puis sommer ces nombres. Le nombre d'éléments renvoyés par l'algorithme sera égal au nombre d'exécutions de l'instruction (7).

Algorithme 10 MMT

Fonction $\text{SUBISD}_{\text{MMT}}(H, s, p, \ell_2, S_0, S_2, A)$
  $(H', H'', s', s'', W_0, W_1, W_2, W_3) \leftarrow \text{INITMMT}(H, s, p, \ell_2, S_0, S_2)$
  $T_0 \leftarrow \text{INITSTRUCT}(); T_2 \leftarrow \text{INITSTRUCT}()$
  $L \leftarrow \emptyset$
  Pour $e_0 \in W_0$
    $\text{STOCKER}(T_0, e_0 H^t, e_0)$ (1)
  Pour $e_2 \in W_2$
    $\text{STOCKER}(T_2, e_2 H^t, e_2)$ (2)
  Pour $a \in A$
    $T_{01} \leftarrow \text{INITSTRUCT}()$
    Pour $e_1 \in W_1$
      $\mathscr{E}_0 \leftarrow \text{LIRE}(T_0, e_1 H^t + a)$ (3)
      Pour $e_0 \in \mathscr{E}_0$
        $\text{STOCKER}(T_{01}, (e_0 + e_1) H'^t, e_0 + e_1)$ (4)
    Pour $e_3 \in W_3$
      $\mathscr{E}_2 \leftarrow \text{LIRE}(T_2, s - e_3 H^t - a)$ (5)
      Pour $e_2 \in \mathscr{E}_2$
        $\mathscr{E}_{01} \leftarrow \text{LIRE}(T_{01}, s' - (e_2 + e_3) H'^t)$ (6)
        Pour $e_{01} \in \mathscr{E}_{01}$
          $L \leftarrow L \cup \{e_{01} + e_2 + e_3\}$ (7)
  Retourner L

(1)	$\lvert W_0 \rvert$
(2)	$\lvert W_2 \rvert$
(3)	$\lvert A \rvert \lvert W_1 \rvert$
(4)	$\lvert A \rvert \lvert W_1 \rvert \mathbb{E}(\lvert \mathscr{E}_0 \rvert)$
(5)	$\lvert A \rvert \lvert W_3 \rvert$
(6)	$\lvert A \rvert \lvert W_3 \rvert \mathbb{E}(\lvert \mathscr{E}_2 \rvert)$
(7)	$\lvert A \rvert \lvert W_3 \rvert \mathbb{E}(\lvert \mathscr{A}_2 \rvert) \mathbb{E}(\lvert \mathscr{E}_{01} \rvert)$

En supposant que les éléments insérés dans T_0, T_2 et T_{01} y sont insérés uniformément (ce qui est naturel si H est aléatoire), nous obtenons les espérances des ensembles $\mathscr{E}_0$, $\mathscr{E}_1$ et $\mathscr{E}_{01}$ en multipliant le nombre d'insertions dans chaque structure par la probabilité pour qu'un élément donné se trouve dans un emplacement donné. Le nombre d'insertions dans T_0 et T_2 est donné respectivement par le nombre d'exécutions des instructions (1) et (2). T_{01} étant réinitialisée à chaque nouvelle valeur de a, le nombre d'insertions dans T_{01} est égal au nombre d'exécutions de l'instruction (4) pour une valeur de

a donnée ; soit $|W_1|\mathbb{E}(|\mathscr{E}_0|)$.

$$\mathbb{E}(|\mathscr{E}_0|) = \frac{|W_0|}{2^{\ell_2}} \qquad \mathbb{E}(|\mathscr{E}_2|) = \frac{|W_2|}{2^{\ell_2}}$$

$$\mathbb{E}(|\mathscr{E}_{01}|) = \frac{|W_1|\mathbb{E}(|\mathscr{E}_0|)}{2^{\ell-\ell_2}} = \frac{|W_0||W_1|}{2^{\ell}}$$

Si l'on pose $L = |W_0| = |W_1| = |W_2| = |W_3| = \binom{(k+\ell)/2}{p/4}$, le coût moyen de l'algorithme est donc :

$$K_{SubISD} = |W_0| + |W_2| + \\ |A|\left(|W_1| + \frac{|W_0||W_1|}{2^{\ell_2}} + |W_3| + \frac{|W_2||W_3|}{2^{\ell_2}} + \frac{|W_0||W_1||W_2||W_3|}{2^{\ell+\ell_2}}\right)$$

$$K_{SubISD} = 2L + |A|\left(2L + 2\frac{L^2}{2^{\ell_2}} + \frac{L^4}{2^{\ell+\ell_2}}\right)$$

Le nombre moyen d'éléments renvoyés est :

$$\mu = |A||W_3|\mathbb{E}(|\mathscr{E}_2|)\mathbb{E}(|\mathscr{E}_{01}|) = |A|\frac{L^4}{2^{\ell+l2}}$$

Il faut sélectionner A tel que $|A| \ll 2^{\ell_2}$; ainsi le nombre de doubles (candidats générés plusieurs fois) est limité et on peut faire l'hypothèse $\mu_d = \mu$. On obtient alors pour le facteur de travail de l'algorithme

$$\mathrm{WF}_{\mathrm{MMT}} = \frac{1}{\mathcal{P}}\left(2L + |A|\left(2L + \frac{2L^2}{2^{\ell_2}} + \frac{L^4}{2^{\ell+\ell_2}}\right) + \frac{|A|L^4}{2^{\ell+\ell_2}}\right)$$

Il faut par contre choisir A suffisamment grand pour rentabiliser la construction des structures T_0 et T_2 (instructions (1) et (2)) ainsi que l'élimination de Gauss. Leurs coûts peuvent alors être négligés. Choisir ℓ_2 suffisamment petit par rapport à $\log_2(L)$, permet d'obtenir des ensembles $\mathscr{E}_0$ et $\mathscr{E}_2$ non vides, de rentabiliser les interrogations aux structures T_0 et T_2 (instructions (3) et (5)) et de négliger leurs coûts dans la formule. Celle-ci devient :

$$\mathrm{WF}_{\mathrm{MMT}} = \frac{2|A|}{\mathcal{P}}\left(\frac{L^2}{2^{\ell_2}} + \frac{L^4}{2^{\ell+\ell_2}}\right)$$

En approximant (comme pour le facteur de travail de l'algorithme Stern/Dumer) que le minimum est atteint lorsque les deux termes de la sommes sont égaux

nous obtenons que le facteur de travail est minimum lorsque $\ell \approx \log_2(L^2)$. De même que pour $\mathrm{WF}_{\mathrm{SD}}$ il faudra choisir ℓ légèrement supérieur à cette valeur pour se rapprocher de la vraie valeur minimum.

Pour résumer, le facteur de travail est

$$\mathrm{WF}_{\mathrm{MMT}}(p) \approx \frac{4|A|L^4}{\mathcal{P}2^{\ell+\ell_2}} = \frac{c}{\mathcal{E}(p,\ell)2^{\ell}}$$

où c est une constante ; p sera choisi pour minimiser cette formule et ℓ sera choisi légèrement supérieur à $\log_2(\binom{(k+\ell)/2}{p/4}^2)$. Il est intéressant de remarquer que si A et ℓ_2 sont choisis tels que proposés précédemment, leurs valeurs ne rentrent pas en compte dans le calcul du facteur de travail.

La variante MMT est supérieure à la version Stern/Dumer car même si l'expression des facteurs de travail est similaire, la valeur optimale du paramètre ℓ est supérieure pour l'algorithme MMT.

11.3 Optimisations

Cette section utilise les notations de la figure 11.5.

Critère d'abandon prématuré d'un candidat

L'idée d'abandon prématuré d'un candidat fut évoquée dans [45] puis utilisée dans [15]. Lorsqu'un candidat est fourni par l'algorithme SubISD, il faut tester ce candidat sur le reste de la matrice H_0. Cependant, il n'est pas nécessaire de calculer la somme correspondante à ce candidat sur toute sa longueur ; dès que le poids de la somme dépasse $w-p$, on peut considérer le candidat non-valide. Si l'on considère que tous les candidats non-valides vont produire un mot aléatoire de poids moyen $(r-\ell)/2$, un candidat sera éliminé en moyenne après calcul de $2(w-p)$ bits.

De manière générale, si on élimine tous les candidats donnant un mot de poids supérieur à t après avoir calculé d bits de la somme correspondante, la probabilité d'éliminer un candidat qui est la solution est :

$$\mathcal{P}_{\mathrm{m}}(d,t) = 1 - \sum_{i=0}^{t} \frac{\binom{d}{i}\binom{r-d-\ell}{w-p-i}}{\binom{r-\ell}{w-p}}$$

Dans ce même contexte, la probabilité de fausse alarme, c'est-à-dire la probabilité qu'un mot aléatoire (correspondant donc à un mot non-valide) ne soit pas éliminé à ce stade est :

$$\mathcal{P}_{\mathrm{fa}}(d,t) = \sum_{i=0}^{t} \frac{\binom{d}{i}}{2^d}$$

Calcul partiel de la forme échelonnée partielle

Puisqu'il est possible d'éliminer un candidat sans effectuer le test final complet, il n'est pas nécessaire de calculer la forme échelonnée partielle en entier lors de l'élimination de Gauss. Si lors de cette étape, on ne calcule que d lignes de la partie supérieure (pour former la matrice H' ici) et que l'on fixe un seuil t au-delà duquel on considère un mot non-valide, le test final d'un candidat e revient au calcul du poids de Hamming de eH'^t puis si ce poids est inférieur à t, il faut calculer le poids de eH''^t. Ce dernier calcul nécessite de calculer au moins partiellement H'' mais sera effectué avec probabilité $\mathcal{P}_{\mathrm{fa}}(d,t)$, ce qui permet de le rendre aussi négligeable que souhaité. En revanche, il devient possible de manquer la solution ce qui multiplie le nombre de candidats moyen nécessaire, et donc le coût total de l'algorithme, par $\frac{1}{1-\mathcal{P}_{\mathrm{m}}(d,t)}$.

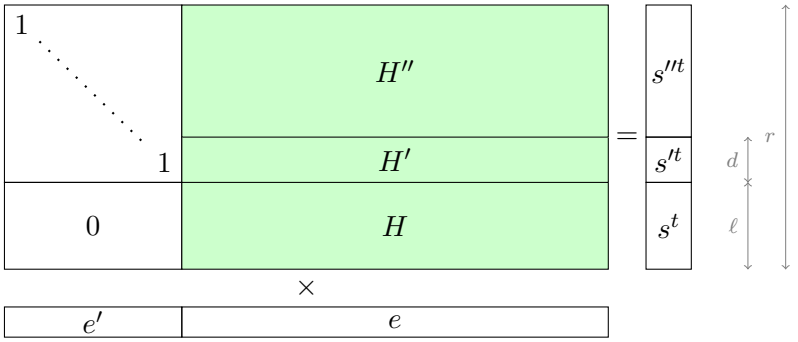


FIGURE 11.5 – Nouveau découpage de la matrice H_0 après élimination de Gauss. Seules H et H' sont calculées initialement.

L'algorithme 11 est une version simplifiée de l'algorithme 4 intégrant ces deux optimisations.

Extension et mémorisation des sommes de colonnes

Les algorithmes SUBISD calculent des sommes de colonnes de H. En pratique ℓ est souvent inférieur à la taille d'un mot machine (qui font typiquement 64 bits aujourd'hui). Il en résulte donc que calculer une somme de colonnes de H n'utilise pas pleinement la capacité d'une unité de calcul. Afin de ne pas gaspiller ces bits sommés, il est possible de choisir d (figure 11.5) tel que $d + \ell$ soit égale à la taille d'un mot machine et de faire en sorte que l'algorithme SUBISD fasse les sommes sur les colonnes de la matrice composée des H et H' empilées. Les d bits supplémentaires du résultat correspondants à la matrice H' seront stockés pour être réutilisés lors du calcul de eH'^t (étape (1) de l'algorithme 11).

Une fois d fixé, le seuil t sera choisi afin de minimiser le coût de l'algorithme. En pratique, il est simple d'exécuter le programme sur quelques

Algorithme 11 Information Set Decoding avec abandon prématuré

Entrée :
Voir l'algorithme 4
un entier $0 < d < r - \ell$,
un entier $0 \leq t \leq d$.

Sortie :
Voir l'algorithme 4

Fonction ISD_EarlyAbort($H_0, s_0, w, p, \ell, d, t$)
  Boucle
    $(H, H', H'', s, s', s'', P) \leftarrow$ InitISD(H_0, ℓ, s_0) ▷ voir figure 11.5
    Pour $e \in$ SubISD(H, s, p)
      $w' \leftarrow$ Poids($eH'^t + s'$) (1)
      Si $w' < t$
        Si Poids($eH''^t + s''$) $+ w' = w - p$
          Retourner $(eH''^t | eH'^t | e)P^{-1}$

itérations pour mesurer les coûts des différentes étapes puis de minimiser le coût total de l'algorithme en faisant varier t sur l'intervalle $[\![1, \min(d, w-p)]\!]$.

Sacrifice de candidats

Lors d'une fusion de listes par indexation (voir section 11.1.2), il peut être coûteux de résoudre les collisions dans la structure de données. Une collision dans la structure se produit lorsque un élément doit être rangé à un indice déjà occupé. Il existe est possible de gérer ces collisions en chaînant les éléments ou en utilisant un adressage ouvert, c'est-à-dire l'utilisation d'une méthode de sondage qui trouvera un nouvel emplacement pour l'élément. Ces méthodes ont l'inconvénient d'imposer l'ajout de contrôle et des accès mémoires potentiellement aléatoires lors de l'insertion ou la consultation de la structure. Il est cependant possible d'ignorer ces collisions et de simplement conserver le premier ou dernier élément que l'on voudra ranger à un indice donné. Cela permet d'éviter ces inconvénients mais toutes les collisions entre les deux listes ne sont plus trouvées. Du point de vue des algorithmes SubISD, tous les candidats potentiels ne sont plus générés donc le nombre d'itérations de l'algorithme global augmente mais cela réduit le coût d'une itération. La figure 11.6 compare le nombre d'éléments générés par microsecondes lors de 100 fusions de deux listes de 2^ℓ éléments aléatoires de ℓ bits dans le cas où les collisions dans la structure de données sont gérées (l'équivalent d'une table de hachage où les collisions sont résolues via chaînage) et celui où elles ne le sont pas (un simple tableau).

On pourrait penser que cette optimisation pourrait se dégrader avec le rapport (taille de la liste à indexer)/(nombre de bits de chaque élément)

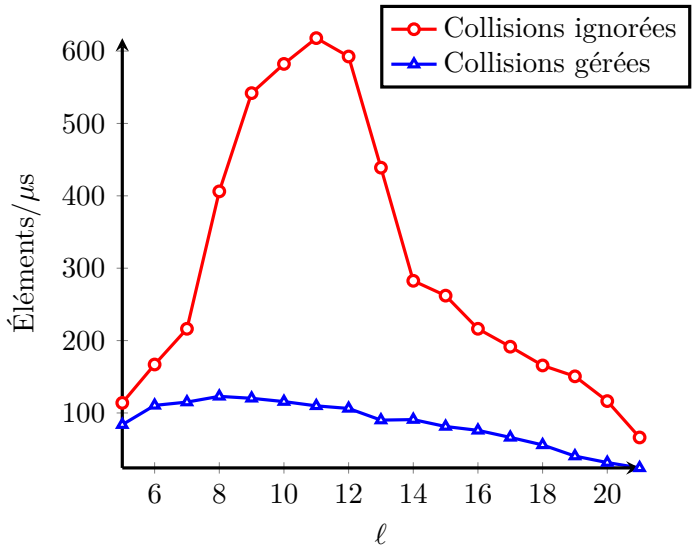


FIGURE 11.6 – Nombre d'éléments générés par µs lors de 100 fusions de deux listes de 2^ℓ éléments aléatoires de ℓ bits.

de par l'augmentation du nombre d'éléments devant être rangé à un même indice. En pratique on constate la tendance inverse. La figure 11.7 est équivalente à la figure 11.6 mais fait varier la taille des listes en fixant la taille des éléments (11 bits ici).

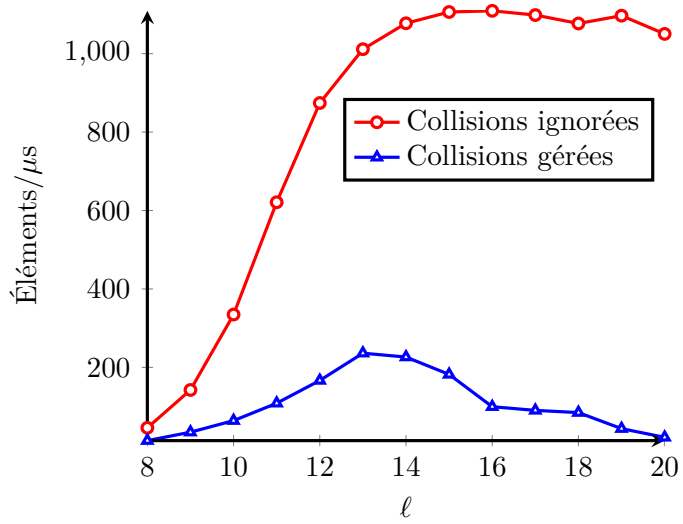


FIGURE 11.7 – Nombre d'éléments générés par µs lors de 100 fusions de deux listes de 2^ℓ éléments aléatoires de 11 bits.

Parcourir les mots de poids constants

Toutes les variantes de l'algorithme 4 nécessitent de parcourir tout ou partie des mots binaires de longueur et de poids donnés et de les multiplier par une matrice. Il est possible de faire cela de deux façons différentes. Supposons que l'on veuille parcourir tous les mots de poids p de longueur n et les multiplier par une matrice $H \in mats\ell n$. La première consiste à écrire une fonction qui à partir d'un mot de poids p calcule le mot suivant selon une relation d'ordre prédéfinie. Il reste ensuite à multiplier le mot par H et à traiter le résultat avant d'itérer. Il est préférable que la relation d'ordre minimise la distance de Hamming entre deux mots successifs afin de rentabiliser le calcul des sommes partielles. L'algorithme 12 est un exemple d'une telle fonction.

Algorithme 12 Parcours de mots de poids constant

Entrée :

Un entier p,
un entier n,
un mot binaire t de poids p sous forme d'un tableau de p entiers $\in [\![0, n[\![$. Ces entiers représentent les positions non nulles de t. Les éléments de t vérifient $t[i] < t[i+1]\ \forall i \in [\![0, p-1[\![$.

Sortie :

Le mot binaire u suivant t selon la relation d'ordre, ou t si t n'a pas de successeur,
le plus grand i tel que $u[i] \neq t[i]$, ou -1 si t n'a pas de successeur.

Fonction SUIVANT(t, p, n)
  $u \leftarrow t$
  $i \leftarrow p - 1$
  Tant que $i \geq 0$ **et** $t[i] \geq n - p + i$
    $i \leftarrow i - 1$
  Si $i < 0$ **Retourner** $u, -1$
  Pour $j \leftarrow i + 1, p$
    $u[j] \leftarrow u[j-1] + 1$
  Retourner u, i

Commentaires :

Retourner i indique à une fonction appelante à partir de quel indice le tableau de sortie diffère du tableau d'entrée.

La deuxième approche consiste à imbriquer p boucles de tel façon que le p-uplet formé des indices de chaque boucle parcoure l'ensemble des p-uplets d'entiers appartenant à $[\![0, n[\![$. Cette approche n'est pas flexible car elle impose d'utiliser une procédure de génération de code mais elle est plus rapide.

En pratique, les algorithmes SubISD calculent un tel produit puis stockent le résultat dans une structure de données à un emplacement indexé sur ℓ bits. De ce fait, lorsque ce paramètre croît, le nombre d'echec d'accès au cache L1 (puis L2) augmente. Cela rend le calcul du produit négligeable en comparaison du coût des accès mémoire et donc amenuise le bénéfice de cette optimisation.

Pistes à explorer

Lors d'une fusion de grandes listes par indexation (voir section 11.1.2), les accès aléatoires en mémoire sont ce qui coûte le plus cher. Il est cependant possible de réduire la taille de la structure à interroger. En effet, il est possible de commencer par interroger une structure de taille plus petite qui nous dira s'il y a ou non un élément rangé à cet emplacement dans la structure principale. Ce comportement, ressemblant à celui d'une mémoire cache, peut être mis en place via l'utilisation d'un tableau d'octet (rapide à interroger), un *bit field* (plus lent mais plus compacte) ou un filtre de *Bloom* (encore plus lent et plus compacte mais introduit une probabilité de faux positifs) [21].

Autre moyen de limiter les échecs d'accès à la mémoire cache, l'algorithme 6.15 de [38] est un algorithme réalisant une recherche de collisions dans une liste en utilisant des piles. Celui-ci répartit les éléments dans des piles de taille inférieure à la liste initiale puis effectue une recherche de collision dans chaque pile. La mise en œuvre de cette technique adaptée à la fusion de liste est en cours d'étude.

Comportement que nous n'avons pas su exploiter, si l'on modifie l'algorithme 9 et que l'on insère un tri de la matrice H (soit sur sa globalité soit sur les deux matrices issues de la restriction de H aux supports S et $\bar{S}$) on observe un motif dans l'évolution de la valeur $e_0 H^t$, c'est-à-dire de l'index où sera rangé l'élément e_0 dans la structure T. Cette évolution est représentée figure 11.8. Ce comportement peut améliorer la localité spatiale des données puisque l'emplacement où sera écrite la donnée de l'itération suivante est la plupart du temps proche (ou du moins semble prévisible) de l'emplacement où est écrite la donnée de l'itération en cours. Ce comportement se produit également dans la deuxième boucle, cette fois sur l'indice où est interrogé la structure T ; la remarque précédente s'applique donc cette fois sur les données lues.

Ce potentiel gain de localité d'accès aux données n'est sûrement pas suffisant pour que les accès mémoires bénéficient des mécanismes de cache. Il est peut-être possible d'aiguiller le mécanisme de préchargement de données pour tirer profit de ce comportement.

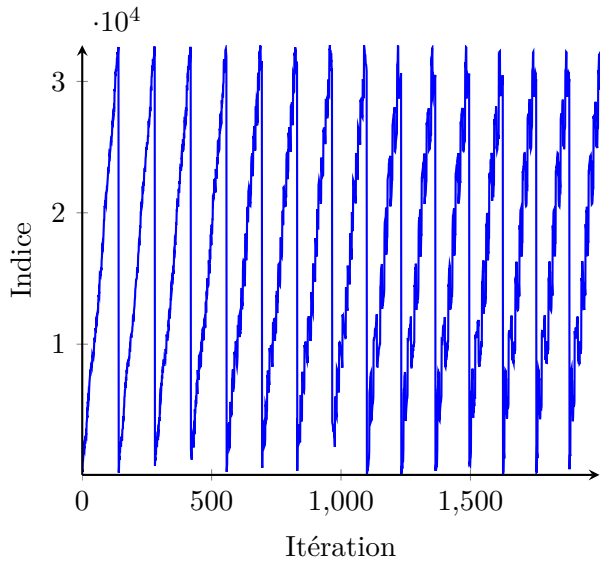


FIGURE 11.8 – Évolution de l'indice mémoire accédé lors des 2000 premières itérations de l'instruction (1) de l'algorithme 9 après tri de H. Ici $H \in \{0,1\}^{15\times 285}$ et $p = 4$

12 | Mise en œuvre logicielle

L'une des contributions de cette thèse est un logiciel mettant un œuvre les variantes de Stern/Dumer et de May, Meurer et Thomae incluant les optimisations décrites section 11.3. Cette section détaille les choix effectués lors de la réalisation de ce logiciel.

Les colonnes de la matrice H sont manipulées via une représentation en colonne , c'est-à-dire qu'une colonne est stockée dans un tableau de r/b mots machines où r est la taille de la colonne et b le nombre de bits que peut contenir un mot machine. En effet, la majorité des opérations effectuées sont des sommes de section de colonnes ; sommer deux colonnes revient donc à sommer les mots machines correspondants. Cela permet aussi d'échanger simplement deux colonnes puisqu'il suffit d'échanger les adresses des tableaux.

La bibliothèque *M4RI* [4] est utilisée pour l'algèbre linéaire. Elle dispose de fonctions de manipulations de matrices binaires telles que le découpage ou la transposition mais surtout met en œuvre la méthode des quatre russes qui permet de gagner un facteur logarithmique sur l'élimination de Gauss.

Lors de la mise sous forme échelonnée partielle de la matrice H_0, la sous-matrice H_{mod}, version étendue de H de hauteur 64, représentée figure 12.1 est calculée [1]. Les colonnes de la sous-matrice H'_{mod} seront calculé plus tard au besoin. Chaque colonne de H_{mod} tient alors dans un mot machine. Les algorithmes SubISD manipulent la matrice H mais plutôt que de tronquer les colonnes de H_{mod} avant de les sommer, leurs sommes sont calculées sur H_{mod} (les 64 bits) puis conservées en mémoire pour usage futur avant d'être tronquées pour l'algorithme de fusion.

Lorsqu'un candidat e (un mot de poids p qui vérifie $eH^t = s$) est trouvé par l'algorithme SubISD, le mot $eH^t_{\text{mod}} + s_{\text{mod}}$ est calculé (en utilisant les sommes conservées en mémoire précédemment) et son poids est evalué. Si ce poids dépasse un certain seuil, alors le candidat est rejeté ; sinon les p colonnes de H'_{mod} correspondantes sont calculées (en utilisant la matrice de passage qui a mis H_0 sous cette forme), sommées puis le poids de cette somme est évalué.

Le seuil appliqué au poids de $eH^t_{\text{mod}} + s_{\text{mod}}$ permet de décider s'il est

1. par conséquence, l'une des contraintes du logiciel est que ℓ ne peut excéder 64.

préférable d'appliquer le test final (calculer le poids de $eH'^t_{\text{mod}} + s'_{\text{mod}}$) ou s'il vaut mieux considérer ce candidat invalide (quitte à manquer la solution).

Si l'on reprend les notations de la section 11.3, le coût de la vérification d'un candidat généré par SubISD revient au coût du calcul du poids de $eH^t_{\text{mod}} + s_{\text{mod}}$ (2 additions et un poids puisque l'on somme deux éléments des listes de SubISD au syndrome) auquel s'ajoute le coût du calcul du poids de $eH'^t_{\text{mod}} + s'_{\text{mod}}$ multiplié par la probabilité d'effectuer ce calcul $\mathcal{P}_{\text{fa}}(64, t)$.

En revanche, le coût total est multiplié par la probabilité de mettre de côté un candidat qui était la vraie solution : $\mathcal{P}_{\text{m}}(64, t)$

Si ℓ n'est pas trop proche de 64, il est possible de positionner le seuil t de telle sorte que $\mathcal{P}_{\text{fa}}(64, t)$ et $\mathcal{P}_{\text{m}}(64, t)$ soit très faible, ce qui permet de pouvoir négliger le coût du test final sans pour autant faire croître enormément le coût global.

Il est donc possible de calculer la valeur de ℓ qui minimise ces deux probabilités mais il est également possible de mesurer l'évolution des performances du programme en faisant varier ce seuil (le programme utilise la valeur heuristique $t = \frac{64-\ell}{4}$ si le seuil ne lui est pas précisé).

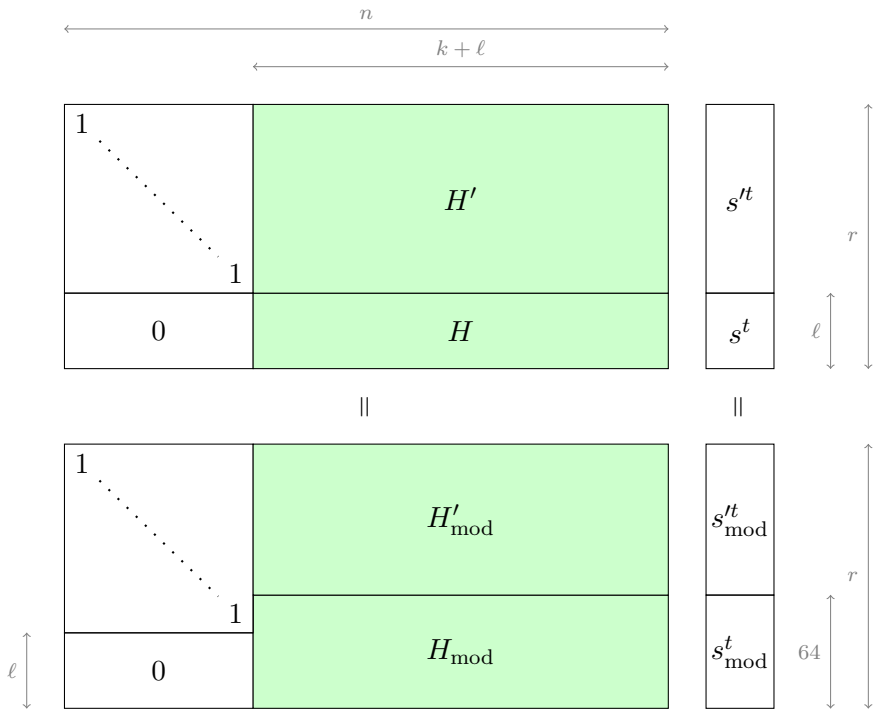


FIGURE 12.1 – Le logiciel calcule H_{mod}, une version étendue de H.

13 | Challenges Wild McEliece

En 2011, Bernstein, Lange et Peters ont mis en ligne une page web [16] regroupant un ensemble de challenges cryptographiques concrets afin d'encourager la communauté des cryptographes à étudier la sécurité des cryptosystèmes basés sur les codes. Ces challenges ont été créés dans le contexte d'un système de McEliece utilisant des codes de Goppa non-binaires [18] mais certains sont basés sur des codes binaires. Les problèmes sont de difficulté croissante et sont des décodages de codes sur des corps finis ayant des alphabets de taille variant entre 2 et 32. Les challenges basés sur des codes binaires sont utilisés afin de mesurer les performances des algorithmes et du logiciel présenté chapitre 12. Les challenges basés sur des codes non-binaires n'ont pas été étudiés.

Les challenges binaires sont au nombre de 61 et sont présentés sous la forme d'une clé publique et d'un chiffré. Il s'agit en réalité de la variante de Niederreiter donc sont donnés une matrice de parité et un syndrome. Les auteurs invitent les challengers à :

- retrouver les textes clairs correspondant aux chiffrés ;
- retrouver les clés secrêtes correspondant aux clés publiques ;
- mesurer les performances des cryptanalyses publiques pour les plus petits challenges afin de permettre à la communauté de constater les améliorations des algorithmes d'attaque ;
- donner des estimations de la difficulté de résolution des plus grands challenges.

Le logiciel a pu résoudre 23 des 61 challenges proposés en utilisant l'algorithme Stern/Dumer avec le paramètre $p = 4$. Le tableau 13.1 répertorie pour chaque challenge résolu le paramètre ℓ utilisé, le nombre de cycles processeur requis pour effectuer une itération de l'algorithme, le nombre théorique d'itération que l'on s'attend à devoir exécuter avant de trouver la solution ainsi que le nombre d'itérations qui ont été réalisées avant de trouver la solution. Le tableau 13.2 donne une estimation du nombre théorique d'itération que l'on s'attend à devoir réaliser avant de trouver la solution.

			Nombre d'itérations	
Challenge	ℓ	Cycles/itération ($\log_2$)	Attendu ($\log_2$)	Mesuré ($\log_2$)
5	15	21.17	13.32	13.30
6	15	21.28	15.10	14.72
7	15	21.53	16.98	16.76
9	15	21.54	17.87	17.76
10	17	22.35	19.51	16.73
11	16	22.29	20.42	19.90
12	16	22.34	21.36	22.24
13	17	22.77	23.26	20.86
14	17	22.80	24.19	22.16
15	17	22.67	25.24	24.13
16	17	23.04	27.07	23.01
17	17	22.97	28.07	25.87
18	17	23.04	29.04	28.69
19	17	23.12	30.00	24.89
20	17	23.16	30.95	30.30
21	17	23.23	31.90	31.74
22	17	23.30	32.83	32.90
23	18	23.55	33.83	30.48
24	18	23.74	35.66	34.67
25	17	23.45	35.73	34.63
26	17	23.57	36.67	27.68
27	18	23.74	37.72	32.21
28	18	23.57	38.67	27.92

TABLE 13.1 – Résultats de l'application du logiciel sur les 23 premiers challenges. Ajouter la colonne *Cycles/itération* et une colonne *Nombre d'itérations* donne le $\log_2$ de l'effort de travail attendu/mesuré

			Nombre d'itérations
Challenge	ℓ	Cycles/itération ($\log_2$)	Attendu ($\log_2$)
32	17	23.70	38.58
34	19	24.32	39.16
36	19	24.36	40.18
38	19	24.35	42.09
40	19	24.38	43.16
42	20	25.05	45.06
44	19	24.42	46.08
46	19	24.54	47.02
48	19	24.75	48.88
50	19	24.75	49.90
52	19	24.75	50.92
54	20	25.36	52.80
56	20	25.25	53.83
58	19	25.09	54.75
60	19	25.04	55.71
62	20	25.24	56.79
64	20	25.48	58.58
68	20	25.41	60.66
72	20	25.47	62.63
76	20	25.43	64.60
80	20	25.92	66.39
84	21	26.42	69.35
88	21	26.43	71.33
92	20	25.92	72.34
96	20	26.09	74.32
100	20	26.02	76.25
104	20	26.01	77.27
108	20	25.91	79.23
112	20	25.86	80.25
128	20	25.77	80.27
136	21	26.94	82.81
144	22	27.44	85.75
152	22	27.58	88.66
160	21	27.16	90.65
168	22	27.64	93.58
176	22	27.59	96.58
184	21	27.38	98.55
192	22	27.83	101.45

TABLE 13.2 – Estimation de l'effort de travail à fournir pour résoudre les challenges restants. Ajouter la colonne Cycles/itération et la colonne Nombre d'itérations donne le $\log_2$ de l'effort de travail attendu

14 | Attaque d'un schéma de chiffrement basé sur des codes convolutifs

Les travaux présentés dans ce chapitre ont fait l'objet d'une publication lors de la conférence PQCrypto 2013 [44].

14.1 Introduction

Löndahl et Johansson ont proposé en 2012 [47] une variante du cryptosystème de McEliece remplaçant les codes de Goppa par des codes convolutifs. Cette modification se veut rendre les attaques structurelle plus difficiles puisque la matrice génératrice publique de ce schéma contient de grandes parties générées entièrement aléatoirement. Deux schémas sont proposés, l'un consiste à étendre un code de Goppa en y ajoutant la matrice génératrice d'un code convolutif évolutif. Nous montrons ici que ce schéma peut être attaqué en recherchant des mots de code de poids faible dans le code public du schéma et en les utilisant pour démêler la partie convolutive de la partie Goppa. Il ne reste ensuite qu'à casser la partie Goppa de ce schéma, ce qui peut être fait en moins d'un jour de calcul.

La proposition d'utiliser des codes convolutifs émise dans [47] s'insère dans le fil de recherche présenté section 1.3.2 visant à proposer des alternatives aux codes de Goppa dans le système de McEliece. La nouveauté intéressante de ce schéma est le fait que la clé secrète est composée de grandes parties générées complètement aléatoirement, dépourvues donc de structure algébrique contrairement aux codes de Reed–Solomon, les codes algébriques géométriques, les codes de Goppa ou les codes de Reed–Muller.

Dans [47], deux schémas sont proposés. Le premier envisage un système ou la clé secrète est simplement la matrice génératrice d'un code convolutif *tail-biting* évolutif. Des paramètres supposés rendre le schéma résistant aux attaques de complexité temporelle 2^{80} opérations élémentaires et permettant une complexité de décodage raisonnable y sont suggérés. L'inconvénient de cette construction est que la complexité du décodage augmente exponentiel-

lement avec le niveau de sécurité souhaité. Les auteurs donnent cependant un deuxième schéma qui, lui, passe à l'échelle et qui est construit en partant d'un code de Goppa et en l'étendant en y adjoignant la matrice génératrice d'un code convolutif évolutif.

Nous étudions la sécurité de ce second schéma. La proposition plaide que la structure convolutive du code ne peut être retrouvée de par la suffisamment grande distance minimale du code dual. Cependant, nous montrons ici que cette défense supplémentaire peut être attaquée en recherchant des mots de code de poids faible dans le code public du schéma. En utilisant une procédure de filtrage adaptée de ces mots de code, nous parvenons à démêler la partie convolutive de la partie Goppa de la matrice publique.

L'élément principal qui permet à cette attaque de fonctionner est le phénomène suivant : le code public de ce schéma contient des sous-codes de support bien plus petit que celui du code public alors que leurs rendements restent proches de celui du code public. Le support de ces mots peut être facilement trouvé par des algorithmes de recherche de mots de poids faible. Il est intéressant de remarquer que le schéma de signature KKS [39] a été cassé par cette même approche [58]. Le support de ces sous-codes révélant la structure convolutive, il suffit de poinçonner le code public pour ne conserver que la partie Goppa. Après cela, déchiffrer un message chiffré devient possible vis-à-vis des paramètres données dans [47], car ceux-ci sont suffisamment faibles pour que les algorithmes génériques de décodage de codes linéaires fonctionnent en temps raisonnable. En effet, dans ce contexte, l'attaque ne nécessite que quelques heures de calcul. Il semble possible de modifier les paramètres du schéma pour éviter ce genre d'attaque. Afin de donner un aperçu du nouveau niveau de sécurité de ce schéma, une version améliorée de cette attaque est décrite et sa complexité est analysée dans la section 14.5.1. Celle-ci suggère que le schéma pourrait être réparé en le paramétrant de manière plus conservatrice. Quelques indications sur la manière de procéder sont donnée section 14.5.3.

14.2 Un schéma de McEliece basé sur des codes convolutifs

Le schéma peut se résumer de la façon suivante :

Clé secrète :
- $\boldsymbol{G}_{\text{sec}}$ une matrice génératrice $k \times n$ construite par blocs telle que montré figure 14.1 ;
- $\boldsymbol{P}$ une matrice de permutation $n \times n$;
- $\boldsymbol{S}$ une matrice aléatoire inversible $k \times k$ sur $\mathbb{F}_2$.

Clé publique : $\boldsymbol{G}_{\text{pub}} \stackrel{\text{def}}{=} \boldsymbol{S}\boldsymbol{G}_{\text{sec}}\boldsymbol{P}$.

Chiffrement : Le chiffré $\boldsymbol{c} \in \mathbb{F}_2^n$ d'un texte clair $\boldsymbol{m} \in \mathbb{F}_2^k$ est obtenu

en tirant aléatoirement $\boldsymbol{e}$ dans $\mathbb{F}_2^n$ de poids t et en calculant $\boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{m}\boldsymbol{G}_{\text{pub}} + \boldsymbol{e}$.

Déchiffrement : Il consiste en les étapes suivantes :

1. Calculer $\boldsymbol{c}' \stackrel{\text{def}}{=} \boldsymbol{c}\boldsymbol{P}^{-1} = \boldsymbol{m}\boldsymbol{S}\boldsymbol{G}_{\text{sec}} + \boldsymbol{e}\boldsymbol{P}^{-1}$ et utiliser l'algorithme de décodage du code de matrice génératrice $\boldsymbol{G}_{\text{sec}}$ pour retrouver $\boldsymbol{m}\boldsymbol{S}$ partant de $\boldsymbol{c}'$;
2. Multiplier le résultat du décodage par $\boldsymbol{S}^{-1}$ pour retrouver $\boldsymbol{m}$.

Ce qui permet à ce schéma de fonctionner est le fait que si t est correctement choisi alors la partie Goppa du mot peut être décodé avec grande probabilité, ce qui permet au décodeur séquentiel du code convolutif évolutif de décoder les erreurs restantes. Nous dénoterons désormais par $\mathscr{C}_{\text{pub}}$ le code de matrice génératrice $\boldsymbol{G}_{\text{pub}}$ et par $\mathscr{C}_{\text{sec}}$ le code de matrice génératrice $\boldsymbol{G}_{\text{sec}}$.

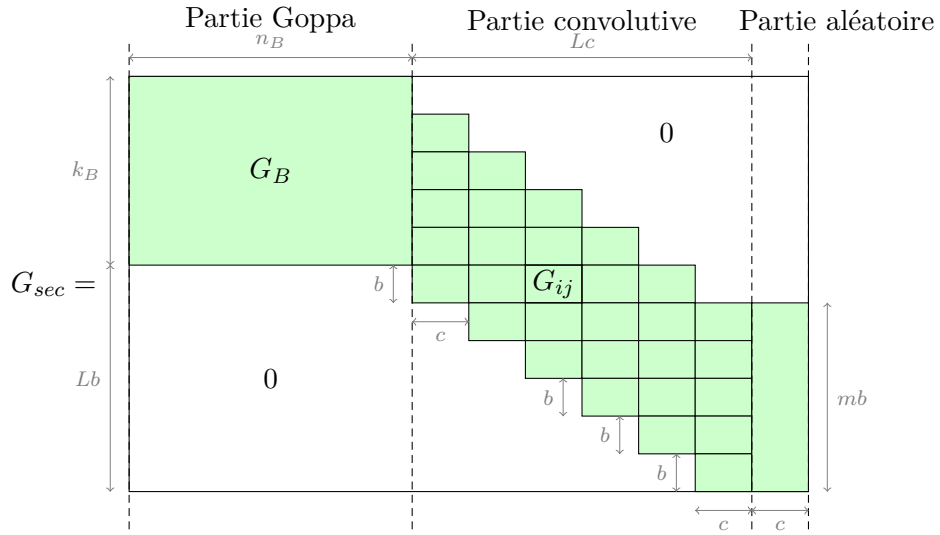


FIGURE 14.1 – La matrice génératrice secrète. Les zones non-blanches indiquent les éléments non-nuls de la matrice. G_B est une matrice génératrice d'un code de Goppa binaire de longueur n_B et de dimension k_B. À cette matrice est concaténée la matrice génératrice d'un code convolutif évolutif transformant b bits d'information en c bits de donnée (les blocs G_{ij} sont donc de taille $b \times c$) ainsi que c colonnes aléatoires. La dimension du code final est donc $k \stackrel{\text{def}}{=} k_B + Lb$ et sa longueur est $n \stackrel{\text{def}}{=} n_B + (L+1)c$ où L est la taille de la fenêtre de temps sur laquelle s'étend le codage convolutif.

14.3 Description de l'attaque

Le but de cette section est d'expliquer l'idée sous-jacente de l'attaque qui est une attaque de récupération de message tirant parti d'une récupération

partielle de la clé. L'attaque est divisée en deux parties. La première consiste en une récupération partielle de la clé visant à retrouver quelles positions du chiffré correspondent à la partie convolutive du code. La deuxième partie consiste en une attaque de récupération de message tirant parti du fait que si la partie convolutive est révélée, alors un attaquant peut, avec forte probabilité, déchiffrer un message s'il est capable de décoder un mot de code linéaire de longueur n_B bruité en moins de $t_B \stackrel{\text{def}}{=} t\frac{n_B}{n}$ positions (il s'agit du nombre moyen d'erreur que la partie Goppa doit décoder).

14.3.1 Démêler la structure convolutive

Les auteurs de [47] ont choisi les paramètres de leur schéma de telle sorte qu'il soit difficile de trouver des mots de poids faible dans le code dual du code public $\mathscr{C}_{\text{pub}}$. L'article plaide que la seule distinction entre le code public et un code aléatoire est la présence de la structure convolutive en terme d'équation de parité de poids faible. Par exemple, les paramètres $(n, k, c, b, t) = (1800, 1200, 30, 20, 45)$ sont proposés et les auteurs proposent d'écarter les codes ayant des équations de parité de poids inférieur à 125 lors de la construction. Cependant, le fait que la structure de $\mathscr{C}_{\text{pub}}$ mène de façon naturelle à des mots de poids faible dans le code lui-même n'est pas pris en compte. En effet, le nombre de mots de codes de poids inférieur ou égal à c est très grand ($\approx 2^{b-1}$). Cela provient du fait que le sous-code généré par les c dernières lignes de $\boldsymbol{G}_{\text{sec}}$ (et permuté par $\boldsymbol{P}$) a un support de taille $2c$ et une dimension b. Par conséquence, tout algorithme cherchant des mots de poids inférieur à c devrait trouver les mots de ce sous-code. Le support de ces mots révèle les $2c$ dernières positions de $\boldsymbol{G}$. En poinçonnant ces colonnes, nous obtenons un code contenant un sous-code de dimension b et de support de taille c, généré par les pénultièmes lignes de $\boldsymbol{G}_{\text{sec}}$. On peut donc répéter le processus précédent mais en cherchant cette fois des mots de poids inférieur à $c/2$ pour révéler les c colonnes du bloc précédent. En d'autres termes, nous récupérons un premier sous-code de dimension b et ayant pour support les $2c$ dernières positions de $\mathscr{C}_{\text{sec}}$. Puis nous récupérons un deuxième sous-code de dimension b ayant pour support les $3c$ dernières positions de $\mathscr{C}_{\text{sec}}$ et ainsi de suite jusqu'à obtenir, en échangeant les colonnes appropriées, la matrice génératrice G' d'un code équivalent à $\mathscr{C}_{\text{pub}}$ qui aurait la forme présentée figure 14.2.

Plus formellement, l'algorithme 13 permet de retrouver une matrice génératrice d'un tel code.

Nous supposons ici que :

– la fonction MatriceGénératriceCodePoinçonné prend en entrée un code $\mathscr{C}$ de longueur n et un ensemble ordonné de positions $\mathcal{L}$, sous-ensemble de $[\![1, n]\!]$, et retourne une matrice génératrice du code $\mathscr{C}$ poinçonné en les positions appartenant) $\mathcal{L}$;

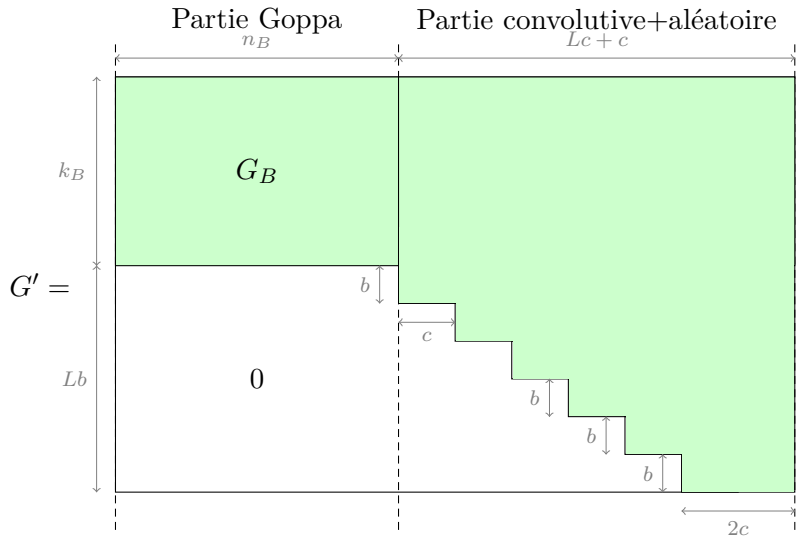


FIGURE 14.2 – La matrice génératrice d'un code équivalent obtenu par notre approche. $\boldsymbol{G}'_B$ est la matrice génératrice d'un code de Goppa équivalent au code de matrice génératrice $\boldsymbol{G}_B$.

- CHOISIRW est une fonction ajustant le poids des mots recherchés en fonction de l'itération (ce paramétrage est détaillé section 14.4) ;
- SUPPORT($\mathscr{C}$) retourne le support (ordonné) du code $\mathscr{C}$
- l'opérateur || est l'opérateur de concaténation de listes ;
- la fonction POIDSFAIBLE prend en entrée un code $\mathscr{C}$ et un poids w. Elle retourne une matrice génératrice d'un sous-code de $\mathscr{C}$ obtenu en cherchant des mots de code de poids inférieur ou égaux à w. Un certain nombre de mots de code de poids $\leq w$ sont produits et les positions impliquées dans au moins t mots de codes sont placés dans une liste $\mathcal{L}$ (ou t est un seuil dépendant de w, de la longueur n et de la dimension k du code, ainsi que du nombre de mots de codes produits par l'appel précédent à cette fonction). Cela signifie que la position i est sélectionnée dès qu'au moins t éléments c dans $\mathscr{C}$ pour lesquels $c_i = 1$ (voir algorithme 14).
- la fonction MATRICEGÉNÉRATRICEÉTENDUE prend en entrée une matrice génératrice d'un code $\mathscr{C}'$, un ensemble ordonné de position $\mathcal{L}$ et un code $\mathscr{C}$ tels que $\mathscr{C}'$ est le résultat du poinçonnage de $\mathscr{C}$ en les positions appartenant à $\mathcal{L}$. Elle retourne une matrice génératrice de $\mathscr{C}''$, sous-code permuté de $\mathscr{C}$ dont les positions sont réordonnées de telle façon que les premières positions correspondent au code $\mathscr{C}'$ et les autres positions à la liste ordonnée $\mathcal{L}$. Ce code $\mathscr{C}''$ correspond aux mots du code $\mathscr{C}'$ qui sont étendues comme étant des mots du code $\mathscr{C}$ sur les position appartenant à $\mathcal{L}$ d'une façon linéaire arbitraire.

Algorithme 13 Un algorithme pour trouver $\boldsymbol{G}'$.

Entrée :

La matrice génératrice publique $\boldsymbol{G}_{\text{pub}}$.

Sortie :

Une matrice génératrice $\boldsymbol{G}'$ d'un code équivalent à $\mathscr{C}_{\text{pub}}$ ayant la forme présentée fig. 14.2.

Fonction ConstruireCodeÉquivalent($\boldsymbol{G}_{\text{pub}}$)
  $\mathcal{L} \leftarrow []$
  Pour $i = L, \ldots, 1$
    $\boldsymbol{G} \leftarrow$ MatriceGénératriceCodePoinçonné($\mathscr{C}_{\text{pub}}, \mathcal{L}$)
    $\boldsymbol{G} \leftarrow$ PoidsFaible($\boldsymbol{G}, w$)
    $w \leftarrow$ ChoisirW(i)
    $\boldsymbol{G}_i \leftarrow$ MatriceGénératriceÉtendue($\boldsymbol{G}, \mathcal{L}, \mathscr{C}_{\text{pub}}$)
    $\mathcal{L} \leftarrow$ Support($\boldsymbol{G}$)$||\mathcal{L}$
  $\boldsymbol{G} \leftarrow$ MatriceGénératriceCodePoinçonné($\mathscr{C}_{\text{pub}}, \mathcal{L}$)
  $\boldsymbol{G}_0 \leftarrow$ MatriceGénératriceÉtendue($\boldsymbol{G}, \mathcal{L}, \mathscr{C}_{\text{pub}}$)
  $\boldsymbol{G}'$ est la concaténation des lignes de $\boldsymbol{G}_0, \boldsymbol{G}_1, \ldots, \boldsymbol{G}_L$.
  Retourner $\boldsymbol{G}'$

14.3.2 Décoder les messages

Si nous sommes capables de décoder le code généré par la matrice $\boldsymbol{G}'_B$, alors les algorithmes standard de décodage de code convolutifs pourront décoder les $(L+1)c$ dernières positions. Soit $\boldsymbol{G}'_B$ la matrice génératrice d'un code équivalent au code de Goppa secret choisi pour le schéma spécifié figure 14.2. Un tel code peut être décoder par des algorithmes de décodage de code linéaires génériques (présentés partie II). Ces algorithmes fonctionnent en temps raisonnable vis-à-vis des paramètres proposés dans [47].

14.4 Mise en œuvre de l'attaque pour les paramètres proposés

Nous avons appliqué l'attaque sur les paramètres proposés dans [47]. Ceux-ci sont reporté dans le tableau 14.1.

Table 14.1 – Paramètres pour le deuxième schéma proposés dans [47].

n	n_B	k	k_B	b	c	L	m	t (nombre d'erreurs)
1800	1020	1160	660	20	30	25	12	45

Ajuster correctement le paramètre w de la fonction PoidsFaible est la

Algorithme 14 La fonction PoidsFaible

Entrée :
Une matrice G de taille $k \times n$ génératrice d'un code $\mathscr{C}$,
un entier w.

Sortie :
une matrice $\boldsymbol{G}'$ génératrice d'un sous-code de $\mathscr{C}$ obtenu à partir des supports d'un sous-ensemble de mots de $\mathscr{C}$ de poids $\leq w$.

Fonction PoidsFaible($\boldsymbol{G}, w$)
  $\mathscr{C} \leftarrow$ RechercheMotsPoidsFaible($\boldsymbol{G}, w$)
  Initialiser un tableau tab de taille n à zéro
  $t \leftarrow$ Seuil($w, n, k, |\mathscr{C}|$)
  Pour tout $c \in \mathscr{C}$
    Pour $i \in [1..n]$
      Si $c_i = 1$
        tab[i] $\leftarrow$ tab[i] + 1
  $\mathcal{L} \leftarrow []$
  Pour $i \in [1..n]$
    Si tab[i] $\geq$ t
      $\mathcal{L} \leftarrow \mathcal{L}||\{i\}$
  $\boldsymbol{G}' \leftarrow$ CodeRaccourci($\boldsymbol{G}, \mathcal{L}$)
  Retourner $\boldsymbol{G}'$.

Commentaires :
RechercheMotsPoidsFaible($\boldsymbol{G}, w$) produit un ensemble de combinaisons linéaire de ligne de $\boldsymbol{G}$ de poids $\leq w$
CodeRaccourci($\boldsymbol{G}, \mathcal{L}$) produit une matrice génératrice du sous-code de $\mathscr{C}$ formé des mots de $\mathscr{C}$ dont les coordonnées hors de $\mathcal{L}$ sont nulles.

clé permettant de trouver les 60 dernières positions du code. Si w est choisi trop grand, les mots retrouvés ne permettent pas de discriminer comme souhaité les dernières positions du code. Par exemple, la figure 14.4 donne les fréquences des positions impliqués dans les supports des mots de codes de poids inférieur à 22 trouvés par l'algorithme de Dumer [26] (voir partie II)

On peut voir sur cette figure que ces fréquences discriminent les 90 dernières positions du code et non les 60 désirées. Par contre, choisir $w = 18$ discrimine correctement ces positions, comme montré figure 14.4.

La figure 14.4 a été produite à partir de 3900 mots de code générés en une heure et trente minutes par le programme présenté chapitre 12 sur un Intel Xeon W3550 (3 GHz). La récupération d'un message, consistant à décoder une moyenne de 25.5 erreurs dans un code de dimension 660 et de longueur 1020, a une complexité en temps $\approx 2^{42}$. Celle-ci peut être exécutée par ce même programme sur cette même machine en 6.5 heures en moyenne.

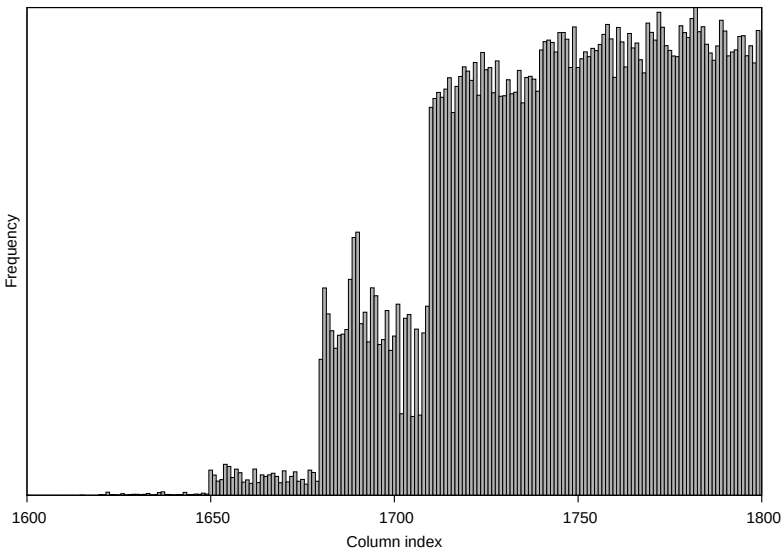


FIGURE 14.3 – Les fréquences des positions du code impliquées dans le support des mots de codes de poids $\leq$ 22 générés par l'algorithme de Dumer.

14.5 Analyse de la sécurité du schéma

14.5.1 Une attaque améliorée

Le but de cette section est de donner une analyse brute de la sécurité du schéma. Nous n'analyserons pas l'attaque détaillée section 14.3 puisque, même si elle suffit à casser le deuxième schéma proposé dans [47], elle n'est pas la plus efficace. Nous donnerons un aperçu d'une meilleure attaque ainsi qu'une ébauche d'analyse. Le problème fondamental du schéma est l'existante d'un sous-code $\mathscr{C}$ de $\mathscr{C}_{\text{pub}}$ de support très restreint (de taille $2c$ ici), à savoir le code généré par les b dernière lignes de $\boldsymbol{G}$ permuté par la matrice de permutation secrète $\boldsymbol{P}$. Par exemple, il existe $\approx 2^{b-1}$ mots de code c de poids inférieur à c qui peuvent être trouvé par un algorithme de recherche de mots de poids faible et qui révèlent le support de $\mathscr{C}$. C'est là l'idée sous-jacente de notre attaque. Cependant, il existe d'autres sous-codes de support plutôt restreint qui donnent des mots de code de poids faible, à savoir les codes $\mathscr{C}_s$ générés par les $s \times b$ dernières lignes de $\boldsymbol{G}$ pour s allant de 2 à L. Le support de $\mathscr{C}_s$ est de taille $(s+1)c$. On peut remarquer que leur rendement s'approche du rendement $\frac{2}{3}$ (qui est plus ou moins le rendement du code final) lorsque s augmente. Ce phénomène aide les algorithmes de recherche

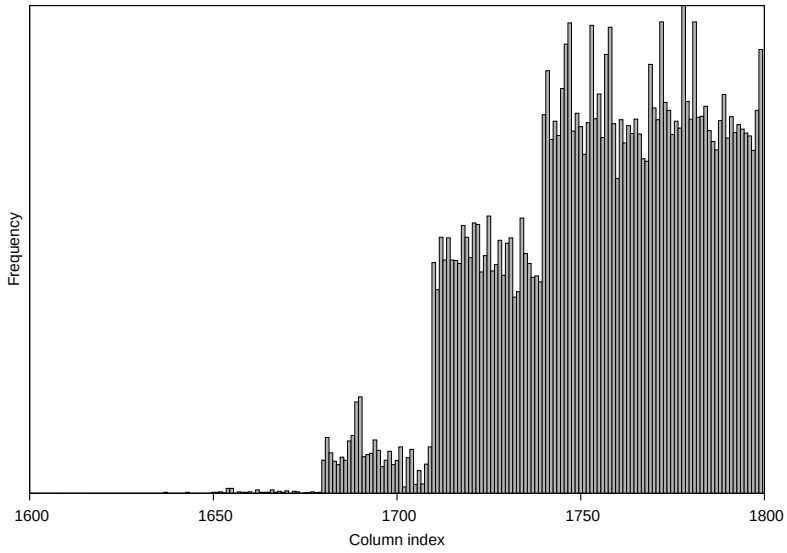


FIGURE 14.4 – Les fréquences des positions du code impliquées dans le support des mots de codes de poids $\leq$ 18 générés par l'algorithme de Dumer.

de mots de petit poids.

Une amélioration de notre attaque consiste à utiliser un algorithme de recherche de mots de code de poids faible pour trouver l'un des mots de $\mathscr{C}_s$ puis d'utiliser ce mot pour amorcer la recherche du support entier de $\mathscr{C}_s$. Cela correspond à l'idée de l'attaque du schéma de signature KKS expliquée par l'algorithme 2 de la section 4.4 de [58]. Cette approche utilise le mot de code trouvé c pour trouver de nouveaux mots appartenant au même sous-code de support restreint en imposant à l'algorithme de recherche de mot de petit poids le choix un ensemble d'information sans intersection avec le support de c La complexité de l'attaque est donc dominée par le coût de la recherche d'un seul mot de code de $\mathscr{C}$, lorsqu'il est possible d'identifier efficacement les candidats (en vérifiant leur poids ici). Remarquons qu'il est vraisemblable que $\mathscr{C}$ soit le sous-code $\mathscr{C}$ de dimension b ayant le plus petit support. Cela est précisément la notion retranscrite par le poids de Hamming généralisé d'un code [72], w_i étant défini comme la taille du plus petit support d'un sous-code de dimension i. En d'autres termes, w_1 est la distance minimale du code et dans notre cas il est vraisemblable que $w_b = 2c$ (et plus généralement $w_{sb} = (s+1)c$ pour $s = 1..L$). Exprimé différemment, le problème qui devrait être difficile est le suivant

Problem 1. *Trouver l'un des sous-codes de dimension $s \times b$ dont la taille*

du support est le $s \times b$-ième poids de Hamming généralisé de $\mathscr{C}_{pub}$.

Nous allons maintenant nous concentrer sur l'approche suivante pour résoudre ce problème. Considérons un algorithme recherchant des mots de poids faible dans un code de dimension k tel que ceux présentés dans la partie II. Nous exécutons un tel algorithme et nous intéressons à la complexité de retrouver un mot appartenant à $\mathscr{C}$. Cette approche est celle qui a permis de casser avec succès le schéma KKS [58] et qui est le candidat naturel pour casser le schéma proposé dans [47].

Pour analyser un tel algorithme, nous utiliserons les hypothèses simplificatrices suivantes :

- Le coût de vérification d'un des ensembles d'information $\mathcal{I}$ est de l'ordre de $O\left(\mathscr{L} + \frac{\mathscr{L}^2}{2^l}\right)$ où $\mathscr{L} = \sqrt{\binom{k+l}{p}}$. Nous négligeons ici le coût de mise sous forme systématique de la matrice de parité et ne considérons pas les améliorations récentes [49, 8]. Cette approximation est faite a des fins de simplicité.
- Dénotons par k' la dimension du sous-code $\mathscr{C}$, par n' la taille de son support $\mathcal{J}$. Nous supposons que le code résultant du poinçonnage de $\mathscr{C}$ en toutes les positions n'appartenant pas à $\mathcal{J}$ se comporte comme un code aléatoire de dimension k' et de longueur n'.

La complexité d'un tel algorithme est donnée par la proposition suivante.

Proposition 1. *Soit*

- $f(x)$ *la fonction définie sur les nombres réels positifs par* $f(x) \stackrel{def}{=} \max\left(x(1-x/2), 1-\frac{1}{x}\right)$;
- $\pi(s) \stackrel{def}{=} \frac{\binom{n'}{s}\binom{n-n'}{k+l-s}}{\binom{n}{k+l}}$;
- $\lambda(s) \stackrel{def}{=} \binom{s}{p} 2^{k'-s}$;
- $C(k,l,p) \stackrel{def}{=} \mathscr{L} + \frac{\mathscr{L}^2}{2^l}$ *où* $\mathscr{L} \stackrel{def}{=} \sqrt{\binom{k+l}{p}}$;
- $\Pi \stackrel{def}{=} \sum_{s=1}^{n'} \pi(s) f(\lambda(s))$.

Le coût de la recherche d'un mot de poids faible appartenant à $\mathscr{C}$ est alors de l'ordre de

$$O\left(\frac{C(k,l,p)}{\Pi}\right).$$

14.5.2 Preuve de la proposition 1

Notre premier outil est une borne inférieure sur la probabilité qu'a un ensemble donné $X \subseteq \mathbb{F}_2^n$ ait une intersection non-nulle avec un code linéaire aléatoire $\mathscr{C}_{\text{rand}}$ de dimension k et de longueur n tiré uniformément. Ce lemme donne une borne inférieure fine même lorsque X est très grand et quand il existe un grand écart entre les quantités $\mathbf{prob}(X \cap \mathscr{C}_{\text{rand}} \neq \emptyset) = \mathbf{prob}(\cup_{x \in X}\{x \in \mathscr{C}_{\text{rand}}\})$ et $\sum_{x \in X} \mathbf{prob}(x \in \mathscr{C}_{\text{rand}})$.

Lemme 2. *Soit X un sous-ensemble de $\mathbb{F}_2^n$ de taille m et f la fonction définie par $f(x) \stackrel{def}{=} \max\left(x(1-x/2), 1-\frac{1}{x}\right)$. Si l'on nomme x la quantité $\frac{m}{2^{n-k}}$, alors*

$$\mathbf{prob}(X \cap \mathscr{C}_{rand} \neq \emptyset) \geq f(x).$$

Ce lemme est donné et prouvé dans [58].

Terminons désormais la preuve de la proposition 1. Dénotons $\mathcal{J}$ le support de $\mathscr{C}$:

$$\mathcal{J} \stackrel{\text{def}}{=} \mathsf{supp}(\mathscr{C}).$$

Commençons par calculer le nombre attendu d'ensemble $\mathcal{I}$ qu'il faut prendre en compte avant de trouver un élément de $\mathscr{C}$. Un tel évènement se produit précisément lorsque il y a un mot non-nul dans $\mathscr{C}$ dont la restriction à $\mathcal{I} \cap \mathcal{J}$ est de poids p. Soit $\mathscr{C}_{\mathcal{I}\cap\mathcal{J}}$ la restriction des mots de $\mathscr{C}$ aux position appartenant) $\mathcal{I} \cap \mathcal{J}$, c'est-à-dire

$$\mathscr{C}_{\mathcal{I}\cap\mathcal{J}} \stackrel{\text{def}}{=} \{(c_i)_{i\in\mathcal{I}\cap\mathcal{J}} : (c_i)_{1\leq i\leq n} \in \mathscr{C}\}.$$

Soit X l'ensemble des mots binaires non-nuls de support $\mathcal{I} \cap \mathcal{J}$ et de poids p. Dénotons W la taille de $\mathcal{I} \cap \mathcal{J}$. La probabilité qu'a W d'être égal à s est précisément

$$\mathscr{C}_{\mathcal{I}\cap\mathcal{J}} \stackrel{\text{def}}{=} \{(c_i)_{i\in\mathcal{I}\cap\mathcal{J}} : (c_i)_{1\leq i\leq n} \in \mathscr{C}\}.$$

La probabilité Π qu'a le choix d'un certain $\mathcal{I}$ de donner, parmi les mots de codes examinés par l'algorithme, un mot de $\mathscr{C}$ peut s'exprimer

$$\begin{aligned}\Pi &= \sum_{s=1}^{n'} \mathbf{prob}(W = s)\mathbf{prob}(X \cap \mathscr{C}_{\mathcal{I}\cap\mathcal{J}} \neq \emptyset) \\ &\geq \sum_{s=1}^{n'} \pi(s) f(\lambda(s))\end{aligned}$$

14.5.3 Réparer le schéma

Une réparation possible consiste à augmenter la taille de la partie aléatoire (correspondant ici aux c dernières colonnes de $\boldsymbol{G}$). Plutôt que de prendre cette taille égale à c comme suggéré dans [47], sa taille peut être augmentée afin de contrecarrer l'algorithme décrit section 14.5.1. Notons r le nombre de colonnes aléatoires ajoutées à la fin de la partie convolutive de $\boldsymbol{G}_{\text{sec}}$. Si l'on choisit $r = 140$, alors l'attaque susmentionnée est capable

de retourner un élément de $\mathscr{C}$ (le sous-code permuté correspondant aux b dernières lignes de $\boldsymbol{G}_{\text{sec}}$) en $\approx 2^{80}$ opérations. Comme précédemment, notons $\mathscr{C}_s$ le sous-code permuté de $\mathscr{C}_{\text{pub}}$ généré par les $s \times b$ dernières lignes de $\boldsymbol{G}_{\text{sec}}$. Nous pouvons utiliser l'analyse précédente pour estimer la complexité de l'obtention d'un élément de $\mathscr{C}_s$ par l'algorithme précédent. Les résultats sont réunis dans le tableau 14.2.

TABLE 14.2 – Complexité d'obtention d'au moins un élément de $\mathscr{C}_s$ par l'algorithme décrit section 14.5.1

s	1	5	10	15	20	21	22	25
complexité (bits)	80.4	72.1	65.1	61.0	59.4	59.3	59.4	59.8

Ce tableau montre que, dans ce cas, la menace principale ne vient pas de la recherche de mot de poids faible provenant de $\mathscr{C}_1$, mais des mots de poids modéré provenant de $\mathscr{C}_{20}$. Les mots de $\mathscr{C}_{20}$ ont un poids moyen de $\frac{r+20c}{2} = 370$. Conserver tous les candidats de poids inférieur à cette quantité lors de l'exécution de l'algorithme décrit section 14.5.1 permet vraisemblablement de filtrer la vaste majorité des mauvais candidats et de conserver avec forte probabilité les éléments de $\mathscr{C}_{20}$. De tels candidats peuvent être utilisés tel que décrit section 14.5.1 pour vérifier si ils appartiennent ou non à un sous-code de grande dimension et de support restreint.

Il existe une façon simple d'expliquer ce phénomène. Il faut remarquer que le rendement de $\mathscr{C}$ vaut $\frac{b}{c+r}$, ce qui est bien plus faible que le rendement du schéma global qui est proche de $\frac{b}{c}$. Cependant, lorsque s augmente, le rendement de $\mathscr{C}_s$ s'approche de $\frac{b}{c}$, puisque son rendement est $\frac{sb}{sc+r} = \frac{b}{c+r/s}$. Supposons un instant que le rendement de $\mathscr{C}_s$ soit $\frac{b}{c}$. Dans ce cas, mettre $\boldsymbol{G}_{\text{pub}}$ sous forme systématique (ce qui revient à utiliser l'algorithme précédent avec $p = 1$ et $l = 0$) va vraisemblablement révéler une grande partie du support de $\mathscr{C}_s$ puisqu'il suffit d'examiner le support des lignes ayant un poids proche de $\frac{sc+r}{2}$ (ce phénomène a déjà été observé dans [59]). Cela peut s'expliquer de la façon suivante. Choisissons $\mathcal{I}$ de taille k, la dimension de $\mathscr{C}_{\text{pub}}$, comme ensemble d'information de $\mathscr{C}_{\text{pub}}$. Alors, puisque le rendement de $\mathscr{C}_s$ est égal à celui de $\mathscr{C}_{\text{pub}}$, la taille de $\mathcal{I} \cap \mathcal{J}$ (où $\mathcal{J}$ est le support de $\mathscr{C}_s$) a de bonnes chances d'être inférieure ou égale à la dimension de $\mathscr{C}_s$. Cela implique qu'il est possible d'obtenir des mots de $\mathscr{C}_s$ en choisissant n'importe quel ensemble d'information $\mathcal{I}$ de poids 1 non nul sur $\mathcal{I} \cap \mathcal{J}$ (et donc de poids 1 ici). Plus généralement, même si $\mathcal{I} \cap \mathcal{J}$ est légèrement plus grand que la dimension de $\mathscr{C}_s$ nous espérons être capables d'obtenir des mots de $\mathscr{C}_s$ dès que p est plus grand que la distance de Gilbert-Varshamov de la restriction $\mathscr{C}'_s$ de $\mathscr{C}_s$ à $\mathcal{I} \cap \mathcal{J}$, car il y a dans ce cas de bonne chance que ce code poinçonné ait des mots de poids p. Cette distance de Gilbert-Varshamov sera

petite dans ce cas, car le rendement de $\mathscr{C}_s$ est très proche de 1 (on s'attend à ce qu'elle vaille $\frac{\dim(\mathscr{C}_s)}{|\mathcal{I}\cap\mathcal{J}|}$).

Quoi qu'il en soit, il est clair qu'il est possible de paramétrer le schéma (en particulier en augmentant r) de telle façon que les algorithmes de recherche de mots de poids faible soient incapables de trouver les sous-codes $\mathscr{C}_s$ avec une complexité inférieure à un certain seuil. Cependant, tous ces codes doivent être pris en compte. De plus, les attaques sur le code dual doivent également être reconsidérée ; [47] ne considère que les attaques sur le dual cherchant des mots de code de poids faible, mais il est évident que la technique utilisé pour trouver les sous-codes $\mathscr{C}_s$ fonctionne également sur le code dual. Qui plus est, même si par construction la restriction de $\mathscr{C} = \mathscr{C}_1$ à son support devrait se comporter comme un code aléatoire, cela n'est plus vrai pour $\mathscr{C}_s$ avec s supérieur à un, à cause de la structure convolutive. L'analyse esquissée section 14.5.1 devrait être adaptée légèrement pour ce cas et devrait prendre en compte les améliorations récentes dans le domaine des algorithmes de recherche de mots de poids faible [49, 8]. Pour finir, le choix des paramètres nécessite également une étude soigneuse de la probabilité d'échec du décodage séquentiel.

Bibliographie

[1] M. Baldi, M. Bianchi, F. Chiaraluce, J. Rosenthal, and D. Schipani. Enhanced public key security for the McEliece cryptosystem. submitted, 2011. arxiv :1108.2462v2[cs.IT].

[2] M. Baldi, M. Bodrato, and G. Chiaraluce. A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes. In *Security and Cryptography for Networks (SCN)*, pages 246–262, 2008.

[3] M. Baldi and G. F. Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC codes. In *IEEE International Symposium on Information Theory*, pages 2591–2595, Nice, France, Mar. 2007.

[4] G. Bard and M. Albrecht. M4RI(e)- Linear Algebra over $\mathbf{F}_2$ (and $\mathbf{F}_{2^e}$). Free Open Source Software. `http://m4ri.sagemath.org/index.html`.

[5] E. Barker and A. Roginsky. Transitions : Recommendation for transitioning the use of cryptographic algorithms and key lengths, 2011. Published as NIST Special Publication 800-131A, `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`.

[6] A. Becker. *The representation technique - Applications to hard problems in cryptography.* Thèse de doctorat, Université de Versaille Saint-Quentin-en-Yvelines, 2012.

[7] A. Becker, A. Joux, A. May, and A. Meurer. Decoding Random Binary Linear Codes in $2^{n/20}$: How 1+1=0 Improves Information Set Decoding. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

[8] A. Becker, A. Joux, A. May, and A. Meurer. Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding. In *Eurocrypt 2012*, LNCS. Springer, 2012.

[9] T. P. Berger, P. Cayrel, P. Gaborit, and A. Otmani. Reducing Key Length of the McEliece Cryptosystem. In B. Preneel, editor, *Progress in Cryptology - Second International Conference on Cryptology in Africa (AFRICACRYPT 2009)*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97, Gammarth, Tunisia, June 21-25 2009.

[10] T. P. Berger and P. Loidreau. How to Mask the Structure of Codes for a Cryptographic Use. *Designs Codes and Cryptography*, 35(1) :63–79, 2005.

[11] E. Berlekamp. *Algebraic Coding Theory.* Aegen Park Press, 1968.

[12] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3) :384–386, May 1978.

[13] D. Bernstein. *Minimum number of bit operations for multiplication*, 2009.

[14] D. Bernstein, J. Buchmann, and J. Ding, editors. *Post-Quantum Cryptography.* Springer, 2009.

[15] D. J. Bernstein, T. Lange, and C. Peters. Attacking and Defending the McEliece Cryptosystem. In *PQCrypto*, volume 5299 of *LNCS*, pages 31–46, 2008.

[16] D. J. Bernstein, T. Lange, and C. Peters. *Cryptanalytic challenges for wild McEliece*, 2011.

[17] D. J. Bernstein, T. Lange, and C. Peters. Smaller decoding exponents : ball-collision decoding. In *Proceedings of Crypto 2011*, volume 6841 of *LNCS*, pages 743–760, 2011.

[18] D. J. Bernstein, T. Lange, and C. Peters. Wild McEliece Incognito. In *PQCrypto*, pages 244–254, 2011.

[19] J.-L. Beuchat, N. Sendrier, A. Tisserand, and G. Villard. FPGA Implementation of a Recently Published Signature Scheme. Rapport de recherche 5158, INRIA, 2004.

[20] B. Biswas and V. Herbert. Efficient Root Finding of Polynomials over Fields of Characteristic 2. In *WEWoRC 2009*, LNCS. Springer-Verlag, 2009.

[21] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13 :422–426, 1970.

[22] N. Courtois, M. Finiasz, and N. Sendrier. How to Achieve a McEliece-based Digital Signature Scheme. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174, Gold Coast, Australia, 2001. Springer.

[23] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based Digital Signature Scheme. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 157–174. Springer, 2001.

[24] A. Couvreur, P. Gaborit, V. Gauthier, A. Otmani, and J. Tillich. Distinguisher-Based Attacks on Public-Key Cryptosystems Using Reed-Solomon Codes. In *Proceedings of WCC 2013*, Apr. 2013. to appear, see also arxiv.

[25] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, Nov. 1976.

[26] I. Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991.

[27] I. Dumer. On Minimum Distance Decoding of Linear Codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991.

[28] J.-C. Faugère, V. Gauthier, A. Otmani, L. Perret, and J.-P. Tillich. A Distinguisher for High Rate McEliece Cryptosystems. In *ITW 2011*, pages 282–286, Paraty, Brazil, Oct. 2011.

[29] J.-C. Faugère, V. Gauthier, A. Otmani, L. Perret, and J.-P. Tillich. A Distinguisher for High Rate McEliece Cryptosystems. In *Proceedings of the Information Theory Workshop 2011, ITW 2011*, pages 282–286, Paraty, Brasil, 2011.

[30] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 279–298. Springer, 2010.

[31] C. Faure and L. Minder. Cryptanalysis of the McEliece cryptosystem over hyperelliptic curves. In *Proceedings of the eleventh International Workshop on Algebraic and Combinatorial Coding Theory*, pages 99–107, Pamporovo, Bulgaria, June 2008.

[32] M. Finiasz. Parallel-CFS : Strengthening the CFS McEliece-Based Signature Scheme. In A. Biryukov, G. Gong, and D. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *LNCS*, pages 159–170. Springer, 2010.

[33] M. Finiasz and N. Sendrier. Security Bounds for the Design of Code-based Cryptosystems. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, 2009.

[34] V. Gauthier, A. Otmani, and J.-P. Tillich. A Distinguisher-Based Attack on a Variant of McEliece's Cryptosystem Based on Reed-Solomon Codes. *CoRR*, abs/1204.6459, 2012.

[35] N. Howgrave-Graham and A. Joux. New Generic Algorithms for Hard Knapsacks. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 235–256. Springer, 2010.

[36] H. Janwa and O. Moreno. McEliece Public Key Cryptosystems Using Algebraic-Geometric Codes. *Designs Codes and Cryptography*, 8(3) :293–307, 1996.

[37] T. Johansson and F. Jönsson. On the Complexity of Some Cryptographic Problems Based on the General Decoding Problem. *IEEE-IT*, 48(10) :2669–2678, Oct. 2002.

[38] A. Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2009.

[39] G. Kabatianskii, E. Krouk, and B. J. M. Smeets. A Digital Signature Scheme Based on Random Error-Correcting Codes. In *IMA Int. Conf.*, volume 1355 of *Lecture Notes in Computer Science*, pages 161–167. Springer, 1997.

[40] D. Kravitz. Digital signature algorithm. US patent 5231668, July 1991.

[41] G. Landais. Implementation of several CSD solving algorithms. Free Open Source Software. `https://gforge.inria.fr/projects/collision-dec/`.

[42] G. Landais and N. Sendrier. McEliece based signature scheme. Free Open Source Software. `https://gforge.inria.fr/projects/cfs-signature/`.

[43] G. Landais and N. Sendrier. Implementing CFS. In S. Galbraith and M. Nandi, editors, *Indocrypt 2012*, volume 7668 of *LNCS*, pages 474–488. Springer, Dec. 2012.

[44] G. Landais and J. Tillich. An efficient attack of a McEliece cryptosystem variant based on convolutional codes. In *Proceedings of PQCrypto 2013*, LNCS, jun 2013. to appear.

[45] P. Lee and E. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. Günther, editor, *Advances in Cryptology - EUROCRYPT '88*, volume 330 of *LNCS*, pages 275–280. Springer, 1988.

[46] J. Leon. A Probabilistic Algorithm for Computing Minimum Weights of Large Error-Correcting Codes. *IEEE Transactions on Information Theory*, 34(5) :1354–1359, Sept. 1988.

[47] C. Löndahl and T. Johansson. A New Version of McEliece PKC Based on Convolutional Codes. In *ICICS*, pages 461–470, 2012.

[48] J. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15(1) :122–127, Jan. 1969.

[49] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $O(2^{0.054n})$. In D. H. Lee and X. Wang, editors, *Asiacrypt 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

[50] R. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep., Jet Prop. Lab., California Inst. Technol., Pasadena, CA*, pages 114–116, Jan. 1978.

[51] R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[52] L. Minder and A. Shokrollahi. Cryptanalysis of the Sidelnikov cryptosystem. In *Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 347–360, Barcelona, Spain, 2007.

[53] R. Misoczki and P. S. L. M. Barreto. Compact McEliece Keys from Goppa Codes. In *Selected Areas in Cryptography (SAC 2009)*, Calgary, Canada, Aug. 13-14 2009.

[54] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto. MDPC-McEliece : New McEliece Variants from Moderate Density Parity-Check Codes. *IACR Cryptology ePrint Archive*, 2012 :409, 2012.

[55] N. Nethercote et al. *Cachegrind : a cache and branch-prediction profiler*, 2002.

[56] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2) :159–166, 1986.

[57] A. Otmani, J. Tillich, and L. Dallot. Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes. *Special Issues of Mathematics in Computer Science*, 3(2) :129–140, Jan. 2010.

[58] A. Otmani and J.-P. Tillich. An Efficient Attack on All Concrete KKS Proposals. In B.-Y. Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 98–116. Springer, 2011.

[59] R. Overbeck. *Public Key Cryptography based on Coding Theory.* Phd thesis, Technische Universität Darmstadt, 2007.

[60] R. Overbeck and N. Sendrier. Code-based cryptography. In D. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-Quantum Cryptography*, pages 95–145. Springer, 2009.

[61] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2) :203–207, Mar. 1975.

[62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions*, IT-8 :S5–S9, 1962.

[63] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2) :120–126, Feb. 1978.

[64] N. Sendrier. Finding the permutation between equivalent codes : the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4) :1193–1203, July 2000.

[65] N. Sendrier. Decoding One Out of Many. In B.-Y. Yang, editor, *PQ-Crypto 2011*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011.

[66] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5) :1484–1509, 1997.

[67] V. Sidelnikov. A public-key cryptosystem based on Reed-Muller codes. *Discrete Mathematics and Applications*, 4(3) :191–207, 1994.

[68] V. Sidelnikov and S. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 1(4) :439–444, 1992.

[69] J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1989.

[70] V. G. Umana and G. Leander. Practical Key Recovery Attacks On Two McEliece Variants, 2009. IACR Cryptology ePrint Archive 509.

[71] D. Wagner. A Generalized Birthday Problem. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.

[72] V. K.-W. Wei and K. Yang. On the generalized Hamming weights of product codes. *Trans. Inf. Theory*, 39(5) :1709–1713, 1993.

[73] C. Wieschebrink. Two NP-complete Problems in Coding Theory with an Application in Code Based Cryptography. In *2006 IEEE International Symposium on Information Theory*, pages 1733–1737, 2006.

[74] C. Wieschebrink. Cryptanalysis of the Niederreiter Public Key Scheme Based on GRS Subcodes. In *PQCrypto*, pages 61–72, 2010.