Implantation sécurisée de protocoles cryptographiques basés sur les codes correcteurs d'erreurs

Tania Richmond

Thèse

pour obtenir le grade de docteur
de l'Université Jean Monnet

Discipline : **Informatique**

Équipe : **Systèmes Embarqués Sécurisés et Architectures Matérielles**

Implantation sécurisée de protocoles cryptographiques basés sur les codes correcteurs d'erreurs

Tania Richmond

Thèse soutenue publiquement le 24 Octobre 2016 devant le jury suivant :

Rapporteurs :

Marine Minier	**Loria, France**
Arnaud Tisserand	**IRISA, France**

Directeur de thèse :

Viktor Fischer	**LaHC, France**

Co-encadrant de thèse :

Pierre-Louis Cayrel	**LaHC, France**

Examinateur :

Jean-Claude Bajard	**LIP6, France**

« La vraie faute est celle qu'on ne corrige pas. »
(Confusius)

À Bernard et Pépé Michel,
qui n'auront pas eu le temps de voir cette thèse aboutir.

Remerciements

La thèse est synonyme de dépression. On a des hauts et des bas, et c'est là qu'un soutien de la part des collègues, de la famille et des amis s'avère primordial. C'est la raison pour laquelle des remerciements s'imposent.
La thèse, c'est comme une sortie en mer. Le temps peut changer d'un moment à l'autre, il est difficile de garantir ce que l'on va y trouver, on peut se perdre ou s'échouer, un coup de soleil est vite attrapé, bref, on sait quand on part mais ni exactement quand on revient ni forcément comment.

Je commencerai donc par remercier Viktor Fischer et Pierre-Louis Cayrel pour avoir accepté de m'embarquer dans cette aventure de pirates. On n'arrête jamais d'apprendre et j'ai beaucoup appris à vos côtés, en électronique avec Viktor et en cryptographie basée sur les codes avec Pierre-Louis. Merci également d'avoir pris sur vos temps personnels, surtout ces derniers mois, afin que cette thèse puisse aboutir sans s'éterniser davantage.
Je souhaite également dire un grand merci pour la lourde responsabilité que Marine Minier et Arnaud Tisserand ont acceptée en étant rapporteurs de cette thèse, et de l'avoir fait malgré les délais très tendus. Je remercie également Jean-Claude Bajard pour avoir non seulement voulu assister à ma soutenance, mais en plus m'accorder le privilège d'être membre de mon jury. Je remercie aussi son premier thésard, Laurent-Stéphane Didier, pour son déplacement depuis Toulon afin d'assister à ma soutenance, même en n'étant pas membre du jury.

Cette thèse s'est effectuée au laboratoire Hubert Curien, dans l'équipe SESAM (anciennement CAT), dont je remercie chaleureusement les membres : Alain Aubert (pour le temps consacré à essayer de comprendre mon charabia de "matheuse" et nos discussions sur l'électronique et les réseaux), Florent Bernard (notamment pour m'avoir aidée à faire le lien entre les mathématiques et l'électronique, ainsi que les repas chez toi), Nathalie Bochard (pour avoir répondu à mes questions sur des termes français ou régionaux que je ne connaissais pas), Lilian Bossuet, Robert Fouquet, Luboš Gaspar (pour tes conseils et ta visite à Košice), Pierre Bayon (pour ton aide sur les tentatives d'attaques), Abdelkarim Cherkaoui, Patrick Haddad (pour tes conseils de grand frère thésard), Cédric Marchand (pour la via ferrata), Brice Colombier (capitaine de l'équipe de foot), Vlad Dragoi (pour notre collaboration). Je passe maintenant à ceux qui ne font pas partie de l'équipe, mais au moins de la pause café et de l'équipe de foot. Je remercie donc Olivier Alata, Cécile Barat, Anne-Claire Bayard-Legrand, Loïc Denis et Christophe Ducottet. L'équipe administrative fait également un travail formidable et je remercie en particulier Claude Aubry (pour m'avoir permise d'entrer dans le labo les trois premiers mois et nos discussions sur les îles du Pacifique), Julie Debiesse, Anne-Laure Rozier et Patrick Vincent.

For no-French speakers, I will write in English :
I would like to thank Miloš Drutarovský (to allow me to stay three months in Košice in Slovakia), Martin Petrvalský (for our work together and helpful advices to discover your country) and the Department of Electronics and Multimedia Communications at the Technical University of Košice, in particular Ludmila Maceková (for our swimming sessions and to have shared with me your history) and Mária Gamcová (for

our ballet and opera evenings with our daughter).
I would also like to thank Oto Petura and your wife Eugenika (for your kindness, welcome in Slovakia and to spend time together, it was my great pleasure to meet you), Miroslav Repko (for traveling through all Slovakia with me, to make me feel better and to spend time together with your friends and family), Cuauhtemoc Mancillas-López and Brisbane Ovilla Martínez for our conversations in Spanglish.

Une thèse de doctorat est l'aboutissement d'un travail de plusieurs années à réfléchir sur un même sujet, mais aussi à de nombreuses années de spécialisation après le baccalauréat. C'est pourquoi, je souhaite remercier des personnes sans lesquelles mon parcours n'aurait pas été le même. Je commencerai par l'Université de Nouvelle-Calédonie : Éric Edo (pour avoir mentionné la cryptographie comme une application de l'arithmétique en première année de Licence, sans quoi je n'aurais pas eu la curiosité de chercher à savoir ce que c'était et avoir accepté de m'encadrer sur un projet en cryptographie en fin de Licence), Gilles Taladoire (pour m'avoir initiée à l'algorithmique et la programmation, ce qui m'a ouvert les portes de l'informatique) et Nazha Selmaoui-Folcher (pour ses cours de mathématiques pour l'informatique, la réunion des deux me plaisant beaucoup j'ai cherché à les combiner).
Ensuite, je souhaite remercier l'ensemble des enseignants du Master Cryptologie et Sécurité Informatique à l'Université de Bordeaux. Je suis très heureuse d'avoir eu le courage de quitter le Caillou et d'avoir suivi cette formation. Je remercie en particulier Gilles Zémor, pour m'avoir encadrée sur des projets passionnants et continuer de m'apprendre des choses en cryptologie et théorie des codes.

Je ne me serais peut-être pas tourner vers la recherche sans le stage de Master 1 dans l'équipe GRACE (anciennement TANC) du Laboratoire d'Informatique de l'École Polytechnique. Je souhaite donc remercier l'équipe et plus particulièrement Daniel Augot (pour m'avoir proposé ce stage qui fût très enrichissant), Morgan Barbier (pour ton encadrement, ton soutien, ton suivi de mon devenir depuis et ton invitation au séminaire de Caen) et Guillaume Quintin (pour tes conseils en programmation et en théorie des codes). Arrivé après, je remercie également Alain Couvreur pour ses conseils utiles sur les codes de Goppa notamment.

Mon histoire avec le laboratoire IMATH a commencé avec mon stage de Master 2. Je remercie donc très sincèrement Pascal Véron pour avoir accepté de m'encadrer, m'avoir ouvert la porte de la cryptographie basée sur les codes, de continuer à me supporter, répondre à mes nombreuses questions et me conseiller, enfin pour être un exemple de Maître de Conférence en Informatique.
Je remercie l'IMATH pour son chaleureux accueil à chacune de mes visites et m'avoir permis de finir ma thèse dans de bonnes conditions : Jean-Jacques Alibert (ne lui parlez pas de moto, de course à pieds ou de parpaing), Thierry Astruc, Yves Aubry (alias "Georges"), Guy Bouchitté (pour faire en sorte que cette jeunesse travaille un peu), Thierry Champion (pour ne faire de la recherche que de 17h à 17h30 avec Guy), Laurent-Stéphane Didier (pour tes histoires abracadabrantes et ton suivi de mon devenir), Mehmet Ersoy, Gloria Faccanoni (la reine de LaTeX), Cédric Galusinski (pour être un directeur de labo décontracté), Valérie Gillot (comme une maman), Frédéric Golay (pour travailler presque allongé), Fabien Herbaut (pour tes encouragements et ton adorable petite famille), Philippe Langevin (pour ramener un peu de rigueur mathématique dans l'équipe), Nicolas Méloni (pour ton accueil, tes relectures et les soirées fléchettes ou attaque de zombies), Sofiane Meradji (ne

lui parlez pas de foot, pour ta gentillesse), Christian Nguyen (pour les soirées jeux de plateau), Antonin Novotny (pour exploser des records de nombres de publications), Catherine Pideri (Catherine, ... c'est Catherine !), Jacques Schneider, Pierre Seppecher, Jacques Wolfmann (pour l'exemple en tant que passionné des codes correcteurs d'erreurs), Lyudmila Yushchenko, Jean-Pierre Zanotti (pour tes conseils en pédagogie), Thomas Altazin, Christophe Bourel, Serhii Dyshko, Didier Jesslé, Jalal Lakhlili, Ilaria Lucardesi, David Maltese, Giuseppe Rosi. Merci au duo "déjanté" (Fabinou et Nico) pour m'avoir traitée de "PrincesseBurger", offert un "BurgerAnniversaire" et bien d'autres délires, surtout le mardi. Ce laboratoire est sans conteste comme une grande famille.

Je remercie la communauté du groupe de travail Codage et Cryptographie ainsi que celle du groupe d'Arithmétique pour toutes les conférences, qui sont à chaque fois un réel plaisir, notamment les journées C2.

Je souhaite remercier les amis, notamment Marion Candeau (pour tes conseils et encouragements), Iuliia Tkachenko (pour être mon binôme, tes encouragements et nos week-ends filles, à découvrir Montpellier et ses environs), Jérôme Milan (pour tes conseils, encouragements et nos sorties musées). Je remercie également la bande du lycée (et leurs familles), toujours en contact : les familles Ferrand et Jacono, la famille Chevrot et en particulier Ken, Alexandre Guiot, la famille Vincent et en particulier Ludovic, la famille Sinem et en particulier Xavier (pour ta sagesse). Je remercie très sincèrement Hugo Ferrand (pour continuer à m'encourager quoi qu'il arrive).

Je remercie la famille (qui m'a vue partir à l'autre bout du monde il y a 6 ans), notamment mes parents, Marraine et mes grands-parents. Je remercie très sincèrement ma sœur, Vaïai (alias "Pierre Richard", pour ton soutien, ta compréhension et tes tentatives de faire ressortir mon côté femme).

Enfin merci à ceux que je n'ai pas cités (toutes mes excuses !), mais qui ont quand même joué un rôle ces dernières années.

Résumé

Le premier protocole cryptographique basé sur les codes correcteurs d'erreurs a été proposé en 1978 par Robert McEliece. La cryptographie basée sur les codes est dite post-quantique car il n'existe pas à l'heure actuelle d'algorithme capable d'attaquer ce type de protocoles en temps polynomial, même en utilisant un ordinateur quantique, contrairement aux protocoles basés sur des problèmes de théorie des nombres. Toutefois, la sécurité du cryptosystème de McEliece ne repose pas uniquement sur des problèmes mathématiques. L'implantation, logicielle ou matérielle, a également un rôle très important pour sa sécurité et l'étude de celle-ci face aux attaques par canaux auxiliaires/cachés n'a débuté qu'en 2008. Des améliorations sont encore possibles.

Dans cette thèse, nous proposons de nouvelles attaques sur le déchiffrement du cryptosystème de McEliece, utilisé avec les codes de Goppa classiques, ainsi que des contre-mesures correspondantes. Les attaques proposées sont des analyses de temps d'exécution ou de consommation d'énergie. Les contre-mesures associées reposent sur des propriétés mathématiques et algorithmiques. Nous montrons qu'il est essentiel de sécuriser l'algorithme de déchiffrement en le considérant dans son ensemble et non pas seulement étape par étape.

Table des matières

Notations

Afin d'unifier les notations dans cette thèse, les conventions suivantes ont été admises.

Nombres	κ	:	paramètre de sécurité d'un cryptosystème
	n	:	longueur d'un code linéaire
	k	:	dimension d'un code linéaire
	r	:	co-dimension d'un code linéaire ($= n - k$)
	d	:	distance minimale d'un code linéaire
	p	:	entier premier
	q	:	puissance de p
	m	:	paramètre d'un code de Hamming, degré de l'extension de corps pour les codes de Goppa
	t	:	capacité de correction d'un code linéaire, degré du polynôme de Goppa
	ε	:	poids d'un vecteur erreur
	w_H	:	poids de Hamming d'un mot
	d_H	:	distance de Hamming
	ν_q	:	cardinal de la boule de Hamming
	ϱ	:	rendement d'un code linéaire
	ρ	:	rayon de recouvrement d'un code linéaire
	λ_i	:	scalaires dans le corps de base $\mathbb{F}_q$
	μ_i	:	nombre de mots de code de poids i
	a_i	:	coordonnée i du vecteur A
	b_i	:	coordonnée i du vecteur B
	c_i	:	coordonnée i du vecteur C
	x	:	variable d'un polynôme ($\in \mathbb{F}_q$)
	y	:	variable d'un polynôme ($\in \mathbb{F}_q$)
	$h_{i,j}$	:	composante i, j de la matrice $\mathcal{H}$
	α_i	:	élément i du support de Goppa $\mathscr{L}$
	β	:	racine primitive de $\mathbb{F}_{2^m}$

Vecteurs	M	:	message (en clair, $\in \mathbb{F}_q^k$)
	A	:	mot de l'espace ambiant quelconque ($\in \mathbb{F}_q^n$)
	B	:	message encodé (mot de l'espace ambiant)
	E	:	vecteur d'erreurs (mot de l'espace ambiant)
	C	:	message encodé avec des erreurs (mot de l'espace ambiant),
	$\tilde{C}$	:	texte chiffré (pour McEliece)
	X	:	variable des polynômes ($\in \mathbb{F}_q^n$) variable du polynôme de Goppa
	H_i	:	ligne i de la matrice $\mathcal{H}$
	H_j	:	colonne j de la matrice $\mathcal{H}$
	$S_{\mathcal{H}}(C), S_C$	:	syndrome de C (calculé avec la matrice $\mathcal{H}$)
Polynômes	G	:	polynôme générateur d'un code cyclique, polynôme de Goppa (privé pour McEliece)
	H	:	polynôme de contrôle d'un code cyclique
	P	:	polynôme
	σ	:	polynôme localisateur d'erreurs
	η	:	polynôme évaluateur d'erreurs
	$\mathfrak{a}$	:	somme des monômes de puissances paires de σ
	$\mathfrak{b}$	:	somme des monômes de puissances impaires de σ
	S	:	polynôme syndrome
	T	:	inverse du syndrome
	R	:	polynôme de la racine de $T(X) + X$ (dans Patterson)
	$W_{\mathscr{C}}$	:	énumérateur de poids du code $\mathscr{C}$
Matrices	$\mathcal{G}$	:	matrice génératrice d'un code linéaire (privée pour McEliece)
	$\mathcal{I}_k$	:	matrice identité de taille $k \times k$
	$0_{m,n}$	:	matrice nulle de taille $m \times n$
	$\mathcal{A}$	:	partie redondance de la matrice $\mathcal{G}$ sous forme systématique
	$\tilde{\mathcal{G}}$	:	matrice génératrice d'un code linéaire (publique pour McEliece)
	$\mathcal{H}$	:	matrice de parité d'un code linéaire
	$\mathcal{X}$	:	matrice inversible (triangulaire inférieure des coefficients du polynôme de Goppa)
	$\mathcal{Y}$	:	matrice de Vandermonde (de t lignes et n colonnes) (des éléments du support de Goppa)
	$\mathcal{Z}$	:	matrice diagonale (des inverses des évaluations du polynôme de Goppa sur les éléments du support)
	$\bar{\mathcal{H}}$	:	produit $\mathcal{Y} \cdot \mathcal{Z}$ (matrice de parité d'un code de Goppa vu comme un code alternant)
	$\hat{\mathcal{H}}$	:	produit $\mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$ (matrice de parité d'un code de Goppa vue comme la matrice donnant le fameux syndrome)
	$\mathcal{S}$	:	matrice carrée inversible (de confusion, privée pour McEliece)
	$\mathcal{Q}$	:	matrice carrée inversible (de confusion, privée pour Niederreiter)
	$\mathcal{P}$	:	matrice de permutation (carrée inversible, privée pour McEliece et Niederreiter)

Ensembles	$\mathbb{N}$	:	entiers naturels
	$\mathbb{F}$	:	corps fini
	$\mathscr{C}$	:	code linéaire
	$\mathscr{C}^{\perp}$	:	code dual
	$\mathscr{M}$	:	ensemble de matrices
	Γ	:	code de Goppa
	s_k	:	clé privée (de McEliece)
	p_k	:	clé publique (de McEliece)
	$\mathscr{L}$	:	support de Goppa
	$\mathcal{B}(B,t)$	:	boule de Hamming (de centre B et de rayon t)
	$\mathscr{W}_{n,t}$	:	ensemble des mots de longueur n et de poids t
	$\mathscr{R}_t$	:	ensemble des racines d'un polynôme scindé P de degré t
Opérateurs	$\frac{d}{dX}$	:	dérivée par rapport à la variable X
	$\oplus$	:	opérateur OU EXCLUSIF
	$\vert$	:	opérateur OU
	&	:	opérateur ET (bit à bit)
	$\wedge$	:	opérateur ET
	$>> i$	:	décalage de i rang vers la droite
	$\triangleq$	:	par définition
Algorithmes	$KeyGen$	:	génération de clé(s) (*Key Generation* en Anglais)
	Enc	:	chiffrement (*Encryption* en Anglais)
	Dec	:	déchiffrement (*Decryption* en Anglais)
	$Enco$	:	encodage (*Encoding* en Anglais)
	$Deco$	:	décodage (*Decoding* en Anglais)

Introduction

La cryptologie est l'art de dissimuler un message. Le concept est apparu dans l'Antiquité, notamment avec le chiffrement de César. La cryptologie comprend la cryptographie, domaine où l'on cherche des techniques sûres pour cacher une information (exemple : décalage des lettres dans l'alphabet pour le chiffrement de César), et la cryptanalyse, domaine où l'on cherche des failles dans les techniques utilisées pour cacher un message afin de retrouver de l'information, partielle ou totale, sur le message ou le secret (exemple : deviner le décalage utilisé dans le chiffrement de César).

On appelle codage une méthode de transformation qui convertit la représentation d'une information en une autre. La théorie des codes sert à détecter le bruit ajouté à un message (les erreurs) lors d'une transmission sur un canal dit bruité (où des erreurs de transmission sont possibles) et à corriger ce bruit. Pour faire cela, l'idée est de rajouter de la redondance d'information (à celle du message initial), afin de pouvoir repérer les erreurs, quand il n'y en a pas trop, et de les corriger.

Nous nous intéresserons uniquement aux codes linéaires dans cette thèse. Cela signifie que la redondance est linéairement dépendante de l'information initiale contenue dans le message.
Un code correcteur linéaire possède un algorithme d'encodage simple, en temps polynomial (raisonnable). Le problème du décodage d'un code aléatoire quant à lui, a été démontré NP-difficile en 1978 dans [BMvT78]. En revanche, il devient plus facile pour certaines classes de codes dont il existe des algorithmes de décodage en temps polynomial, d'où la construction et l'étude de ces classes.

Une implantation est, d'un point de vue logiciel, la traduction d'un algorithme dans un langage de programmation, d'un point de vue matériel, la description d'un circuit dans un langage (de description). L'implantation d'un cryptosystème est un compromis entre la sécurité et l'efficacité. C'est pourquoi, la cryptanalyse et l'implantation doivent être considérées simultanément. Selon les méthodes algorithmiques et le support que nous choisissons d'utiliser, l'implantation et par conséquent les attaques, ne seront pas les mêmes.

Une attaque par canal auxiliaire est une attaque qui exploite les lois de la physique afin d'obtenir de l'information contenue dans un canal associé à une implantation, un circuit (pour extraire des secrets manipulés par des cartes à puce ou des processeurs cryptographiques par exemple). Le canal auxiliaire n'a pas pour but de transmettre volontairement de l'information. Il laisse s'échapper de l'information, qu'un observateur doit savoir interpréter pour la comprendre. En revanche, des contre-mesures algorithmiques ou physiques peuvent être proposées pour bloquer ou limiter l'attaque.
Les principales attaques par canaux auxiliaires sont l'attaque temporelle et l'attaque par consommation d'énergie. La première exploite les variations de temps de traitement d'un programme, qui dépendent de la taille des données. La deuxième exploite de façon instantanée la consommation d'énergie du système dans sa globalité, qui dépend des traitements à effectuer sur les données. Une attaque semblable est l'attaque électromagnétique. Elle exploite les émanations d'une zone d'un circuit en

fonction des traitements qui y sont effectués sur les données.

De manière générale, il y a deux concepts d'attaques en cryptanalyse : les attaques passives, lors desquelles un attaquant ne fait qu'observer ce qu'il se passe, en prenant des mesures si nécessaire, mais sans intervenir ; et les attaques actives, lors desquelles un attaquant peut modifier ou contrôler le texte chiffré (créer une faute) afin de comprendre ce qu'il se passe dans le programme ou le circuit. On appelle cela une attaque par injection de faute.

Les protocoles cryptographiques basés sur les codes correcteurs d'erreurs sont dits post-quantiques. Les protocoles post-quantiques doivent leur nom à leur supposée résistance face aux attaques réalisables avec un ordinateur quantique. Ces attaques [Sho94] permettraient de résoudre en temps polynomial les problèmes de la factorisation [RSA78] et du logarithme discret [JMV01] (problèmes à la base de nombreux cryptosystèmes actuels). Il est communément admis que les protocoles à clef publique basés sur des problématiques de la théorie des codes correcteurs d'erreurs [BMvT78] résisteraient à ces attaques quantiques. (Il n'existe pas, à l'heure actuelle, d'algorithme qui résolve ces problèmes en temps polynomial.)

Les protocoles cryptographiques basés sur les codes correcteurs d'erreurs sont à la fois rapides (en théorie) et possèdent une excellente complexité théorique. Il s'agit d'un domaine de recherche encore jeune qui provient de solides bases historiques.
La communauté cryptographique s'est principalement intéressée aux protocoles basés sur les problèmes de la théorie des nombres. De plus, le principe des attaques par canaux auxiliaires n'est apparu qu'en 1996 [Koc96]. En revanche, le premier protocole de chiffrement à clé publique basé sur les codes correcteurs d'erreurs date d'il y a bientôt une quarantaine d'années [McE78], mais l'étude de ce protocole face à ce type d'attaques n'a commencé qu'il y a un peu moins de dix ans [STM+08]. De nombreux protocoles cryptographiques également basés sur les problèmes de la théorie des codes ont été proposés depuis [Nie86, Ste94, Vér97, KKS97, CFS01, BS08, CVEYA11, BCS13].

Cette thématique a de nombreuses perspectives et retombées tant en France qu'à l'international. L'innovation qu'apportent de tels cryptosystèmes est forte car un avènement de l'ordinateur quantique rendrait vulnérable tous les autres moyens de sécurisation des données. L'entreprise américaine CISCO, spécialisée dans le matériel de réseaux informatiques, a d'ailleurs annoncé qu'avec les avancées actuelles de la recherche, il ne faudra attendre qu'une dizaine d'années avant que l'ordinateur quantique ne voie le jour [EF09, 6ème prédiction technologique]. De plus, nous ne sommes pas à l'abri d'une grande avancée sur le problème du logarithme discret [BGJT14]. Tout ceci reste encore à vérifier, mais quand on sait à quel point la sécurité des données est importante, on peut comprendre les risques encourus si des alternatives ne sont pas rapidement développées. Enfin, l'Agence Nationale de Sécurité américaine (*National Security Agency* (NSA) en Anglais) [NSA15] a déclaré sur son site officiel, le 19 Août 2015, que les sociétés et consommateurs qui veulent se mettre à la cryptographie peuvent utiliser les protocoles actuels (avec les paramètres appropriés), mais qu'ils devront se préparer à une migration vers des protocoles résistants aux attaques par ordinateur quantique.

L'objectif de cette thèse est de fournir des éléments nécessaires à la concep-

tion et l'implantation de protocoles cryptographiques basés sur les codes correcteurs d'erreurs, afin de les sécuriser. Une implantation est dite sécurisée si elle est résistante face aux attaques par canaux auxiliaires. Plusieurs implantations, partielles ou complètes (dans différents langages), ont servi pour cette thèse et cela sur divers supports (logiciels et matériels). Le travail a consisté à analyser le comportement de ces implantations face à des attaques par canaux auxiliaires.

Après une introduction à la cryptologie et à la théorie des codes dans la Partie I, nous détaillerons dans la Partie II les protocoles de chiffrement basés sur les codes correcteurs d'erreurs, leurs implantations et les attaques par canaux auxiliaires de l'état de l'art, avant de finir par la présentation de mes résultats dans la Partie III et de conclure cette thèse.

Première partie

La cryptologie et les codes correcteurs d'erreurs

Chapitre 1

Cryptologie

La cryptologie est l'art de dissimuler un message. Le concept est apparu dans l'Antiquité avec César notamment. La cryptologie comprend la cryptographie et la cryptanalyse. La cryptographie est le domaine où l'on cherche des techniques sûres pour cacher une information. La cryptanalyse est le domaine où l'on cherche des failles dans les techniques utilisées pour cacher un message afin retrouver de l'information, partielle ou totale, sur celui-ci par exemple. Nous donnons, dans la suite de ce chapitre, des définitions plus formelles.

Définition 1.1 (Cryptologie) *(Étymologiquement, la cryptologie signifie science du secret.) C'est l'ensemble des principes, méthodes et techniques dont l'application assure le chiffrement et le déchiffrement des données, afin d'en préserver la confidentialité, l'intégrité et l'authenticité. Elle englobe deux approches complémentaires que nous définissons ci-après : la cryptographie et la cryptanalyse.*

1.1 Cryptographie

Définition 1.2 (Cryptographie) *(Étymologiquement, la cryptographie signifie écriture secrète.) Il s'agit de transformer un message clair en un message chiffré, de l'étude et de la conception de procédés de chiffrement. On dit aussi que l'on rend inintelligible le message.*

Les objectifs de la cryptographie sont d'assurer la protection de l'information de différentes manières :

- Confidentialité : Garantir le caractère secret de l'information (mécanisme pour transmettre des données de telle sorte que seul le destinataire autorisé puisse les lire).
- Intégrité : Empêcher la modification de l'information (mécanisme pour s'assurer que les données reçues n'aient pas été modifiées durant la transmission).
- Authenticité : Garantir l'origine de l'information (mécanisme pour permettre d'identifier des personnes ou des entités et de certifier cette identité).
- Non-répudiation : Mécanisme pour enregistrer un acte ou un engagement d'une personne ou d'une entité de telle sorte que celle-ci ne puisse pas nier avoir accompli cet acte ou pris cet engagement.

Les principes de base de la cryptographie sont dûs à Kerckhoffs [Ker83] :

- Compromis entre la performance et la sécurité : Un protocole sûr vaut mieux qu'un protocole efficace.
- La simplicité : Un protocole ne doit jamais essayer de faire plus que ce qu'il est sensé faire.
- Le maillon faible : Un protocole n'est jamais plus sûr que sa composante la plus faible.
- Le raisonnement paranoïaque : Un protocole avec une faiblesse, aussi petite soit-elle, est un protocole qui n'est plus sûr.

— Le modèle de sécurité : Un protocole n'est jamais parfait. L'essentiel est d'obtenir le niveau de sécurité souhaité.

Ces principes font ressortir l'importance, d'une part de la partie algorithmique (c'est-à-dire la complexité calculatoire d'un problème ainsi que la rapidité à le résoudre en utilisant la bonne clé) et d'autre part de la partie implantation (c'est-à-dire l'adaptation de l'algorithme dans un environnement donné).
La cryptographie est principalement connue pour les cryptosystèmes (ou systèmes de chiffrement) mais ce ne sont pas les seuls protocoles. Nous les développerons juste après. Le chiffrement permet d'assurer la confidentialité lors d'une communication. Afin d'expliquer le fonctionnement des différents protocoles cryptographiques, nous prendrons les personnages Alice et Bob comme acteurs de nos scénarios de communication.

1.1.1 Fonction à sens unique et fonction à trappe

Définition 1.3 (Fonction à sens unique) *Une fonction à sens unique est une fonction facile à calculer dans un sens, mais difficile à inverser.*

Définition 1.4 (Fonction à trappe) *Une fonction à trappe est une fonction à sens unique qui devient facile à inverser lorsqu'on connaît une information secrète en plus.*

Les fonctions à sens-unique suffisent pour montrer l'existence de plusieurs primitives cryptographiques :
— signature
— générateurs pseudo-aléatoires
— code d'authentification de message (*Message Authentication Code* (MAC) en Anglais)

Pour le chiffrement, il faut aussi une trappe.

1.1.2 Chiffrement symétrique

Historiquement, la première forme de cryptographie semble être apparue dans l'Antiquité avec le chiffrement de César. Le chiffrement symétrique est également appelé chiffrement à clé secrète. Nous allons voir pourquoi.
Un système de chiffrement symétrique est constitué de trois algorithmes :

Génération de clé (*Key generation* en Anglais) : C'est un algorithme probabiliste qui prend en entrée un paramètre de sécurité κ et produit une clé privée s_k. On note $KeyGen(\kappa) = s_k$.

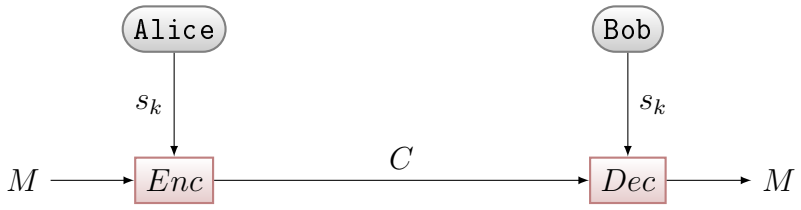
Chiffrement (*Encryption* en Anglais) : C'est un algorithme déterministe qui prend en entrées un message clair M et la clé secrète s_k, et produit le message chiffré c. On note $Enc(M, s_k) = C$.

Déchiffrement (*Decryption* en Anglais) : C'est un algorithme déterministe qui prend en entrées un chiffré C et la clé secrète s_k, et retourne un message clair M. On note $Dec(C, s_k) = M$.

Un système symétrique est donc un protocole cryptographique qui a les paramètres suivants : (M, C, s_k, Enc, Dec), où M est un message à chiffrer, C est le chiffré correspondant à M, s_k est la clé privée utilisée pour chiffrer M en C puis pour déchiffrer C en M, Enc est l'algorithme de chiffrement et Dec est l'algorithme de déchiffrement. Un système de chiffrement est dit symétrique si c'est une même clé

qui est utilisée pour chiffrer et déchiffrer (la génération de clé produit une seule clé) : $KeyGen(\kappa) = s_k$. La Figure 1.1 est une représentation schématique d'un système de chiffrement symétrique.

FIGURE 1.1 – Système de chiffrement symétrique



Le principal inconvénient à la cryptographie à clé secrète est l'échange de clés. Une solution est apparue il y a une quarantaine d'années [DH76], donnant naissance à un nouveau type de chiffrement.

1.1.3 Chiffrement asymétrique

Si le concept de chiffrement est antique, celui du chiffrement asymétrique en revanche est plus moderne, puisqu'il a été proposé en 1976 par Diffie et Hellman [DH76]. Le chiffrement asymétrique est également appelé chiffrement à clé publique. Nous allons voir pourquoi.
Un système de chiffrement asymétrique est constitué de trois algorithmes :

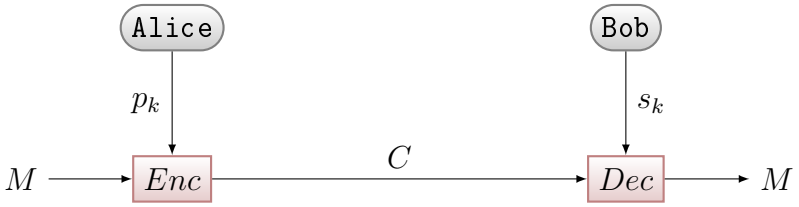
Génération de clés : C'est un algorithme probabiliste qui prend en entrée un paramètre de sécurité κ et produit un couple de clés (p_k, s_k), respectivement publique et privée. On note $KeyGen(\kappa) = (p_k, s_k)$.

Chiffrement : C'est un algorithme déterministe qui prend en entrées un message clair M et la clé publique p_k, et produit le message chiffré C. On note $Enc(M, p_k) = C$.

Déchiffrement : C'est un algorithme déterministe qui prend en entrées un chiffré C et la clé privée s_k, et rend en sortie un message clair M. On note $Dec(C, s_k) = M$.

Un système asymétrique est donc un protocole cryptographique qui a les paramètres suivants : $(M, C, (p_k, s_k), Enc, Dec)$, où M est un message à chiffrer, C est le chiffré correspondant à M, p_k est la clé publique utilisée pour chiffrer M en C, s_k est la clé privée utilisée pour déchiffrer C en M, Enc est l'algorithme de chiffrement et Dec est l'algorithme de déchiffrement. Un système de chiffrement est dit asymétrique si une clé sert à chiffrer et une autre clé associée à la première sert à déchiffrer (la génération de clé produit un couple de clés) : $KeyGen(\kappa) = (p_k, s_k)$. La Figure 1.2 est une représentation schématique d'un système de chiffrement symétrique.

FIGURE 1.2 – Système de chiffrement asymétrique



Remarque 1.1 *D'après [MVOV96], pour éviter toute ambiguïté, une convention communément faite est d'utiliser le terme de clé privée en association avec les cryptosystèmes dits à clé publique, et le terme clé secrète en association avec les cryptosystèmes symétriques :*
Il faut deux parties ou plus pour partager un secret, mais une clé est vraiment privée lorsqu'une seule partie la connaît.

Revenons sur certains termes employés dans un système de chiffrement. Alice chiffre son message en clair (*plaintext* en Anglais) pour obtenir le chiffré (ou cryptogramme, *ciphertext* en Anglais), qu'elle envoie à Bob. Bob déchiffre le chiffré pour obtenir un message en clair (*cleartext* en Anglais). Si le message déchiffré par Bob est le même que le message chiffré par Alice, alors il n'y a pas eu d'erreur. (Une différence est faite dans la terminologie anglaise, mais pas toujours dans la française.) Un attaquant, que nous appellerons Ève, qui regarde les communications pour chercher à obtenir de l'information sans autorisation, va essayer de décrypter le chiffré pour obtenir le message ou la clé. La différence entre Bob et Ève et, a fortiori entre déchiffrer et décrypter, c'est que Bob possède la clé pour le déchiffrement et pas Ève. Contrairement à la cryptographie à clé secrète, la cryptographie à clé publique n'a pas de problème d'échange de clés. En revanche, un annuaire de clés publiques est nécessaire et les questions de stockage et de non divulgation de la clé privée lors du chiffrement et du déchiffrement demeurent.
La génération d'un couple de clés pour un système de chiffrement asymétrique est basée sur des fonctions à trappe, facile à calculer dans un sens et difficile pour la réciproque, sauf si on connaît la trappe (le secret). En effet, il doit être difficile pour un attaquant de retrouver s_k même s'il connaît p_k mais l'une est liée à l'autre.

1.1.4 Autres types de protocoles cryptographiques

Nous nous intéresserons particulièrement au chiffrement asymétrique par la suite, mais nous décrivons brièvement dans cette sous-section les autres principaux types de protocoles cryptographiques afin de compléter les généralités.

Signature

La signature est un principe cryptographique qui assure l'authentification de l'expéditeur, l'intégrité des données signées et la non-répudiation par l'expéditeur. L'équivalent non numérique est par exemple la signature manuelle sur un document administratif, car elle a les mêmes objectifs. Un protocole de signature est composé de trois algorithmes : la génération de clé (couple clé publique - clé privée), la signature et la vérification. Le principe est grossièrement l'inverse de celui du chiffrement à clé publique. Alice possède deux clés p_k et s_k respectivement publique et privée.

Elle utilise sa clé privée s_k pour signer son message et l'envoie à Bob. Pour vérifier que c'est bien Alice qui a signé le message reçu, Bob utilise la clé publique p_k d'Alice avec l'algorithme de vérification.

Identification

L'identification est un principe cryptographique qui assure uniquement l'authentification de l'expéditeur, contrairement à la signature qui assure en plus l'intégrité du message et la non-répudiation par l'expéditeur. Les termes employés pour un protocole d'identification sont les suivants : Alice utilise un algorithme pour prouver son identité auprès de Bob. Alice est appelée prouveur. Bob utilise un algorithme pour vérifier l'identité d'Alice. Bob est appelé vérifieur.

Fonction de hachage

Une fonction de hachage permet de passer d'un message de taille variable à un bloc de taille fixée au départ, comme une empreinte du message. Le résultat est appelé haché. Les principales propriétés d'une fonction de hachage sont :

— la résistance à la détermination d'une (première) pré-image : Étant donné un haché, il n'est pas possible de retrouver un antécédent de ce haché.
— la résistance à la détermination d'une deuxième pré-image : Étant donnés un message et son haché, il n'est pas possible de trouver un deuxième message qui ait le même haché que le premier message.
— la résistance aux collisions : Il n'est pas possible de retrouver deux messages distincts ayant un même haché.

Les fonctions de hachage sont très souvent utilisées en cryptographie par rapport à leurs propriétés. Une fonction de hachage doit être difficile à inverser. Le problème est qu'une fonction de hachage n'est pas injective et qu'elle est soumise au paradoxe des anniversaires. Une fonction est injective si, quelques soient deux éléments de l'ensemble de définition de la fonction, leurs images sont égales implique que ces éléments sont égaux. Le paradoxe des anniversaires est la probabilité que, dans une liste de i éléments tirés aléatoirement et indépendamment dans un ensemble à N éléments, cette liste comporte deux fois le même élément. Pour illustrer ce phénomène, prenons l'exemple suivant : On souhaite connaître le nombre minimal de personnes qu'il faut pour être sûr qu'au moins deux personnes ont la même date d'anniversaire. En répondant sans réfléchir, on pourra dire 366 (365 personnes avec des dates d'anniversaire différentes + 1). En réalité, si on prend un groupe de 23 personnes comme ensemble à N éléments, la probabilité que deux personnes aient la même date d'anniversaire est supérieure à $\frac{1}{2}$. Il suffit donc de prendre un groupe beaucoup plus petit que 366 pour avoir une probabilité élevée d'avoir deux personnes ayant la même date d'anniversaire. Il est également souhaitable qu'une petite modification de l'entrée d'une fonction de hachage entraîne un important changement dans le haché (effet avalanche).

Générateur pseudo-aléatoire

Un générateur (de nombres) pseudo-aléatoire (*Pseudo-random number generator* en Anglais) produit à partir d'une graine (quelques bits par exemple), une suite de nombres qui a de bonnes propriétés statistiques sur la répartition de ses bits, autrement dit la suite générée semble aléatoire. Un générateur de nombres aléatoires est souvent la base d'un algorithme de génération de clés en cryptographie.

Il est donc important que sa sortie reste imprévisible par un attaquant pour un usage cryptographique. Le terme "pseudo" vient du fait que la suite est générée de manière déterministe, la suite étant une récurrence d'état initial la graine. Sans la connaissance de cette graine, il doit être impossible *a priori* pour un attaquant de déterminer le terme suivant à partir d'une suite pseudo-aléatoire pour un usage cryptographique.

Les principales faiblesses des protocoles cryptographiques résident dans l'adaptation des algorithmes aux supports, faisant ainsi apparaître des erreurs dans l'implantation et menant à des fuites de mémoire ou à des variations de comportement du support, détectables par un attaquant. Ces faiblesses sont exploitées en cryptanalyse.

1.2 Cryptanalyse

Définition 1.5 (Cryptanalyse) *(Étymologiquement, la cryptanalyse signifie analyse du cryptogramme.) Cela regroupe les techniques pour décrypter des chiffrés, évaluer les faiblesses possibles et effectuer des attaques. On dit aussi que c'est la science de la découverte/reconstitution des informations cachées par un protocole cryptographique.*

1.2.1 Différents niveaux de sécurité

On évalue le niveau de sécurité d'un protocole cryptographique en fonction de la pire attaque connue contre celui-ci.

Nous les énumérons ici de la plus simple à la plus élaborée :

- **Attaque à texte chiffré seul (*ciphertext-only attack* ou *known ciphertext attack* en Anglais) :** L'attaquant ne connaît que le chiffré. Elle est possible pour le chiffrement si l'attaquant peut récupérer le chiffré passant sur un canal entre Alice et Bob.
- **Attaque à texte clair connu (message et chiffré connus, *known-plaintext attack* en Anglais) :** L'attaquant connaît des couples de textes clairs/chiffrés. Pour le chiffrement asymétrique, un attaquant peut toujours en forger.
- **Attaque à clair choisi (*Chosen Plaintext Attack* (CPA) en Anglais) :** L'attaquant peut obtenir le chiffrement de textes clairs de son choix avant et après avoir récupéré le texte chiffré qui l'intéresse. Ce type d'attaque est toujours possible sur un système de chiffrement à clé publique car l'attaquant dispose de la clé publique (et de l'algorithme de chiffrement).
- **Attaque à chiffré choisi (*Chosen Ciphertext Attack* (CCA) en Anglais) :** L'attaquant peut obtenir le déchiffrement de textes chiffrés de son choix avant d'obtenir et d'attaquer le chiffré qui l'intéresse. On parle d'attaque adaptative à chiffrés choisis (*Adaptive Chosen Ciphertext Attack* (CCA2) en Anglais) lorsque l'attaquant peut obtenir le déchiffrement de textes chiffrés en adaptant ses choix avant et après avoir récupéré le chiffré qui l'intéresse (à l'exception de celui-ci).

1.2.2 Principes des attaques par canaux auxiliaires et/ou par injection de faute

Une attaque par canal auxiliaire est une attaque qui exploite les lois de la physique afin d'obtenir de l'information contenue dans des canaux associés à une implantation, un circuit (pour extraire des secrets manipulés par des cartes à puce ou des accélérateurs cryptographiques par exemple). Le canal auxiliaire n'a pas pour but de transmettre volontairement de l'information. Il laisse s'échapper de l'information, qu'un attaquant doit savoir interpréter pour la comprendre. En revanche, des contre-mesures algorithmiques ou physiques peuvent être proposées pour fermer ou limiter un canal auxiliaire. La notion d'attaque par canal auxiliaire est apparue pour la première fois en 1996 dans [Koc96].
De manière générale, il y a deux concepts d'attaques en cryptanalyse : les attaques dites passives et les attaques dites actives. Les attaques passives sont celles lors desquelles un attaquant ne fait qu'observer ce qu'il se passe, en prenant des mesures si nécessaire, mais sans intervenir. Les attaques actives en revanche sont celles lors desquelles un attaquant peut modifier le texte chiffré, créer une faute, ou autre, afin de comprendre ce qu'il se passe dans l'algorithme ou l'implantation. Une attaque par canal auxiliaire sans injection de faute est dite passive et avec injection de faute est dite active. Nous développons ci-après les différents types d'attaque.

Attaque temporelle (*Timing Attack* en Anglais)

Kocher proposa en 1996 [Koc96] une analyse du temps mis par une implantation d'un système de chiffrement à clé publique basé sur le problème de factorisation d'un nombre entier [RSA78]. Cet article a révolutionné la cryptanalyse, car la communauté s'est rendue compte que des attaques liées au comportement du support de l'implantation étaient possibles en fonction de ce qu'il se passe ou non dans les algorithmes.
Ce type d'attaque consiste à mesurer et analyser le temps d'exécution, du déchiffrement pour revenir à l'exemple de RSA, afin de retrouver une information secrète. Cette attaque dépend de l'algorithme utilisé et des différences d'opérations faites en fonction du bit considéré de l'exposant pour RSA. De manière plus générale, le temps mis pour effectuer certaines opérations cryptographiques est étroitement lié à l'implantation cible, qu'elle soit logicielle ou matérielle. Ce type d'attaque requiert que l'attaquant connaisse les détails de l'implantation.

Attaque par consommation (*Power consumption attack* en Anglais)

Kocher fût également co-auteur de la proposition d'une attaque par consommation de courant en 1999 [KJJ99] contre un système de chiffrement symétrique. L'attaque consiste à observer à l'aide d'un oscilloscope la consommation de courant d'un processeur. Le but est de déterminer si le bit de la clé secrète n pour calculer x^n (*un message élevé à la puissance la clé secrète*) est un 0 ou un 1 pour en revenir à l'exemple de RSA. Le test **Si** dans l'algorithme provoque ce constat. L'idée générale d'une contre-mesure est d'éviter des branches où les opérations/opérandes ne sont pas les mêmes en fonction d'un choix/test.

Analyse différentielle de consommation (*Differential Power Analysis* (DPA) en Anglais) : Un cas particulier d'analyse de consommation est l'analyse différentielle. Pour effectuer une attaque de ce type, l'attaquant doit disposer de

plusieurs courbes de consommation pour une même clé secrète/privée. Le principe est de faire une moyenne de toutes ces courbes pour éliminer les bruits parasites qui fausseraient les relevés de consommation et donc l'hypothèse de clé. Le bruit peut provenir de processus en cours simultanément sur un ordinateur, de la qualité et de la précision des outils de mesure, ... L'attaquant n'a pas besoin de connaître le fonctionnement du support en détails. Ce type d'analyse est apparu pour la première fois dans [KJJ99].

Analyse simple de consommation (*Simple Power Analysis* (SPA) en Anglais) : À l'inverse d'une analyse différentielle, une analyse simple de consommation est une attaque faite par rapport à une seule courbe. Sur celle-ci, l'attaquant doit être capable de reconnaître la forme de la consommation pour une opération donnée et ainsi déterminer où en est l'exécution de l'implantation. Dans l'exemple de RSA, c'est une analyse de ce type qui peut être faite sur le canal auxiliaire. Le succès d'une telle attaque dépend directement de la connaissance que l'attaquant a de l'implantation et du fonctionnement du support.

Autres attaques : par rayonnement électromagnétique, par émission sonore, optique

Après le temps et l'énergie, des attaques exploitant d'autres phénomènes physiques sont possibles. Lors de leur fonctionnement, un support cible peut émettre des radiations, un composant électronique peut faire du bruit, un écran peut changer de couleur. Ce sont d'autres canaux et donc d'autres sources d'information qu'un attaquant est susceptible d'exploiter.

Les attaques, temporelles, par consommation, par rayonnement électromagnétique, par émission sonore et optiques, sont appelées de manière générale attaques par canaux auxiliaires (*Side-Channel Attacks* (SCA) en Anglais). En pratique, les attaques par canaux auxiliaires ne sont pas triviales à mettre en place, mais restent possibles. Depuis une vingtaine d'année, la communauté cryptographique doit prendre en compte cette nouvelle possibilité d'attaque pour la sécurité des protocoles cryptographiques en plus de la sécurité prouvable (preuves mathématiques de difficulté de résolution de certains problèmes). Lorsqu'un moyen pour fermer un canal auxiliaire est proposé, on parle de contre-mesure.

Attaque par injection de faute (*Fault Injection* en Anglais)

Le principe d'une attaque par injection de faute a été proposé en 1997 dans [BDL97]. C'est une attaque active bien évidemment. L'injection de faute peut se faire par augmentation de la tension électrique, par perturbation lumineuse (avec un laser par exemple), ou par impulsion électromagnétique. Il n'est pas rare que le support soit détérioré par l'attaque.

Nous allons nous intéresser par la suite à un système de chiffrement à clé publique élaboré par McEliece en 1978. Celui-ci étant basé sur les codes correcteurs d'erreurs, nous ferons dans le chapitre suivant, quelques rappels sur ce domaine. Le cœur de cette thèse portera sur des attaques par canaux auxiliaires contre ce système de chiffrement.

Chapitre 2

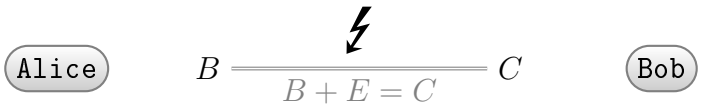
Codes correcteurs d'erreurs

"**R**omeo **I**ndia **C**harlie **H**otel **M**ike **O**scar **N**ovember **D**elta" (en utilisant l'alphabet de l'OTAN), ou comment éviter que quelqu'un écrive mon nom comme une célèbre marque de fromage à raclette dont je tairai le nom, c'est-à-dire avec deux erreurs ? !

2.1 Généralités

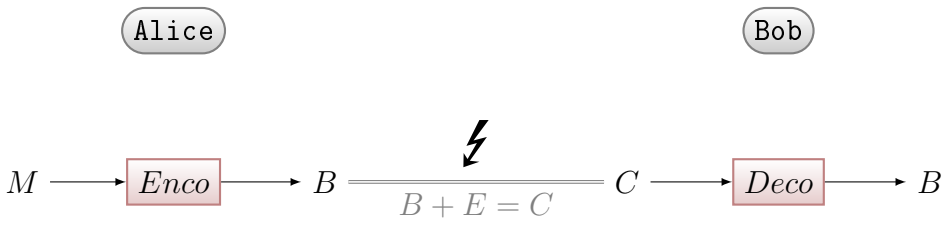
Une communication est une transmission d'information. C'est en 1948 que la quantification de l'information est apparue, dans un article dû à Claude E. Shannon [Sha48]. Cet article posa les bases de ce que l'on appelle aujourd'hui, la théorie de l'information. Un problème se posa alors, les erreurs dues au canal de transmission. On parle dans ce cas de canal bruité. Cela peut être, par exemple, un câble ethernet, une liaison par bluetooth. Il faudra attendre 1950 pour qu'une première solution soit proposée, par Richard W. Hamming [Ham50]. Cet article posa les bases de ce que l'on appelle aujourd'hui, la théorie des codes. En effet, Hamming proposa les premiers codes correcteurs d'erreur.
On appelle codage une méthode de transformation qui convertit la représentation d'une information en une autre. La théorie des codes sert à corriger le bruit ajouté à un message lors d'une transmission sur un canal bruité. La Figure 2.1 schématise ce problème.

Figure 2.1 – Utilité des codes correcteurs d'erreurs



Pour corriger ce problème, l'idée est de rajouter de la redondance d'information à celle du message initial, afin de pouvoir repérer les erreurs, quand il n'y a pas eu trop de bruit, et de les corriger. Par la suite, nous utiliserons le terme de "code" pour désigner un "code correcteur d'erreur(s)". Nous nous intéresserons uniquement à des codes linéaires dans cette thèse. Cela signifie que la redondance est linéairement dépendante de l'information.
Un code linéaire a un algorithme d'encodage simple, en temps polynomial. Le problème du décodage d'un code linéaire quant à lui, a été montré NP-difficile en 1978 dans [BMvT78]. En revanche, il devient plus facile pour certaines classes de codes dont il existe des algorithmes de décodage en temps polynomial, d'où la construction et l'étude de ces classes. Une représentation schématique des codes peut être trouvée dans la thèse de Vincent Herbert [Her11, page 23]. La Figure 2.2 présente les principes d'encodage (noté *Enco*) et de décodage (noté *Deco*), qui seront développés dans la Section 2.2.

FIGURE 2.2 – Principes d'encodage et de décodage



2.2 Codes linéaires

2.2.1 Vocabulaire pour débuter

Soit $\mathbb{F}_q$ un corps fini à q éléments, où q est une puissance d'un nombre premier p.

Définition 2.1 (Code linéaire) *On dit que $\mathscr{C}$ est un code linéaire de paramètres $[n,k]_q$ si et seulement si $\mathscr{C}$ est un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension k.*

En d'autres termes, un code linéaire est un ensemble de vecteurs de longueur n, qui est stable par addition et par multiplication par un scalaire, dont une base contient k vecteurs (indépendants). (Un scalaire est un élément de l'ensemble auquel appartiennent les composantes de la matrice.)

Définition 2.2 (Mot de code) *On appelle mot de code un élément de $\mathscr{C}$. En d'autres termes, c'est un vecteur de longueur n appartenant au code $\mathscr{C}$, c'est donc une combinaison linéaire à coefficients dans $\mathbb{F}_q$ des éléments d'une base de cet ensemble.*

Définition 2.3 (Espace ambiant) *On appelle espace ambiant l'espace-vectoriel $\mathbb{F}_q^n$. En d'autres termes, c'est l'espace de tous les mots de longueur n et à coefficients dans $\mathbb{F}_q$.*

Proposition 2.1 *Le cardinal de $\mathscr{C}$ est q^k, où k est sa dimension en tant que $\mathbb{F}_q$-espace-vectoriel.*

2.2.2 Rappels d'algèbre linéaire en dimension finie

Définition 2.4 (Rang d'une matrice) *Le rang d'une matrice quelconque $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$, noté $\mathrm{rg}(\mathcal{G})$, est égal au plus grand entier s tel que l'on puisse extraire une matrice carrée de taille $s \times s$ de $\mathcal{G}$ qui soit inversible.*

Remarque 2.1 *Soit $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$, avec $k \leqslant n$. Alors :*

$$0 \leqslant \mathrm{rg}(\mathcal{G}) \leqslant k.$$

De manière plus générale, le rang d'une matrice a pour borne supérieure le minimum entre le nombre de lignes et le nombre de colonnes de cette matrice.

Propriété 2.1 *Soient $\mathscr{A}$ et $\mathscr{B}$ des matrices de même format.*

1. *Si $\mathscr{A}$ est une matrice carrée inversible, alors $\mathrm{rg}(\mathscr{A} \cdot \mathscr{B}) = \mathrm{rg}(\mathscr{B})$.*

2. *Si $\mathscr{B}$ est une matrice carrée inversible, alors* $\mathrm{rg}(\mathscr{A} \cdot \mathscr{B}) = \mathrm{rg}(\mathscr{A})$.

Définition 2.5 (Opérations élémentaires sur une matrice) *Sur les lignes (respectivement sur les colonnes) :*

1. *La suppression d'une ligne (respectivement d'une colonne) ne contenant que des 0.*
2. *La multiplication de tous les éléments d'une ligne (respectivement une colonne) par un scalaire non nul.*
3. *L'ajout à une ligne (respectivement une colonne) d'une combinaison linéaire des autres lignes (respectivement colonnes).*
4. *L'échange de deux lignes (respectivement colonnes).*

Définition 2.6 (Matrices équivalentes) *On dit que deux matrices sont équivalentes si et seulement si on peut passer de l'une à l'autre via des opérations élémentaires.*

Théorème 2.1 *Deux matrices équivalentes ont même rang.*

Théorème 2.2 (de la dimension) *Soit $\mathscr{F}$ une famille d'éléments de dimension n d'un ensemble fini $\mathscr{E}$. Les propriétés suivantes sont équivalentes :*

1. *$\mathscr{F}$ est une base de $\mathscr{E}$.*
2. *$\mathscr{F}$ est une famille libre et contient n éléments.*
3. *$\mathscr{F}$ est une famille génératrice de $\mathscr{E}$ et contient n éléments.*
4. *$\mathscr{F}$ est une famille libre et génératrice de $\mathscr{E}$.*

Définition 2.7 (Rang d'une famille de vecteurs) *Soit $\mathscr{E}$ un espace-vectoriel et $\mathscr{F} = \{E_1, E_2, \ldots, E_n\}$ une famille d'éléments de $\mathscr{E}$. On appelle rang de $\mathscr{F}$, noté $\mathrm{rg}(\mathscr{F})$, la dimension du sous-espace-vectoriel de $\mathscr{E}$ engendré par $\mathscr{F}$.*

2.2.3 Matrice génératrice, encodage

Définition 2.8 (Matrice génératrice) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$. On dit que $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ est une matrice génératrice de $\mathscr{C}$ si et seulement si ses lignes forment une base de $\mathscr{C}$. Donc $\mathcal{G}$ est de rang k et on a :*

$$\mathscr{C} \triangleq \left\{ M \cdot \mathcal{G} | M \in \mathbb{F}_q^k \right\}.$$

Proposition 2.2 *Soient $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$ et $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ une matrice génératrice de $\mathscr{C}$. Alors $\mathcal{G}$ est de rang k, noté $\mathrm{rg}(\mathcal{G}) = k$.*

Preuve *Soient $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$ et $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ une matrice génératrice de $\mathscr{C}$. On a, par définition :*

$$\dim_{\mathbb{F}_q^n}(\mathscr{C}) = k.$$

D'après le Théorème 2.2 (de la dimension) et comme les lignes $\mathcal{G}$ forment une base de $\mathscr{C}$ (d'après la Définition 2.8), alors on a :

$$\mathrm{rg}(\textit{lignes de } \mathcal{G}) = dim_{\mathbb{F}_q^n}(\mathscr{C}) = k.$$

□

Remarque 2.2 (Rang plein) *On dit que $\mathcal{G}$ est de rang plein, car k est la valeur maximale de $\mathrm{rg}(\mathcal{G})$ et $\mathrm{rg}(\mathcal{G}) = k$.*

Définition 2.9 (Fonction d'encodage) *Soient $\mathscr{C}$ un code linéaire de paramètres $[n,k]_q$ et $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ une matrice génératrice de $\mathscr{C}$. La fonction d'encodage associée à $\mathcal{G}$ est l'application suivante :*

$$\begin{array}{rcl} f_{\mathcal{G}} : \mathbb{F}_q^k & \longrightarrow & \mathbb{F}_q^n \\ M & \longmapsto & B = M \cdot \mathcal{G}, \end{array}$$

L'image de la fonction d'encodage est le code linéaire associé : $Im(f_{\mathcal{G}}) = \mathscr{C}$.

Proposition 2.3 *Soient $\mathscr{C}$ un code linéaire de paramètres $[n,k]_q$, $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ une matrice génératrice de $\mathscr{C}$ et $\mathcal{S} \in \mathscr{M}_{k,k}(\mathbb{F}_q)$ une matrice carrée inversible. Alors $\mathcal{S} \cdot \mathcal{G}$ est aussi une matrice génératrice du code $\mathscr{C}$.*

Définition 2.10 (Codes équivalents par permutation) *Soient deux codes linéaires $\mathscr{C}_1$ et $\mathscr{C}_2$, de paramètres $[n,k]_q$, avec respectivement comme matrice génératrice $\mathcal{G}_1 \in \mathscr{M}_{k,n}(\mathbb{F}_q)$ et $\mathcal{G}_2 \in \mathscr{M}_{k,n}(\mathbb{F}_q)$. On dit que $\mathscr{C}_1$ et $\mathscr{C}_2$ sont équivalents si $\mathcal{G}_1 = \mathcal{S} \cdot \mathcal{G}_2 \cdot \mathcal{P}$, où $\mathcal{S}$ est une matrice inversible dans $\mathscr{M}_{k,k}(\mathbb{F}_q)$ et $\mathcal{P}$ est une matrice de permutation dans $\mathscr{M}_{n,n}(\mathbb{F}_q)$.*

2.2.4 Distance de Hamming, capacité de correction

Définition 2.11 (Support) *Dans l'espace ambiant, le support d'un vecteur $A = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$ est l'ensemble des indices de ses coordonnées non nulles. On le note :*

$$Supp(A) \triangleq \{i; a_i \neq 0\}.$$

Définition 2.12 (Poids de Hamming) *Dans l'espace ambiant, le poids de Hamming d'un vecteur $A = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$ est le nombre de coordonnées non nulles de celui-ci. On le note $w_H(A)$. Autrement dit, on a :*

$$w_H(A) \triangleq |\{i; a_i \neq 0\}| = |Supp(A)|.$$

Remarque 2.3 *Le poids minimal des lignes de $\mathcal{G}$ n'est pas nécessairement le poids minimal du code $\mathscr{C}$.*

Notation 2.1 *Soit $\mathscr{C}$ un code linéaire de longueur n. On appelle μ_i le nombre de mots de code de poids i, pour $0 \leqslant i \leqslant n$.*

Définition 2.13 (Énumérateur de poids) *Soient $\mathscr{C}$ un code linéaire de longueur n et μ_i le nombre de mots de code de poids i, pour $0 \leqslant i \leqslant n$. Alors, le polynôme*

$$W_{\mathscr{C}}(x,y) = \sum_{i=0}^{n} \mu_i \cdot x^i \cdot y^{n-i}$$

d'inconnues x et y est appelé l'énumérateur de poids de $\mathscr{C}$.

Définition 2.14 (Distance de Hamming) *Dans l'espace ambiant, la distance de Hamming d_H entre deux mots, $A = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$ et $B = (b_1, \ldots, b_n) \in \mathbb{F}_q^n$, est le nombre de coordonnées distinctes entre les deux :*

$$d_H(A,B) \triangleq |\{i; a_i \neq b_i\}|. \tag{2.1}$$

Propriété 2.2 *Soient A et B deux éléments de $\mathbb{F}_2^n$, $\oplus$ l'opérateur OU EXCLUSIF bit à bit (c'est-à-dire l'addition coordonnée par coordonnée dans $\mathbb{F}_2$) et $\wedge$ l'opérateur ET bit à bit (c'est-à-dire la multiplication coordonnée par coordonnée dans $\mathbb{F}_2$). On a alors :*

— $w_H(A) = d_H(A, 0)$, *où 0 dénote le mot tout à zéro,*
— $d_H(A, B) = w_H(A \oplus B) = w_H(A) \oplus w_H(B) - 2 \cdot w_H(A \wedge B)$

Définition 2.15 (Distance minimale) *La plus petite distance non nulle entre deux mots de code (différents) est appelée distance minimale du code. On la note d.*

Définition 2.16 (Paramètres d'un code linéaire) *On dit qu'un code est de type (ou a pour paramètres) $[n, k, d]_q$ s'il est à valeurs dans $\mathbb{F}_q$ et si on a :*

n : *longueur d'un mot de code*

k : *dimension du code*

d : *distance minimale du code*

Remarque 2.4 *Lorsque la distance minimale du code n'intervient pas dans les propos, nous simplifierons la notation $[n, k, d]_q$ par $[n, k]_q$. De plus, si le cardinal du corps n'intervient pas, nous allégerons encore cette notation par $[n, k]$.*

Définition 2.17 (Boule de Hamming) *Soient $A \in \mathbb{F}_q^n$ et ε un entier positif. On définit la boule de Hamming $\mathcal{B}$ de centre A et de rayon ε par :*

$$\mathcal{B}(A, \varepsilon) \triangleq \{C \in \mathbb{F}_q^n ; d_H(A, C) \leqslant \varepsilon\}.$$

Proposition 2.4 *Soit ε un entier positif. Pour tout $A \in \mathbb{F}_q^n$, la cardinalité de la boule de Hamming $\mathcal{B}$ de centre A et de rayon ε est donnée par :*

$$\nu_q(n, \varepsilon) \triangleq |\mathcal{B}(A, \varepsilon)| = \sum_{i=0}^{\varepsilon} \binom{n}{i} \cdot (q-1)^i.$$

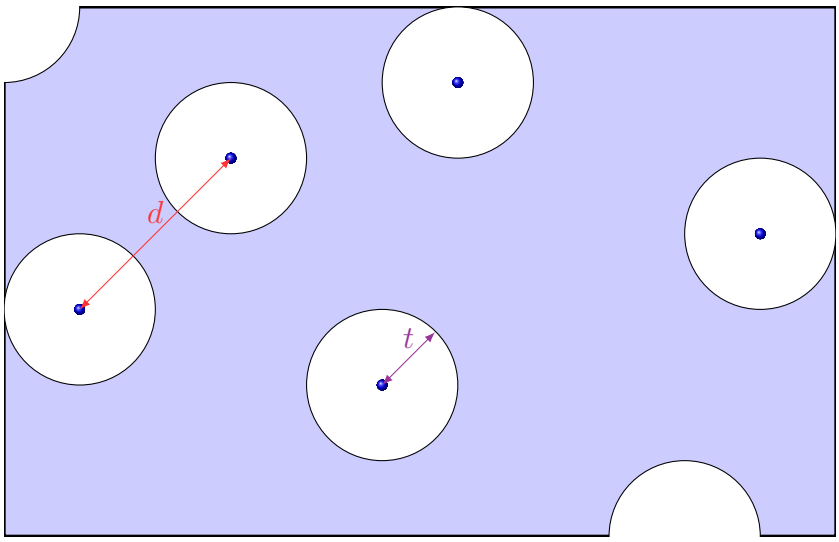
Définition 2.18 (Capacité de détection) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k, d]_q$. La capacité de détection de $\mathscr{C}$ est la quantité $d-1$. On dit alors que $\mathscr{C}$ est $(d-1)$-détecteur.*

Définition 2.19 (Capacité de correction) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k, d]_q$. La capacité de correction de $\mathscr{C}$, notée t, est définie par :*

$$t \triangleq \left\lfloor \frac{d-1}{2} \right\rfloor$$

On dit alors que $\mathscr{C}$ est t-correcteur.

FIGURE 2.3 – Capacité de correction t d'un code $\mathscr{C}$



Remarque 2.5 *Les codes sont parfois qualifiés de détecteurs et de correcteurs. Nous nous intéresserons uniquement à l'aspect correction dans cette thèse.*

Définition 2.20 (Rendement) *On appelle rendement noté ϱ d'un code de paramètres $[n, k]$ le ratio $\varrho \triangleq \dfrac{k}{n}$.*

Définition 2.21 (Rayon de recouvrement) *Le rayon de recouvrement d'un code linéaire $\mathscr{C}$ de paramètres $[n, k]_q$, noté ρ, est défini de la manière suivante :*

$$\rho \triangleq \max_{A \in \mathbb{F}_q^n} \left\{ d_H(\mathscr{C}, A) \right\} \triangleq \max_{A \in \mathbb{F}_q^n} \left\{ \min_{B \in \mathscr{C}} d_H(B, A) \right\}. \tag{2.2}$$

Définition 2.22 (Code parfait) *Un code est dit parfait si son rayon de recouvrement est égal à sa capacité de correction. Autrement dit, un code est parfait (ou t-parfait) s'il est t-correcteur et si toute erreur détectée est corrigée.*

2.2.5 Produit scalaire, code dual

Définition 2.23 (Produit scalaire) *Soient $A = (a_1, \ldots, a_n)$ et $B = (b_1, \ldots, b_n)$ dans $\mathbb{F}_q^n$. Le produit scalaire entre A et B est :*

$$< A, B > \triangleq \sum_{i=1}^{n} a_i \cdot b_i.$$

Définition 2.24 (Code dual) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$. Le code dual (ou code orthogonal), noté $\mathscr{C}^\perp$, est l'espace-vectoriel orthogonal de $\mathscr{C}$ pour le produit scalaire sur $\mathbb{F}_q^n$ défini précédemment. On note :*

$$\mathscr{C}^\perp \triangleq \left\{ A \in \mathbb{F}_q^n \middle| < B, A >= 0, \forall B \in \mathscr{C} \right\}.$$

Remarque 2.6 *Soient $A, B \in \mathbb{F}_q^n$. On dit que A et B sont orthogonaux si et seulement si $< A, B >= 0$.*

Définition 2.25 (Co-dimension de $\mathscr{C}$) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$. La dimension du code dual est la co-dimension de $\mathscr{C}$. On la notera r, avec :*

$$r \triangleq n - k.$$

2.2.6 Matrice de contrôle

Définition 2.26 (Matrice de contrôle) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n,k]_q$. On appelle matrice de contrôle (ou matrice de parité) de $\mathscr{C}$, toute matrice génératrice du code dual $\mathscr{C}^\perp$, notée $\mathcal{H}$. On a $\mathcal{H} \in \mathscr{M}_{r,n}(\mathbb{F}_q)$ et :*

$$\mathscr{C} \triangleq \left\{B \in \mathbb{F}_q^n | B \cdot {}^t\mathcal{H} = 0_{1,r}\right\}.$$

Proposition 2.5 *Soient $\mathscr{C}$ un code linéaire sur $\mathbb{F}_2$ et $\mathcal{H}$ une matrice de contrôle de $\mathscr{C}$. La distance minimale d'un code $\mathscr{C}$ est égale au nombre minimal de colonnes de $\mathcal{H}$ dont la somme vaut zéro (c'est-à-dire colonnes linéairement dépendantes).*

Proposition 2.6 *Soient $\mathscr{C}$ un code linéaire de paramètres $[n,k,d]_2$ et $\mathcal{H}$ une matrice de contrôle de $\mathscr{C}$. $d \geqslant 3$ si et seulement si $\mathcal{H}$ ne contient ni colonne nulle ni colonnes identiques.*

Définition 2.27 (Fonction syndrome) *Soient $\mathscr{C}$ un code linéaire et $\mathcal{H}$ une matrice de contrôle de $\mathscr{C}$. On définit la fonction syndrome associée à $\mathcal{H}$ de la manière suivante :*

$$\begin{array}{rcl} S_{\mathcal{H}} : \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^r \\ C & \longmapsto & C \cdot {}^t\mathcal{H} \end{array}$$

où $S_{\mathcal{H}}(C) = C \cdot {}^t\mathcal{H}$ est appelé le syndrome du mot C.

Pour alléger les notations, il est possible que le syndrome de C soit noté S ou S_C dans la suite de cette thèse.

Remarque 2.7 *On a $\forall B \in \mathscr{C}, B \cdot {}^t\mathcal{H} = 0_{1,r}$ et $\mathcal{G} \cdot {}^t\mathcal{H} = 0_{k,r} \in \mathscr{M}_{k,r}(\mathbb{F}_q)$.*

Remarque 2.8 *Avec $C = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$, notons que :*

$$S_{\mathcal{H}}(C) = \sum_{j=1}^{n} c_j \cdot H_j,$$

où H_j correspond à la j-ème colonne de $\mathcal{H}$. Autrement dit, $S_{\mathcal{H}}(C)$ correspond au produit scalaire entre le vecteur C et chaque colonne H_j de $\mathcal{H}$.

Définition 2.28 (Coset-leader) *Soient $N \in \mathbb{F}_q^r$ et $E \in \mathbb{F}_q^n$. On dit que E est le coset-leader de N si et seulement si $S_{\mathcal{H}}(E) = N$ et $w_H(E)$ est le plus petit poids d'un vecteur de l'espace ambiant ayant pour syndrome N.*

Définition 2.29 (Forme systématique) *Soient $\mathscr{C}$ un code linéaire de paramètres $[n,k]$ et $\mathcal{G}$ une matrice génératrice de $\mathscr{C}$. On dit que $\mathcal{G}$ est sous forme systématique si*

$$\mathcal{G} = \left[\mathcal{I}_k | \mathcal{A}\right],$$

où $\mathcal{I}_k$ est la matrice identité dans $\mathscr{M}_{k,k}(\mathbb{F}_q)$ et $\mathcal{A}$ une matrice dans $\mathscr{M}_{k,r}(\mathbb{F}_q)$.

Proposition 2.7 *Soient $\mathscr{C}$ un code linéaire de paramètres $[n,k]$ et $\mathcal{G}$ une matrice génératrice de $\mathscr{C}$ sous forme systématique. Le code $\mathscr{C}$ admet alors pour matrice de contrôle (sous forme systématique) :*

$$\mathcal{H} = \left[-{}^t\mathcal{A} | \mathcal{I}_r\right].$$

Remarque 2.9 *Lorsqu'un code linéaire a une matrice génératrice sous forme systématique, alors tout mot du code est composé du mot source suivi de la redondance (c'est-à-dire de l'information suivie de la redondance d'information).*

Un mot de code peut a priori être corrigé par n'importe qui, à condition de savoir quel code a été utilisé (quelle est sa structure), afin d'utiliser l'algorithme adéquat (c.f. Section 2.3 pour le problème du décodage). Donc, il n'y a pas vraiment de secret. Par contre, on peut vouloir chiffrer ce mot de code avant de l'envoyer ou cacher la structure du code. C'est là que la théorie des codes rejoint la cryptologie. Nous proposons au lecteur à la fin de ce chapitre une description des codes de Goppa classiques binaires irréductibles (c.f. Section 2.5), utilisés en cryptographie, et du cryptosystème de McEliece, qui les utilise, dans le chapitre suivant (c.f. Chapitre 3).

2.3 Problème du décodage

Cette section est inspirée de [PW95, Bar11].

On pourrait penser que, comme l'encodage par un code linéaire d'un mot est une simple multiplication entre un vecteur et une matrice, le décodage serait facile, mais il n'en est rien. En effet, c'est un problème qui a été démontré NP-complet en 1978 pour un syndrome donné et un mot de poids fixé [BMvT78].
Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$. Tout d'abord, on peut énumérer tous les mots de l'espace ambiant en les séparant de la manière suivante :

- Tous les mots de $\mathscr{C}$ (il y en a q^k) ;
- puis tous les mots de $\mathscr{C}$ avec une erreur de poids 1,
  pour toutes les erreurs de poids 1 (c'est-à-dire $\binom{n}{1}$ vecteurs erreurs possibles) ;
- ensuite tous les mots de $\mathscr{C}$ avec une erreur de poids 2,
  pour toutes les erreurs de poids 2 (c'est-à-dire $\binom{n}{2}$ vecteurs erreurs possibles) ;
- et ainsi de suite ...
- jusqu'à ce que tous les mots de l'espace ambiant soient écrits (c'est-à-dire les q^n mots).

Ensuite, différentes méthodes sont possibles.

2.3.1 Décodage (à poids) borné

La première méthode est une force brute sur l'hypothèse du poids de l'erreur, appelée le décodage borné. Comme tous les mots de l'espace ambiant ont précédemment été écrits, lorsqu'on veut décoder un mot, il "suffit" de chercher où il se trouve de la manière suivante :

- Pour commencer, on suppose qu'il n'y a pas eu d'erreur. On cherche donc si le mot est un mot du code.
- Si ce n'est pas le cas, on suppose qu'il y a eu une erreur de poids 1. On cherche donc si le mot est dans la liste des mots avec une erreur de poids 1.
- Si ce n'est pas le cas, on suppose qu'il y a eu un erreur de poids 2.
- Et ainsi de suite, ...
- jusqu'à ce qu'on trouve le mot.

Posons le problème de manière formelle.

Problème 2.1 (Décodage (à poids) borné) *Soient $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$ et $A \in \mathbb{F}_q^n$ un mot de l'espace ambiant. Le problème du décodage borné par ε consiste à trouver un mot de code $B \in \mathscr{C}$ tel que $d_H(A, B) \leqslant \varepsilon$.*

2.3.2 Décodage unique

Un problème découlant directement du décodage borné est le décodage unique.

Problème 2.2 (Décodage unique) *Soient $\mathscr{C}$ un code linéaire de paramètres $[n, k, d]_q$, où $d = 2t + 1$, de capacité de correction t et $A \in \mathbb{F}_q^n$ un mot de l'espace ambiant. Le problème du décodage unique consiste à trouver un mot de code $B \in \mathscr{C}$ tel que $d_H(A, B) \leqslant t$. Autrement dit, le décodage unique est le décodage borné par t.*

2.3.3 Décodage complet

Du décodage borné, on en vient également de manière naturelle au décodage complet, qui consiste à trouver le mot de code le plus proche du mot de l'espace ambiant.

Problème 2.3 (Décodage complet) *Soient $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$ et $A \in \mathbb{F}_q^n$ un mot de l'espace ambiant. Le problème du décodage complet consiste à trouver un mot de code $B \in \mathscr{C}$ tel que $d_H(A, B) = d_H(A, \mathscr{C})$.*

2.3.4 Décodage par syndrome (*Syndrome Decoding* (SD) en Anglais)

Un problème équivalent au problème du décodage complet est le décodage par syndrome. C'est un problème central sur lequel sont basés les protocoles cryptographiques utilisant les codes correcteurs d'erreurs. En effet, la résolution de ce problème permet de retrouver le message.

Problème 2.4 (Décodage par syndrome) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n, k]_q$, de capacité de correction t. Soient $\mathcal{H}$ une matrice de contrôle de $\mathscr{C}$ et $C \in \mathbb{F}_q^n$ un mot de l'espace ambiant. Le problème du décodage par syndrome consiste à trouver le coset-leader E tel que $S_{\mathcal{H}}(E) = S_{\mathcal{H}}(C)$.*
Autrement dit, étant donnés $\mathcal{H} \in \mathscr{M}_{r,n}(\mathbb{F}_q)$ une matrice de contrôle de $\mathscr{C}$, $S_{\mathcal{H}}(C)$ un vecteur de longueur r sur $\mathbb{F}_q$ et ε un entier fixé, existe-il un vecteur E de longueur n sur $\mathbb{F}_q$ tel que $E \cdot {}^t\mathcal{H} = S_{\mathcal{H}}(C)$ et $w_H(E) \leqslant \varepsilon$?

Ce problème a été démontré NP-complet en 1978 par Berlekamp, McEliece et van Tilborg [BMvT78].

Remarque 2.10 *Le décodage par syndrome revient à chercher le plus petit ensemble des indices $\mathcal{J} \subset \{1, 2, \ldots, n\}$ tel qu'il existe des $\lambda_j \in \mathbb{F}_q^*$ pour $j \in \mathcal{J}$ et*

$$\sum_{j \in \mathcal{J}} \lambda_j \cdot H_j = S_{\mathcal{H}}(C), \tag{2.3}$$

où les H_j sont des colonnes de $\mathcal{H}$.

Remarque 2.11 *Lorsque $q = 2$, l'équation (2.3) devient :*

$$\sum_{j \in \mathcal{J}} H_j = S_{\mathcal{H}}(C).$$

Le mot de code le plus proche de C est donc $C \oplus \overbrace{\sum_{j \in \mathcal{J}} e_j}^{E} = B$, où $e_j = 1$ si $j \in \mathcal{J}$ et 0 sinon.

2.3.5 Décodage par ensemble d'information (*Information Set Decoding* (ISD) en Anglais)

Une méthode possible pour deviner où sont les erreurs découle du raisonnement suivant. On sait que, sur les n symboles d'un mot de code, k coordonnées sont celles du message qui a été encodé et les r autres correspondent à la redondance d'information. L'idée est d'essayer de deviner où sont les k bits d'information, en espérant qu'il n'y ait pas eu d'erreur sur ces coordonnées (d'où le nom d'ensemble d'information). Évidemment, cet algorithme est probabiliste, puisque il faut essayer plusieurs fois, a priori, avant de trouver un ensemble d'information.

Problème 2.5 (Décodage par ensemble d'information) *Soit $\mathscr{C}$ un code linéaire de paramètres $[n,k]_q$, de capacité de correction t. Soient $\mathcal{G}$ une matrice génératrice de $\mathscr{C}$, $B \in \mathbb{F}_q^n$ un mot de code, $E \in \mathbb{F}_q^n$ un vecteur erreur de poids au plus t et $C = B + E \in \mathbb{F}_q^n$ un mot de l'espace ambiant. Le problème du décodage par ensemble d'information consiste à trouver k coordonnées de C telles que, mises bout à bout pour former le mot $M \in \mathbb{F}_q^k$, on puisse retrouver le mot de code B après encodage : $f_{\mathcal{G}}(M) \triangleq M \cdot \mathcal{G} = B$.*
Autrement dit (version duale), soient $\mathcal{H} \in \mathscr{M}_{r,n}(\mathbb{F}_q)$ une matrice de contrôle de $\mathscr{C}$ et N un vecteur de $\mathbb{F}_q^r$ (un syndrome fixé). Le problème du décodage par ensemble d'information consiste à trouver un vecteur $E \in \mathbb{F}_q^n$ de poids au plus t tel que $N = S_{\mathcal{H}}(E) \triangleq E \cdot {}^t\mathcal{H}$.

La première proposition a été faite par McEliece lui-même comme méthode pour attaquer son propre cryptosystème, tous deux développés dans le même article [McE78]. De nombreuses améliorations ont été faites par la suite et c'est encore un sujet très étudié en recherche de nos jours.

2.3.6 Décodage en liste

Nous nous sommes cantonnés au décodage unique dans cette thèse. Le problème du décodage en liste ne sera donc pas abordé par la suite, mais pour compléter cette section, en voici l'énoncé.

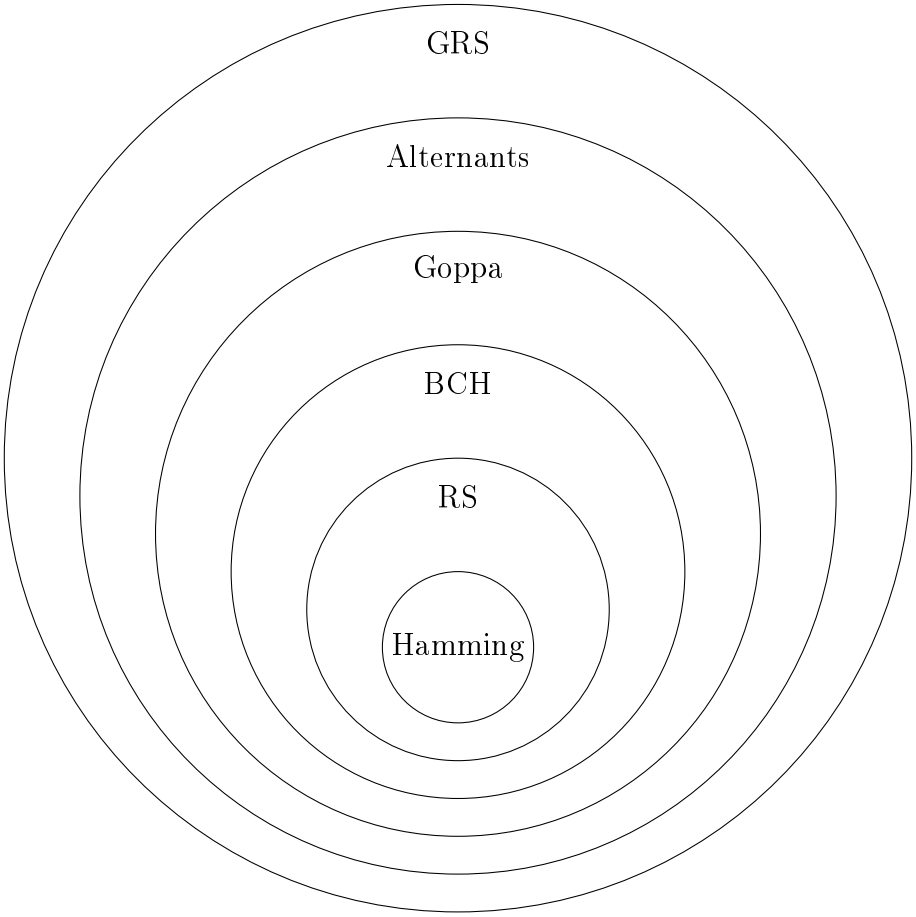
Problème 2.6 (Décodage en liste) *Soit $\mathscr{C}$ un code. Le problème du décodage en liste juqu'à ε consiste à trouver, pour tout mot de l'espace ambiant $A \in \mathbb{F}_q^n$, l'ensemble des mots $B \in \mathscr{C}$, s'il en existe, tels que $d_H(A,B) \leqslant \varepsilon$.*

Remarque 2.12 *L'avantage de cette méthode est que, pour un code linéaire t-correcteur, on peut prendre $\varepsilon > t$ et avoir une liste des candidats qui vérifient cette condition.*

2.4 Quelques exemples de codes linéaires

Nous présentons, dans cette section, les codes linéaires les plus connus, dont certaines propriétés nous servirons dans la suite de cette thèse.

FIGURE 2.4 – Classes de codes



Nous commencerons par les plus simples : les codes de Hamming.

2.4.1 Codes de Hamming

Cette sous-section est inspirée des cours de Gilles Zémor sur les codes linéaires[1] *et du livre [PW95]. Le lecteur pourra également se référer à [MS77, Chapitre 1].*

Comme mentionné au début de ce chapitre, les codes de Hamming ont été introduits en 1950 par Richard Hamming [Ham50], lors de ses travaux sur un modèle de calculateur à carte perforée de faible fiabilité. Il construit alors le premier code correcteur véritablement efficace, capable de corriger une erreur.

Cas binaire

Définition 2.30 (Code de Hamming binaire) *Soit $q = 2$. Soit m un entier positif supérieur ou égal à 2. Un code de Hamming binaire, dit de paramètre m, est un code linéaire défini sur $\mathbb{F}_2$, de longueur $n = 2^m - 1$, de dimension $k = 2^m - 1 - m = n - m$, et de distance minimale $d = 3$. Les colonnes d'une matrice de contrôle d'un code de Hamming binaire sont tous les vecteurs non nuls de $\mathbb{F}_2^m$.*

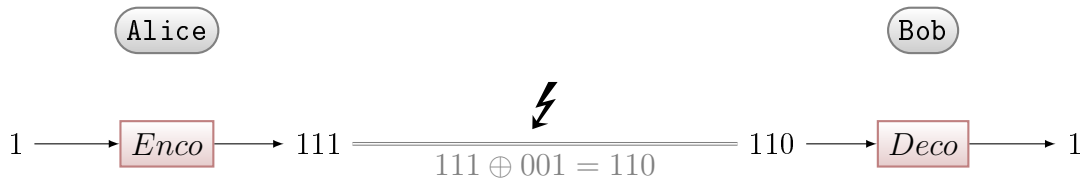
1. https://www.math.u-bordeaux.fr/~gzemor/codes06.pdf
https://www.math.u-bordeaux.fr/~gzemor/arit06.pdf

Les codes de Hamming binaires sont des codes 1-correcteurs à redondance minimale. Ce sont des codes parfaits, autrement dit ils ne contiennent aucune redondance d'information inutile. De plus, pour une longueur de code donnée, il n'existe pas de code plus compact pour une même capacité de correction.
Pour encoder un message $M \in \mathbb{F}_2^k$, on calcule $B = M \cdot \mathcal{G} \in \mathbb{F}_2^n$. Il y a donc $r = n - k$ bits de redondance. Pour décoder un message reçu $C \in \mathbb{F}_2^n$, on peut utiliser le décodage par syndrome de la façon suivante. On calcule $S_{\mathcal{H}}(C) = C \cdot {}^t\mathcal{H}$. S'il est nul, alors $C \in \mathscr{C}$, sinon, $C = B \oplus E$, où $B \in \mathscr{C}$ et E est l'erreur, d'où $S_{\mathcal{H}}(C) = S_{\mathcal{H}}(B) + S_{\mathcal{H}}(E) = S_{\mathcal{H}}(E)$. De là, on en déduit le bit erroné : comme les codes de Hamming sont 1-correcteurs, le poids de l'erreur vaudra donc 1 et le syndrome sera la colonne de $\mathcal{H}$ dont l'indice correspond à l'indice du bit erroné dans E.

Exemple 1 : Le code de Hamming de paramètres $[3, 1, 3]_2$
(aussi connu sous le nom de code à répétition de longueur 3)

Le code de Hamming de paramètres $[3, 1, 3]_2$ est plus connu sous le nom de code à répétition. En reprenant la Figure 2.2 dans laquelle on remplace M par le message "1", cela donne comme mot de code "111". Avec une erreur en 3ème position sur le canal, le mot reçu est alors "110", que nous pouvons corriger en "111" (car il y a plus de "1" que de "0") et décoder en "1". La Figure 2.5 schématise cet exemple.

FIGURE 2.5 – Code à répétition



Exemple 2 : Le code de Hamming de paramètres $[7, 4, 3]_2$

Ce code est souvent pris comme exemple, car il est à la fois une illustration amusante du principe d'un code correcteur d'erreur et est un peu plus élaboré que la simple répétition de l'information (c.f. [PW95, p. 18]).
Présentons-le sous la forme d'un jeu de devinette. L'idée de ce jeu est que quelqu'un choisisse un nombre M entre 0 et 15, puis qu'une autre personne devine ce nombre en posant 7 questions bien précises à la première. Les réponses doivent être 1 pour oui et 0 pour non. Il est possible de répondre au plus 1 seule fois de manière incorrecte.
Sans erreur : Prenons comme exemple le nombre $M = 3$. Voici la liste de ces 7 questions et les réponses associées à notre exemple (sans triche) :

1. L'entier M est-il $\geqslant 8$?
   $0 \Rightarrow$ L'ensemble de réponse est $\{0, 1, 2, 3, 4, 5, 6, 7\}$.
2. L'entier M est-il dans $\{4, 5, 6, 7, 12, 13, 14, 15\}$?
   $0 \Rightarrow$ L'ensemble de réponse est $\{0, 1, 2, 3\}$.
3. L'entier M est-il dans $\{2, 3, 6, 7, 10, 11, 14, 15\}$?
   $1 \Rightarrow$ L'ensemble de réponse est $\{2, 3\}$.
4. L'entier M est-il impair ?
   $1 \Rightarrow$ L'ensemble de réponse est $\{3\}$.

5. L'entier M est-il dans $\{1, 2, 4, 7, 9, 10, 12, 15\}$?
   $0 \Rightarrow$ L'ensemble de réponse est $\{3\}$.
6. L'entier M est-il dans $\{1, 2, 5, 6, 8, 11, 12, 15\}$?
   $0 \Rightarrow$ L'ensemble de réponse est $\{3\}$.
7. L'entier M est-il dans $\{1, 3, 4, 6, 8, 10, 13, 15\}$?
   $1 \Rightarrow$ L'ensemble de réponse est $\{3\}$.

On peut remarquer que 4 réponses suffisent à retrouver le nombre choisi. Posons $C = (c_1, c_2, c_3, c_4, c_5, c_6, c_7)$ le vecteur formé par les réponses. Cela donne pour notre exemple $C = (0, 0, 1, 1, 0, 0, 1)$. Pour vérifier qu'il n'y a pas eu triche pour une réponse, calculons les sommes suivantes (dans $\mathbb{F}_2$) :

1. $s_1 = c_4 + c_5 + c_6 + c_7 = 1 + 0 + 0 + 1 = 0$
2. $s_2 = c_2 + c_3 + c_6 + c_7 = 0 + 1 + 0 + 1 = 0$
3. $s_3 = c_1 + c_3 + c_5 + c_7 = 0 + 1 + 0 + 1 = 0$

Si les trois sommes sont nulles ($s_1 = s_2 = s_3 = 0$), alors il n'y a pas eu de triche (ce qui est le cas ici).
Avec erreur : Reprenons notre exemple $M = 3$ mais avec une réponse incorrecte cette fois-ci (la 6ème). Le vecteur formé par les réponses devient alors $C = (0, 0, 1, 1, 0, 1, 1)$. Les vérifications donnent :

1. $s_1 = c_4 + c_5 + c_6 + c_7 = 1 + 0 + 1 + 1 = 1$
2. $s_2 = c_2 + c_3 + c_6 + c_7 = 0 + 1 + 1 + 1 = 1$
3. $s_3 = c_1 + c_3 + c_5 + c_7 = 0 + 1 + 0 + 1 = 0$

Comme $S = (s_1, s_2, s_3)_2 = (110)_2 \neq 0$ (en écriture binaire), il y a eu triche et il faut donc changer la réponse à la question $S = (110)_2 = 6_{10}$.
Explication : Analysons tout ceci d'un point de vue mathématique maintenant. Le nombre que l'on cherche est compris entre 0 et 15, autrement dit son écriture binaire est sur 4 bits. Sans erreur, on retrouve directement ce nombre après les 4 premières questions, car l'intersection des réponses possibles à ces 4 questions est un ensemble à un seul élément (le nombre choisi). C'est également la raison pour laquelle c_1, c_2, c_3 et c_4 n'apparaissent que dans une seule somme pour déterminer S. Quant aux 3 dernières questions, elles servent à détecter s'il y a eu triche ou non. C'est pourquoi c_5 et c_6 sont dans deux sommes de vérification et c_7 dans les trois. En effet, s'il y a une erreur, au moins une de ces sommes de vérification ne sera pas nulle.
Pour faire le lien avec les notions précédentes, écrivons la matrice $\mathcal{H}$ associée à ce système de 3 équations de vérification, où chaque équation sera une ligne de cette matrice. On obtient alors :

$$\mathcal{H} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

On peut remarquer que tous les nombres de 1 à 7 sont écrits en binaire dans les colonnes de cette matrice. Il devient évident alors qu'il s'agit de la matrice de contrôle d'un code de Hamming (de paramètres $[7, 4, 3]_2$) et que le vecteur S correspond au syndrome de C.

Cas q-aire

Définition 2.31 (Code de Hamming q-aire) *Soit m un entier positif supérieur ou égal à 2. Soit q la puissance $m^{\text{ième}}$ d'un nombre premier. Un code de Hamming q-aire est un code linéaire défini sur $\mathbb{F}_q$, obtenu en prenant comme colonnes d'une matrice de contrôle l'ensemble maximal des vecteurs non nuls de $\mathbb{F}_q^m$ avec la propriété qu'aucun n'est multiple d'un autre. Ce code est de longueur $n = (q^m - 1)/(q - 1)$, de dimension $k = n - m$, et de distance minimale $d = 3$.*

Les codes de Hamming sont des cas particuliers des codes BCH. Voyons maintenant cette classe plus générale de codes.

2.4.2 Codes de Bose, Chaudhuri et Hocquenghem (BCH)

Cette sous-section est inspirée du cours de Gilles Zémor sur les codes linéaires[2] *et du livre [McE02, Chapitre 9]. Le lecteur pourra également se référer à [MS77, Chapitres 3, 7 et 9].*

Les codes BCH ont été introduits séparément par Alexis Hocquenghem en 1959 [Hoc59] et par Raj Bose et Dijen (Ray-)Chauduri en 1960 [BRC60]. Les codes BCH binaires ont d'abord été construits comme une généralisation des codes de Hamming binaires, donnant des codes 2-correcteur. Une explication détaillée est donnée dans [McE02, Chapitre 9]. L'idée générale est la suivante : Rappelons qu'une matrice de contrôle d'un code de Hamming binaire de longueur $n = 2^m - 1$ est donnée par :

$$\mathcal{H} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix},$$

où $(v_1, v_2, \dots, v_n)$ sont tous les vecteurs (colonnes) non nuls de $\mathbb{F}_{2^m}$ dans un ordre donné. La matrice $\mathcal{H}$ étant de taille $m \times n$, m symboles de contrôle sont donc utilisés pour corriger une erreur (vecteur de poids 1). Il est alors assez intuitif de vouloir rajouter m symboles supplémentaires afin d'être capable de corriger deux erreurs (vecteur de poids 2). Cette hypothèse nous mène à une matrice de contrôle de la forme suivante :

$$\tilde{\mathcal{H}} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \\ w_1 & w_2 & \dots & w_n \end{bmatrix},$$

où $(w_1, w_2, \dots, w_n)$ sont tous les vecteurs (colonnes) non nuls de $\mathbb{F}_{2^m}$ dans un ordre donné, mais différent de celui des v_i (sinon la matrice ne serait pas de rang plein et ne pourrait donc pas servir de matrice de contrôle d'un code de longueur n). Comme les v_i sont tous distincts, on peut considérer les w_i comme étant l'image par une certaine fonction f des v_i, de $\mathbb{F}_{2^m}$ dans lui-même et ré-écrire $\tilde{\mathcal{H}}$ comme :

$$\tilde{\mathcal{H}} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \\ f(v_1) & f(v_2) & \dots & f(v_n) \end{bmatrix}.$$

Le choix de cette fonction f dépend directement de la fonction syndrome $S_{\tilde{\mathcal{H}}}$. En effet, le but ici est que la matrice de contrôle $\tilde{\mathcal{H}}$ nous permette de corriger jusqu'à 2 erreurs. Notons i et j les indices des colonnes correspondant aux 2 erreurs dans le vecteur E et $S_{\tilde{\mathcal{H}}}(E) = (s_1, s_2)$ le syndrome de E. On obtient alors le système suivant :

$$\begin{cases} v_i & + & v_j & = & s_1 \\ f(v_i) & + & f(v_j) & = & s_2 \end{cases},$$

2. `https://www.math.u-bordeaux.fr/~gzemor/codes06.pdf`

que l'on veut pouvoir résoudre avec au plus une seule solution. Un premier mauvais choix serait de prendre f linéaire, donnant ainsi deux équations dépendantes au système précédent. On doit donc prendre f non-linéaire. Un second mauvais choix serait de prendre f comme la fonction carrée, car nous travaillons ici en caractéristique 2 et donc $(v_i + v_j)^2 = v_i^2 + v_j^2$, donnant encore une fois deux équations dépendantes au système précédent.
En utilisant le fait que toute fonction $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ peut être représentée par un polynôme et que les polynômes de degrés $\leqslant 2$ ne fonctionnent pas, on en arrive à choisir $f : x \mapsto x^3$ et donc à une matrice de contrôle de la forme suivante :

$$\tilde{\mathcal{H}} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \\ v_1^3 & v_2^3 & \dots & v_n^3 \end{bmatrix}.$$

De manière générale, pour corriger t erreurs, on en vient à la définition qui suit.

Définition 2.32 (Code BCH binaire) *Soit $q = 2$. Soit m un entier positif supérieur ou égal à 2. Un code BCH binaire t-correcteur est un code linéaire défini sur $\mathbb{F}_2$, obtenu en prenant comme matrice de contrôle la matrice dont la première ligne est constituée de tous les vecteurs non nuls de $\mathbb{F}_2^m$, la deuxième est constitué des éléments de la première au cube, la troisième des puissances 5 des éléments de la première, ..., jusqu'à la ligne t qui correspond aux puissances $2t-1$ des éléments de la première. Ce code est de longueur $n = 2^m - 1$, de dimension $k \geqslant n - tm$, et de distance minimale $d \geqslant 2t + 1$.*

Théorème 2.3 *Soit $\bar{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n)$ tous les éléments non nuls distincts de $\mathbb{F}_{2^m}$ (dans un certain ordre). Soit un entier positif $t \leqslant 2^{m-1} - 1$. Le code BCH sur $\mathbb{F}_{2^m}$ de longueur $n = 2^m - 1$ et de dimension $k \geqslant n - m \cdot t$, capable de corriger t erreurs admet comme matrice de contrôle :*

$$\mathcal{H} = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \alpha_1^3 & \alpha_2^3 & \dots & \alpha_n^3 \\ \alpha_1^5 & \alpha_2^5 & \dots & \alpha_n^5 \\ \vdots & \vdots & \dots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \dots & \alpha_n^{2t-1} \end{bmatrix},$$

où chaque composante est remplacée par la colonne correspondante de m éléments de $\mathbb{F}_2$.

Définition 2.33 (Code BCH général) *Soit $\mathbb{F}_q$ un corps fini. Soient n et m deux entiers tels que $\operatorname{pgcd}(n, q) = 1$ et m le plus petit entier vérifiant $q^m \equiv 1 \mod n$. Soient b un certain entier positif et δ la distance désignée (c'est-à-dire $2 \leqslant \delta \leqslant n$). Soit α une racine primitive $n^{\text{ième}}$ de l'unité dans $\mathbb{F}_{q^m}$. Alors le code BCH général est défini sur $\mathbb{F}_q$ par :*

$$BCH_{n,k}(\alpha, b, \delta) = \left\{ (v_1, v_2, \dots, v_n) \in \mathbb{F}_q^n ; \forall i \in [\![0, \delta - 2]\!], \sum_{j=1}^{n} v_j \cdot \alpha^{(b+i)(j-1)} = 0 \in \mathbb{F}_{q^m} \right\}.$$

Remarque 2.13 *Un code BCH ainsi défini est dit :*
- *au sens strict si $b = 1$,*
- *primitif si $n = q^m - 1$.*

Nous allons maintenant voir un autre cas particulier de codes BCH généraux, les codes de Reed-Solomon.

2.4.3 Codes de Reed-Solomon (RS)

Cette sous-section est inspirée de [WY09, p. 425]. Le lecteur pourra également se référer à [MS77, Chapitre 10].

Les codes de Reed-Solomon ont été introduits par Irving Reed et Gustave Solomon en 1960 [RS60]. Ils sont en un sens une généralisation des codes BCH binaires au cas q-aire ($q > 2$). Mais ils restent des cas particuliers des codes BCH généraux.

Définition 2.34 (Code RS) *Soit $\mathbb{F}_q$ un corps fini. Soient k et n deux entiers tels que $k \leqslant n$. Soient $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ des éléments distincts de $\mathbb{F}_q$. Un code de Reed-Solomon $RS_{n,k}(\bar{\alpha})$ sur $\mathbb{F}_q$ de longueur n et de dimension k est défini par :*

$$RS_{n,k}(\bar{\alpha}) = \left\{(P(\alpha_1), P(\alpha_2), \ldots, P(\alpha_n)) \in \mathbb{F}_q^n; P \in \mathbb{F}_q[X], \deg(P) < k\right\}.$$

Ce code est de longueur n, de dimension k et de distance minimale $d = n - k + 1$.

Propriété 2.3 *Une matrice génératrice $\mathcal{G}$ du code de Reed-Solomon défini précédemment est donnée par :*

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_n^{k-1} \end{bmatrix}$$

Théorème 2.4 *Soit $\mathbb{F}_q$ un corps fini et $\mathbb{F}_{q^m}$ son extension de degré m. Le code RS $RS_{n,k}$ de longueur $n = q^m - 1$ est défini par :*

$$RS_{n,k} = \left\{(v_1, v_2, \ldots, v_n) \in \mathbb{F}_{q^m}^n; \forall i \in [\![1, 2t]\!], \sum_{j=1}^{n} v_j \alpha^{(j-1)i} = 0 \in \mathbb{F}_{q^m}\right\},$$

où α est une racine primitive fixée de $\mathbb{F}_{q^m}$.

Propriété 2.4 *Un code RS de longueur $n = q - 1$ est un code BCH de même longueur sur $\mathbb{F}_q$.*

Les codes de Reed-Solomon sont utilisés notamment pour la lecture des CD/DVD, car ils ont un bon comportement vis à vis des erreurs groupées causées par exemple par une rayure [VO82]. Ils sont aussi utilisés par des satellites pour la transmission d'images, comme celui d'exploration de Jupiter *Galileo* [DRTV07, Chapitre 4, page 254].

2.4.4 Codes de Reed-Solomon généralisés (*Generalized Reed-Solomon*, noté GRS)

Cette sous-section est inspirée de [WY09, p. 425]. Le lecteur pourra également se référer à [MS77, Chapitre 10].

Définition 2.35 (Code GRS) *Un code de Reed-Solomon généralisé $GRS_k(\bar{\alpha}, \bar{v})$ sur $\mathbb{F}_{q^m}$ de longueur $n \leqslant q^m - 1$ et de dimension k est défini par un ensemble appelé son support $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, qui est un n-uplet d'éléments non nuls ordonnés de $\mathbb{F}_{q^m}$ deux à deux distincts, et par un vecteur appelé son multiplicateur $\bar{v} = (v_1, v_2, \ldots, v_n)$, qui est une suite d'éléments non nuls de $\mathbb{F}_{q^m}$. On a alors :*

$$GRS_k(\bar{\alpha}, \bar{v}) = \left\{(v_1 \cdot P(\alpha_1), v_2 \cdot P(\alpha_2), \ldots, v_n \cdot P(\alpha_n)); P \in \mathbb{F}_{q^m}[X], \deg(P) < k\right\}.$$

Un code GRS correspond donc à un code RS dont on multiplie les colonnes par des éléments non nuls de $\mathbb{F}_{q^m}$. La multiplication d'une colonne par un élément non nul ne modifiant pas la distance minimale, les codes GRS ont les mêmes paramètres que les codes RS.

Nous allons maintenant introduire les codes alternants, qui sont des restrictions au corps de base de codes GRS.

2.4.5 Codes alternants

Cette sous-section est inspirée de [WY09, p. 425]. Le lecteur pourra également se référer à [MS77, Chapitre 12].

Définition 2.36 (Code alternant) *Soit $GRS_{n-r}(\bar{\alpha}, \bar{v})$ un code GRS de paramètres $[n, n-r, r+1]_{q^m}$ construit sur une extension $\mathbb{F}_{q^m}$ de $\mathbb{F}_q$. Un code alternant $A_k(\bar{\alpha}, \bar{y})$ est défini comme l'ensemble des mots du code $GRS_{n-r}(\bar{\alpha}, \bar{v})$ dont les coordonnées sont à coefficients dans le sous-corps $\mathbb{F}_q$.*

Proposition 2.8 *Soit $GRS_{n-r}(\bar{\alpha}, \bar{v})$ un code GRS de paramètres $[n, n-r, r+1]_{q^m}$ construit sur une extension $\mathbb{F}_{q^m}$ de $\mathbb{F}_q$. Le code alternant $A_k(\bar{\alpha}, \bar{y})$ correspondant (sur $\mathbb{F}_q$) est un code de paramètres $[n, k, d]_q$ avec $n - mr \leqslant k \leqslant n - r$ et $d \geqslant r + 1$.*

Théorème 2.5 *Soit $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ des éléments distincts de $\mathbb{F}_{q^m}$. Soit $\bar{y} = (y_1, y_2, \ldots, y_n)$ des éléments non nuls de $\mathbb{F}_{q^m}$ (non nécessairement distincts). Soit $r \in \mathbb{N}^*$ et soit $\mathcal{Y}$ la matrice de Vandermonde à r lignes de $\bar{\alpha}$, donnée par :*

$$\mathcal{Y} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_n^{r-1} \end{bmatrix}. \tag{2.4}$$

Soit $\mathcal{Z}$ la matrice diagonale de $\bar{y}$ donnée par :

$$\mathcal{Z} = \begin{bmatrix} y_1 & 0 & \ldots & 0 \\ 0 & y_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & y_n \end{bmatrix}. \tag{2.5}$$

Soit $\mathcal{H} = \mathcal{Y} \cdot \mathcal{Z}$, c'est-à-dire :

$$\mathcal{H} = \begin{bmatrix} y_1 & y_2 & \ldots & y_n \\ \alpha_1 \cdot y_1 & \alpha_2 \cdot y_2 & \ldots & \alpha_n \cdot y_n \\ \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{r-1} \cdot y_1 & \alpha_2^{r-1} \cdot y_2 & \ldots & \alpha_n^{r-1} \cdot y_n \end{bmatrix}. \tag{2.6}$$

Alors le code alternant de $\bar{\alpha}$ et $\bar{y}$, noté $A(\bar{\alpha}, \bar{y})$, est le code dont une matrice de contrôle est la matrice $\mathcal{H}$ vue sur $\mathbb{F}_q$, c'est-à-dire que chaque coordonnée $h_{i,j} \in \mathbb{F}_{q^m}$ de $\mathcal{H}$ est écrite comme un vecteur colonne à m composantes sur $\mathbb{F}_q$. Cette matrice sur $\mathbb{F}_q$ sera notée $\mathcal{H}_q$.

Les codes de Goppa, que nous allons définir ci-après, sont des cas particuliers de codes alternants.

2.5 Codes de Goppa (classiques)

Les codes de Goppa sont une classe de codes correcteurs d'erreurs linéaires qui a été proposée par Valery Denlsovlch Goppa en 1970 dans [Gop70]. On distingue les codes de Goppa "classiques" des codes de Goppa "géométriques". Ces derniers généralisent les premiers et ont été proposés plus tardivement dans [Gop83].

Comme à l'origine Robert J. McEliece prévoyait l'usage des codes de Goppa classiques binaires irréductibles pour son cryptosystème [McE78], nous utiliserons essentiellement cette sous-classe de codes linéaires dans cette thèse, pour nos attaques contre ce cryptosystème.

2.5.1 Définition et propriétés

Définition 2.37 (Code de Goppa classique) *Soit q une puissance d'un nombre premier p. Soient $m > 0$, $n \leqslant q^m$, $\mathscr{L} \triangleq \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$ et $G(X) \in \mathbb{F}_{q^m}[X]$ un polynôme de degré t tel que $\forall i \in \{1, 2, \ldots, n\}$, $G(\alpha_i) \neq 0$. Le code de Goppa de longueur n est défini par :*

$$\Gamma(\mathscr{L}, G) \triangleq \left\{ C = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n ; \sum_{i=1}^{n} \frac{c_i}{X - \alpha_i} \equiv 0 \bmod G(X) \right\}.$$

On dit que $\mathscr{L}$ est le support et G est le polynôme de Goppa de $\Gamma(\mathscr{L}, G)$.

Proposition 2.9 *La dimension de ce code est $k \geqslant n - mt$ et sa distance minimale $d \geqslant t + 1$.*

Remarque 2.14 *On dit que $\Gamma(\mathscr{L}, G)$ est irréductible si et seulement si G est irréductible dans l'anneau de polynôme auquel il appartient. De plus, dans le cas binaire on a $d \geqslant 2t+1$. (Cela découle de la proposition suivante.) Le code $\Gamma(\mathscr{L}, G)$ est alors t-correcteur, où $t = \deg(G)$.*

Proposition 2.10 *Soit $G(X) \in \mathbb{F}_{2^m}[X]$ un polynôme de Goppa sans racine multiple sur $\mathbb{F}_{2^m}$. Alors $\Gamma(\mathscr{L}, G) = \Gamma(\mathscr{L}, G^2)$.*

Preuve *La preuve se trouve dans [SKHN76].*

Remarque 2.15 *Dans cette thèse, on considérera le plus souvent $n = 2^m$, G irréductible, et on aura $k = n - mt$.*

Il y a deux façons de voir les codes de Goppa : celle provenant des codes alternants et celle provenant des codes BCH. Ces deux visions des choses nous mènent à deux formes de matrices de contrôle bien particulières.

Théorème 2.6 *Soient $\mathscr{L} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \subseteq \mathbb{F}_{q^m}$ un sous-ensemble du corps à q^m éléments et $G(X)$ un polynôme de degré t sur $\mathbb{F}_{q^m}$. Une matrice de contrôle du code $\bar{\mathcal{H}}_q$ de Goppa $\Gamma(\mathscr{L}, G)$ peut être construite à partir d'une matrice $\bar{\mathcal{H}}$ dans $\mathbb{F}_{q^m}$*

de la façon suivante :

$$\begin{aligned}
\bar{\mathcal{H}} &= \underbrace{\begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & \dots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \end{pmatrix}}_{\mathcal{Y}} \cdot \underbrace{\begin{pmatrix} \frac{1}{G(\alpha_1)} & 0 & \dots & 0 \\ 0 & \frac{1}{G(\alpha_2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{G(\alpha_n)} \end{pmatrix}}_{\mathcal{Z}} \\
&= \begin{pmatrix} \frac{1}{G(\alpha_1)} & \frac{1}{G(\alpha_2)} & \dots & \frac{1}{G(\alpha_n)} \\ \frac{\alpha_1}{G(\alpha_1)} & \frac{\alpha_2}{G(\alpha_2)} & \dots & \frac{\alpha_n}{G(\alpha_n)} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\alpha_1^{t-1}}{G(\alpha_1)} & \frac{\alpha_2^{t-1}}{G(\alpha_2)} & \dots & \frac{\alpha_n^{t-1}}{G(\alpha_n)} \end{pmatrix} \qquad (2.7)
\end{aligned}$$

en considérant une base de $\mathbb{F}_{q^m}$ sur $\mathbb{F}_q$ (comme pour les codes alternants). Le code de Goppa $\Gamma(\mathscr{L}, G)$ est bien défini comme étant l'ensemble des vecteurs $x \in \mathbb{F}_q^n$ tels que $\bar{\mathcal{H}}_q \cdot {}^t x = 0$.

Remarque 2.16 *La matrice $\bar{\mathcal{H}}_q$ est une matrice de contrôle d'un code de Goppa $\Gamma(\mathscr{L}, G)$ vu comme un code alternant.*

Afin d'illustrer cette section, nous allons prendre un exemple simple de code de Goppa.

Exemple (Code de Goppa classique binaire irréductible) *Soit $\Gamma(\mathscr{L}, G)$ un code de Goppa classique binaire irréductible de paramètres $[8, 2, 5]_2$, avec $L = \mathbb{F}_2[X]/(X^3 + X + 1)$ et $G(X) = X^2 + X + 1$. Regardons comment construire ce code.*

Soient $\mathbb{F}_8 = \mathbb{F}_2[X]/(X^3+X+1)$ et $\beta = X \mod (X^3+X+1)$ une racine primitive de $\mathbb{F}_8$. On considère le code de Goppa classique binaire irréductible $\Gamma(\mathscr{L}, G)$, où

$$\begin{aligned}
\mathcal{L} &= \{\alpha_1, \alpha_2, \dots, \alpha_8\} \\
&= \{0, 1, \beta, \beta^2, \dots, \beta^6\} \\
&= \mathbb{F}_{2^3},
\end{aligned}$$

et $G(X) = X^2 + X + 1$. On a donc $t = \deg(G(X)) = 2$ (le code est 2-correcteur), car G est bien irréductible. De plus, on a $m = 3$, $n = 2^3 = 8$, $k \geqslant n - mt = 8 - 3 \times 2$ et $d \geqslant t + 1$ (ici on a même $d \geqslant 2t + 1$, car G est irréductible). On peut également pré-calculer les tables suivantes :

X	X	X	$(1, \beta, \beta^2)$	$G(X)$	$G^{-1}(X)$	$X \cdot G^{-1}(X)$
α_1	0	0	$(0,0,0)$	1	1	0
α_2	1	1	$(1,0,0)$	1	1	1
α_3	β	β	$(0,1,0)$	$\beta^2+\beta+1$	β^2	$\beta+1$
α_4	β^2	β^2	$(0,0,1)$	$\beta+1$	$\beta^2+\beta$	β^2+1
α_5	β^3	$\beta+1$	$(1,1,0)$	$\beta^2+\beta+1$	β^2	$\beta^2+\beta+1$
α_6	β^4	$\beta^2+\beta$	$(0,1,1)$	β^2+1	β	$\beta^2+\beta+1$
α_7	β^5	$\beta^2+\beta+1$	$(1,1,1)$	β^2+1	β	β^2+1
α_8	β^6	β^2+1	$(1,0,1)$	$\beta+1$	$\beta^2+\beta$	$\beta^2+\beta$

Voici la table de la somme de deux éléments dans $\mathbb{F}_8$:

$\oplus$	0	1	β	β^2	β^3	β^4	β^5	β^6
0	0	1	β	β^2	β^3	β^4	β^5	β^6
1	1	0	β^3	β^6	β	β^5	β^4	β^2
β	β	β^3	0	β^4	1	β^2	β^6	β^5
β^2	β^2	β^6	β^4	0	β^5	β	β^3	1
β^3	β^3	β	1	β^5	0	β^6	β^2	β^4
β^4	β^4	β^5	β^2	β	β^6	0	1	β^3
β^5	β^5	β^4	β^6	β^3	β^2	1	0	β
β^6	β^6	β^2	β^5	1	β^4	β^3	β	0

Voici la table du produit de deux éléments dans $\mathbb{F}_8$:

$\cdot$	0	1	β	β^2	β^3	β^4	β^5	β^6
0	0	0	0	0	0	0	0	0
1	0	1	β	β^2	β^3	β^4	β^5	β^6
β	0	β	β^2	β^3	β^4	β^5	β^6	1
β^2	0	β^2	β^3	β^4	β^5	β^6	1	β
β^3	0	β^3	β^4	β^5	β^6	1	β	β^2
β^4	0	β^4	β^5	β^6	1	β	β^2	β^3
β^5	0	β^5	β^6	1	β	β^2	β^3	β^4
β^6	0	β^6	1	β	β^2	β^3	β^4	β^5

Construisons une matrice de contrôle de ce code en utilisant la forme $\bar{\mathcal{H}}_G = \mathcal{Y} \cdot \mathcal{Z} \in \mathscr{M}_{t,n}(\mathbb{F}_{2^3})$ calculée avec G :

$$\begin{aligned}\bar{\mathcal{H}}_G &= \begin{pmatrix} \frac{G(\alpha_1)^{-1}}{\alpha_1 \cdot G(\alpha_1)^{-1}} & \frac{G(\alpha_2)^{-1}}{\alpha_2 \cdot G(\alpha_2)^{-1}} & \frac{G(\alpha_3)^{-1}}{\alpha_3 \cdot G(\alpha_3)^{-1}} & \frac{G(\alpha_4)^{-1}}{\alpha_4 \cdot G(\alpha_4)^{-1}} & \frac{G(\alpha_5)^{-1}}{\alpha_5 \cdot G(\alpha_5)^{-1}} & \frac{G(\alpha_6)^{-1}}{\alpha_6 \cdot G(\alpha_6)^{-1}} & \frac{G(\alpha_7)^{-1}}{\alpha_7 \cdot G(\alpha_7)^{-1}} & \frac{G(\alpha_8)^{-1}}{\alpha_8 \cdot G(\alpha_8)^{-1}} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & \beta^2 & \beta^4 & \beta^2 & \beta & \beta & \beta^4 \\ 0 & 1 & \beta^3 & \beta^6 & \beta^5 & \beta^5 & \beta^6 & \beta^3 \end{pmatrix}\end{aligned}$$

Notons que $\bar{\mathcal{H}}_{G,2} \in \mathcal{M}_{mt,n}(\mathbb{F}_2)$, une véritable matrice de contrôle de $\Gamma(\mathscr{L}, G)$, est à valeurs dans $\mathbb{F}_2$, avec $\mathcal{B} = (1, \beta, \beta^2)$ pour base de $\mathbb{F}_{2^3}$, donc $|\mathcal{B}| = 3 = m$. Mais le plus souvent la matrice de contrôle sera écrite dans le gros corps.

$$\bar{\mathcal{H}}_{G,2} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Pour trouver $\mathcal{G}$ à partir de $\bar{\mathcal{H}}_G$, il faut mettre $\bar{\mathcal{H}}_{G,2}$ sous forme systématique (notée $\tilde{\mathcal{H}}_G$), grâce à la méthode du pivot de Gauss, suivie d'une permutation des colonnes 6 et 7. On obtient alors :

$$\tilde{\mathcal{H}}_G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Comme $\tilde{\mathcal{H}}_G = [I_6|A]$, on aura alors $\tilde{\mathcal{G}} = [{}^tA|I_2]$, d'où :

$$\tilde{\mathcal{G}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Maintenant, retrouvons $\mathcal{G} \in \mathcal{M}_{k,n}(\mathbb{F}_2)$ *en appliquant la permutation inverse (de celle appliquée lors du Pivot de Gauss). On obtient donc :*

$$\mathcal{G} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

On obtient alors comme code de Goppa :

$$\Gamma(\mathscr{L}, G) = \{(0,0,0,0,0,0,0,0), (1,1,1,1,0,1,0,0), (1,1,0,0,1,0,1,1), (0,0,1,1,1,1,1,1)\}.$$

On peut rapidement vérifier que l'on a $k = 2$ *et* $d = 5$.

Construisons maintenant une matrice de contrôle de ce code en utilisant la forme $\bar{\mathcal{H}}_{G^2} = \mathcal{Y} \cdot \mathcal{Z} \in \mathscr{M}_{2t,n}(\mathbb{F}_{2^3})$ *calculée avec* G^2*, afin de vérifier que l'on retrouve bien le même code.*

On rappelle que $G^2(X) = (X^2 + X + 1)^2 = X^4 + X^2 + 1$. *On a donc* $2t = \deg(G^2(X)) = 4$. *On peut aussi pré-calculer la table suivante :*

X	X	X	$(1, \beta, \beta^2)$	$G(X)$	$G^2(X)$	$G^{-2}(X)$	$X \cdot G^{-2}(X)$	$X^2 \cdot G^{-2}(X)$	$X^3 \cdot G^{-2}(X)$
α_1	0	0	$(0,0,0)$	1	1	1	0	0	0
α_2	1	1	$(1,0,0)$	1	1	1	1	1	1
α_3	β	β	$(0,1,0)$	β^5	β^3	β^4	β^5	β^6	1
α_4	β^2	β^2	$(0,0,1)$	β^3	β^6	β	β^3	β^5	1
α_5	β^3	$\beta+1$	$(1,1,0)$	β^5	β^3	β^4	1	β^3	β^6
α_6	β^4	$\beta^2+\beta$	$(0,1,1)$	β^6	β^5	β^2	β^6	β^3	1
α_7	β^5	$\beta^2+\beta+1$	$(1,1,1)$	β^6	β^5	β^2	1	β^5	β^3
α_8	β^6	β^2+1	$(1,0,1)$	β^3	β^6	β	1	β^6	β^5

On obtient :

$$\bar{\mathcal{H}}_{G^2} = \begin{pmatrix} 1 & 1 & \beta^4 & \beta & \beta^4 & \beta^2 & \beta^2 & \beta \\ 0 & 1 & \beta^5 & \beta^3 & 1 & \beta^6 & 1 & 1 \\ 0 & 1 & \beta^6 & \beta^5 & \beta^3 & \beta^3 & \beta^5 & \beta^6 \\ 0 & 1 & 1 & 1 & \beta^6 & 1 & \beta^3 & \beta^5 \end{pmatrix},$$

d'où

$$\bar{\mathcal{H}}_{G^2,2} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Après la méthode du pivot de Gauss, on obtient :

$$\tilde{\mathcal{H}}_{G^2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

En supprimant les six dernières lignes, linéairement dépendantes des précédentes, on remarque que l'on retrouve la matrice $\tilde{\mathcal{H}}_G$ construite avec G. Donc pour trouver la matrice $\mathcal{G}$, il suffit de passer par la matrice $\bar{\mathcal{H}}_G$ construite avec G, car cela demande moins de calculs, mais pour le calcul du syndrome et avoir deux fois plus de coordonnées permettant de corriger deux fois plus d'erreurs (et donc pouvoir corriger non pas une erreur comme avec G mais deux avec G^2), il faut passer par $\bar{\mathcal{H}}_{G^2}$.

Une autre construction possible d'une matrice de contrôle pour un code de Goppa $\Gamma(\mathscr{L}, G)$ provient de la Définition 2.37, elle-même une généralisation des codes BCH.

Théorème 2.7 *Soient $\mathscr{L} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \subseteq \mathbb{F}_{q^m}$ un sous-ensemble du corps à q^m éléments et $G(X) = g_t \cdot X^t + g_{t-1} \cdot X^{t-1} + \ldots + g_1 \cdot X + g_0$ un polynôme de degré t sur $\mathbb{F}_{q^m}$. Une matrice de contrôle du code de Goppa $\Gamma(\mathscr{L}, G)$ peut être construite dans $\mathbb{F}_{q^m}$ de la façon suivante :*

$$\begin{aligned} \hat{\mathcal{H}} &= \underbrace{\begin{pmatrix} g_t & 0 & \ldots & 0 \\ g_{t-1} & g_t & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ g_1 & \ldots & g_{t-1} & g_t \end{pmatrix}}_{\mathcal{X}} \cdot \underbrace{\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \ldots & \alpha_n^{t-1} \end{pmatrix}}_{\mathcal{Y}} \cdot \underbrace{\begin{pmatrix} \frac{1}{G(\alpha_1)} & 0 & \ldots & 0 \\ 0 & \frac{1}{G(\alpha_2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{1}{G(\alpha_n)} \end{pmatrix}}_{\mathcal{Z}} \\ &= \begin{pmatrix} \frac{g_t}{G(\alpha_1)} & \frac{g_t}{G(\alpha_2)} & \ldots & \frac{g_t}{G(\alpha_n)} \\ \frac{g_{t-1}+\alpha_1 g_t}{G(\alpha_1)} & \frac{g_{t-1}+\alpha_2 g_t}{G(\alpha_2)} & \ldots & \frac{g_{t-1}+\alpha_n g_t}{G(\alpha_n)} \\ \vdots & \vdots & \ldots & \vdots \\ \frac{\sum_{i=1}^{t} \alpha_1^{i-1} g_i}{G(\alpha_1)} & \frac{\sum_{i=1}^{t} \alpha_2^{i-1} g_i}{G(\alpha_2)} & \ldots & \frac{\sum_{i=1}^{t} \alpha_n^{i-1} g_i}{G(\alpha_n)} \end{pmatrix}. \end{aligned} \tag{2.8}$$

Remarque 2.17 *Comme $\mathcal{X}$ est inversible, les matrices $\bar{\mathcal{H}} = \mathcal{Y} \cdot \mathcal{Z}$ et $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$ représentent deux fonctions syndromes ayant le même noyau, à savoir le code de Goppa $\Gamma(\mathscr{L}, G)$.*

Comme pour tout code correcteur d'erreur, la matrice de contrôle permet de calculer le syndrome d'un mot reçu, ici le polynôme syndrome de $C \in \mathbb{F}_2^n$, qui est en fait l'écriture sous forme polynomiale du syndrome défini dans le chapitre précédent :

$$S_C(X) = C \cdot {}^t\mathcal{H} \cdot {}^t(X^{t-1}, \ldots, X, 1).$$

Théorème 2.8 (Polynôme syndrome) *Le polynôme syndrome d'un mot reçu C est donné par :*

$$S_C(X) = -\sum_{i=1}^{n} c_i \times \frac{1}{G(\alpha_i)} \times \frac{G(X) - G(\alpha_i)}{X - \alpha_i}.$$

Remarque 2.18 *On a :*

$$S_C(X) \equiv \sum_{i=1}^{n} \frac{c_i}{X - \alpha_i} \mod G(X).$$

Propriété 2.5 *Soit $G(X) \in \mathbb{F}_{q^m}[X]$ un polynôme de Goppa. $\forall \beta \in \mathbb{F}_{q^m}^*$, $\beta \cdot G(X)$ et $G(X)$ définissent le même code.*

Preuve *(D'après [Vér92, Chapitre IV].)*
Les polynômes $\beta \cdot G(X)$ et $G(X)$ ont les mêmes racines.

Remarque 2.19 *On peut donc se restreindre au cas où $G(X)$ est unitaire.*

Propriété 2.6 *Le nombre de polynômes irréductibles unitaires de degré t à coefficients dans $\mathbb{F}_{2^m}$ est d'environ $2^{mt}/t$.*

Propriété 2.7 *Le nombre de codes de Goppa irréductibles t-correcteurs de longueur n est d'environ $\binom{q^m}{n} \times \frac{q^{mt}}{t}$.*

Preuve *Pour construire $\mathscr{L}$, on a le choix de $\binom{q^m}{n}$ éléments. Pour avoir un polynôme de Goppa, de degré t, qui soit irréductible, on a le choix parmi environ $\frac{q^{mt}}{t}$ polynômes.*

On connaît en outre des algorithmes polynomiaux de décodage pour les codes de Goppa, donnés dans la Sous-section 2.5.2. Le décodage des codes de Goppa reposant essentiellement sur la résolution d'une équation, celle-ci est couramment appelée "équation-clé".

2.5.2 Décodage (résolution de l'équation-clé)

Considérons le mot erroné $\tilde{C} = C + E$, où C est un mot du code de Goppa $\Gamma(\mathscr{L}, G)$ et $E = (e_1, e_2, \ldots, e_n)$ un vecteur erreur de poids $w_H(E) \leqslant t$. Le décodage de $\tilde{C}$ consiste à résoudre une certaine équation-clé. Mais avant de la présenter, nous avons besoin des définitions suivantes.

Définition 2.38 (Polynôme localisateur d'erreurs) *Le polynôme localisateur de l'erreur E dans $\tilde{C} = C + E$ est défini par :*

$$\sigma_E(X) = \prod_{\substack{i=1 \\ e_i \neq 0}}^{n} (X - \alpha_i).$$

Définition 2.39 (Polynôme évaluateur d'erreurs) *Le polynôme évaluateur de l'erreur E dans $\tilde{C} = C + E$ est défini par :*

$$\eta_E(X) = \sum_{\substack{i=1 \\ e_i \neq 0}}^{n} \prod_{\substack{j=1 \\ e_j \neq 0 \\ j \neq i}}^{n} (X - \alpha_j).$$

Par la suite, les polynômes localisateur d'erreurs et évaluateur d'erreurs pourront respectivement être abrégés PLE et PEE.

Remarque 2.20 *Pour $\sigma_E(X)$ et $\eta_E(X)$ définis comme précédemment, on a alors*

$$\eta_E(X) = \frac{d\sigma_E(X)}{dX}.$$

Proposition 2.11 *En utilisant les définitions précédentes du PLE et du PEE, corriger $\tilde{C}$ en C, si l'on connaît $S_{\tilde{C}}(X)$, revient à résoudre ce que l'on appelle l'équation-clé :*

$$S_{\tilde{C}}(X) \cdot \sigma_E(X) \equiv \eta_E(X) \mod G(X). \tag{2.9}$$

Preuve *Soit $\tilde{C} = C+E$, où $C \in \Gamma(\mathscr{L}, G)$ et $w_H(E) \leqslant t$. D'après la Remarque 2.18, on a :*

$$S_{\tilde{C}}(X) \equiv \sum_{i=1}^{n} \frac{\tilde{c}_i}{X - \alpha_i} \mod G(X).$$

De plus, comme $\tilde{C} = C + E$, on peut décomposer cette somme de la façon suivante :

$$\sum_{i=1}^{n} \frac{\tilde{c}_i}{X - \alpha_i} = \underbrace{\sum_{i=1}^{n} \frac{c_i}{X - \alpha_i}}_{= 0 \text{ car } C \in \Gamma(\mathscr{L}, G)} + \sum_{i=1}^{n} \frac{e_i}{X - \alpha_i}.$$

On obtient donc :

$$S_{\tilde{C}}(X) \equiv \sum_{\substack{i=1 \\ e_i=1}}^{n} \frac{1}{X - \alpha_i} \mod G(X).$$

En remarquant que :

$$\frac{\eta_E(X)}{\sigma_E(X)} = \sum_{\substack{i=1 \\ e_i=1}}^{n} \frac{1}{X - \alpha_i},$$

où $\sigma_E(X)$ ne peut être nul par définition mais vaut au moins 1, on a alors :

$$S_{\tilde{C}}(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \mod G(X).$$

□

Remarque 2.21 $S_{\tilde{C}}(X) = S_E(X)$.

Plusieurs algorithmes permettent de décoder un mot reçu d'un code de Goppa, autres que la recherche exhaustive. Tout d'abord, donnons les grandes lignes du décodage (Algorithme 1).

Algorithme 1 Décodage d'un code de Goppa

Entrée(s): $\tilde{C} = C + E$ un mot reçu tel que $C \in \Gamma(\mathscr{L}, G)$ un code de Goppa t-correcteur, E un vecteur erreur de poids $w_H(E) \leqslant t$, $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ le support du code et G le polynôme de Goppa.

Sortie(s): $C \in \Gamma(\mathscr{L}, G)$.

1: Calculer le syndrome de $\tilde{C}$: $S_{\tilde{C}}(X)$.
2: Résoudre l'équation-clé : $S_{\tilde{C}}(X) \cdot \sigma_E(X) = \eta_E(X) \mod G(X)$.
3: //(afin d'obtenir le polynôme localisateur d'erreurs $\sigma_E(X)$)
4: Construire le vecteur erreur E tel que $e_i = 1$ si $\sigma_E(\alpha_i) = 0$ et $e_i = 0$ sinon.
5: //Cela revient à chercher les racines de σ_E sur le support $\mathscr{L}$.
6: **Retourner** $C = \tilde{C} - E$.

Passons maintenant à une explication détaillée de chaque méthode pour la résolution de l'équation : les algorithmes d'Euclide étendu, de Berlekamp-Massey et de Patterson.

Remarque 2.22 *La forme de la matrice de contrôle utilisée pour le décodage dépend de l'algorithme choisi.*

Algorithme d'Euclide étendu (*Extended Euclidean Algorithm* (EEA) en Anglais)

Cet algorithme détermine le plus grand diviseur commun (pgcd) de deux polynômes $a(X)$ et $b(X)$ appartenant à $\mathbb{F}_{2^m}[X]$ avec $b(X) \neq 0$ et $\deg(a) \geqslant \deg(b)$, ainsi que deux polynômes $u(X)$ et $v(X)$, le tout vérifiant la relation de Bézout de la manière suivante (voir [MS77]) :

$$\text{pgcd}(a(X), b(X)) = u(X) \cdot a(X) + v(X) \cdot b(X).$$

Remarque 2.23 *Dans notre cas, cette équation devient :*

$$\text{pgcd}(S_{\tilde{C}}(X), G(X)) = u(X) \cdot S_{\tilde{C}}(X) + v(X) \cdot G(X),$$

avec $S_{\tilde{C}}(X)$ le polynôme syndrome de $\tilde{C}$ et $\deg(S_{\tilde{C}}) < \deg(G) = t$. Comme on a toujours $S_{\tilde{C}}(X) \equiv \dfrac{\eta_E(X)}{\sigma_E(X)} \mod G(X)$, on aura alors :

$$\text{pgcd}(\underbrace{S_{\tilde{C}}(X)}_{=a(X)}, \underbrace{G(X)}_{=b(X)}) = \eta_E(X) \mod G(X) \text{ et } u(X) = \sigma_E(X) \mod G(X).$$

Algorithme 2 Décodage par Euclide étendu

Entrée(s): Les polynômes $a(X)$ et $b(X)$ appartenant à $\mathbb{F}_{2^m}[X]$ avec $b(X) \neq 0$ et $\deg(a) \geqslant \deg(b)$.

Sortie(s): Le pgcd $d(X)$ de $a(X)$ et $b(X)$ et deux polynômes $u(X)$ et $v(X)$ tels que $d(X) = u(X) \cdot a(X) + v(X) \cdot b(X)$ avec $\deg(d) < \deg(b)$.

1: $r_{-1} \leftarrow a(X)$
2: $r_0 \leftarrow b(X)$
3: $u_0 \leftarrow 1$
4: $u_1 \leftarrow 0$
5: $v_0 \leftarrow 0$
6: $v_1 \leftarrow 1$
7: $i \leftarrow 1$
8: **Tant que** $r_i \neq 0$ **faire**
9: $\quad q_{i+1}(X) \leftarrow quotient(r_{i-1}(X), r_i(X))$
10: $\quad r_{i+1}(X) \leftarrow r_{i-1}(X) - q_{i+1}(X).r_i(X)$
11: $\quad u_{i+1}(X) \leftarrow u_{i-1}(X) - q_{i+1}(X).u_i(X)$
12: $\quad v_{i+1}(X) \leftarrow v_{i-1}(X) - q_{i+1}(X).v_i(X)$
13: $\quad i \leftarrow i + 1$
14: **Fin tant que**
15: $i \leftarrow i - 1$
16: **Retourner** $(r_i(X), u_i(X), v_i(X))$.

Remarque 2.24 *Le calcul du polynôme syndrome $S_{\tilde{C}}(X)$ se fait en utilisant la matrice de contrôle $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$, calculée avec le polynôme G^2.*

Exemple *En utilisant le même exemple que précédemment, c'est-à-dire $\Gamma(\mathscr{L}, G)$ le code de Goppa classique binaire irréductible de paramètres $[8, 2, 5]_2$, avec $\mathscr{L} = \mathbb{F}_2[X]/(X^3+X+1)$ et $G(X) = X^2+X+1$, la matrice de contrôle $\hat{\mathcal{H}}$ avec $G^2(X) = X^4+X^2+1$ est :*

$$\hat{\mathcal{H}}_{G^2} = \begin{pmatrix} \frac{g_4}{G^2(\alpha_1)} & \frac{g_4}{G^2(\alpha_2)} & \cdots & \frac{g_4}{G^2(\alpha_8)} \\ \frac{g_3+\alpha_1\cdot g_4}{G^2(\alpha_1)} & \frac{g_3+\alpha_2\cdot g_4}{G^2(\alpha_2)} & \cdots & \frac{g_3+\alpha_8\cdot g_4}{G^2(\alpha_8)} \\ \frac{g_2+\alpha_1\cdot g_3+\alpha_1^2\cdot g_4}{G^2(\alpha_1)} & \frac{g_2+\alpha_2\cdot g_3+\alpha_2^2\cdot g_4}{G^2(\alpha_2)} & \cdots & \frac{g_2+\alpha_8\cdot g_3+\alpha_8^2\cdot g_4}{G^2(\alpha_8)} \\ \frac{g_1+\alpha_1\cdot g_2+\alpha_1^2\cdot g_3+\alpha_1^3\cdot g_4}{G^2(\alpha_1)} & \frac{g_1+\alpha_2\cdot g_2+\alpha_2^2\cdot g_3+\alpha_2^3\cdot g_4}{G^2(\alpha_2)} & \cdots & \frac{g_1+\alpha_8\cdot g_2+\alpha_8^2\cdot g_3+\alpha_8^3\cdot g_4}{G^2(\alpha_8)} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & \beta^4 & \beta & \beta^4 & \beta^2 & \beta^2 & \beta \\ 0 & 1 & \beta^5 & \beta^3 & 1 & \beta^6 & 1 & 1 \\ 1 & 0 & \beta^3 & \beta^6 & \beta^6 & \beta^5 & \beta^3 & \beta^5 \\ 0 & 0 & \beta^4 & \beta & \beta^2 & \beta^2 & \beta & \beta^4 \end{pmatrix} \in \mathcal{M}_{2t,n}(\mathbb{F}_{2^m}).$$

Supposons que l'on ait reçu le mot de code erroné $\tilde{C} = (1, 0, 1, 0, 0, 0, 0, 0)$.
En utilisant $\hat{\mathcal{H}}_{G^2}$ calculée juste avant, on obtient comme syndrome :

$$S_{\tilde{C}} = \underbrace{\begin{pmatrix} 1+\beta^4 \\ 0+\beta^5 \\ 1+\beta^3 \\ 0+\beta^4 \end{pmatrix}}_{\substack{1^{\text{ère}} \text{ et } 3^{\text{ème}} \\ \text{colonnes de } \hat{\mathcal{H}}_{G^2}}} = \begin{pmatrix} \beta^5 \\ \beta^5 \\ \beta \\ \beta^4 \end{pmatrix}.$$

Le polynôme syndrome est alors $S_{\tilde{C}}(X) = \beta^5 \cdot X^3 + \beta^5 \cdot X^2 + \beta \cdot X + \beta^4$.
L'algorithme d'Euclide étendu prend en entrées $S_{\tilde{C}}(X) = \beta^5 \cdot X^3 + \beta^5 \cdot X^2 + \beta \cdot X + \beta^4$ et $G^2(X) = X^4 + X^2 + 1$.
Il retourne : $\beta^5 = G^2(X) \cdot (\beta^2 \cdot X + \beta^5) + S_{\tilde{C}}(X) \cdot (\beta^4 \cdot X^2 + \beta^5 X)$.
On obtient alors comme polynôme localisateur d'erreurs $\sigma_E(X) = \beta^4 \cdot X^2 + \beta^5 \cdot X$.
Après avoir parcouru tous les éléments du support, on trouve comme racines 0 et β.
Il y a deux erreurs, en position 1 et 3 : $E = (1, 0, 1, 0, 0, 0, 0, 0)$.
Le mot de code (sans erreur) est donc $C = (0, 0, 0, 0, 0, 0, 0, 0)$.

Algorithme de Berlekamp-Massey

Cet algorithme a d'abord été proposé par Elwyn Berlekamp pour les registres à décalage à rétroaction linéaire (LFSR) [Ber68], puis par James Massey en l'adaptant au décodage des codes BCH [Mas69]. En remarquant qu'un code BCH primitif est un code de Goppa avec $G(X) = X^{2t}$ [Ber73], on peut alors utiliser l'algorithme de Berlekamp-Massey avec de légères modifications pour décoder les codes de Goppa [Pat75, Section 4].

Remarque 2.25 *Le calcul du polynôme syndrome $S_{\tilde{C}}(X)$ se fait en utilisant la matrice de contrôle $\bar{\mathcal{H}} = \mathcal{Y} \cdot \mathcal{Z}$, calculée avec le polynôme G^2.*

On a $S_{\tilde{C}}(X)$ le polynôme syndrome du mot reçu $\tilde{C}$, calculé à partir de la matrice de contrôle $\bar{\mathcal{H}}_{G^2}$, et on cherche à résoudre l'équation-clé :

$$S_{\tilde{C}}(X) \cdot \sigma_E(X) = \eta_E(X) \mod G(X).$$

Plus précisément, on cherche le polynôme localisateur $\sigma_E(X)$ de l'erreur E dans $\tilde{C} = C + E$. Cette équation-clé est sensiblement la même pour les codes BCH.

Algorithme 3 Décodage par Berlekamp-Massey

Entrée(s): $S_0, \ldots, S_{2t-1} \in \mathbb{F}_{2^m}$ une séquence (les coefficients du polynôme syndrome du mot reçu $\tilde{C}$).

Sortie(s): Le polynôme minimal $f(X) \in \mathbb{F}_{2^m}[X]$ qui engendre cette séquence et sa complexité linéaire $L \in \mathbb{N}$ (c'est-à-dire le degré de f).

$L \leftarrow 0$
$m \leftarrow -1$
$f \leftarrow 1$ // $f \in \mathbb{F}_{2^m}[X]$
$g \leftarrow 1$ // $g \in \mathbb{F}_{2^m}[X]$
$d_m \leftarrow 1$
Pour $k = 0$ à $2t - 1$ **faire**
  $d_k \leftarrow S_k + \sum_{i=1}^{L} f_i \cdot S_{k-i}$ // d est le défaut
  Si $d_k \neq 0$ **Alors**
    $h \leftarrow f$ // h est une variable locale auxiliaire pour stocker la valeur de f
    $f \leftarrow f + \frac{d_k}{d_m} \cdot g \cdot X^{k-m}$ // + car nous sommes en caractéristique 2, sinon -
    Si $L \leqslant \frac{k}{2}$ **Alors**
      $L \leftarrow k + 1 - L$ // $L = \deg(f^k) + 1$
      $m \leftarrow k$ // m est 1 + le degré du dernier f qui engendre $S_0, \ldots, S_{k-1}$
      $g \leftarrow h$ // g est le dernier f qui engendre $S_0, \ldots, S_{k-1}$ c'est-à-dire $f^{(k-1)}$
    Fin si
  Fin si
  $k \leftarrow k + 1$
Fin pour
Retourner $(L, f(X))$

Remarque 2.26 *On aura alors $f(X)$ comme étant le polynôme réciproque de $\sigma_E(X)$ si $e_i = 0$ pour i correspondant à la position de 0 dans le support $\mathscr{L}$ et $f(X)$ comme étant le réciproque de $\dfrac{d\sigma_E(X)}{dX}$ si $e_i = 1$.*

Exemple *En utilisant le même exemple que précédemment, c'est-à-dire $\Gamma(\mathscr{L}, G)$ le code de Goppa classique binaire irréductible de paramètres $[8, 2, 5]_2$, avec $\mathscr{L} = \mathbb{F}_2[X]/(X^3 + X + 1)$ et $G(X) = X^2 + X + 1$, la matrice de contrôle $\bar{\mathcal{H}}$ avec $G^2(X) =$*

$X^4 + X^2 + 1$ *est :*

$$\bar{\mathcal{H}}_{G^2} = \begin{pmatrix} \frac{1}{G^2(\alpha_1)} & \frac{1}{G^2(\alpha_2)} & \cdots & \frac{1}{G^2(\alpha_8)} \\ \frac{\alpha_1}{G^2(\alpha_1)} & \frac{\alpha_2}{G^2(\alpha_2)} & \cdots & \frac{\alpha_8}{G^2(\alpha_8)} \\ \frac{\alpha_1^2}{G^2(\alpha_1)} & \frac{\alpha_2^2}{G^2(\alpha_2)} & \cdots & \frac{\alpha_8^2}{G^2(\alpha_8)} \\ \frac{\alpha_1^3}{G^2(\alpha_1)} & \frac{\alpha_2^3}{G^2(\alpha_2)} & \cdots & \frac{\alpha_8^3}{G^2(\alpha_8)} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & \beta^4 & \beta & \beta^4 & \beta^2 & \beta^2 & \beta \\ 0 & 1 & \beta^5 & \beta^3 & 1 & \beta^6 & 1 & 1 \\ 0 & 1 & \beta^6 & \beta^5 & \beta^3 & \beta^3 & \beta^5 & \beta^6 \\ 0 & 1 & 1 & 1 & \beta^6 & 1 & \beta^3 & \beta^5 \end{pmatrix}.$$

Supposons que l'on ait reçu le mot de code erroné $\tilde{C} = (1, 0, 1, 0, 0, 0, 0, 0)$.
En utilisant $\bar{\mathcal{H}}_{G^2}$ *calculée juste avant, on obtient comme syndrome :*

$$S_{\tilde{C}} = \begin{pmatrix} 1 + \beta^4 \\ 0 + \beta^5 \\ 0 + \beta^6 \\ 0 + 1 \end{pmatrix} = \begin{pmatrix} \beta^5 \\ \beta^5 \\ \beta^6 \\ 1 \end{pmatrix}.$$

Le polynôme syndrome est alors $S_{\tilde{C}}(X) = \beta^5 + \beta^5 \cdot X + \beta^6 \cdot X^2 + X^3$.
Voici la trace de l'algorithme de Berlekamp-Massey pour ce cas :
L'entrée est : $(S_0, S_1, S_2, S_3) = (\beta^5, \beta^5, \beta^6, 1)$
$L = 0, m = -1, f(X) = 1, g(X) = 1, d_{-1} = 1$

<u>$k = 0$</u>

- $d_0 = S_0 + \underbrace{\sum_{i=1}^{0} f_i \cdot S_{0-i}}_{=0} = S_0 = \beta^5$

<u>$d_0 \neq 0$</u>

- $h(X) = f(X) = 1$
- $f(X) = f(X) + \frac{d_0}{d_{-1}} \cdot g(X) \cdot X^{0-(-1)} = 1 + \frac{\beta^5}{1} \cdot X^1 = 1 + \beta^5 \cdot X$

<u>$L = 0 \leqslant \frac{0}{2}$</u>

- $L = 0 + 1 - 0 = 1$
- $m = 0$
- $g(X) = 1$

<u>$k = 1$</u>

- $d_1 = S_1 + \sum_{i=1}^{1} f_i \cdot S_{1-i} = S_1 + f_1 \cdot S_0 = \beta^5 + \beta^5 \cdot \beta^5 = \beta^5 + \beta^3 = \beta^2$

<u>$d_1 \neq 0$</u>

- $h(X) = 1 + \beta^5 \cdot X$
- $f(X) = 1 + \beta^5 \cdot X + \frac{d_1}{d_0} \cdot 1 \cdot X^{1-0} = 1 + \beta^5 \cdot X + \frac{\beta^2}{\beta^5} \cdot X = 1 + \beta^5 \cdot X + \beta^4 \cdot X = 1 + X$

$\underline{L = 1 > \frac{1}{2}}$

$\underline{k = 2}$

- $d_2 = S_2 + \sum_{i=1}^{1} f_i \cdot S_{2-i} = S_2 + f_1 \cdot S_1 = \beta^6 + 1 \cdot \beta^5 = \beta$

$\underline{d_2 \neq 0}$

- $h(X) = 1 + X$
- $f(X) = 1 + X + \frac{d_2}{d_0} \cdot 1 \cdot X^{2-0} = 1 + X + \beta^3 \cdot X^2$

$\underline{L = 1 \leqslant \frac{2}{2}}$

- $L = 2 + 1 - 1 = 2$
- $m = 2$
- $g(X) = 1 + X$

$\underline{k = 3}$

- $d_3 = S_3 + \sum_{i=1}^{2} f_i S_{3-i} = S_3 + f_1 \cdot S_2 + f_2 \cdot S_1 = 1 + 1 \cdot \beta^6 + \beta^3 \cdot \beta^5 = 1 + \beta^6 + \beta = \beta^2 + \beta = \beta^4$

$\underline{d_3 \neq 0}$

- $h(X) = 1 + x + \beta^3 \cdot X^2$
- $f(X) = 1 + x + \beta^3 \cdot X^2 + \frac{d_3}{d_2} \cdot (1 + X) \cdot X^{3-2} = 1 + X + \beta^3 \cdot X^2 + \frac{\beta^4}{\beta} \cdot (1 + X) \cdot X$
  $= 1 + X + \beta^3 \cdot X^2 + \beta^3 \cdot (1 + X) \cdot X = 1 + X + \beta^3 \cdot X^2 + \beta^3 \cdot X + \beta^3 X^2 = 1 + \beta \cdot X$

$\underline{L = 2 > \frac{3}{2}}$

L'algorithme retourne : $(L, f(X)) = (2, 1 + \beta \cdot X)$.
Le polynôme engendrant $(\beta^5, \beta^5, \beta^6, 1)$ *est* $f(X) = 1 + \beta \cdot X$ *et il faut* $L = 2$ *valeurs pour obtenir les autres. On obtient alors comme polynôme réciproque à* $f(X)$ *le polynôme* $\tilde{\sigma}(X) = X + \beta$. *Or,* $\deg f < L$ *donc on multiplie* $\tilde{\sigma}(X)$ *par* X *pour obtenir le polynôme localisateur d'erreurs* $\sigma(X) = X \cdot (X + \beta)$. *Ses racines sont donc* 0 *et* β. *Il y a des erreurs en position 1 et 3. Le mot de code était* $c = (0, 0, 0, 0, 0, 0, 0, 0)$.

Algorithme de Patterson

Cet algorithme a été proposé par Nicholas Patterson en 1975 dans [Pat75, Section 5] pour un décodage efficace des codes de Goppa binaires, utilisant des propriétés de la caractéristique pour réduire de moitié la taille des polynômes dans l'équation-clé.

Remarque 2.27 *Le calcul du polynôme syndrome* $S_{\tilde{C}}(X)$ *se fait en utilisant la matrice de contrôle* $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$, *calculée avec le polynôme* G. *C'est également le polynôme de Goppa* G *qui est utilisé dans l'Algorithme 4.*

On a $S_{\tilde{C}}(X)$ le polynôme syndrome du mot reçu $\tilde{C}$, calculé à partir de la matrice de contrôle $\hat{\mathcal{H}}_G$, et on cherche à résoudre l'équation-clé :

$$S_{\tilde{C}}(X) \cdot \sigma_E(X) = \eta_E(X) \mod G(X), \qquad \text{avec } \deg(\sigma_E) \leqslant t \text{ et } \deg(\eta_E) \leqslant (t-1).$$

L'algorithme de Patterson, présenté ici dans l'Algorithme 4, nous permet de nous ramener à la résolution d'une équation équivalente en caractéristique 2 :

$$R_{\tilde{C}}(X) \cdot \mathfrak{b}(X) = \mathfrak{a}(X) \mod G(X), \qquad \text{avec } \deg(\mathfrak{a}) \leqslant t/2 \text{ et } \deg(\mathfrak{b}) \leqslant (t-1)/2,$$

où $t = \deg(G)$. Le polynôme R est explicité ci-après.

Algorithme 4 Décodage par Patterson

Entrée(s): $S_{\tilde{C}}(X)$ le polynôme syndrome de $\tilde{C} = C + E$ (un mot de code erroné avec au plus t erreurs) et le code de Goppa $\Gamma(\mathscr{L}, G)$ de support $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ et avec le polynôme de Goppa G.

Sortie(s): $C \in \Gamma(\mathscr{L}, G)$ le mot de code sans erreur.

1: **Si** $S_{\tilde{C}}(X) = 0$ **Alors**
2: $\quad \sigma_E(X) = 1$
3: **Sinon**
4: $\quad$ Inverser le polynôme syndrome $T(X) = S_{\tilde{C}}^{-1}(X) \mod G(X)$. (EEA)
5: $\quad$ **Si** $T(X) = X$ **Alors**
6: $\quad\quad \sigma_E(X) = X$
7: $\quad$ **Sinon**
8: $\quad\quad$ Calculer $s(X) = \sqrt{X} \mod G(X)$.
$\quad\quad$ *(En posant* $s(X) = X^{2^{tm-1}} \mod G(X)$*.)*
9: $\quad\quad$ Calculer $R(X) = \sqrt{T(X) + X} \mod G(X)$.
$\quad\quad$ *En posant* $h(X) = T(X) + X$*, on utilise la formule suivante pour obtenir* $R(X) = \sqrt{h(X)} \mod G(X)$ *:*

$$R(X) = \underbrace{\sum_{i=0}^{(t-1)/2} h_{2i}^{2^{m-1}} \cdot X^i}_{\textit{monômes de degrés pairs}} + \underbrace{\sum_{i=0}^{(t/2)-1} h_{2i+1}^{2^{m-1}} \cdot X^i \cdot s(X)}_{\textit{monômes de degrés impairs}}$$

$\quad\quad$ *où* $s(X) = \sqrt{X} \mod G(X)$.
10: $\quad\quad$ Trouver $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ tels que $\mathfrak{a}(X) = \mathfrak{b}(X) \cdot R(X) \mod G(X)$, (EEA)
$\quad\quad$ avec $\deg(\mathfrak{a}) \leqslant t/2$ et $\deg(\mathfrak{b}) \leqslant (t-1)/2$.
11: $\quad\quad$ Construire le polynôme $\sigma_E(X)$ tel que : $\sigma_E(X) = \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X)$.
12: $\quad$ **Fin si**
13: **Fin si**
14: Construire le vecteur erreur E tel que $e_i = 1$ si $\sigma_E(\alpha_i) = 0$ et $e_i = 0$ sinon.
15: **Retourner** $C = \tilde{C} - E$.

Remarque 2.28 *Les étapes 4 et 10 se font à l'aide de l'algorithme d'Euclide étendu. Notant également que* $\mathrm{pgcd}(S_{\tilde{C}}(X), G(X))$ *est une constante car* $\deg(S_{\tilde{C}}) < \deg(G)$*, les racines de* $S_{\tilde{C}}(X)$ *sont des éléments du support* $\mathscr{L}$ *et que* $G(X)$ *est irréductible, donc* $T(X) = S_{\tilde{C}}^{-1}(X) \mod G(X)$ *existe. Pour le PLE* $\sigma_E(X)$ *construit ainsi, on a bien* $\deg(\sigma_E) \leqslant t$.

Expliquons maintenant pourquoi cet algorithme fonctionne, dont une preuve peut être trouvée dans [MS77, Chapitre 12, Théorème 16]. On cherche à résoudre l'équation-clé $\eta_E(X) = S_{\tilde{C}}(X) \cdot \sigma_E(X) \mod G(X)$, d'inconnue $\sigma_E(X)$, avec $\eta_E(X) = \frac{d\sigma_E(X)}{dX}$, où $\sigma_E(X)$, $\eta_E(X)$, $S_{\tilde{C}}(X)$ et $G(X)$ sont dans $\mathbb{F}_{2^m}[X]$ et avec $\deg(G) = t$. Posons $\sigma_E(X) = \mathfrak{a}(X)^2 + X \cdot \mathfrak{b}(X)^2$. On rappelle que c'est toujours possible en prenant dans $\mathfrak{a}$ les monômes de degrés pairs et dans $\mathfrak{b}$ les monômes de degrés impairs, car nous travaillons en caractéristique 2. De plus, comme $\frac{d\sigma_E(X)}{dX} = \mathfrak{b}(X)^2$, on a :

$$\begin{aligned}
\eta_E(X) &= S_{\tilde{C}}(X) \cdot \sigma_E(X) \mod G(X) \\
\mathfrak{b}(X)^2 &= S_{\tilde{C}}(X) \cdot (\mathfrak{a}(X)^2 + X \cdot \mathfrak{b}(X)^2) \mod G(X) \\
S_{\tilde{C}}(X) \cdot \mathfrak{a}(X)^2 &= \mathfrak{b}(X)^2 + X \cdot S_{\tilde{C}}(X) \cdot \mathfrak{b}(X)^2 \mod G(X) \\
S_{\tilde{C}}(X) \cdot \mathfrak{a}(X)^2 &= \mathfrak{b}(X)^2 \cdot (1 + X \cdot S_{\tilde{C}}(X)) \mod G(X)
\end{aligned}$$

Comme G est irréductible, $S_{\tilde{C}}(X)$ peut être inversé modulo $G(X)$.
Posons $h(X) = X + S_{\tilde{C}}^{-1}(X) \mod G(X)$. On a alors $\mathfrak{a}(X)^2 = h(X) \cdot \mathfrak{b}(X)^2 \mod G(X)$.
Puisque l'application $f(X) \mapsto f(X)^2 \mod G(X)$ est bijective et linéaire sur $\mathbb{F}_2^{tm}$, il existe un unique polynôme $R(X)$ tel que $R^2(X) = h(X) \mod G(X)$, d'où l'équation

$$\begin{aligned} \mathfrak{a}(X) &= R(X) \cdot \mathfrak{b}(X) \mod G(X), \\ \text{avec } R(X) &= \sqrt{h(X)} \mod G(X) = \sqrt{X + S_{\tilde{C}}^{-1}(X)} \mod G(X). \end{aligned}$$

Nous avons noté $T(X)$ le polynôme $S_{\tilde{C}}^{-1}(X) \mod G(X)$ dans l'Algorithme 4. Le couple des polynômes $(\mathfrak{a}(X), \mathfrak{b}(X))$ est l'unique solution de l'équation :

$$\begin{cases} R(X) \cdot \mathfrak{b}(X) = a(X) \mod G(X) \\ \deg(\mathfrak{a}) \leqslant t/2 \\ \deg(\mathfrak{b}) \leqslant (t-1)/2 \end{cases}$$

L'équation-clé a donc été réduite à une équation (toujours modulo G) sur des polynômes de degrés deux fois plus petits.

Exemple *En utilisant le même exemple que précédemment, c'est-à-dire $\Gamma(\mathscr{L}, G)$ le code de Goppa classique binaire irréductible de paramètres $[8, 2, 5]_2$, avec $\mathscr{L} = \mathbb{F}_2[X]/(X^3 + X + 1)$ et $G(X) = X^2 + X + 1$, la matrice de contrôle $\hat{\mathcal{H}}$ avec $G(X) = X^2 + X + 1$ est :*

$$\begin{aligned} \hat{\mathcal{H}}_G &= \begin{pmatrix} \frac{g_2}{G(\alpha_1)} & \frac{g_2}{G(\alpha_2)} & \cdots & \frac{g_2}{G(\alpha_8)} \\ \frac{g_1 + \alpha_1 \cdot g_2}{G(\alpha_1)} & \frac{g_1 + \alpha_2 \cdot g_2}{G(\alpha_2)} & \cdots & \frac{g_1 + \alpha_8 \cdot g_2}{G(\alpha_8)} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & \beta^2 & \beta^4 & \beta^2 & \beta & \beta & \beta^4 \\ 1 & 0 & \beta^5 & \beta^3 & \beta^3 & \beta^6 & \beta^5 & \beta^6 \end{pmatrix}. \end{aligned}$$

Supposons que l'on ait reçu le mot de code erroné $\tilde{C} = (1, 0, 1, 0, 0, 0, 0, 0)$.
En utilisant $\hat{\mathcal{H}}_G$ calculée juste avant, on obtient comme syndrome :

$$S_{\tilde{C}} = \begin{pmatrix} 1 + \beta^2 \\ 1 + \beta^5 \end{pmatrix} = \begin{pmatrix} \beta^6 \\ \beta^4 \end{pmatrix}.$$

Le polynôme syndrome est alors $S_{\tilde{C}}(X) = \beta^6 \cdot X + \beta^4$.
Voici la trace de l'Algorithme 4 (de Patterson) pour ce cas :

Ligne 1 : *On a $S_{\tilde{C}}(X) \neq 0$, donc on passe directement à la ligne 4.*

Ligne 4 : *Comme on obtient après division de $G(X)$ par $S_{\tilde{C}}(X)$ la relation suivante :*

$$\beta^6 = \underbrace{(X^2 + X + 1)}_{=G(X)} - (\beta \cdot X + \beta^5) \cdot \underbrace{(\beta^6 \cdot X + \beta^4)}_{=S_{\tilde{C}}(X)}$$

et que l'on souhaite obtenir l'inverse modulo $G(X)$ de $S_{\tilde{C}}(X)$, en notant que $\beta^6 = \beta^{-1}$, on a alors :

$$1 = \beta \cdot \underbrace{(X^2 + X + 1)}_{=G(X)} - (\beta^2 \cdot X + \beta^6) \cdot \underbrace{(\beta^6 \cdot X + \beta^4)}_{=S_{\tilde{C}}(X)}$$

On a donc :

$$\begin{aligned} T(X) &= S_{\tilde{C}}^{-1}(X) \mod G(X) \\ &= \beta^2 \cdot X + \beta^6 \end{aligned}$$

Ligne 5 : *On a $T(X) \neq X$, donc on passe directement à la ligne 8.*

Ligne 8 : *Calculons maintenant la racine carrée de X modulo $G(X)$ dans $\mathbb{F}_{2^m}[X]$. On a $t = 2$ et $m = 3$, donc on cherche à réduire $X^{2^{2\times 3-1}} = X^{2^5}$ modulo $G(X) = X^2 + X + 1$.*
Comme $X^{32} = (X^2+X+1)\cdot(X^{30}+X^{29}+X^{27}+X^{26}+X^{24}+X^{23}+X^{21}+X^{20}+X^{18}+X^{17}+X^{15}+X^{14}+X^{12}+X^{11}+X^9+X^8+X^6+X^5+X^3+X^2+1)+(X+1)$, on obtient alors :

$$\begin{aligned} s(X) &= \sqrt{X} \mod G(X) \\ &= X^{32} \mod G(X) \\ &= (X+1) \mod G(X) \end{aligned}$$

Ligne 9 : *Posons $h(X) = T(X) + X$. On a alors :*

$$\begin{aligned} h(X) &= T(X) + X \\ &= (\beta^2 \cdot X + \beta^6) + X \\ &= \beta^6 \cdot X + \beta^6 \end{aligned}$$

On veut construire $R(X)$ à l'aide de $h(X)$ de sorte que $R(X) = \sqrt{h(X)} \mod G(X)$. En appliquant la formule, on obtient :

$$\begin{aligned} R(X) &= h_0^{2^2} \cdot X^0 + h_1^{2^2} \cdot X^0 \cdot s(X) \\ &= (\beta^6)^4 \cdot 1 + (\beta^6)^4 \cdot 1 \cdot (X+1) \\ &= \beta^3 + \beta^3 \cdot X + \beta^3 \\ &= \beta^3 \cdot X \end{aligned}$$

Ligne 10 : *On cherche maintenant $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ tels que $\mathfrak{a}(X) = \mathfrak{b}(X) \cdot R(X) \mod G(X)$, avec $\deg(\mathfrak{a}) \leqslant t/2$ et $\deg(\mathfrak{b}) \leqslant (t-1)/2$.*
Après division de $G(X)$ par $R(X)$ on obtient la relation suivante :

$$1 = (X^2 + X + 1) - (\beta^3 \cdot X) \cdot (\beta^4 \cdot X + \beta^4).$$

Le problème est que ce n'est pas la solution, car la condition sur le degré du polynôme $\mathfrak{b}(X)$ n'est pas vérifiée, puisque $\deg(\beta^4 \cdot X + \beta^4) = 1$, tandis que $\deg(\mathfrak{b}) \leqslant (t-1)/2 = 1/2$. Mais nous pouvons mettre β^4 en facteur dans $(\beta^4 \cdot X + \beta^4)$, ré-écrire cette relation modulo $G(X)$ et remarquer que $(X+1) = \frac{1}{X} \mod G(X)$. On arrive alors à l'équation suivante :

$$\begin{aligned} 1 &= (\beta^3 \cdot X) \cdot \beta^4 \cdot (X+1) \mod G(X) \\ 1 &= (\beta^3 \cdot X) \cdot \beta^4 \cdot \frac{1}{X} \mod G(X) \\ X &= (\beta^3 \cdot X) \cdot \beta^4 \mod G(X) \end{aligned}$$

Cette fois les conditions sur les degrés sont vérifiées et on obtient alors :

$$\begin{aligned} \mathfrak{a}(X) &= X \\ \mathfrak{b}(X) &= \beta^4 \end{aligned}$$

Ligne 11 : *On peut maintenant construire $\sigma_E(X)$ tel que $\sigma_E(X) = \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X)$. On a alors :*

$$\begin{aligned} \sigma_E(X) &= \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X) \\ &= X^2 + X \cdot (\beta^4)^2 \\ &= X^2 + X \cdot \beta \end{aligned}$$

Ligne 14 : *La factorisation de* $\sigma_E(X)$ *est évident. On a donc* $\sigma_E(X) = X \cdot (X + \beta)$*. Les erreurs sont donc aux positions dans le support* $\mathscr{L}$ *de 0 et* β*. On obtient donc comme vecteur erreur* $E = (1, 0, 1, 0, 0, 0, 0, 0)$*. Le mot reçu corrigé est donc* $C = (0, 0, 0, 0, 0, 0, 0, 0)$*.*

L'algorithme de Patterson nous intéressera particulièrement dans la suite de cette thèse, puisque l'objectif ici est d'étudier les attaques par canaux auxiliaires contre celui-ci.

Dans la deuxième partie de cette thèse, nous allons présenter les systèmes de chiffrement à clé publique basés sur les codes correcteurs d'erreurs, de manière algorithmique et implantations d'une part et d'un point de vue attaques d'autre part.

Deuxième partie

Algorithmes, implantations et attaques (état de l'art)

Chapitre 3

Algorithmes de chiffrement et leurs implantations

Dans la partie précédente, nous avons parlé des généralités sur la cryptologie et vu une introduction aux codes correcteurs d'erreurs. Dans ce chapitre, nous allons nous intéresser aux systèmes de chiffrement basés sur les codes correcteurs d'erreurs, expliquant ainsi la terminologie "protocoles cryptographiques basés sur les codes correcteurs d'erreurs" du titre de cette thèse.
Historiquement, le premier cryptosystème basé sur les codes correcteurs d'erreurs a été proposé par McEliece en 1978 [McE78], soit la même année que le célèbre cryptosystème RSA basé sur le problème de la factorisation des entiers [RSA78]. Cependant, l'intérêt de la communauté pour le cryptosystème de McEliece fût plus modéré que celui pour le cryptosytème de RSA en raison de la taille des clés (que nous verrons ci-après). Avec les améliorations technologiques depuis, ceci n'est plus un réel problème de nos jours. De plus, cet intérêt s'accroît nettement depuis une dizaine d'année, car la cryptographie basée sur les codes correcteurs d'erreurs est l'une des cryptographies candidates à la cryptographie dite post-quantique. En effet, depuis que Shor a proposé un algorithme quantique pour résoudre en temps polynomial les problèmes de théorie des nombres [Sho94], sur lequel le cryptosystème RSA notamment est basé, et par rapport aux avancées en matière de construction d'un ordinateur quantique capable d'exécuter un tel algorithme [EF09, 6ème prédiction technologique], il est important de proposer une solution alternative à la cryptographie actuellement utilisée.
Dans le Chapitre 1, nous avons vu qu'un système de chiffrement à clé publique était constitué de trois algorithmes : la génération des clés, le chiffrement et le déchiffrement. Dans la suite de ce chapitre, nous allons présenter les systèmes de chiffrement asymétriques basés sur les codes ayant été proposés jusqu'à présent, en détaillant pour chacun ces trois algorithmes.

3.1 Cryptosystème de McEliece

Le premier protocole cryptographique basé sur les codes correcteurs d'erreurs est le système de chiffrement à clé publique proposé par Robert McEliece en 1978 [McE78]. Les algorithmes de génération de clés, de chiffrement et de déchiffrement sont constitués de la façon suivante.

3.1.1 Génération de clés

Comme ce cryptosystème est basé sur les codes, nous avons besoin de deux entiers n et t lors de la génération des clés, qui définiront respectivement la longueur et la capacité de correction d'un code linéaire, où t sera beaucoup plus petit que n. La génération des clés du cryptosystème de McEliece est présentée dans l'Algorithme 5.

Algorithme 5 Génération de clés de McEliece

Entrée(s): n et t deux entiers, avec $n \leqslant 2^m$.
Sortie(s): $p_k = (\tilde{\mathcal{G}}, t)$ la clé publique et $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \mathscr{C})$ la clé privée associée.

1: Choisir un code linéaire $\mathscr{C}$ de longueur n et t correcteur.
On notera k la dimension du code $\mathscr{C}$.
2: Prendre une matrice $\mathcal{G} \in \mathscr{M}_{k,n}(\mathbb{F}_2)$, génératrice de $\mathscr{C}$.
3: Choisir aléatoirement une matrice inversible $\mathcal{S} \in \mathscr{M}_{k,k}(\mathbb{F}_2)$.
4: Choisir aléatoirement une matrice de permutation $\mathcal{P} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$.
5: Calculer la matrice génératrice $\tilde{\mathcal{G}} = \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P}$.
6: $s_k \leftarrow (\mathcal{S}, \mathcal{G}, \mathcal{P}, \mathscr{C})$
7: $p_k \leftarrow (\tilde{\mathcal{G}}, t)$
8: **Retourner** (p_k, s_k).

Notons que n et k sont des données publiques puisque ces entiers sont donnés par la taille de $\tilde{\mathcal{G}}$. Une variante possible est de rajouter une étape dans l'Algorithme 5 pour calculer les inverses des matrices $\mathcal{S}$ et $\mathcal{P}$, c'est-à-dire les matrices $\mathcal{S}^{-1}$ et $\mathcal{P}^{-1}$, puis de les mettre dans la clé privée s_k à la place des matrices $\mathcal{S}$ et $\mathcal{P}$, donnant ainsi $s_k = (\mathcal{S}^{-1}, \mathcal{G}, \mathcal{P}^{-1}, \mathscr{C})$, car ce sont les inverses qui seront utilisées par la suite dans l'algorithme de déchiffrement (donné dans l'Algorithme 7) et non pas les matrices $\mathcal{S}$ et $\mathcal{P}$ directement. Une fois la génération de clés effectuée, voici comment utiliser ce cryptosystème.

3.1.2 Chiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du message à chiffrer et de la clé publique pour la phase de chiffrement. Le chiffrement de McEliece est présenté dans l'Algorithme 6.

Algorithme 6 Chiffrement de McEliece

Entrée(s): $p_k = (\tilde{\mathcal{G}}, t)$ la clé publique et $M \in \mathbb{F}_2^k$ un message à chiffrer.
Sortie(s): $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré associé à M.

1: Encoder le message $C = M \cdot \tilde{\mathcal{G}}$.
2: Générer aléatoirement un vecteur erreur $E \in \mathbb{F}_2^n$ de poids $w_H(E) = t$.
3: Calculer $\tilde{C} = C \oplus E$.
4: **Retourner** $\tilde{C}$.

On peut constater que le chiffrement est grossièrement un encodage. On peut donc supposer que le déchiffrement est grossièrement un décodage. Voyons cela.

3.1.3 Déchiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du texte chiffré à déchiffrer et de la clé privée pour la phase de déchiffrement. Le déchiffrement de McEliece est présenté dans l'Algorithme 7.

Algorithme 7 Déchiffrement de McEliece

Entrée(s): $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \mathscr{C})$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair associé à $\tilde{C}$.
1: Calculer $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$.
2: Décoder $\tilde{C}_p$ pour retrouver $M \cdot \mathcal{S} \cdot \mathcal{G}$.
3: Récupérer $\tilde{M} = M \cdot \mathcal{S}$ à partir de $M \cdot \mathcal{S} \cdot \mathcal{G}$.
4: Calculer $M = \tilde{M} \cdot \mathcal{S}^{-1}$.
5: **Retourner** M.

À la première ligne, on obtient : $\tilde{C}_p = M \cdot \mathcal{S} \cdot \mathcal{G} \oplus E \cdot \mathcal{P}^{-1}$. Remarquons ensuite que, pour passer de $M \cdot \mathcal{S} \cdot \mathcal{G}$ à $M \cdot \mathcal{S}$ à la troisième ligne, il suffit d'appliquer l'algorithme de Gauss à $\mathcal{G}$ (comme $\mathcal{G}$ est de rang plein, Remarque 2.2), pour obtenir une triangularisation de la matrice. Cela revient à trouver l'inverse à droite de $\mathcal{G}$, noté $\mathcal{G}_r^{-1}$, c'est-à-dire tel que $\mathcal{G} \cdot \mathcal{G}_r^{-1} = \mathcal{I}_k$. On obtient ainsi $M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{G}_r^{-1} = M \cdot \mathcal{S}$. En revanche, si la matrice $\mathcal{G}$ a été prise sous forme systématique (c'est-à-dire $\mathcal{G} = \left[\mathcal{I}_k | \mathcal{A}\right]$), il suffit de prendre les k premiers bits de $M \cdot \mathcal{S} \cdot \mathcal{G}$ pour obtenir $M \cdot \mathcal{S}$. Enfin, le calcul de la quatrième ligne est possible car $\mathcal{S}$ a été choisie comme étant une matrice inversible, donc $\mathcal{S}^{-1}$ existe.

3.1.4 Sécurité

Le principe de décodage par ensemble d'information (*Information Set Decoding* (ISD) en Anglais) pour cryptanalyser ce cryptosystème a été proposé par Robert McEliece lui-même dans son article [McE78], puis bien plus tard dans [Par89] et avec de multiples améliorations depuis. Plus récemment, l'attaque présentée dans [BLP08] réduit la sécurité du cryptosystème de 2^{80} à 2^{62} pour les paramètres donnés dans [McE78]. Pour un niveau de sécurité de 2^{80}, les paramètres requis sont donc $m = 11$ et $t = 27$, donnant comme longueur de code $n = 2048$ et dimension de code $k = 1751$. L'attaque la plus récente a été proposée à Eurocrypt 2012 [BJMM12].

Une version dite "moderne" de McEliece, nommée ainsi dans [HG13] pour la différencier de la version "classique", correspond à une matrice génératrice sous forme systématique afin de réduire l'espace mémoire nécessaire pour stocker la matrice génératrice publique $\tilde{\mathcal{G}}$. La matrice $\mathcal{S}$ est alors choisie de telle sorte que $\tilde{\mathcal{G}} = [\mathcal{I}_k | \mathcal{A}]$ en utilisant l'algorithme de Gauss-Jordan.

3.1.5 Implantations

Nous présentons les différentes implantations existantes du cryptosystème de McEliece dans les Tableaux 3.1 et 3.2. La première publication d'implantation logicielle [PBGV92] n'est pas souvent citée. Elle est probablement passée inaperçue car le grand intérêt pour ce cryptosystème, notamment des points de vue attaque par canaux auxiliaires et cryptographie post-quantique, a commencé il y a une dizaine d'années. La deuxième implantation est un code source disponible sur internet mais laissé anonymement. Autour des années 2000, aucune implantation de ce cryptosystème n'a été proposée. Il faut attendre les tentatives d'implantations embarquées et sécurisées pour voir émerger tout un panel de propositions.
Cette course des implantations a repris avec HyMES en 2008 [BS08], qui n'apparaît volontairement pas dans les tableaux suivants pour la simple et bonne raison que la Section 3.3 est consacrée à cette implantation logicielle. De même avec McBits en 2013 [BCS13], pour laquelle la Section 3.4 lui est consacrée. Une autre implantation logicielle, avec Sage, a été proposée en 2011 [Ris11].
Certaines publications mixent des implantations logicielles et embarquées [Str13a, Cho16], tandis que d'autres mixent des implantations embarquées et matérielles [EGHP09, HvMG13, vMOG15]. Notons d'ailleurs que [EGHP09] est la première implantation embarquée du cryptosystème de McEliece. La première implantation matérielle, quant à elle, a été proposée la même année dans [SWM+09].
Les implantations embarquées, sur microcontrôleurs [Pau10, Hey11, Str12b, vMG14b] ou sur carte à puce [Str10a], ont été réalisées pour démontrer qu'il était possible de le faire et donc d'utiliser ce cryptosystème en pratique, ou dans le but d'effectuer des attaques par canaux auxiliaires contre celles-ci par la suite (c.f. Chapitre 4). Il en va de même par exemple pour l'implantation matérielle [vMG14a], qui fût attaquée par [CEvMS15], puis améliorée dans [CEvMS16].

Tableau 3.1 – Implantations de McEliece (jusqu'à 2009 inclus)

Titre	Auteur(s)	Année	Référence
A Software Implementation of the McEliece Public-Key Cryptosystem	Preneel Bosselaers Govaerts Vandewalle	1992	[PBGV92]
goppa_ code.c[3]	"Prometheus"	1995	[Pro95]
MicroEliece : McEliece for Embedded Devices	Eisenbarth Güneysu Heyse Paar	2009	[EGHP09]
A Novel Processor Architecture for McEliece Cryptosystem and FPGA Platforms	Shoufan Wink Molter Huss Strentzke	2009	[SWM+09]

3. http://www.eccpage.com/

Tableau 3.2 – Implantations de McEliece (depuis 2010)

Titre	Auteur(s)	Année	Référence
A Smart Card Implementation of the McEliece PKC	Strenzke	2010	[Str10a]
Post-Quantum Cryptography on Embbeded Devices : Efficient Implementation of the McEliece Public-Key Scheme based on Quasi-Dyadic Goppa Codes	Paustjan	2010	[Pau10]
Implementation of McEliece Based on Quasi-dyadic Goppa Codes for Embedded Devices	Heyse	2011	[Hey11]
How SAGE helps to implement Goppa Codes and McEliece PKCs	Risse	2011	[Ris11]
Solutions for the Storage Problem of McEliece Public and Private Keys on Memory-Constrained Platforms	Strenzke	2012	[Str12b]
Efficient implementation of a CCA2-secure variant of McEliece using generalized Srivastava codes	Cayrel Hoffmann Persichetti	2012	[CHP12]
Smaller Keys for Code-based Cryptography : QC-MDPC McEliece Implementations on Embedded Devices	Heyse von Maurich Güneysu	2013	[HvMG13]
A Side-Channel Secure and Flexible Platform-Independent Implementation of the McEliece PKC	Strenzke	2013	[Str13a]
Lightweight Code-based Cryptography : QC-MDPC McEliece Encryption on Reconfigurable Devices	von Maurich Güneysu	2014	[vMG14a]
Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices	von Maurich Güneysu	2014	[vMG14b]
Implementing QC-MDPC McEliece Encryption	von Maurich Oder Güneysu	2015	[vMOG15]
Masking Large Keys in Hardware : A Masked Implementation of McEliece	Chen Eisenbarth von Maurich Steinwandt	2016	[CEvMS16]

3.2 Cryptosystème de Niederreiter

Le deuxième protocole cryptographique basé sur les codes correcteurs d'erreurs est le système de chiffrement à clé publique proposé par Harald Niederreiter en 1986 [Nie86]. Pour simplifier les notations, les coordonnées des vecteurs seront numérotées à partir de zéro dans cette section. Les algorithmes de génération de clés, de chiffrement et de déchiffrement sont constitués de la façon suivante.

3.2.1 Génération de clés

Comme ce cryptosystème est basé sur les codes, nous avons encore besoin de deux entiers n et t lors de la génération des clés, qui définiront respectivement la longueur et la capacité de correction d'un code linéaire, où t sera beaucoup plus petit que n. La génération des clés du cryptosystème de Niederreiter est présentée dans l'Algorithme 8.

Algorithme 8 Génération de clés de Niederreiter

Entrée(s): n et t deux entiers.
Sortie(s): $p_k = (\tilde{\mathcal{H}}, t)$ la clé publique et $s_k = (\mathcal{Q}, \mathcal{H}, \mathcal{P}, \mathscr{C})$ la clé privée associée.

1: Choisir un code linéaire $\mathscr{C}$ de longueur n et t correcteur.
On notera k la dimension du code $\mathscr{C}$ et r la co-dimension.
2: $\ell = \lfloor \log_2 \binom{n}{t} \rfloor$.
3: Prendre une matrice de contrôle $\mathcal{H} \in \mathscr{M}_{r,n}(\mathbb{F}_2)$ de $\mathscr{C}$.
4: Choisir aléatoirement une matrice inversible $\mathcal{Q} \in \mathscr{M}_{r,r}(\mathbb{F}_2)$.
5: Choisir aléatoirement une matrice de permutation $\mathcal{P} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$.
6: Calculer la matrice génératrice $\tilde{\mathcal{H}} = \mathcal{Q} \cdot \mathcal{H} \cdot \mathcal{P}$.
7: $s_k \leftarrow (\mathcal{Q}, \mathcal{H}, \mathcal{P}, \mathscr{C})$
8: $p_k \leftarrow (\tilde{\mathcal{H}}, t, \ell)$
9: **Retourner** (p_k, s_k).

Notons que n et k sont des données publiques puisque ces entiers sont donnés par la taille de $\tilde{\mathcal{H}}$. Une variante possible est de rajouter une étape dans l'Algorithme 8 pour calculer les inverses des matrices $\mathcal{Q}$ et $\mathcal{P}$, c'est-à-dire les matrices $\mathcal{Q}^{-1}$ et $\mathcal{P}^{-1}$, puis de les mettre dans la clé privée s_k à la place des matrices $\mathcal{Q}$ et $\mathcal{P}$, donnant ainsi $s_k = (\mathcal{Q}^{-1}, \mathcal{H}, \mathcal{P}^{-1}, \mathscr{C})$, car ce sont les inverses qui seront utilisés par la suite dans l'algorithme de déchiffrement (donné dans l'Algorithme 11) et non pas les matrices $\mathcal{Q}$ et $\mathcal{P}$ directement. Une fois la génération de clés effectuée, voici comment utiliser ce cryptosystème.

3.2.2 Chiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du message à chiffrer et de la clé publique pour la phase de chiffrement. Le chiffrement de Niederreiter est présenté dans l'Algorithme 9.

Algorithme 9 Chiffrement de Niederreiter

Entrée(s): $p_k = (\tilde{\mathcal{H}}, t, \ell)$ la clé publique et $M \in \mathbb{F}_2^\ell$ un message à chiffrer.
Sortie(s): $\tilde{S} \in \mathbb{F}_2^r$ le texte chiffré associé à M.

1: Représenter M comme un vecteur erreur E de longueur n et de poids t,
2: //grâce à un algorithme d'encodage en poids constant.
3: Calculer $\tilde{S} = \tilde{\mathcal{H}} \cdot {}^tE$,
4: //fonction syndrome.
5: **Retourner** $\tilde{S}$.

On peut constater que le chiffrement est grossièrement une mise sous forme de syndrome du message, au lieu d'un encodage comme c'est le cas pour le cryptosystème de McEliece. L'encodage en poids constant mentionné ci-dessus peut se faire d'une manière simple comme proposée dans [Cov73], donnée ici dans l'Algorithme 10. Notons $\phi_{n,t} : [\![0, \binom{n}{t}[\![\to \mathscr{W}_{n,t}$ cette fonction d'encodage, où $\mathscr{W}_{n,t}$ est l'ensemble des mots de longueurs n et de poids t. Cette fonction correspond à l'algorithme de décodage énumératif [OS09]. D'autres méthodes existent, mais celle-ci est la plus efficace en terme de rendement (c'est-à-dire de taux d'information).

Algorithme 10 Décodage énumératif

Entrée(s): $x \in [\![0, \binom{n}{t}[\![$
Sortie(s): E un vecteur de n bits et de poids t, où les t entiers $0 \leqslant i_1 < i_2 < \ldots < i_t \leqslant n-1$ correspondent aux indices des positions des bits non nuls dans E.

1: $j \leftarrow t$
2: **Tant que** $j > 0$ **faire**
3: $\quad i_j \leftarrow$ `inverse-binomial`(x, j)
4: $\quad$ //où `inverse-binomial`(x, j) retourne l'entier i tel que $\binom{i}{j} \leqslant x < \binom{i+1}{j}$
5: $\quad x \leftarrow x - \binom{i_j}{j}$
6: $\quad j \leftarrow j - 1$
7: **Fin tant que**
8: **Retourner** E.

L'entier $x \in [\![0, \binom{n}{t}[\![$ correspond à celui ayant pour écriture binaire $M \in \mathbb{F}_2^\ell$.

3.2.3 Déchiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du texte chiffré à déchiffrer et de la clé privée pour la phase de déchiffrement. Le déchiffrement de Niederreiter est présenté dans l'Algorithme 11.

Algorithme 11 Déchiffrement de Niederreiter

Entrée(s): $s_k = (\mathcal{Q}, \mathcal{H}, \mathcal{P}, \mathscr{C})$ la clé privée, $\tilde{S} \in \mathbb{F}_2^r$ le texte chiffré.
Sortie(s): $M \in \mathbb{F}_2^\ell$ le texte clair associé à $\tilde{C}$.
1: Calculer $S = \mathcal{Q}^{-1} \cdot \tilde{S}$.
2: Décoder $S = \mathcal{H} \cdot \mathcal{P} \cdot {}^tE$ pour retrouver $\tilde{E} = \mathcal{P} \cdot {}^tE$,
3: //décodage par syndrome.
4: Calculer ${}^tE = \mathcal{P}^{-1} \cdot \tilde{E}$.
5: Représenter E comme un message M,
6: //fonction réciproque de l'encodage en poids constant.
7: **Retourner** M.

L'algorithme de décodage utilisé à l'étape 2 dépend de la famille de code dans laquelle le code $\mathscr{C}$ a été choisi et de l'algorithme de décodage le plus efficace connu pour celle-ci. Quant à la fonction réciproque de l'encodage en poids constant, $\phi_{n,t}^{-1} : \mathscr{W}_{n,t} \to [\![0, \binom{n}{t}[\![$, elle est donnée dans par la formule suivante (c.f. [OS09]) :

$$\begin{aligned} \phi_{n,t}^{-1} : \quad \mathscr{W}_{n,t} &\to [\![0, \binom{n}{t}[\![\\ (i_1, i_2, \ldots, i_t) &\mapsto \binom{i_1}{1} + \binom{i_2}{2} + \ldots + \binom{i_t}{t}. \end{aligned}$$

3.2.4 Sécurité

Il a été démontré en 1994 que les cryptosystèmes de McEliece et de Niederreiter sont équivalents si on utilise la même famille de codes [LDW94]. Le cryptosystème de Niederreiter est souvent présenté comme le dual du cryptosystème de McEliece, car les algorithmes de chiffrement et de déchiffrement sont grossièrement opposés par rapport à leurs utilisations des matrices génératrice et de contrôle dans ces deux cryptosystèmes.
Le cryptosystème de Niederreiter est de type sac à dos. En effet, pour réaliser un cryptosystème à clé publique, il faut s'appuyer sur un problème inversible mais fortement asymétrique d'un point de vue calculatoire : une fonction à trappe. Dans le cas présent, il s'agit de la fonction syndrome et du décodage par syndrome.
Le cryptosystème de Niederreiter a pour avantage qu'il ne nécessite pas de générateur d'aléa, contrairement à la génération des clés dans le cryptosystème de McEliece. L'inconvénient en revanche est qu'un message doit être un mot de longueur n et de poids t, afin que le décodage soit toujours possible. Il faut donc posséder un algorithme d'encodage en poids constant efficace.
Une version dite "moderne" de Niederreiter, nommée ainsi dans [HG13], correspond à une matrice de contrôle sous forme systématique afin de réduire l'espace mémoire nécessaire pour stocker la matrice de contrôle publique $\tilde{\mathcal{H}}$. La matrice $\mathcal{P}$ est alors choisie de telle sorte que $\tilde{\mathcal{H}} = [\mathcal{I}_r | \mathcal{A}]$ en utilisant l'algorithme de Gauss-Jordan.

3.2.5 Implantations

Nous présentons les différentes implantations existantes du cryptosystème de Niederreiter dans le Tableau 3.3. Contrairement au cryptosystème de McEliece, les implantations du cryptosystème de Niederreiter sont essentiellement embarquées [Hey10, vMHG16] ou matérielles [HG12, HG13, HC15]. La seule implantation logicielle est proposée dans [BBMR14]. Même si les deux cryptosystèmes ont une sécu-

rité prouvée équivalente, il serait intéressant dans l'avenir de s'intéresser davantage à celui de Niederreiter.

TABLEAU 3.3 – Implantations de Niederreiter

Titre	Auteur(s)	Année	Référence
Low-Reiter : Niederreiter Encryption Scheme for Embedded Microcontrollers	Heyse	2010	[Hey10]
Towards One Cycle per Bit Asymmetric Encryption : Code-Based Cryptography on Reconfigurable Hardware	Heyse Güneysu	2012	[HG12]
Code-based cryptography on reconfigurable hardware : tweaking Niederreiter encryption for performance	Heyse Güneysu	2013	[HG13]
Scaling efficient code-based cryptosystems for embedded platforms	Biasi Barreto Misoczki Ruggiero	2014	[BBMR14]
An Application Specific Instruction Set Processor (ASIP) for the Niederreiter Cryptosystem	Hu Cheung	2015	[HC15][4]
IND-CCA Secure Hybrid Encryption from QC-MDPC Niederreiter	von Maurich Heberle Güneysu	2016	[vMHG16]

3.3 Schéma de chiffrement hybride de McEliece

Une idée de modification du cryptosystème de McEliece est apparue une dizaine d'années après sa publication dans [Par89] : utiliser l'erreur pour faire passer de l'information en plus. Celle-ci semble être passée inaperçue. Elle fût de nouveau mentionnée par Nicolas Sendrier en 2001 [Sen02] puis fût utilisée dans l'implantation proposée par Bhaskar Biswas et Nicolas Sendrier en 2008 [BS08], appelé cryptosystème de McEliece hybride (*Hybrid McEliece Encryption Scheme* (HyMES) en Anglais). Cette idée permet d'augmenter le taux d'information. Les algorithmes de génération de clés, de chiffrement et de déchiffrement sont constitués de la façon suivante. Comme pour le schéma original, les auteurs ont proposé d'utiliser les codes de Goppa.

3.3.1 Génération de clés

Comme ce cryptosystème est basé sur les codes, nous avons besoin de deux entiers n et t lors de la génération des clés, qui définiront respectivement la longueur et la capacité de correction d'un code linéaire, où t sera beaucoup plus petit que n. La génération des clés de HyMES est présentée dans l'Algorithme 12.

4. `https://github.com/davidhoo1988/NiederreiterCryptoprocessor`

Algorithme 12 Génération de clés de HyMES

Entrée(s): n et t deux entiers.
Sortie(s): $p_k = (\mathcal{A}, t)$ la clé publique et $s_k = (\mathscr{L}, G)$ la clé privée associée.

1: $\ell \leftarrow \lfloor \log_2 \binom{n}{t} \rfloor$.
2: Choisir un code de Goppa $\Gamma(\mathscr{L}, G)$ de longueur n et t correcteur, dont le support contient $n = 2^m$ éléments.
On notera k la dimension du code $\mathscr{C}$.
3: Prendre une matrice $\mathcal{A}$ telle que la matrice $\mathcal{G} = [\mathcal{I}_k | \mathcal{A}] \in \mathscr{M}_{k,n}(\mathbb{F}_2)$ soit génératrice de $\Gamma(\mathscr{L}, G)$.
4: $s_k \leftarrow (\mathscr{L}, G)$
5: $p_k \leftarrow (\mathcal{A}, t, \ell)$
6: **Retourner** (p_k, s_k).

Maintenant que les clés ont été générées, voici comment utiliser ce cryptosystème.

3.3.2 Chiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du message à chiffrer et de la clé publique pour la phase de chiffrement. Le chiffrement de HyMES est présenté dans l'Algorithme 13.

Algorithme 13 Chiffrement de HyMES

Entrée(s): $p_k = (\mathcal{A}, t, \ell)$ la clé publique, $M \in \mathbb{F}_2^k$ un message à chiffrer et $\tilde{M} \in \mathbb{F}_2^\ell$ une information supplémentaire.
Sortie(s): $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré associé à $(M, \tilde{M})$.

1: Encoder le message M : $C = M \cdot [\mathcal{I}_k | \mathcal{A}]$.
2: Encoder en mot de poids t et de longueur n le message $\tilde{M}$: $E = \varphi(\tilde{M})$.
3: Calculer $\tilde{C} = C \oplus E$.
4: **Retourner** $\tilde{C}$.

Pour l'algorithme d'encodage en poids constant, il s'agit du même que pour le cryptosystème de Niederreiter, voir l'Algorithme 10.

3.3.3 Déchiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du texte chiffré à déchiffrer et de la clé privée pour la phase de déchiffrement. Le déchiffrement de HyMES est présenté dans l'Algorithme 14.

Algorithme 14 Déchiffrement de HyMES

Entrée(s): $s_k = (\mathscr{L}, G)$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair et $\tilde{M} \in \mathbb{F}_2^\ell$ l'information supplémentaire, tous deux associés à $\tilde{C}$.

1: Décoder $\tilde{C}$ pour retrouver C et E.
2: Récupérer M : les k premiers bits de $\tilde{C} \oplus E$.
3: Calculer $\tilde{M} = \varphi^{-1}(E)$.
4: **Retourner** $(M, \tilde{M})$.

3.3.4 Sécurité

Deux différences sont notables dans la version hybride par rapport à la version originale. La première a été précisée au début de cette section. Il s'agit de l'utilisation de l'erreur pour faire passer de l'information en plus. La deuxième est remarquable dans l'algorithme de chiffrement : la matrice génératrice publique est sous forme systématique. De plus, les auteurs ont démontré dans leur article [BS08] que ces changements n'entraînent en aucun cas une diminution du niveau de sécurité de ce cryptosystème comparé au cryptosystème de McEliece sous les hypothèses algorithmiques de la difficulté du décodage (décodage par syndrome [BMvT78] et décodage à poids borné d'un code de Goppa [Fin04]) et de la distinction entre un code de Goppa et un code aléatoire. Cependant la deuxième hypothèse n'est pas toujours vraie (lorsque le rendement est élevé) et une preuve a été proposée depuis [FGUO+13]. Toutefois, la sécurité du cryptosystème d'HyMES repose sur la même fonction à sens unique que le cryptosystème de McEliece.

3.3.5 Implantations

Nous présentons les différentes implantations existantes du cryptosystème d'HyMES dans le Tableau 3.3, qui est la version dite "hybride" du cryptosystème de McEliece. Nous avons mis dans ce tableau trois implantations car celles proposées par Falko Strenzke dans [Str13a, Str14], appelées FLEA et McEliece in Botan, sont basées sur l'implantation d'HyMES [BS08].

Tableau 3.4 – Implantations logicielles d'HyMES

Titre	Auteur(s)	Année	Référence
McEliece cryptosystem implementation : theory and practice	Biswas Sendrier	2008	[BS08][5]
A Side-Channel Secure and Flexible Platform-Independent Implementation of the McEliece PKC	Strenzke	2013	[Str13a]
Botan's Implementation of the McEliece PKC	Strenzke	2014	[Str14][6]

3.4 McBits

Une nouvelle variante du cryptosystème de McEliece, nommée McBits, a été proposée en 2013 par Daniel Bernstein, Tung Chou et Peter Schwabe dans [BCS13]. Cette proposition vise à clore tout type d'attaque temporelle contre McEliece (c.f. Chapitre 4), puisque l'implantation est non seulement proclamée comme étant (extrêmement) rapide, mais en temps constant qui plus est. Pour arriver à ce résultat, les grandes lignes composant le cryptosystème de McEliece ont été gardées, mais les algorithmes utilisés dans ces étapes diffèrent. La principale nouveauté réside dans l'application de la transformée de Fourier rapide (*Fast Fourier Transform* (FFT)

5. HyMES : `https://www.rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes`
6. `http://cryptosource.de/news/mce_in_botan_en.html`

en Anglais) additive pour la recherche des racines du polynôme localisateur d'erreurs, la transposée de la transformée de Fourier rapide additive pour le calcul du syndrome et enfin l'utilisation d'un réseau de tri pour éviter les attaques sur la mémoire cache (au lieu de tables de recherche) concernant la permutation. Les algorithmes de génération de clés, de chiffrement et de déchiffrement sont constitués de la façon suivante.

3.4.1 Génération de clés

Comme ce cryptosystème est basé sur les codes, nous avons besoin essentiellement de deux entiers m et t lors de la génération des clés, mais de manière plus détaillée, de cinq entiers qui sont m, q, n, t et k, dont certains dépendent des autres. Ces entiers définissent respectivement le degré de l'extension du corps, la cardinalité de l'extension du corps sur lequel le code est défini, la longueur du code, la capacité de correction du code et enfin la dimension du code. La génération des clés de McBits est présentée dans l'Algorithme 15.

Algorithme 15 Génération de clés de McBits

Entrée(s): m et t deux entiers au moins égaux à 2 chacun.
Sortie(s): $p_k = (\mathcal{K})$ la clé publique et $s_k = (\mathscr{L}, G)$ la clé privée associée.

1: $q \leftarrow 2^m$
2: Choisir n tel que $n \leqslant q$
3: $k \leftarrow n - mt$
4: Choisir un code de Goppa $\Gamma(\mathscr{L}, G)$ de longueur n et t correcteur, dont le support $\mathscr{L}$ contient n éléments distincts de $\mathbb{F}_q$ et où G est un polynôme irréductible de degré t à coefficients dans $\mathbb{F}_q$.
5: Calculer la matrice secrète $\bar{\mathcal{H}}_G$ de taille $t \times n$ sur $\mathbb{F}_q$.
6: Remplacer toutes les composantes par un vecteur colonne de m bits afin d'obtenir la matrice de contrôle $\mathcal{H}$ du code de Goppa $\Gamma(\mathscr{L}, G)$ de taille $tm \times n$ sur $\mathbb{F}_2$.
7: $s_k \leftarrow (\mathscr{L}, G)$
8: Appliquer l'algorithme de Gauss sur $\mathcal{H}$ pour obtenir $\mathcal{K}$,
tel que $\mathcal{K} = [\mathcal{I}_{tm} | \mathcal{A}] \in \mathscr{M}_{mt,n}(\mathbb{F}_2)$.
9: $p_k \leftarrow (\mathcal{K})$
10: **Retourner** (p_k, s_k).

Notons que les paramètres m, q, n, t et k sont également publics (en plus de $\mathcal{K}$). Une fois la génération de clés effectuée, voici comment utiliser ce cryptosystème.

3.4.2 Chiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du message à chiffrer et de la clé publique pour la phase de chiffrement. Le chiffrement de McBits est présenté dans l'Algorithme 16.

Algorithme 16 Chiffrement de McBits

Entrée(s): $p_k = (\mathcal{K})$ la clé publique, M un message à chiffrer, $hash$ une fonction de hachage, enc un algorithme de chiffrement par flot et $auth$ un algorithme d'authentification.

Sortie(s): $\tilde{C}$ le texte chiffré associé à (M, E).

1: Générer un vecteur $E \in \mathbb{F}_2^n$ de poids t.
2: $W \leftarrow \mathcal{K} \cdot {}^tE$
3: Calculer $hash(E)$ pour obtenir une chaîne de bits de longueur $s+a$, en tant que concaténation de deux clés (key_{enc}, key_{aut}).
4: Chiffrer le message M en utilisant l'algorithme de chiffrement par flot enc et la clé key_{enc} pour obtenir C.
5: Calculer l'empreinte A de C en utilisant l'algorithme d'authentification $auth$ avec la clé key_{aut}.
6: $\tilde{C} \leftarrow (A, W, C)$.
7: **Retourner** $\tilde{C}$.

Les clés key_{enc} et key_{aut} sont chacune de 256 bits si la fonction de hachage utilisée est SHA-512. La taille du message n'est pas fixée à l'avance, puisque le chiffrement se fait non pas par encodage mais en utilisant un chiffrement par flot (cryptographie symétrique). Cela explique également pour la taille de C et a fortiori celle de $\tilde{C}$ ne sont également pas déterminées à l'avance, mais dépendent directement de l'utilisation.
On peut constater que l'algorithme de chiffrement est plus compliqué que pour le cryptosystème de McEliece original, puisque les personnes communiquant de manière sécurisée en utilisant le cryptosystème de McBits doivent également se mettre d'accord sur une fonction de hachage, un algorithme de chiffrement par flot et un algorithme d'authentification. Les exemples donnés dans [BCS13] sont respectivement SHA-512 [7] voire SHA-3 [8] , Salsa20 [Ber08] et Poly1305 [Ber05]. Ce protocole permet de garantir davantage que la confidentialité, mais aussi l'intégrité et l'authenticité du texte chiffré.

3.4.3 Déchiffrement

Comme dans tout système de chiffrement à clé publique, nous avons besoin du texte chiffré à déchiffrer et de la clé privée pour la phase de déchiffrement. Le déchiffrement de McBits est présenté dans l'Algorithme 17.

7. `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`
8. `http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf`

Algorithme 17 Déchiffrement de McBits

Entrée(s): $s_k = (\mathscr{L}, G)$ la clé privée, $\tilde{C} = (A, W, C)$ le texte chiffré, *hash* une fonction de hachage, *enc* un algorithme de chiffrement par flot et *auth* un algorithme d'authentification.

Sortie(s): M le texte clair associé à $\tilde{C}$.

1: Décoder $W \in \mathbb{F}_2^{mt}$ en utilisant s_k pour obtenir $E \in F_2^n$.
2: Calculer $hash(E)$ pour obtenir une chaîne de bits de longueur $s + a$, tel que $hash(E)$ soit la concaténation (key_{enc}, key_{aut}).
3: Vérifier que l'empreinte A de C est valide en utilisant l'algorithme d'authentification *auth* avec la clé key_{aut}.
4: Déchiffrer le texte C en utilisant l'algorithme de chiffrement par flot *enc* avec la clé key_{enc} pour obtenir M.
5: **Retourner** M.

3.4.4 Implantations

Nous présentons dans le Tableau 3.5 les deux seules implantations répertoriées qui se disent en temps constant. La différence entre McBits et QcBits se trouve dans le choix des codes utilisés : McBits [BCS13] est une implantation proposée pour les codes de Goppa binaires tandis que QcBits [Cho16] est une implantation proposée pour les codes QC-MDPC (QC pour quasi-cycliques, *Quasi-Cyclic* en Anglais, codes non abordés dans cette thèse). Ces deux implantations sont basées sur le découpage en bit des opérations arithmétiques pour permettre une exécution en temps constant. Ces propositions sont très récentes, cela explique pourquoi la communauté ne s'est pas encore forcément approprié toutes les idées novatrices de celles-ci et avoir assez de recul pour proposer des variantes ou attaques. Ceci reste donc une piste pour la suite.

Tableau 3.5 – Implantations logicielles de McBits et QcBits

Titre	Auteur(s)	Année	Référence
McBits : Fast Constant-Time Code-Based Cryptography	Bernstein Chou Schwabe	2013	[BCS13][9]
QcBits : constant-time small-key code-based cryptography	Chou	2016	[Cho16][10]

Nous présentons, dans le chapitre suivant, les attaques par canaux auxiliaires, visant certaines implantations présentées dans ce chapitre. Nous nous intéressons dans cette thèse plus particulièrement aux attaques contre le cryptosystème de McEliece, même si certaines sont facilement adaptables au cryptosystème de Niederreiter.

9. `http://binary.cr.yp.to/mcbits.html`
10. `https://www.win.tue.nl/~tchou/qcbits/`

Chapitre 4

Attaques existantes contre le cryptosystème de McEliece

Nous allons rapidement décrire dans ce chapitre l'essentiel sur les attaques contre le cryptosystème de McEliece, essentiellement celles par canaux auxiliaires, qui représentent le cœur de cette thèse.
L'implantation d'un cryptosystème est un compromis entre la sécurité et l'efficacité. C'est pourquoi, la cryptanalyse et l'implantation doivent être considérées simultanément. Nous présentons, dans le Tableau 4.1, les implantations logicielles et matérielles du cryptosystème de McEliece avec les codes de Goppa classiques, principal intérêt de cette thèse. Nous y avons également placé les implantations d'HyMES et de McBits car ce sont des variantes de McEliece.

Tableau 4.1 – Implantations avec des codes de Goppa classiques

Titre	Auteur(s)	Année	Référence
A Software Implementation of the McELIECE Public-Key Cryptosystem	Preneel, Bosselaers Govaerts, Vandewalle	1992	[PBGV92]
McEliece cryptosystem implementation : theory and practice	Biswas Sendrier	2008	[BS08]
A Novel Processor Architecture for McEliece Cryptosystem and FPGA Platforms	Shoufan, Wink Molter, Huss Strentzke	2009	[SWM+09]
MicroEliece : McEliece for Embedded Devices	Eisenbarth, Güuneysu Heyse, Paar	2009	[EGHP09]
How SAGE helps to implement Goppa Codes and McEliece PKCs	Risse	2011	[Ris11]
Fast and Secure Root-Finding for Code-based Cryptosystems	Strenzke	2012	[Str12a]
Efficient implementation of a CCA2-secure variant of McEliece using generalized Srivastava codes	Cayrel Hoffmann Persichetti	2012	[CHP12]
A Side-Channel Secure and Flexible Platform-Independent Implementation of the McEliece PKC	Strenzke	2013	[Str13a]
McBits : Fast Constant-Time Code-Based Cryptography	Bernstein Chou, Schwabe	2013	[BCS13]

Pour la suite, avant de présenter les attaques par canaux auxiliaires contre le cryptosystème de McEliece, dites "pratiques", nous présentons rapidement les principes des attaques dites "théoriques".

4.1 Attaques théoriques

4.1.1 Attaques par décodage vs. attaques structurelles

Il y a deux types d'attaques théoriques pour effectuer une cryptanalyse du cryptosystème de McEliece : les attaques par décodage et les attaques structurelles.
Une attaque par décodage (ou attaque générique) consiste à voir la matrice génératrice publique $\tilde{\mathcal{G}}$ comme une matrice génératrice d'un code aléatoire et d'essayer de retrouver l'erreur utilisée lors du chiffrement pour retrouver le message initial à partir d'un texte chiffré.
Une attaque structurelle (ou attaque algébrique) consiste à chercher un décodeur du code publique, qui est équivalent au code privé. Le but est de retrouver la matrice de permutation (et a fortiori le code privé).
Une attaque structurelle est donc plus puissante qu'une attaque par décodage, car la première cible la clé privée tandis que la seconde cible un message en clair.

4.1.2 Différentes familles de codes testées en cryptographie

Qui dit "cryptographie basée sur les codes", dit "choix d'un code". Nous présentons dans le Tableau 4.2 les différentes familles de codes qui ont été proposées et les attaques théoriques répertoriées.

Tableau 4.2 – Cryptosystèmes avec différents codes

Noms des codes	Propositions	Attaques
Goppa (classiques binaires)	[McE78]	
Reed-Solomon généralisés	[Nie86]	[SS92, Wie10]
sous-codes	[BL05, Wie06]	[Wie10, MCMMP13, CGGU+14]
Reed-Muller binaires	[Sid94]	[MS07]
Algébriques-géométriques	[JM96]	[MS07, FM08, MCMMP14, MCMMPR14, CMCP16]
sous-codes		[CMCP16]
MDPC	[MTSB13]	

On peut constater que la cryptographie utilisant les codes de Goppa reste encore non-cassée (pour des choix de paramètres judicieux), ce qui explique notre intérêt pour cette famille de codes. Nous ne nous intéressons pas dans cette thèse aux codes MDPC (*Moderate Density Parity-Check* en Anglais, pour des codes ayant des matrices de contrôles dont le poids de chaque ligne est constant), mais cela reste une piste pour la suite de ces travaux.

4.2 Attaques en pratique (par canaux auxiliaires)

La grande majorité des attaques par canaux auxiliaires contre le système de chiffrement à clé publique de McEliece portent sur l'algorithme de Patterson utilisé pour le décodage des codes de Goppa classiques binaires irréductibles lors de la phase de déchiffrement. Comme on peut le constater dans le Tableau 4.3, l'étude des attaques par canaux auxiliaires contre ce cryptosystème n'a débuté qu'en 2008, soit 30 ans après l'apparition de celui-ci.

TABLEAU 4.3 – État de l'art des SCAs

Titre	Auteur(s)	Année	Réf.	Type d'attaques
Side channels in the McEliece PKC	Strenzke Tews Molter Overbeck Shoufan	2008	[STM+08]	TA [11]
A Timing Attack against Patterson Algorithm in the McEliece PKC	Shoufan Strenzke Molter Stöttinger	2009	[SSMS10]	TA poids de l'erreur dans EEA
A Timing Attack against the Secret Permutation in the McEliece PKC	Strenzke	2010	[Str10b]	TA matrice de perm. dans EEA
Practical Power Analysis Attacks on Software Implementations of McEliece	Heyse Moradi Paar	2010	[HMP10]	PA [12] clé privée
McEliece/Niederreiter PKC : sensitivity to fault injection	Cayrel Dusart	2010	[CD10]	FI [13]
Side-Channel Attacks on the McEliece and Niederreiter Public-Key Cryptosystems	Avanzi Hoerder Page Tunstall	2011	[AHPT11]	TA amélioration de [STM+08]
A simple power analysis attack on a McEliece cryptoprocessor	Molter Stöttinger Shoufan Strenzke	2011	[MSSS11]	SPA [14] poids de l'erreur dans EEA
Message-aimed side channel and fault attacks against public key cryptosystems with homomorphic properties	Strenzke	2011	[Str11]	FI, TA message
Timing Attacks against the Syndrome Inversion in Code-based Cryptosystems	Strenzke	2013	[Str13b]	TA polynôme de Goppa
Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices	von Maurich Güneysu	2014	[vMG14b]	(TA,) (S)PA message (ds enc.) clé privée (ds dec.)
Differential Power Analysis of a McEliece Cryptosystem	Chen Eisenbarth von Maurich Steinwandt	2015	[CEvMS15]	DPA [15] (1ère) clé privée

11. TA : Timing Attack
12. PA : Power Attack
13. FI : Fault Injection
14. SPA : Simple Power Analysis
15. DPA : Differential Power Analysis

Peu d'attaques visent la génération de clés et aucune ne concerne le chiffrement puisque toutes les données manipulées durant cette phase sont connues. Nous évoquerons, dans la suite de cette section, les attaques dans l'ordre du déroulement de l'algorithme de Patterson pour le cas du déchiffrement, après avoir développé celles sur la génération de clés. Le lecteur pourra notamment retrouver un bilan de ce type d'attaques contre le système de chiffrement à clé publique basé sur les codes de McEliece dans [CS10], réalisé par Pierre-Louis Cayrel et Falko Strenzke en 2010.

Nous détaillons, dans la suite de ce chapitre, les attaques qui nous intéresserons pour cette thèse. En particulier, nous ne développerons pas les attaques contre le cryptosystème de McEliece utilisé avec les codes MDPC [vMG14b, CEvMS15].

4.2.1 Génération des clés

La seule proposition d'attaque sur la génération de clés a été faite dans [STM+08].

La génération de clés dans le cryptosystème de McEliece (c.f. Algorithme 5) commence par le choix d'un code linéaire (code de Goppa classique binaire irréductible dans notre cas) et se poursuit avec le calcul d'une matrice $\mathcal{G}$, génératrice du code choisi (ici le code de Goppa noté $\Gamma(\mathscr{L}, G)$). Comme il n'existe pas a priori de formule pour construire cette matrice $\mathcal{G}$ directement à l'aide des paramètres du code, on calcule une matrice de contrôle $\mathcal{H}$ (matrice pour laquelle une formule est connue, c.f. Équations (2.7) et (2.8)) grâce au support $\mathscr{L}$ et au polynôme G. Une fois cette matrice $\mathcal{H}$ calculée, le but est de la mettre sous forme systématique, pour ainsi retrouver plus facilement une matrice génératrice $\mathcal{G}$ du code en question.

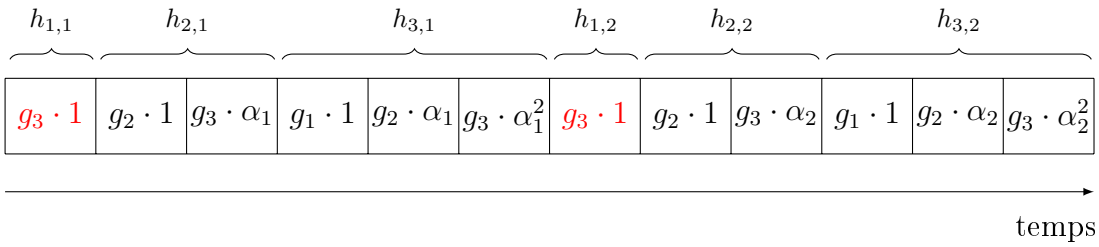
Génération de la matrice de contrôle

La première proposition d'attaque faite dans [STM+08] est la suivante. En supposant que $\mathcal{H}$ soit construite comme le produit $\mathcal{X}\cdot\mathcal{Y}\cdot\mathcal{Z}$, c'est-à-dire $\hat{\mathcal{H}}$ pour respecter les précédentes notations, avec :

$$\begin{aligned}
\mathcal{X} &= \begin{pmatrix} g_t & 0 & 0 & \dots & 0 \\ g_{t-1} & g_t & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \dots & g_t \end{pmatrix}, \\
\mathcal{Y} &= \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \dots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \end{pmatrix}, \\
\text{et } \mathcal{Z} &= \begin{pmatrix} \frac{1}{G(\alpha_1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{G(\alpha_2)} & 0 & \dots & 0 \\ \vdots & \dots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \frac{1}{G(\alpha_n)} \end{pmatrix},
\end{aligned}$$

un canal auxiliaire est alors possible sur la multiplication des coefficients du polynôme de Goppa G avec des puissances d'éléments du support $\mathscr{L}$. La Figure 4.1 illustre ces propos.

FIGURE 4.1 – Calcul de $\hat{\mathcal{H}}$ avec G et $\mathscr{L}$

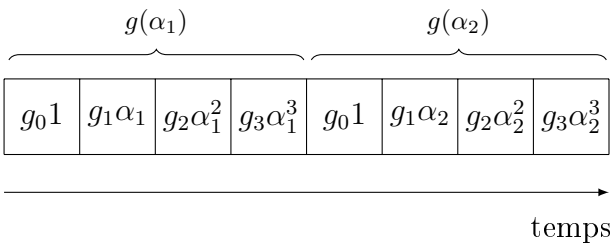


L'attaque consiste à utiliser la trace de consommation des produits faits entre les coefficients de G et les puissances d'éléments du support (lors de la multiplication $\mathcal{X} \cdot \mathcal{Y}$) et d'identifier le début d'une nouvelle colonne de $\hat{\mathcal{H}}$ en utilisant le fait qu'elles commencent toutes par la même valeur (c'est-à-dire par $g_t 1$ dans le calcul $\mathcal{X} \cdot \mathcal{Y}$). Notons que cette information est essentielle pour pratiquer une analyse de consommation. Pour l'exemple donné dans la Figure 4.1, le polynôme de Goppa est de degré 3. De plus, la trace de consommation peut également servir à estimer les coefficients de G sous l'hypothèse que les éléments du support sont connus de l'attaquant (l'ordre lexicographique est utilisé par exemple). La contre-mesure proposée est d'effectuer un masquage pour le calcul des coefficients de $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$. Une manière plus simple d'éviter cette attaque est de prendre une autre matrice de contrôle, $\bar{\mathcal{H}} = \mathcal{Y} \cdot \mathcal{Z}$, ce qui est souvent (pour ne pas dire toujours) le cas, étant donné que, comme la matrice $\mathcal{X}$ est inversible (car $g_t \neq 0$), les matrices $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$ et $\bar{\mathcal{H}} = \mathcal{Y} \cdot \mathcal{Z}$ contrôlent le même code de manière équivalente.

Évaluation du polynôme de Goppa

Une deuxième attaque proposée dans [STM+08] porte sur l'évaluation du polynôme de Goppa G en chaque élément du support $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. En effet, si on a $G(\alpha_i)$ qui est calculé par morceaux, $g_0 \cdot 1, g_1 \cdot \alpha_i, \ldots, g_t \cdot \alpha_i^t$, pour tout i de 1 à n, alors cette suite chronologique, même effectuée en pré-calcul, peut permettre un canal auxiliaire. La figure suivante illustre ce qui se passe :

FIGURE 4.2 – Évaluation du polynôme de Goppa G sur les éléments du support $\mathscr{L}$



Là encore, la contre-mesure qui est proposée est un masquage, restant sur la même idée que pour l'attaque précédente. Une manière plus efficace pour évaluer un polynôme est d'utiliser l'algorithme de Horner [Hor19] plutôt que la simple méthode dite scolaire.

Passons maintenant à la partie la plus critique du cryptosystème, le déchiffrement.

4.2.2 Déchiffrement

Nous rappelons l'usage de l'abréviation PLE pour polynôme localisateur d'erreurs.

A/ Permutation du texte chiffré

Les propositions d'attaques sur la permutation du texte chiffré ont été faites dans [STM+08, HMP10].

Première proposition : [STM+08] L'attaque contre la permutation du texte chiffré (Algorithme 18) proposée dans [STM+08] vise la mémoire cache. Supposons que la matrice de permutation ne soit pas stockée directement comme une matrice dans la mémoire mais plutôt sous forme de table de recherche en fonction des lignes et des colonnes de la matrice. Cette table de recherche est alors utilisée au début du déchiffrement pour calculer $\tilde{C} \cdot \mathcal{P}^{-1}$. Si une implantation simple a été réalisée, alors un accès dans la mémoire sur les adresses dépendantes de la permutation privée sera créé. Un attaquant pourrait ainsi obtenir de l'information sur $\mathcal{P}$. La version ciblée de cette permutation est donnée dans l'Algorithme 18, où les vecteurs seront indexés de 0 à $n-1$ pour concorder avec la proposition de contre-mesure des auteurs donnée ci-après.

Algorithme 18 Permutation du texte chiffré (version intuitive)

Entrée(s): $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ l'inverse de la matrice de permutation privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté.

1: **Pour** $i = 0$ **à** $n-1$ **faire**
2: $\quad j = t_i^{\mathcal{P}^{-1}}$
3: $\quad \tilde{C}_{p_i} = \tilde{C}_j$
4: **Fin pour**
5: **Retourner** $\tilde{C}_p$

Attaque : Les hypothèses de l'attaque sont les suivantes. Pour commencer, supposons que l'attaquant ait accès au processus de déchiffrement exécuté par un système et où le processeur (*Central Processing Unit* (CPU) en Anglais) de l'ordinateur victime supporte simultanément plusieurs tâches. Alors l'attaquant peut exécuter un processus espion en même temps que le déchiffrement. Supposons ensuite que l'attaquant sache où est stocké le texte chiffré $\tilde{C}$ dans la mémoire principale. Il peut par exemple se servir de son processus espion pour écraser le contenu de $\tilde{C}$ de la mémoire du CPU et le remplacer par celui qu'il aura choisi. Il pourra alors effectuer des accès à cette donnée pour obtenir le temps d'accès dans un premier temps, puis déterminer lors du déchiffrement exécuté en parallèle de son programme d'attaque à quelle partie du texte chiffré $\tilde{C}$ le processus attaqué a accédé dans un second temps.
Une dernière hypothèse est que la règle reliant les adresses de la mémoire principale aux blocs de la mémoire cache soit un système dépendant et connu de l'attaquant.

Comme la taille d'un bloc dans la mémoire cache sera plus grande qu'une coordonnée du texte chiffré $\tilde{C}$, il ne pourra pas obtenir une coordonnée exacte de $\tilde{C}$ par accès. En revanche, si la localisation de $\tilde{C}$ dans la mémoire diffère d'une exécution à l'autre, et que $\tilde{C}$ n'a pas la même composition au début d'un bloc, alors l'attaquant sera capable de réduire à une coordonnée cette partie accédée de $\tilde{C}$ par le processus de déchiffrement.
Si toutefois la première hypothèse n'est pas vérifiée, c'est-à-dire que le système où se déroule le processus de déchiffrement ne supporte pas les multi-tâches, alors l'attaquant ne sera pas capable de regarder dans le sous-programme de déchiffrement ce qu'il s'y passe à toutes les itérations. En revanche, il est possible que le système d'exploitation effectue des changements de contexte, c'est-à-dire qu'il passe du programme de déchiffrement au programme espion de l'attaquant. L'attaque serait dans ce cas plus difficile mais pas impossible, sous l'hypothèse que l'attaquant puisse répéter ses mesures assez souvent.

Contre-mesure : Au lieu d'utiliser l'Algorithme 18, la contre-mesure proposée dans [STM+08] est d'utiliser l'Algorithme 19.

Algorithme 19 Permutation du texte chiffré (de [STM+08])

Entrée(s): $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ l'inverse de la matrice de permutation privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté.

1: **Pour** $i = 0$ **à** $n-1$ **faire**
2: $\quad j = t_i^{\mathcal{P}^{-1}}$
3: $\quad \tilde{C}_{p_i} = 0$
4: $\quad$ **Pour** $h = 0$ **à** $n-1$ **faire**
5: $\quad\quad k = \tilde{C}_{p_i}$
6: $\quad\quad \mu = \tilde{C}_h$
7: $\quad\quad s = j \oplus h$
8: $\quad\quad s \mid= s \gg 1$
9: $\quad\quad s \mid= s \gg 2$
10: $\quad\quad s \mid= s \gg 4$
11: $\quad\quad s \mid= s \gg 8$
12: $\quad\quad s \mid= s \gg 16$
13: $\quad\quad s \,\&= 1$
14: $\quad\quad s = \sim (s-1)$
15: $\quad\quad \tilde{C}_{p_i} = (s \,\&\, k) \mid ((\sim s) \,\&\, \mu)$
16: $\quad$ **Fin pour**
17: **Fin pour**
18: **Retourner** $\tilde{C}_p$

On note que les opérateurs $\sim$, &, $\gg$, & =, $\mid$ et $-$ sont utilisés dans le langage de programmation C. L'idée de l'Algorithme 19 est la suivante : Comme dans l'Algorithme 18, que l'on appellera algorithme initial, $t_i^{\mathcal{P}^{-1}}$ est lu dans la ligne 2. L'algorithme initial lit $\tilde{C}_j$ et l'écrit dans $\tilde{C}_{p_i}$. Cette écriture dans $\tilde{C}_{p_i}$ n'est pas critique en elle-même, parce que i est publique, mais comme j dépend de $\mathcal{P}$, une lecture de $\tilde{C}_j$ pourrait révéler de l'information au sujet de j et a fortiori sur $\mathcal{P}$.

Dans l'Algorithme 19, voici ce qu'il se passe : Dans la ligne 3, $\tilde{C}_{p_i}$ est initialisé à 0. Dans la ligne 4, une nouvelle boucle est commencée, où h va de 0 à $n-1$.

Dans toutes les itérations, la valeur de $\tilde{C}_{p_i}$ est stockée dans k et celle de $\tilde{C}_h$ dans μ. Maintenant, il faut distinguer deux cas :

1. $j = h$: Dans ce cas, le but est d'écrire la valeur de μ ($= \tilde{C}_h = \tilde{C}_j$) dans $\tilde{C}_{p_i}$, comme dans l'algorithme initial.
2. $j \neq h$: Dans ce cas, le but est de ne pas modifier $\tilde{C}_{p_i}$. Mais pour créer le même accès mémoire que dans le cas 1, il faut affecter la valeur de k ($= \tilde{C}_{p_i}$) à $\tilde{C}_{p_i}$, et ainsi garder $\tilde{C}_{p_i}$ inchangé.

Afin que cela se fasse sans **Si-Alors-Sinon**, voici l'astuce utilisée dans le nouvel algorithme : La différence (l'opération $\oplus$) entre j et h, notée s, est calculée dans la ligne 7. Si cette différence est nulle, alors $j = h$ et nous nous retrouvons dans le cas 1. Si la différence n'est pas nulle, alors nous nous retrouvons dans le cas 2. Les lignes 8 à 14 se font alors de manière sûre, puisque si s n'est pas nul, tous les bits dans s seront transformés en 1 après exécution de la ligne 14. Ensuite, l'expression $(s \,\&\, k) \mid ((\sim s) \,\&\, \mu)$ évalue k puis k est écrit dans $\tilde{C}_{p_i}$ dans la ligne 15. Si s est nul après la ligne 7, s est encore nul après la ligne 14. Ensuite, l'expression $(s \,\&\, k) \mid ((\sim s) \,\&\, \mu)$ évalue μ puis μ est écrit dans $\tilde{C}_{p_i}$ dans la ligne 15.
L'implantation qu'ils ont réalisée de cet algorithme s'exécute à temps constant (but recherché pour éviter une attaque temporelle), n'effectue aucun bond dépendant de l'entrée secrète, et accède aux adresses mémoire qui dépendent uniquement de l'entrée publique. Malheureusement, cette contre-mesure augmente le temps d'exécution de $\mathcal{O}(2^m)$ à $\mathcal{O}((2^m)^2)$.

Deuxième proposition : [HMP10] Les auteurs décrivent quatre profils d'attaques en fonction des choix d'implantation des étapes consécutives de permutation du texte chiffré et du calcul du syndrome. Nous présentons cela en détails dans la partie concernant le calcul du syndrome (ci-après).

Notre contribution : Nous proposons une attaque par analyse différentielle de consommation sur micro-contrôleur visant la contre-mesure donnée dans l'Algorithme 19 (voir Chapitre 5), en nous basant sur deux des profils d'implantation énoncés dans [HMP10] (voir ci-après).

B/ Calcul du syndrome du chiffré permuté

La seule proposition d'attaque sur le calcul du syndrome du chiffré permuté a été faite dans [HMP10].

Profils : Comme le déclarent Stefan Heyse, Amir Moradi et Christof Paar dans [HMP10], il n'y a pas une unique manière d'implanter le cryptosystème de McEliece, notamment sur un micro-contrôleur. C'est la raison pour laquelle ils proposent quatre profils d'implantation, que nous détaillons ici :

Profil 1 : La permutation du texte chiffré et le calcul du syndrome sont deux opérations effectuées séparément. La permutation est faite sous la forme de produit vecteur-matrice (entre le texte chiffré et l'inverse de la matrice de permutation). Le syndrome est également un produit vecteur-matrice (entre le texte chiffré permuté et la matrice de contrôle $\hat{\mathcal{H}}$ donnée dans l'Équation (2.8) et dont les colonnes sont calculées une à une lorsque nécessaire).

Profil 2 : Similaire au Profil 1 excepté que la matrice $\hat{\mathcal{H}}$ est pré-calculée.

Profil 3 : La stratégie adoptée est de permuter le support $\mathscr{L}$ par un produit vecteur-matrice avec la matrice de permutation. Puis le calcul du syndrome du texte chiffré est réalisé directement en utilisant le support permuté à la place du support lui-même pour le calcul de la matrice de contrôle $\hat{\mathcal{H}}$ (colonne par colonne, grâce à l'Équation (2.8)) et en effectuant le produit vecteur-matrice comme dans le Profil 1.

Profil 4 : Similaire au Profil 2, l'idée est de pré-calculer la matrice de contrôle $\hat{\mathcal{H}}$, mais cette fois-ci en utilisant le support permuté comme dans le Profil 3. La permutation du texte chiffré et le calcul du syndrome sont alors réalisés en une seule opération, à savoir un produit vecteur-matrice (entre le texte chiffré et la matrice de contrôle permutée pré-calculée).

Les auteurs proposent une attaque pour chacun de ces profils.

Principe des attaques : Les auteurs se sont basés sur l'implantation proposée dans [EGHP09] pour effectuer leurs attaques. L'idée principale de ces attaques est basée sur une analyse en fonction du poids de Hamming du texte chiffré (qui est une donnée d'entrée). De manière générale, leurs attaques peuvent être vues en deux phases. La première consiste à collecter des observations via un canal auxiliaire pour des textes chiffrés choisis et de générer une liste de candidats possibles pour chaque élément ciblé du cryptosystème. La seconde consiste à examiner cette liste de candidats pour vérifier quelles hypothèses coïncident avec les paramètres publics du cryptosystème.
Les auteurs font une remarque sur la possibilité d'attaquer le cryptosystème de Mceliece par analyse de consommation (étant donné que c'est la première proposition du type) : comme les calculs sont effectués de manière séquentielle, adopter une stratégie de type "diviser pour régner" est possible afin de retrouver la clé privée octet par octet.

Scénario des attaques : Les auteurs supposent que l'attaquant connaît la clé publique, l'implantation attaquée (incluant support, code source) et est capable de mesurer la consommation lors du déchiffrement pour divers textes chiffrés choisis. La préparation de l'attaque commence par la fabrication d'une base de données des consommations de différentes instructions (opérations arithmétiques, chargement, enregistrement), afin de déterminer ensuite quel type de motif sera à chercher dans l'analyse des traces de consommation par la suite. Le but est de se focaliser par exemple sur le motif le plus facilement reconnaissable par rapport aux autres afin de garantir une meilleure efficacité d'attaque. Une moyenne des traces de consommation peut être envisagée pour une même donnée en entrée afin de réduire le bruit et ainsi affiner le motif pour chaque opération. De plus, il faut remarquer que la consommation sur un micro-contrôleur dépend du poids de Hamming des données manipulées (à cause du pré-chargement et du pré-déchargement des bus de communication). En répétant les mesures une dizaine de fois par entrée (texte chiffré) et en devinant le poids de Hamming de l'entrée à plus ou moins un près, l'attaque sera alors un succès complet. Ayant tout ceci en tête, un attaquant peut ainsi choisir son opération cible et donc le motif associé à rechercher.
Annoncée comme étant une attaque par analyse de consommation, les auteurs expliquent pourquoi ils proposent des SPAs et pas de DPA. En effet, il est difficile d'aligner les traces de consommation pour différents textes chiffrés à cause des variations dans les calculs et dans les temps d'exécution du déchiffrement.

Attaque contre les Profils 1 et 2 : Considérons que l'implantation ciblée corresponde au Profil 1 ou au Profil 2. Une SPA est alors possible sur le calcul du syndrome, produit du texte chiffré permuté et de la matrice de contrôle, pour retrouver la permutation privée.
Le calcul du syndrome étant un produit vecteur-matrice, le résultat peut-être obtenu simplement, en sommant les lignes de la transposée de la matrice de contrôle qui correspondent aux positions des bits non nuls dans le texte chiffré permuté. Le syndrome dépend donc du poids de Hamming du texte chiffré permuté. De plus, le texte chiffré permuté a le même poids de Hamming que le texte chiffré. L'attaquant peut donc choisir tous les textes chiffrés de poids 1 dans le but de retrouver la permutation complète.
Cette attaque n'est pas possible lorsque l'implantation correspond au Profil 3 ou au Profil 4. En effet, les calculs dépendent alors des bits du texte chiffré, la permutation étant dissimulée, l'attaquant n'obtient aucune information supplémentaire pour arriver à ses fins.

Contre-mesure : Les auteurs proposent comme contre-mesure d'effectuer le calcul du syndrome, c'est-à-dire la somme des lignes de la matrice de contrôle dont les indices correspondent aux bits non nuls du texte chiffré permuté dans un ordre qui varie d'un déchiffrement à l'autre. En masquant ici ce calcul par une valeur aléatoire, l'attaquant ne sera alors plus capable de déterminer la matrice de permutation privée.

Attaque contre les Profils 3 et 4 : Ne pouvant obtenir de l'information sur la permutation privée, l'attaquant a tout de même la possibilité d'essayer de retrouver le contenu de la matrice de contrôle. Pour cela, l'idée est la même que pour l'attaque précédente. L'attaquant peut deviner par une SPA ce qu'il se passe en fonction du poids de Hamming du vecteur par lequel la matrice de contrôle est multipliée. Ainsi, en choisissant de nouveau des textes chiffrés de poids 1, il peut essayer de deviner le poids de chaque colonne de la matrice de contrôle. Si les coordonnées des colonnes de la matrice sont stockées bout à bout dans la mémoire, l'attaquant peut arriver à avoir le poids de Hamming de cette matrice octet par octet. En revanche, si chaque coordonnée est stockée de manière bien distincte dans la mémoire, l'attaquant peut alors obtenir davantage d'information sur le poids de chaque coordonnée. L'attaquant obtient ainsi un ensemble de candidats en fonction du poids de Hamming pour chaque coordonnée de la matrice de contrôle.

Contre-mesure : Contrairement à la précédente attaque, un simple changement d'ordre dans le produit vecteur-matrice n'est pas suffisant pour cette attaque. Il serait possible de rajouter des opérations inutiles, mais le temps d'exécution ne doit pas trop augmenter malgré une contre-mesure. Les auteurs proposent donc un masquage booléen de la mémoire avant la multiplication et de démasquer ensuite.
Une autre possibilité énoncée, reprise ensuite dans [CEvMS16], est d'utiliser un masque aussi grand que la matrice de contrôle. Mais ce masque doublerait le temps d'exécution et l'espace mémoire utilisé (pour stocker deux matrices au lieu d'une seule).

Notre contribution : Nous proposons des attaques par analyse de consommation sur micro-contrôleur, pour lesquelles nous nous sommes basés sur les Profils 1 et 2

énoncés dans [HMP10]. La première est une analyse simple, dont l'idée de contre-mesure est celle énoncée ci-dessus. La seconde est une analyse différentielle (voir Chapitre 6).

C/ Inversion du polynôme syndrome modulo le polynôme de Goppa

Les propositions d'attaques sur l'inversion du polynôme syndrome modulo le polynôme de Goppa ont été faites dans [HMP10, Str13b].

La première étape du déchiffrement où intervient le polynôme de Goppa (partie de la clé privée) est l'inversion du polynôme syndrome. Ce polynôme est obtenu en considérant les coordonnées du vecteur syndrome vues comme les coefficients dans $\mathbb{F}_{2^m}$ d'un polynôme. Rappelons que le vecteur syndrome est le résultat de la multiplication entre le texte chiffré permuté et la matrice de contrôle. Le polynôme syndrome a la propriété qu'il est de degré strictement inférieur au degré du polynôme de Goppa. En se plaçant dans le cas où le polynôme de Goppa a été choisi comme étant irréductible, ces deux polynômes sont donc premiers entre eux. On peut donc inverser le polynôme syndrome modulo le polynôme de Goppa.

Première proposition : [HMP10] L'attaque par SPA proposée dans [HMP10] pour retrouver le polynôme de Goppa est basée sur la même idée que pour retrouver la matrice de permutation ou la matrice de contrôle (attaques vues précédemment). Étant donné que le polynôme de Goppa est utilisé à partir de cette étape, on peut supposer que c'est à ce moment là qu'il est chargé depuis la mémoire non volatile. En admettant que l'attaquant puisse savoir où en est l'exécution et mesurer la consommation de courant durant cette étape, il pourra alors retrouver de l'information sur le poids de Hamming des valeurs chargées et ainsi obtenir une liste de candidats potentiels de polynômes.

Deuxième proposition : [Str13b] Soit disant la première attaque visant à retrouver la clé privée et non pas seulement le message en clair, ce fût probablement le cas au moment de l'écriture de [Str13b]. Cette attaque temporelle a pour but de retrouver le support $\mathscr{L}$ utilisé pour construire le code de Goppa, en commençant par localiser la position de l'élément nul de $\mathbb{F}_{2^m}$ dans $\mathscr{L}$. L'idée est de prendre des vecteurs erreurs de petits poids (1, 4 et 6) et de mesurer le temps d'exécution de l'algorithme d'Euclide étendu.

Notre contribution : Nous ré-utilisons l'attaque proposée dans [Str13b] en la combinant avec une autre (présentée ci-après) afin d'améliorer les résultats (voir Chapitre 7).

D/ Détermination du PLE

Les propositions d'attaques sur la détermination du PLE ont été faites dans [SSMS10, Str10b].

Première proposition : [SSMS10] Malgré la contre-mesure concernant l'évaluation du polynôme localisateur d'erreurs proposée dans [STM+08] (sur laquelle nous reviendrons dans le paragraphe suivant), un canal auxiliaire temporel persiste. En effet, dans [SSMS10], une nouvelle contre-mesure permet à la fois de bloquer

les canaux auxiliaires provenant de la détermination et de l'évaluation du polynôme localisateur d'erreurs.
Revenons donc sur la détermination du polynôme localisateur d'erreurs qui se déroule sur deux étapes de l'algorithme de Patterson (voir la sous-section 3.1.3 du Chapitre 3). Ces deux étapes sont, d'une part la recherche des polynômes $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ tels que $\mathfrak{a}(X) = \mathfrak{b}(X) \cdot R(X) \mod G(X)$ où $\deg(\mathfrak{a}) \leqslant \lfloor \frac{t}{2} \rfloor$ et $R(X) = \sqrt{S_{\tilde{C}}^{-1}(X) + X} \mod G(X)$, et d'autre part le calcul du polynôme localisateur d'erreurs à partir de ces deux polynômes $\sigma_E(X) = \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X)$.

Principe de l'attaque : Dans l'algorithme de Patterson, le temps d'exécution de la détermination du polynôme localisateur d'erreurs, c'est-à-dire pour passer de $R(X)$ à $\sigma_E(X)$, dépend du poids de l'erreur. Le problème d'une relation entre le nombre d'itérations dans l'algorithme d'Euclide étendu, noté N, et le poids du vecteur erreur, noté $w_H(E)$, est soulevé. L'attaque diffère selon la parité de t, le poids théorique du vecteur erreur. Voici ce qu'il en résulte :

1. Si t est impair, alors le degré de $\mathfrak{b}(X)$ doit être égal à $\frac{t-1}{2}$ pour déterminer le terme dominant de $\sigma_E(X)$. Le degré de $\mathfrak{a}(X)$ doit être inférieur ou égal à $\frac{t-1}{2}$. De plus, la diminution de t à $t-1$ du nombre d'erreurs $w_H(E)$ dans un texte chiffré (choisi), entraîne la diminution du nombre d'itérations N.
2. Si t est pair, alors le degré de $\mathfrak{a}(X)$ doit être égal à $\frac{t}{2}$ pour déterminer le terme dominant de $\sigma_E(X)$. Le degré de $\mathfrak{b}(X)$ doit être inférieur ou égal à $\frac{t}{2} - 1$. En revanche, la diminution de $w_H(E)$ à $t-1$ dans un texte chiffré (choisi), n'est pas suffisante pour déduire le nombre d'itérations N. Dans ce cas, il faut trouver (et corriger) au moins deux bits de E.

Notons que la probabilité de deviner correctement ℓ coordonnées du vecteur erreur valant 1 est de :

$$\prod_{i=0}^{\ell} \frac{t-i}{2^m - i}.$$

Scénario de l'attaque : Selon la parité de t, le but est de relier un bit flippé ou deux dans un texte chiffré choisi $\tilde{C}$ (et a fortiori dans le vecteur erreur E, car $\tilde{C} = C + E$) à une durée d'exécution du déchiffrement.
Supposons que t soit impair. Une intervention sur un seul bit e_i est suffisante pour diminuer le nombre d'itérations dans l'algorithme d'Euclide étendu si $e_i = 1$. On peut donc distinguer deux cas :

1. Si $e_i = 0$, alors le bit flippé ajoute une erreur, donc $w_H(E) > t$. Par contre, le degré du PLE vaudra encore t et le nombre d'itérations de l'algorithme d'Euclide étendu sera égal à $\frac{t-1}{2}$.
2. Si $e_i = 1$, alors le bit flippé supprime une erreur, donc $w_H(E) = t - 1$. De plus, le nombre d'itérations de l'algorithme d'Euclide étendu sera inférieur ou égal à $\frac{t-1}{2} - 1$ et le temps de déchiffrement sera plus court que dans le cas précédent.

Un attaquant peut répéter cette procédure pour tous les bits de $\tilde{C}$ (et donc de E). Pour le cas où t est pair, la méthode doit être modifiée pour changer deux bits du texte chiffré.

Contre-mesure : L'idée de la contre-mesure est de détecter un arrêt prématuré de l'algorithme d'Euclide étendu et, si c'est le cas, de poursuivre l'exécution de l'algorithme jusqu'à ce que les degrés corrects de $\mathfrak{a}(x)$ et $\mathfrak{b}(x)$ soient atteints, comme pour un texte chiffré "normal", avec $w_H(E) = t$.
La condition d'arrêt de l'algorithme d'Euclide étendu lors de l'attaque peut être représentée de la façon suivante :

— Si t est pair, alors le degré de $\mathfrak{a}(x)$ peut être utilisé pour détecter un texte chiffré tel que $w_H(E) < t$. En particulier, $\deg(\mathfrak{a})$ doit être égal à $\lfloor \frac{t}{2} \rfloor$ pour être sûr que $w_H(E) \geqslant t$.

— Si t est impair, alors un texte chiffré tel que $w_H(E) < t$ peut être détecté par l'examen du degré de $\mathfrak{b}(x)$, qui doit aussi être égal à $\lfloor \frac{t}{2} \rfloor$ dans la dernière itération.

Dans les deux cas, le canal auxiliaire temporel survenant dans le cas où $w_H(E) < t$ est fermé. En poursuivant l'exécution de l'algorithme d'Euclide étendu, cela se termine par le traitement des polynômes de la suite des restes. Ce traitement est effectué en utilisant des coefficients prédéfinis ou des coefficients pseudo-aléatoires qui sont obtenus de manière déterministe à partir du texte chiffré. On note que ce n'est pas recommandé d'utiliser des données aléatoires pour implanter cette contre-mesure, sinon l'attaquant pourrait recueillir de l'information en détectant que le déchiffrement d'un texte chiffré choisi par lui-même n'est pas déterministe (le résultat n'est pas toujours le même). Il pourrait observer cela en répétant l'arrêt du périphérique qui déchiffre le texte chiffré de son choix, et déterminer la variation des quantités en fonction du temps ou de l'énergie consommée à un instant donné. On note cependant que cette contre-mesure mène à la falsification du résultat, mais uniquement dans le cas où les textes chiffrés ont été modifiés. Cette falsification n'est donc pas critique.

Deuxième proposition : [Str10b] Supposons que l'on utilise une matrice génératrice publique $\tilde{\mathcal{G}}$ sous forme systématique. L'hypothèse faite est donc que la matrice inversible $\mathcal{S}$ n'est pas obligatoirement dans la clé privée. On s'intéresse ici à l'effet de textes chiffrés construits en produisant un mot de code correct dans le code public, en multipliant le message M par une matrice génératrice publique $\tilde{\mathcal{G}}$ sous cette forme bien particulière et en ajoutant ensuite un vecteur erreur E de poids $w_H(E) < t$ (donc en ne respectant pas le système au sens où $w_H(E)$ est supposé de poids exactement t).

Attaque : L'idée de cette attaque est de construire un système d'équations linéaires décrivant la permutation secrète. L'attaquant obtient une liste d'équations en générant des textes chiffrés aléatoirement avec une erreur de poids voulu (petit), dans le but d'exprimer un coefficient du PLE comme fonction des positions des erreurs. Il les fait déchiffrer et récupère le temps d'exécution pour chacun d'eux. Après avoir obtenu un certain nombre d'équations, l'attaquant applique l'algorithme du pivot de Gauss à son système d'équations. Ce système peut être écrit sous la forme d'une matrice binaire de taille $l \times n$, où l est le nombre d'équations et $n = 2^m$ pour le nombre d'éléments dans l'extension de corps (correspondant au support choisi). Les coordonnées doivent être dans l'ordre de la résolution du système.

Contre-mesure : L'attaque peut être déjouée en vérifiant et si nécessaire en modifiant le degré du polynôme $R(X) = \sqrt{S^{-1}(X) + X} \mod G(X)$, variable intermédiaire dans l'algorithme de décodage de Patterson utilisé pour le déchiffrement

(voir l'Algorithme 4). Il faut tester si $\deg(R)$ est plus petit que la condition d'arrêt, c'est-à-dire si $\deg(R) = t-1$ après le calcul de $R(X) = \sqrt{S^{-1}(X) + X} \mod G(X)$.

Notre contribution : Nous proposons une analyse de la probabilité qu'aucun des coefficients du PLE ne soit nul, voir Chapitre 8. Nous proposons également une amélioration des attaques temporelles proposées dans [Str13b] et [STM+08, SSMS10] en les combinant, voir Chapitre 7.

E/ Évaluation du PLE

Les propositions d'attaques sur l'évaluation du PLE ont été faites dans [STM+08, AHPT11].

Algorithmes de recherche de racines d'un polynôme dans un corps fini
Dans cette étape de l'algorithme de Patterson, évaluer le polynôme localisateur d'erreurs revient à chercher ses racines dans le corps $\mathbb{F}_{2^m}$.
Pour cela, une idée est de parcourir tous les éléments du corps un à un et tester s'il est racine ou non (méthode par force brute). Une manière un peu plus élaborée est d'utiliser l'algorithme de Chien, qui fait cela de manière plus efficace. Une autre méthode, plus répandue, est celle de l'algorithme de la Trace de Berlekamp. Pour ne pas perdre en généralité, on aura dans la suite $f = \sigma_E$.

Problème 4.1 *On cherche les racines de $f \in \mathbb{F}_{2^m}[X]$ tel que $\deg(f) = t > 0$. On note $n = 2^m$. On ne connaît pas d'algorithme déterministe et polynomial pour factoriser des polynômes sur les corps finis (d'après [Her11]).*

Algorithme de Chien (1964) : Cet algorithme a été proposé par Robert T. Chien dans [Chi64]. Il teste tous les éléments non nuls du corps $\mathbb{F}_{2^m}$ de manière plus intelligente que la force brute afin de déterminer lesquels sont racines de $f(X)$.
On rappelle les informations suivantes utiles avant de présenter l'algorithme à proprement parler. Soit $\alpha \in \mathbb{F}_{2^m}$ un élément primitif. On a :

$$\begin{aligned} f(X) &= f_0 + f_1 X + \ldots + f_t X^t. \\ f(\alpha^i) &= f_0 + f_1 \alpha^i + \ldots + f_t (\alpha^i)^t. \\ f(\alpha^{i+1}) &= f_0 + f_1 \alpha^{i+1} + \ldots + f_t (\alpha^{i+1})^t. \\ &= f_0 + f_1 \alpha^i \alpha + \ldots + f_t (\alpha^i)^t \alpha^t. \end{aligned}$$

On note $f_{i,j} = f_j (\alpha^i)^j$.
On en déduit que l'on obtient l'image de α^i par f de la manière suivante :

$$f(\alpha^i) = \sum_{j=0}^{t} f_j (\alpha^i)^j, \qquad \forall i = 0, \ldots 2^m - 1.$$

Voici en pseudo-code, l'algorithme de Chien :

Algorithme 20 Chien

Entrée(s): $f(X) = \sum_{i=0}^{t} f_i \cdot X^i \in \mathbb{F}_q[X]$.

Sortie(s): *Racines* l'ensemble des racines de f sur $\mathbb{F}_q$, et le nombre r de racines de f c'est-à-dire le cardinal de *Racines*.

1: $Racines \leftarrow \emptyset$
2: $r \leftarrow 0$
3: **Pour** $j = 0$ à t **faire**
4: $\quad a_j \leftarrow f_j$
5: **Fin pour**
6: **Pour** $i = 0$ à $2^m - 1$ **faire**
7: $\quad sum = a_0$
8: $\quad$ **Pour** $j = 1$ à t **faire**
9: $\quad\quad a_j \leftarrow a_j \cdot \alpha^j$
10: $\quad\quad sum \leftarrow sum \oplus a_j$
11: $\quad$ **Fin pour**
12: $\quad$ **Si** $sum = 0$ **Alors**
13: $\quad\quad$ //(si α^i est racine de f)
14: $\quad\quad Racines \leftarrow Racines \cup \alpha^i$
15: $\quad\quad$ //(On ajoute l'élément α^i à l'ensemble *Racines*.)
16: $\quad\quad r \leftarrow r + 1$
17: $\quad$ **Fin si**
18: **Fin pour**
19: **Retourner** $Racines, r$

On remarque qu'il faut avoir une table des exponentielles jusqu'à t de l'élément primitif α.

Algorithme de la Trace de Berlekamp (1970) : Cet algorithme a été proposé par Elwyn R. Berlekamp dans [Ber70]. Il scinde un polynôme $f \in \mathbb{F}_{2^m}[X]$ qui divise $(X^{2^m} - X)$ en facteurs linéaires grâce à une base de $\mathbb{F}_{2^m}$ sur $\mathbb{F}_2$ (utilisée de manière itérative, c'est-à-dire en parcourant les éléments de cette base) et récursivement sur f. Il utilise des propriétés de la fonction Trace. Sa complexité en temps est $\mathcal{O}(mt^2)$ opérations sur $\mathbb{F}_{2^m}$.

Autres algorithmes : Il existe d'autres algorithmes, qui n'ont pas forcément été testés pour notre problème ou qui ne sont pas adaptés par rapport au degré du polynôme. En voici quelques uns :

Algorithme de Cantor et Zassenhaus' (1981) : Cet algorithme a été proposé par David G. Cantor et Hans Zassenhaus' dans [CZ81]. Il est probabiliste et permet de factoriser un polynôme sur $\mathbb{F}_q$, où $q = p^m$ avec p un nombre premier, qui divise $x^{p^r} - x$. Il est adaptable au problème. Sa complexité en temps est $\mathcal{O}((m + \log(t))t^2 \log(t))$ opérations sur $\mathbb{F}_{2^m}$.

Procédures de Zinoviev (1996) : Cet algorithme a été proposé par Victor A. Zinoviev dans [Zin96]. Il trouve le multiple affine $A(x)$ de plus petit degré de n'importe quel polynôme $f(X)$ de degré $\leqslant 10$ sur $\mathbb{F}_{2^m}$. Ensuite, il cherche puis teste toutes les racines de $A(X)$ à $f(X)$. Il utilise pour cela les notions de polynômes affine et linéarisé.

Algorithme de Truong-Jeng-Reed : Cet algorithme a été proposé par T.-K. Truong, J.-H. Jeng et I.S. Reed dans [TJR01].

Algorithme de Fedorenko-Trifonov (2002) : Cet algorithme a été proposé par Sergei V. Fedorenko et Peter V. Trifonov dans [FT02]. C'est une amélioration de la méthode de Truong-Jeng-Reed en utilisant les polynômes affines, pour des polynômes de n'importe quel degré (et pas uniquement $\leqslant 11$).

Algorithme Berlekamp Trace - Zinoviev (2009) : Cet algorithme a été proposé par Bhaskar Biswas et Vincent Herbert dans [BH09].

Première proposition : [STM+08] La dépendance dans l'algorithme de décodage entre le degré du polynôme localisateur d'erreurs $\sigma_E(X)$ et le nombre d'erreurs peut être utilisée comme base d'une attaque à chiffré choisi par canal auxiliaire.

Attaque : Lors de la construction du vecteur erreur dans le décodage, le polynôme $\sigma_E(X)$ est évalué en chaque élément de $\mathscr{L}$, soit n fois. Il est évident que dans une implantation naïve, le temps de l'évaluation augmentera avec le degré de $\sigma_E(X)$.
Alice chiffre par McEliece un message M en $\tilde{C} = M \cdot \tilde{\mathcal{G}} \oplus E$ et envoie le chiffré $\tilde{C}$ à Bob. Ève voit $\tilde{C}$ passer, le garde en mémoire et prépare son attaque. Puis, Ève envoie à Bob des chiffrés modifiés $\tilde{C}_i$, où $\tilde{C}_i$ est $\tilde{C}$ avec le $i^{\text{ème}}$ bit modifié. Elle peut ainsi déterminer la valeur de e_i :

1. Si le degré du polynôme reste le même, car les calculs se font modulo $G(X)$, alors $e_i = 0$.
2. Si le degré du polynôme diminue, car Ève a en fait corrigé une erreur, alors $e_i = 1$.

Une manière de procéder pour construire E est de chronométrer le temps de déchiffrement pour tous les $\tilde{C}_i$ avec $i = 1, \ldots, n$, puis de prendre les t plus petits temps. Les indices correspondants seront ceux où $e_i = 1$ avec une très forte probabilité. Une fois qu'Ève a reconstruit le vecteur erreur, elle peut facilement retrouver le message. On suppose qu'elle est capable de mesurer le temps d'exécution pour chaque déchiffrement et que cette étape est la seule qui pourrait expliquer des différences de temps.
En présence d'une version du cryptosystème résistante aux attaques adaptatives à chiffrés choisis, les chiffrés envoyés par Ève de cette manière ne sont pas traités. Mais cette attaque peut être menée sur une sous-chaîne de $\tilde{C}$.

Contre-mesure : Pour les paramètres $m = 11$ et $t = 50$ et pour deux chronométrages par coordonnées modifiées, des résultats expérimentaux montrent que le bon vecteur E est obtenu une fois sur deux. Une explication de cette efficacité est que le polynôme $\sigma_E(X)$ est évalué n fois dans l'algorithme de Patterson, soit ici 2048. Cela signifie que même la plus petite différence est notable par rapport à l'ensemble. Afin d'éviter ces différences de temps, une idée est d'augmenter artificiellement le degré du polynôme dans le cas où il serait inférieur à t. Pour éviter les différences de temps, tous les coefficients dans le polynôme de degré t doivent être non nuls.

Améliorations de l'attaque : Deux améliorations sont envisageables.
Pour la première, supposons qu'Ève ait trouvé un indice i auquel $e_i = 1$ grâce à

l'attaque initiale. Pour un i fixé, elle peut alors envoyer à Bob des chiffrés modifiés $\tilde{C}_{i,j}$ à partir de $\tilde{C}_i$ (en ayant modifié le $j^{\text{ème}}$ bit de $\tilde{C}_i$), où $j \neq i$, pour les faire déchiffrer. Ainsi, la différence dans les poids d'erreur serait de 2 et les différences de temps plus grandes.
Pour la deuxième, supposons qu'Ève sache quel est le nombre exact de positions erronées (disons k avec $k \leqslant t$) et non-erronées (donc $n-k$). Alors elle peut modifier $\tilde{C}$ en $\tilde{C}_l$ de telle sorte que $w_H(\tilde{C}_l) = k$ et où l représente un choix de k indices parmi n. Les $\tilde{C}_l$ ont toujours le même nombre d'erreurs. Ainsi, l'algorithme de Patterson commencera avec des syndromes différents mais il n'y aura plus de différences considérables entre deux temps de l'attaque initiale, car les polynômes $\sigma_E(X)$ auront le même degré. Ève pourra alors éliminer des différences de temps possibles dues à certains syndromes, en faisant une moyenne de temps de déchiffrement pour chaque $\tilde{C}_l$ et en n'oubliant pas que parmi ceux-ci, lorsque des erreurs sont corrigées, le degré du syndrome diminue et le temps de déchiffrement également. Ainsi, plus il y aura d'erreurs corrigées, plus le temps de déchiffrement sera petit.

Autre possibilité d'attaque : Une attaque sur la consommation d'énergie est également envisageable sur cette étape du déchiffrement, de la même manière que pour l'évaluation du polynôme G en chaque élément du support $\mathscr{L}$, mais là encore les mêmes idées de contre-mesures reviennent.

Deuxième proposition : [AHPT11] La détermination du vecteur erreur par l'évaluation du polynôme localisateur d'erreurs revient à effectuer une recherche des racines du polynôme localisateur d'erreurs dans $\mathscr{L} = \mathbb{F}_{2^m}$. Par définition du polynôme localisateur d'erreurs, cela revient à le factoriser. Cette étape domine le temps total du déchiffrement.

Attaque : Le principe de cette attaque est le même que pour la précédente.
L'idée est de commencer par effectuer des pré-calculs qui serviront de référence ensuite pour l'attaque. Cette phase sera appelée la phase initiale. Le but est de construire un tableau de longueur $t+1$ contenant le temps moyen de déchiffrement en fonction des poids d'erreurs possibles (c'est-à-dire de 0 à t). Ève crée des vecteurs de poids w (avec w compris entre 0 et t), le donne à déchiffrer à Bob, et chronomètre l'exécution de l'algorithme. Elle fait cela un certain nombre de fois (nombre qu'elle aura préalablement choisi) afin de calculer une moyenne par rapport au poids du vecteur erreur. Elle stocke alors ses résultats dans un tableau. On peut imaginer que pour réaliser cela, elle dispose de deux algorithmes supplémentaires à celui de la phase initiale, un qui génère des mots de code aléatoires et un qui génère des vecteurs erreurs de longueur n et de poids de Hamming w.
La phase suivante est celle de l'attaque à proprement parler. Ève intercepte un texte chiffré de longueur n contenant t erreurs envoyé par Alice à Bob. Le but de son attaque est de trouver le message le plus vraisemblable correspondant à ce chiffré. Au départ, Ève sait que le texte chiffré contient w erreurs, où $w = t$. Ensuite, le principe est de parcourir un à un les bits du chiffré en changeant le bit courant et en regardant le temps mis pour le déchiffrement de celui-ci par Bob. Ces temps sont stockés dans un tableau en fonction de l'indice du bit modifié. Ève sélectionne les w plus petit temps. Les indices correspondants lui permettent alors de corriger les erreurs du chiffré de manière hypothétique. Ce nouveau chiffré est ensuite envoyé à Bob pour avoir son temps de déchiffrement. Ève compare alors celui-ci avec ses résultats de la phase initiale. Selon le nombre d'erreurs qu'elle supposera

avoir corrigé, elle diminuera w, recommencera avec ce nouveau chiffré partiellement corrigé et ainsi de suite jusqu'à ce que w vaille 0.

Contre-mesure : Commençons par donner une définition du non-support, utile à la compréhension de la contre-mesure.

Définition 4.1 (Le non-support) *Soit $\mathscr{L} \subseteq \mathbb{F}_{2^m}$ le support du code. Le non-support est défini par l'une des trois manières suivantes :*

1. $$\bar{\mathscr{L}} = \mathbb{F}_{2^m} \backslash \mathscr{L}$$

   si et seulement si $\mathscr{L} \subsetneq \mathbb{F}_{2^m}$,
2. *si on impose $|\mathscr{L}| = 2^m$, par :*

   $$\bar{\mathscr{L}} = \mathbb{F}_{2^{m'}} \backslash \mathscr{L}$$

   avec $m' = m + 1$,
3. *si on impose $\mathscr{L} = \mathbb{F}_{2^m}$, par :*

   $$\bar{\mathscr{L}} = \mathbb{F}_{2^{xm}} \backslash \mathbb{F}_{2^m}$$

   avec $x \geqslant 2$.

Dans le dernier cas, le calcul de l'erreur doit être fait dans l'extension de corps $\mathbb{F}_{(2^m)^x}$.

Le principe proposé ici reprend l'idée énoncée dans l'introduction comme contre-mesure au canal auxiliaire de l'attaque sur l'algorithme d'exponentiation modulaire. En effet, il s'agit d'avoir un polynôme qui soit toujours de même degré, que l'on évaluera ensuite sur le support du code. Pour construire un tel polynôme, ne dépendant pas du poids de l'erreur contenue dans le chiffré donné en entrée de l'algorithme de déchiffrement, mais s'annulant toujours avec les mêmes éléments du support que le PLE, il faut que ce polynôme soit un multiple du PLE. Il sera donc construit comme produit du PLE et d'un polynôme de degré $t - w$, tel que ce dernier ne s'annule pas sur le support. On aura t le degré du polynôme de Goppa et w le poids du vecteur erreur dissimulé dans le texte chiffré $\tilde{C}$ (par conséquent w sera le degré du PLE). Ce polynôme complémentaire au PLE sera construit comme produit de polynômes de degré 1 s'annulant chacun en un élément du non support. Ainsi, le produit du PLE et du polynôme complémentaire s'annulera pour les mêmes éléments du support que le PLE dans $\mathbb{F}_{2^m}$ et ne dépendra pas de w. L'attaque devient alors inutile et le message en sortie sera correct.

Notre contribution : Étude d'une implantation matérielle de l'évaluation du polynôme localisateur d'erreurs (sur FPGA, voir Chapitre 8).

Proposition de contre-mesure : En gardant l'idée d'avoir dans tous les cas un polynôme de degré t à évaluer en chaque élément de $\mathscr{L}$ quelque soit le degré du polynôme localisateur d'erreur et donc le poids de l'erreur, il nous est apparu une nouvelle proposition pour contrer cette attaque.

Idée : Plutôt que de passer par le non-support afin d'augmenter le degré du polynôme syndrome si nécessaire, pourquoi ne pas évaluer un polynôme $f(X)$ qui serait la somme du PLE $\sigma_E(X)$ et d'un monôme de degré t, βX^t, dont le coefficient

β serait un élément primitif de $\mathbb{F}_{2^m}$ par exemple. Le test de l'évaluation ne serait plus $e_i = 1$ lorsque $\sigma_E(\alpha_i) = 0$ mais lorsque $f(\alpha_i) = \beta \cdot \alpha_i^t$. Or, $\beta \cdot \alpha_i^t$ n'est autre qu'une puissance de β. Sa valeur peut donc être pré-calculée est un accès mémoire à la table de recherche suffirait pour vérifier cette égalité.

Risque : Un risque peut tout de même persister en fonction de l'algorithme utilisé pour l'évaluation du polynôme f en chaque élément de $\mathscr{L}$. Ceci est à étudier plus en profondeur. Par exemple, si le polynôme f est creux, alors en utilisant l'algorithme de Horner pour l'évaluation polynomiale, on pourrait remarquer que le calcul reste rapide.

Dans la troisième partie de cette thèse, nous allons présenter les attaques effectuées contre le déchiffrement de McEliece. Elles seront présentées dans l'ordre d'exécution de l'algorithme (utilisant celui de Patterson). Nous finirons cette thèse par une étude de ce qui se passe une fois que la matrice de permutation a été retrouvée (via une attaque par canal auxiliaire par exemple).

Troisième partie

Attaques par canaux auxiliaires contre le cryptosystème de McEliece

Chapitre 5

Attaque par analyse différentielle de consommation du calcul de la permutation

Ce chapitre correspond aux travaux effectués pour [PRD+16a, PRD+16b] :

Differential Power Analysis Attack on the Secure Bit Permutation in the McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
RadioElektronika 2016, p. 132-137, IEEE.

Low-complexity CBDPA Countermeasure for Resource-Constrained Embedded McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
soumis à RadioEngineering 2016.

5.1 Permutation du chiffré

Dans le cryptosystème de McEliece, le déchiffrement débute par une multiplication entre le texte chiffré reçu et l'inverse de la matrice de permutation. Notons que l'inverse d'une matrice de permutation n'est autre que sa transposée. Puisque la clé privée est connue du destinataire légitime pour le déchiffrement, les calculs peuvent être effectués correctement. Nous rappellons dans l'Algorithme 21 les grandes lignes du déchiffrement de McEliece.

Algorithme 21 Déchiffrement de McEliece

Entrée(s): $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \Gamma)$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair associé à $\tilde{C}$.

1: Calculer $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$.
2: $//\tilde{C}_p = M \cdot \mathcal{S} \cdot \mathcal{G} \oplus E \cdot \mathcal{P}^{-1}$
3: Décoder $\tilde{C}_p$ pour retrouver $M \cdot \mathcal{S} \cdot \mathcal{G}$.
4: Récupérer $\tilde{M} = M \cdot \mathcal{S}$ à partir de $M \cdot \mathcal{S} \cdot \mathcal{G}$.
5: Calculer $M = \tilde{M} \cdot \mathcal{S}^{-1}$.
6: **Retourner** M.

Venons-en maintenant au calcul de la permutation. La manière la plus intuitive de le faire est développée dans l'Algorithme 22.

Algorithme 22 Permutation du texte chiffré (version intuitive)

Entrée(s): $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ l'inverse de la matrice de permutation privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté.

1: **Pour** $i = 0$ **à** $n-1$ **faire**
2: $\quad j = t_i^{\mathcal{P}^{-1}}$
3: $\quad \tilde{C}_{p_i} = \tilde{C}_j$
4: **Fin pour**
5: **Retourner** $\tilde{C}_p$

Cependant, cet algorithme s'est révélé vulnérable à une attaque sur la mémoire cache une fois implantée en logiciel [STM+08]. Les auteurs ont proposé à PQCrypto 2008 dans ce même article l'Algorithme 23 comme contre-mesure pour palier à ce problème. Or, nous allons montrer dans la suite de ce chapitre, que ce dernier est lui-même vulnérable à une attaque par analyse de consommation.

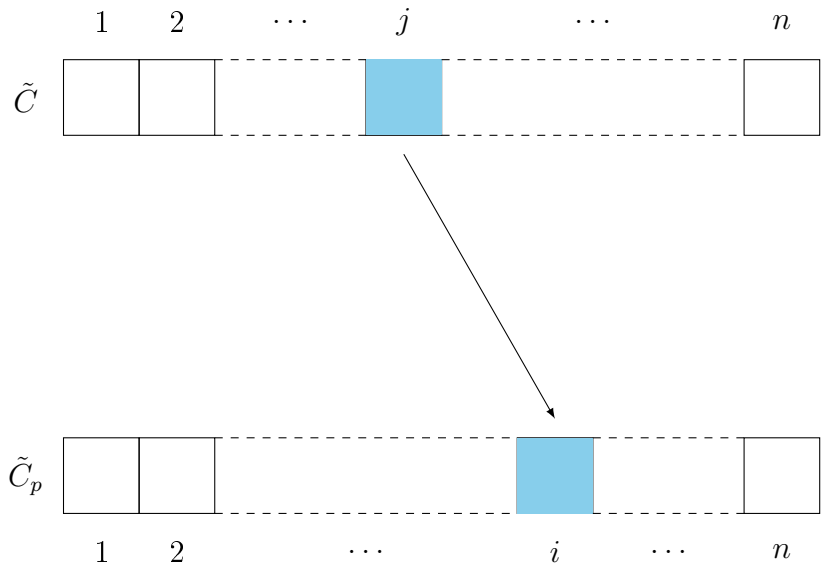
Algorithme 23 Permutation du texte chiffré (de [STM+08])

Entrée(s): $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ l'inverse de la matrice de permutation privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté.

1: **Pour** $i = 0$ **à** $n-1$ **faire**
2: $\quad j = t_i^{\mathcal{P}^{-1}}$
3: $\quad \tilde{C}_{p_i} = 0$
4: $\quad$ **Pour** $h = 0$ **à** $n-1$ **faire**
5: $\quad\quad k = \tilde{C}_{p_i}$
6: $\quad\quad \mu = \tilde{C}_h$
7: $\quad\quad s = j \oplus h$
8: $\quad\quad s \mid= s \gg 1$
9: $\quad\quad s \mid= s \gg 2$
10: $\quad\quad s \mid= s \gg 4$
11: $\quad\quad s \mid= s \gg 8$
12: $\quad\quad s \mid= s \gg 16$
13: $\quad\quad s \,\&= 1$
14: $\quad\quad s = \sim (s-1)$
15: $\quad\quad \tilde{C}_{p_i} = (s \,\&\, k) \mid ((\sim s) \,\&\, \mu)$
16: $\quad$ **Fin pour**
17: **Fin pour**
18: **Retourner** $\tilde{C}_p$

Le but de l'Algorithme 23 est qu'à la fin de celui-ci, pour tout i, on ait $\tilde{C}_{p_i} = \tilde{C}_j$ (voir la Figure 5.1).

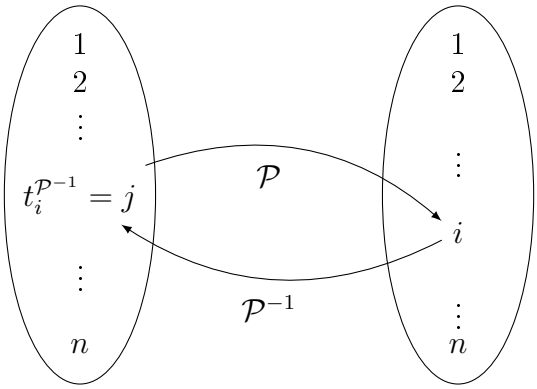
FIGURE 5.1 – Permutation du texte chiffré



Afin de mettre en évidence la vulnérabilité de cet algorithme, décortiquons celui-ci en commençant par les entrées. La permutation est stockée sous forme de tableau, où $j = t_i^{\mathcal{P}^{-1}}$ (élément i du tableau $t^{\mathcal{P}^{-1}}$) est l'image de i par la permutation $\mathcal{P}^{-1}$ (pour la ligne 2), voir Figure 5.2. Quant au texte chiffré $\tilde{C}$, c'est le vecteur de n bits que l'on veut permuter afin d'obtenir $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$ (Algorithme 21, ligne 1, en rouge).

L'indice i, via la première boucle **Pour**, sert à parcourir toutes les coordonnées du texte chiffré permuté (ligne 1). On note $\tilde{C}_{p_i}$ le $i^{\text{ème}}$ bit du texte chiffré permuté $\tilde{C}_p$. Comme dit précédemment, j sera donc l'image de i par la permutation $\mathcal{P}^{-1}$ (ligne 2). Donc i a pour image j par la permutation $\mathcal{P}^{-1}$, autrement dit i est l'image de j par la permutation $\mathcal{P}$, voir Figure 5.2. (Il ne faut pas confondre $t^{\mathcal{P}^{-1}}$ avec sa réciproque : la permutation $\mathcal{P}$.) On note $\tilde{C}_j$ le $j^{\text{ème}}$ bit du texte chiffré $\tilde{C}$.

FIGURE 5.2 – Permutation $\mathcal{P}$ et sa réciproque



Le $i^{\text{ème}}$ bit du texte chiffré permuté est initialisé à 0 à la ligne 3. Ensuite, l'indice de parcours h apparaît via la deuxième boucle **Pour**. C'est là que la contre-mesure intervient, en comparaison avec l'Algorithme 22.

À la ligne 5, k contient le $i^{\text{ème}}$ bit du texte chiffré permuté $\tilde{C}_p$, initialisé à 0 avant la deuxième boucle **Pour**. Puis μ prend la valeur du $h^{\text{ième}}$ bit du texte chiffré $\tilde{C}$, qui est une valeur dont on ne se soucie pas, sauf si $h = j$, puisque l'on veut que $\tilde{C}_{p_i} = \tilde{C}_j$. Cela signifie qu'il faudra mettre à jour le $i^{\text{ème}}$ bit du texte chiffré permuté $\tilde{C}_p$ à la ligne 15 uniquement dans le tour de la deuxième boucle **Pour** pour lequel $h = j$ et ne pas changer sa valeur sinon.

Pour le reste, regardons plutôt quelques traces d'exécution pour diverses valeurs extrêmes (voir Tableau 5.1), afin de mieux comprendre ce que fait la deuxième boucle **Pour**. Pour les indices s, j et h, les bits seront donnés dans le sens poids fort à gauche à poids faible à droite et ce pour 32 bits, car cette contre-mesure, donnée dans [STM+08], avait été implantée en logiciel et pour des données de 32 bits. Pour notre attaque développée ci-après, 16 bits suffisent et la ligne 12 se révélera alors inutile. Les opérations qui interviennent entre les lignes 7 et 14 sont les suivantes :

- $\oplus$ pour le OU EXCLUSIF ;
- $\gg i$ pour le décalage de i rang dans le vecteur vers la droite (c'est-à-dire la division entière par 2^i) ;
- $|$ pour le OU ;
- $\&$ pour le ET bit à bit ;
- $\sim$ pour le complément à 1.

Les indices des bits iront de 0 à $n-1$ et pas de 1 à n comme dans le reste de la discussion, afin de respecter les notations de l'Algorithme 23 .

On remarque que, quelque soit la valeur de s à la ligne 7 (au début du Tableau 5.1), à la ligne 13 de ce même tableau s prendra toujours la valeur $\underbrace{00\ldots0}_{31}1$, <u>sauf</u> dans le cas où $j = h$ (d'où $j \oplus h = \underbrace{00\ldots0}_{32}$) et donc quand s vaut $\underbrace{00\ldots0}_{32}$ à la ligne 7 (jusqu'à la ligne 14, dans la dernière colonne de la partie haute du tableau). De même pour la ligne 14 que pour la ligne 13, s vaudra toujours $\underbrace{11\ldots1}_{32}$, <u>sauf</u> dans le cas où s valait $\underbrace{00\ldots0}_{32}$ à la ligne 7. Cela nous conduit donc à l'étape critique, qui est la mise à jour du $i^{\text{ème}}$ bit du texte chiffré permuté $\tilde{C}_p$ à la ligne 15 (en rouge dans l'Algorithme 23).
À la ligne 15, on peut voir qu'il y a un OU entre deux valeurs :

$$\tilde{C}_{p_i} = \underbrace{(s \,\&\, k)}_{\substack{\text{vrai si } s=11\ldots1 \\ \text{faux sinon}}} \;|\; \underbrace{((\sim s) \,\&\, \mu)}_{\substack{\text{vrai si } s=00\ldots0 \\ \text{faux sinon}}}$$

On peut noter que ces deux valeurs ne peuvent pas être vraies simultanément car s et $\sim s$ sont complémentaires. On en déduit que deux cas possibles sont distinguables :

1. Si la deuxième proposition $((\sim s) \,\&\, \mu)$ est VRAI, cela signifie que $(\sim s) = 11\ldots1$ et donc que $s = 00\ldots0$ à la ligne 7, autrement dit on est dans le cas où $j = h$ et par conséquent où $\tilde{C}_{p_i}$ est mis à jour avec $\mu = \tilde{C}_j$ (car $\mu = \tilde{C}_h$ à la ligne 6).
2. En revanche, si la première proposition $(s \,\&\, k)$ est VRAI, cela signifie que $s \neq 00\ldots0$ à la ligne 7, autrement dit on est dans le cas où $j \neq h$ et par conséquent où $\tilde{C}_{p_i}$ est mis à jour avec k, mais doit rester inchangé. Là encore deux cas sont distinguables :

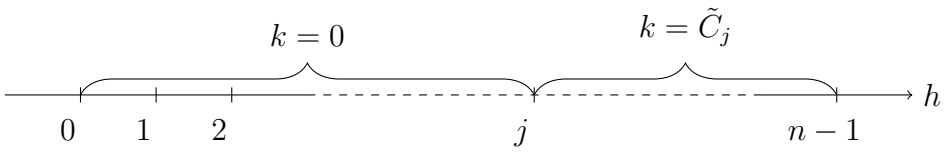
$$k = \begin{cases} 0 & \text{pour } 0 \leqslant h < j \\ \tilde{C}_j & \text{pour } j < h \leqslant n-1 \end{cases}$$

TABLEAU 5.1 – Exécutions partielles de l'Algorithme 23

Instructions	Hypothèses de test			
$7 : s = j \oplus h$	$1\underbrace{00\ldots0}_{31}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$8 : s \mid= s \gg 1$	$11\underbrace{00\ldots0}_{30}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$9 : s \mid= s \gg 2$	$1111\underbrace{00\ldots0}_{28}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$10 : s \mid= s \gg 4$	$\underbrace{11\ldots1}_{8}\underbrace{00\ldots0}_{24}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$11 : s \mid= s \gg 8$	$\underbrace{11\ldots1}_{16}\underbrace{00\ldots0}_{16}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$12 : s \mid= s \gg 16$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$
$13 : s \,\&= 1$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{00\ldots0}_{32}$
$14 : s = \sim(s-1)$	$\underbrace{11\ldots1}_{32}$	$\underbrace{11\ldots1}_{32}$	$\underbrace{11\ldots1}_{32}$	$\underbrace{00\ldots0}_{32}$

Instructions	Hypothèses de test	
$7 : s = j \oplus h$	$\underbrace{00\ldots0}_{16}1\underbrace{00\ldots0}_{15}$	$\underbrace{00\ldots0}_{15}1\underbrace{00\ldots0}_{16}$
$8 : s \mid= s \gg 1$	$\underbrace{00\ldots0}_{16}11\underbrace{00\ldots0}_{14}$	$\underbrace{00\ldots0}_{15}11\underbrace{00\ldots0}_{15}$
$9 : s \mid= s \gg 2$	$\underbrace{00\ldots0}_{16}1111\underbrace{00\ldots0}_{12}$	$\underbrace{00\ldots0}_{15}1111\underbrace{00\ldots0}_{13}$
$10 : s \mid= s \gg 4$	$\underbrace{00\ldots0}_{16}\underbrace{11\ldots1}_{8}\underbrace{00\ldots0}_{8}$	$\underbrace{00\ldots0}_{15}\underbrace{11\ldots1}_{8}\underbrace{00\ldots0}_{9}$
$11 : s \mid= s \gg 8$	$\underbrace{00\ldots0}_{16}\underbrace{11\ldots1}_{16}$	$\underbrace{00\ldots0}_{15}\underbrace{11\ldots1}_{16}0$
$12 : s \mid= s \gg 16$	$\underbrace{00\ldots0}_{16}\underbrace{11\ldots1}_{16}$	$\underbrace{00\ldots0}_{15}\underbrace{11\ldots1}_{17}$
$13 : s \,\&= 1$	$\underbrace{00\ldots0}_{31}1$	$\underbrace{00\ldots0}_{31}1$
$14 : s = \sim(s-1)$	$\underbrace{11\ldots1}_{32}$	$\underbrace{11\ldots1}_{32}$

ou de manière plus visuelle :



Ainsi, $\tilde{C}_{p_i}$ sera inchangé et vaudra bien $\tilde{C}_j$ à la fin de l'algorithme, puisqu'il a été initialisé à 0 à la ligne 3 pour permettre cette mise à jour dans de bonnes conditions.

Les détails de cette vulnérabilité face à notre attaque sont donnés ci-après.

5.2 Attaque par DPA

La première analyse du calcul de la permutation faite d'un point de vue canal auxiliaire a été proposée dans [HMP10]. Les auteurs y ont énoncé quatre profils correspondants à quatre manières différentes d'envisager une implantation de la première étape du déchiffrement de McEliece (permutation du texte chiffré) et de la première étape du décodage des codes de Goppa (calcul du syndrome) selon qu'elles soient fusionnées ou non. En effet, comme ces deux étapes sont consécutives, on peut les voir comme un seul bloc ou non. Ces profils sont expliqués dans la Sous-section 4.2.2 de cette thèse. Pour l'attaque proposée ici, nous avons supposé que nous étions dans le cadre d'une implantation du type "Profile I" ou "Profile II", c'est-à-dire que l'on permute le texte chiffré en le multipliant par la matrice de permutation. Nous rappelons que la différence entre ces deux profils se fait sur la manière d'appréhender le calcul du syndrome et que ce dernier n'était pas notre cible pour cette attaque.

5.2.1 Opération sensible

Dans l'Algorithme 23, l'opération sensible se situe à la ligne 15 (en rouge). En effet, pour h allant de 0 à $j-1$, $\tilde{C}_{p_i}$ sera nul ; et pour h allant de j à $n-1$, $\tilde{C}_{p_i}$ sera égal à μ, autrement dit $\tilde{C}_h$. En d'autres termes, le $j^{\text{ème}}$ bit de $\tilde{C}$ deviendra le $i^{\text{ème}}$ bit de $\tilde{C}_p$ (c'est-à-dire $\mathcal{P}^{-1}(\tilde{C}_j) = \tilde{C}_{p_i}$). Ce changement de valeur s'effectue uniquement lorsque $j = h$ (c'est-à-dire $s = 00\ldots0$) et aucun changement n'est opéré dans les autres cas (c'est-à-dire $s \neq 00\ldots0$). Cette propriété nous a permis de déployer une attaque par analyse différentielle de consommation (*Differential Power Analysis* (DPA) en Anglais) sur une implantation de cet algorithme. En effet, via une attaque par DPA, nous sommes capables de détecter l'instant exact de cette affectation de valeur (c'est-à-dire $\tilde{C}_{p_i} = \mu$) et ainsi de révéler une ligne correspondante dans la matrice de permutation.

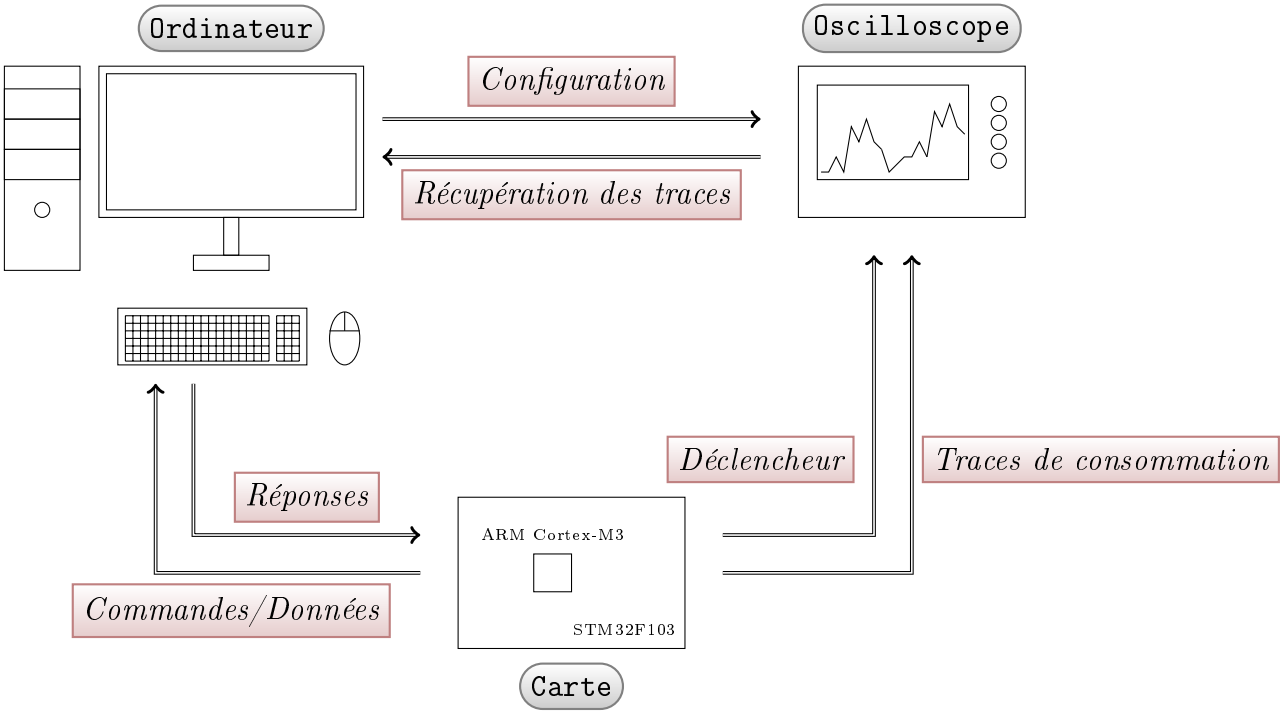
5.2.2 Implantation attaquée

Une implantation logicielle de l'Algorithme 23 a été réalisée sur un microcontrôleur du type STM32F103 de chez STMicroelectronics [16] . Ce microcontrôleur est un système embarqué composé d'un microprocesseur ARM Cortex-M3 de 32 bits

16. `http://www.st.com/web/en/catalog/mmc/FM141/SC1169`

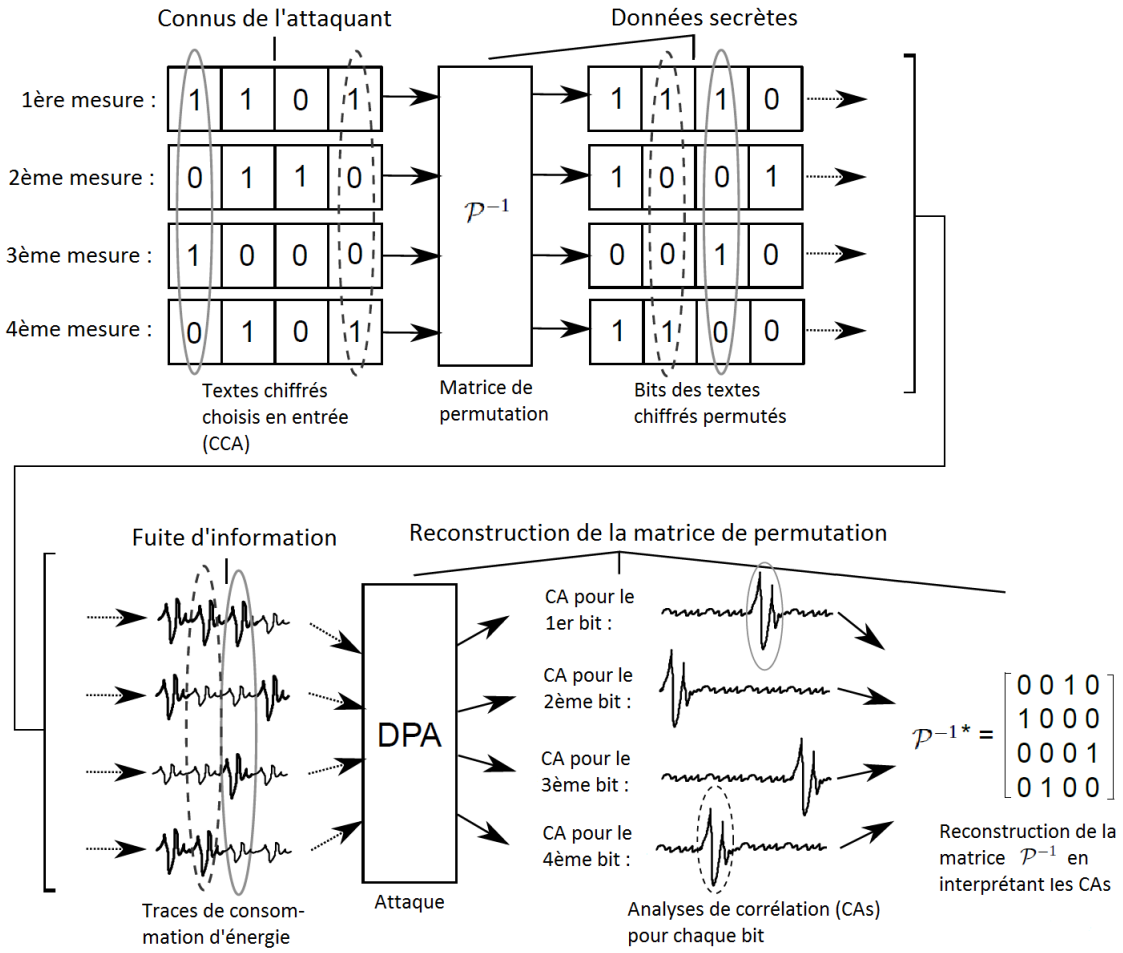
cadencé à 72MHz. Un schéma du banc d'attaque est donné dans la Figure 5.3. Nous avons récupéré les traces de consommation d'énergie de manière instantanée en mesurant la tension aux bornes d'une résistance de 1Ω, en série entre la prise terre de la carte et celle du sol. Le système embarqué utilisé pour l'attaque a été spécialement développé pour des attaques par canaux auxiliaires. Cela permet d'obtenir une fuite d'information nette.

FIGURE 5.3 – Schéma du banc d'attaque par DPA



Les traces de consommation ont été acquises en utilisant deux (des quatres) canaux analogiques d'un oscilloscope de chez Agilent Technologies [Agi]. Toutes les traces nécessaires à l'attaque ont été acquises au rythme de 250×10^6 mesures échantillonnées par seconde. Deux sondes de 500MHz ont été connectées directement à la carte attaquée contenant le microcontrôleur pour effectuer ces mesures. L'acquisition des données a été contrôlée par un programme exécuté tournant sur l'ordinateur inclus dans l'oscilloscope. Ce programme a servi à la fois à envoyer les textes chiffrés au microcontrôleur et à récupérer les traces de consommation à la fin de chaque acquisition.
Le système embarqué envoyait un signal pour débuter les mesures à l'oscilloscope (déclencheur) et commençait le déchiffrement du texte chiffré. L'oscilloscope mesurait la consommation d'énergie durant la première étape du déchiffrement : le calcul de la permutation. Une fois l'acquisition pour un texte chiffré terminée, la trace était envoyée à l'ordinateur, qui la stockait sur son disque dur. Le processus de mesure fût répété ainsi pour le nombre de traces voulues (500 fois pour l'exemple qui suit). La Figure 5.4 représente les principales étapes de l'attaque par DPA sur le calcul de la permutation du texte chiffré. Ces traces furent ensuite analysées et interprétées afin de reconstruire entièrement la matrice de permutation.

FIGURE 5.4 – Exemple pour l'attaque par DPA



5.2.3 Analyse des traces

Commençons par rappeler le principe d'une attaque par DPA. En tant qu'attaquant, pour effectuer une analyse différentielle de consommation, nous avons besoin de plusieurs traces pour une même clé privée. La récupération de celles-ci a été décrite dans la sous-section précédente. La partie de la clé privée que nous avons ciblée était la matrice de permutation. Donc, l'idée est que, pour une même hypothèse de clé privée (et différents textes chiffrés), on effectue plusieurs mesures de la consommation, pour ensuite en faire une moyenne. Cette moyenne sert à minimiser le bruit qui dissimule l'information intéressante pour l'attaque (la consommation au cours de l'opération sensible).
Une fois les traces de consommation récupérées, nous avons utilisé le coefficient de corrélation de Pearson [BCO04] pour le texte chiffré connu en entrée. Nous avons appliqué le poids de Hamming au modèle de fuite des bits individuellement ($H_i \in \{0, 1\}$). Pour l'analyse de corrélation[17] , nous avons ensuite utilisé l'équa-

17. Attention, les notations dans cette formule n'ont rien à voir avec les notations choisies dans cette thèse, notamment pour la théorie des codes.

tion 5.1 :

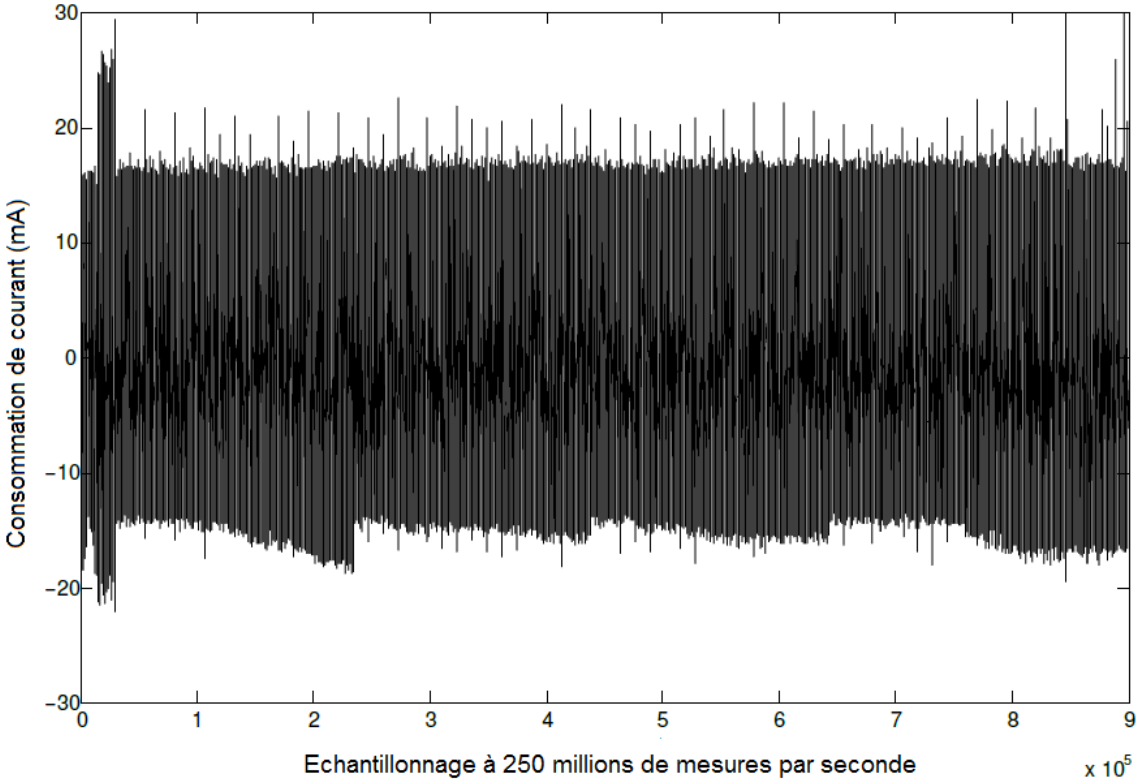
$$r_{H,X}(\eta) = \frac{\sum_{i=1}^{N}[(X_i(\eta) - \bar{X}(\eta))(H_i - \bar{H})]}{\sqrt{\sum_{i=1}^{N}[X_i(\eta) - \bar{X}(\eta)]^2 \sum_{i=1}^{N}(H_i - \bar{H})^2}}, \tag{5.1}$$

où $r_{H,X}(\eta)$ est le coefficient de corrélation de Pearson pour le $\eta^{\text{ème}}$ échantillon mesuré lors de l'exécution du calcul de la permutation du texte chiffré, N est le nombre de traces mesurées, $X_i(\eta)$ est une valeur du $\eta^{\text{ème}}$ échantillon mesuré durant la $i^{\text{ème}}$ mesure ($i^{\text{ème}}$ trace), $\bar{X}(\eta)$ est la valeur moyenne des $\eta^{\text{èmes}}$ échantillons correspondants (pour toutes les traces), H_i est une hypothèse de consommation de courant pour un bit de la donnée en entrée correspondant à la $i^{\text{ème}}$ mesure ($i^{\text{ème}}$ trace) et $\bar{H}$ est une valeur moyenne de toutes les hypothèses H_i.
Nous avons effectué cette analyse de corrélation n fois (où n est la longueur du code, 64 fois pour l'exemple qui suit) pour chaque bit de l'entrée. Nous avons obtenu les positions des bits permutés en cherchant des pics de corrélation durant les analyses, comme dans la Figure 5.6. En comparant les positions des bits permutés aux bits des textes chiffrés pris en entrée, nous avons pu reconstruire la matrice de permutation. Cette attaque par DPA est dite de premier ordre puisqu'elle analyse la consommation de courant pour une seule variable intermédiaire (ici $\tilde{C}_{p_i}$).

5.2.4 Exemple

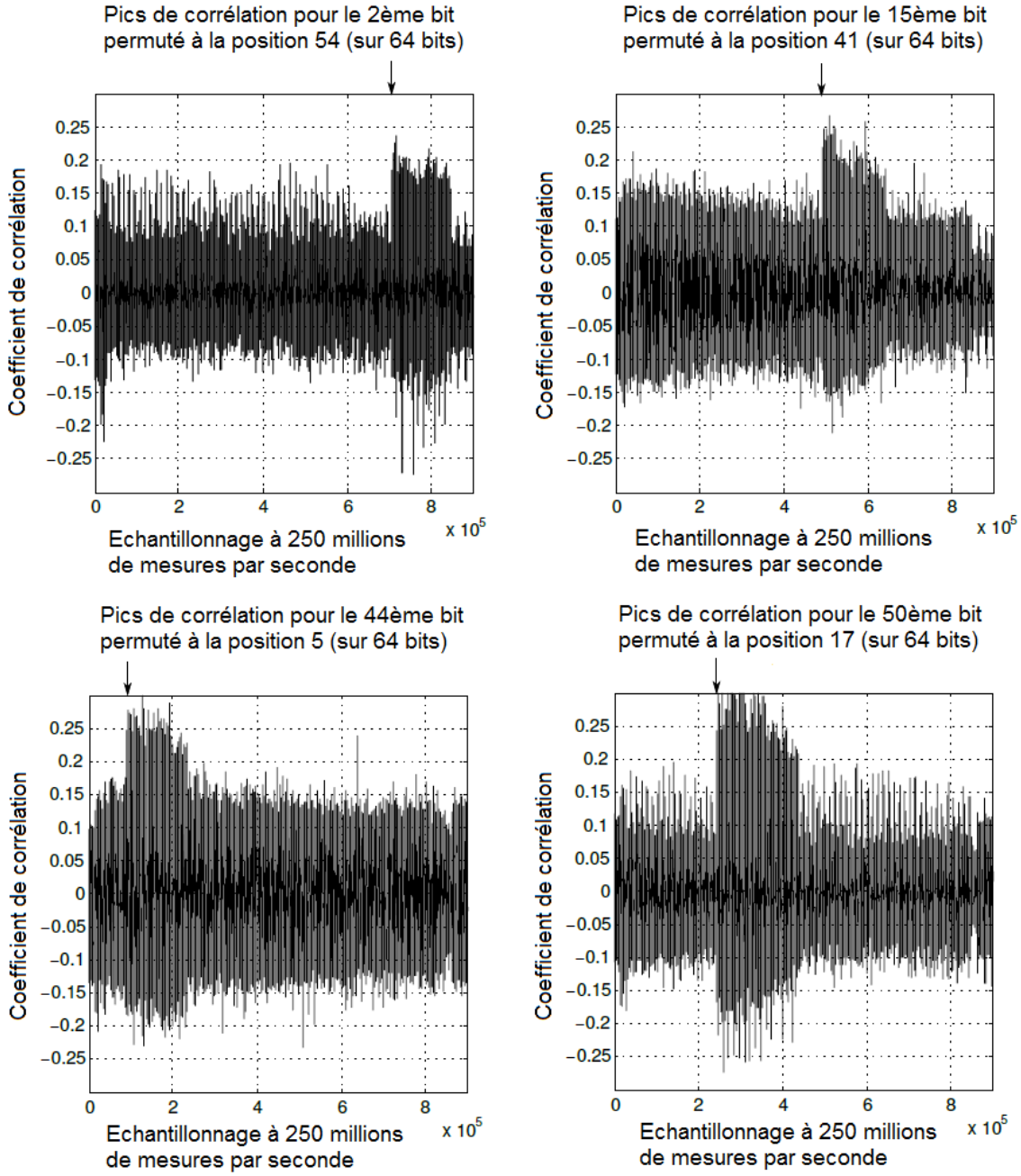
Effectuons cette attaque pour un code binaire de longueur $n = 64$ et visant une matrice de permutation $\mathcal{P}^{-1}$ dans $\mathscr{M}_{n,n}(\mathbb{F}_2)$. La Figure 5.5 est un exemple de trace de consommation. La durée moyenne pour obtenir une trace au rythme de 9×10^5 mesures échantillonnées par seconde est de $1,5$ seconde. Sur la trace de la Figure 5.5, nous pouvons distinguer quatre motifs de variation d'amplitude, notamment dans la moitié négative. Ces motifs sont causés par l'implantation, qui stocke les 64 bits du texte chiffré dans quatre mots de 16 bits. Cela nous permet de distinguer et d'attaquer (puis protéger) chaque mot de manière indépendante. Tout ceci peut bien évidemment être généralisé pour toute longueur de code.

FIGURE 5.5 – Exemple de trace de consommation



Des résultats pour cet exemple sont donnés dans la Figure 5.6. Ce sont quatre analyses de corrélation en utilisant 500 traces. Nous pouvons clairement distinguer l'instant (marqué par une flèche) où un bit des textes chiffrés est manipulé pour la première fois durant le calcul de la permutation. Puisque nous connaissons la position de chaque bit des textes chiffrés pris en entrée de l'algorithme (CCA), et la position des mêmes bits dans les textes chiffrés devinés à partir des analyses de corrélation, nous pouvons reconstruire la matrice de permutation $\mathcal{P}^{-1}$. Notons que la matrice de permutation $\mathcal{P}$ n'est autre que la transposée de $\mathcal{P}^{-1}$, donc si on a l'une, on a l'autre sans calcul supplémentaire. Cette attaque a pris plusieurs minutes (entre 15 et 20 environ) avec 500 traces sur un processeur Intel Core i7 cadencé à 2,4 GHz.

FIGURE 5.6 – Résultats pour l'attaque par DPA



5.3 Contre-mesure

Une technique couramment utilisée comme contre-mesure à une attaque par DPA est le masquage [ZPG13]. Les deux principaux désavantages à l'utilisation de masque comme contre-mesure sont : la nécessité de bits aléatoires et l'augmentation de la complexité calculatoire.

Afin de remédier à la fuite d'information exploitée ici par DPA, nous avons proposé une contre-mesure basée sur les propriétés des codes linéaires. Cela permet d'éviter l'utilisation supplémentaire d'un générateur d'aléa et la complexité n'est pas fondamentalement augmentée. En revanche, nous avons tout de même besoin

d'un masque. L'espace de stockage nécessaire est donc un peu plus grand (n bits supplémentaires).
Les propriétés des codes linéaires mises en avant ici sont les suivantes :

— Les lignes d'une matrice génératrice $\mathcal{G}$ sont des mots de code.
— Tout mot de code a un syndrome nul.

L'idée de notre contre-mesure est d'ajouter un mot de code au texte chiffré au début du déchiffrement. Le texte chiffré sera alors masqué lors de sa permutation et le texte chiffré permuté sera masqué par un mot de code permuté. L'Algorithme 24 détaille cette contre-mesure.

Algorithme 24 Permutation sécurisée du texte chiffré

Entrée(s): $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ la matrice de permutation privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré et $\mathcal{G}$ la matrice génératrice du code de Goppa privé $\Gamma(\mathscr{L}, G)$.
Sortie(s): $\tilde{C}'_p = \tilde{C}_p \oplus B \in \mathbb{F}_2^n$ le chiffré permuté masqué par un mot de code.

1: Choisir aléatoirement $B \in \Gamma(\mathscr{L}, G)$.
2: //une combinaison linéaire des lignes de $\mathcal{G}$
3: $B_p = B \cdot \mathcal{P}$
4: $\tilde{C}' = \tilde{C} \oplus B_p$
5: **Pour** $i = 0$ **à** $n-1$ **faire**
6: $\quad j = t_i^{\mathcal{P}^{-1}}$
7: $\quad \tilde{C}'_{p_i} = 0$
8: $\quad$ **Pour** $h = 0$ **à** $n-1$ **faire**
9: $\quad\quad k = \tilde{C}'_{p_i}$
10: $\quad\quad \mu = \tilde{C}'_h$
11: $\quad\quad s = j \oplus h$
12: $\quad\quad s \mid= s \gg 1$
13: $\quad\quad s \mid= s \gg 2$
14: $\quad\quad s \mid= s \gg 4$
15: $\quad\quad s \mid= s \gg 8$
16: $\quad\quad s \mid= s \gg 16$
17: $\quad\quad s \,\&= 1$
18: $\quad\quad s = \sim (s-1)$
19: $\quad\quad \tilde{C}'_{p_i} = (s \,\&\, k) \mid ((\sim s) \,\&\, \mu)$
20: $\quad$ **Fin pour**
21: **Fin pour**
22: **Retourner** $\tilde{C}'_p$

Pour choisir B dans $\Gamma(\mathscr{L}, G)$ via $\mathcal{G}$ (ligne 1 et commentaire ligne 2), il suffit de prendre une ligne ou une somme de lignes, puisqu'une combinaison linéaire de lignes de $\mathcal{G}$ revient à une simple somme dans $\mathbb{F}_2$. Notons aussi que, comme les lignes de $\mathcal{G}$ forment une base du code linéaire, B est donc bien un mot de code. De plus, lorsque l'on dit, à la première ligne, "choisir aléatoirement", ce qu'il faut retenir c'est que la construction de B doit varier d'un déchiffrement à l'autre. Alors, bien entendu, une manière plus rapide de le faire est d'utiliser un générateur d'aléa, mais ceci est un autre problème.
À la fin de la version sécurisée de l'algorithme, on obtient un texte chiffré permuté

masqué :

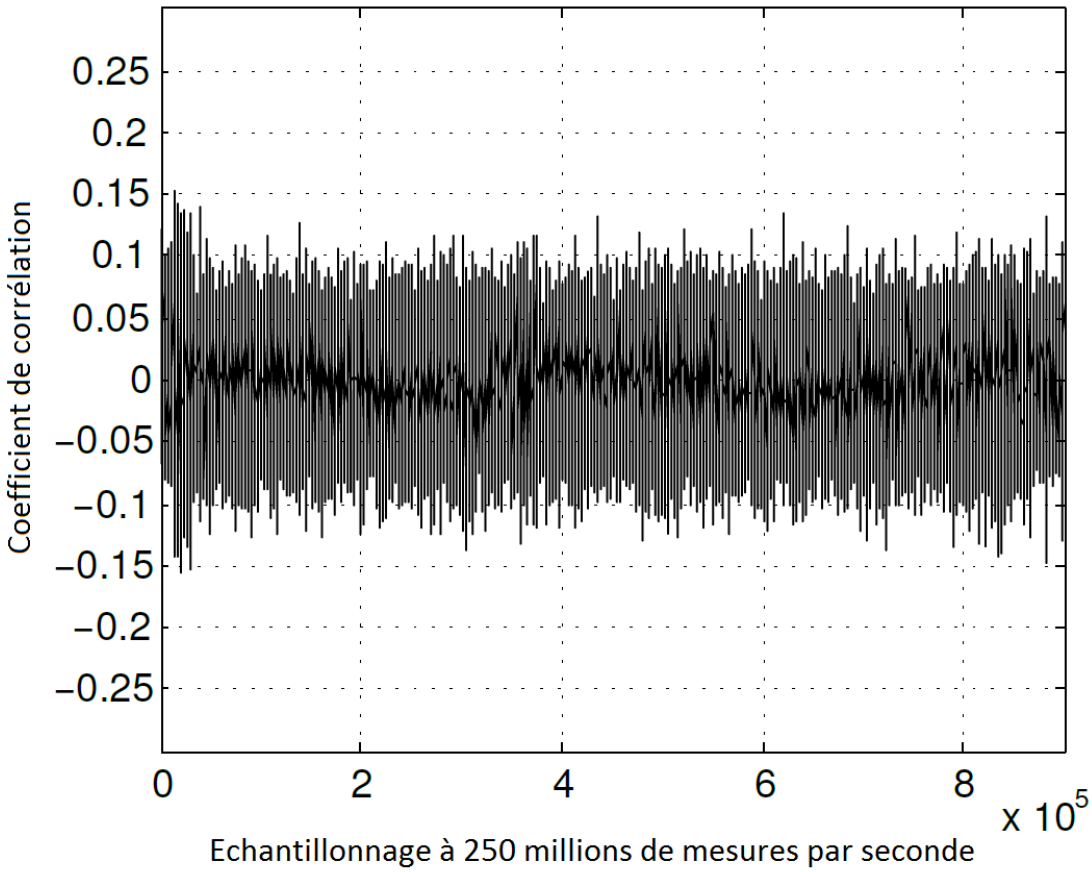
$$\begin{aligned}
\tilde{C}'_p &= \tilde{C}' \cdot \mathcal{P}^{-1} \\
&= (\tilde{C} \oplus B_p) \cdot \mathcal{P}^{-1} \\
&= \tilde{C} \cdot \mathcal{P}^{-1} \oplus (B \cdot \mathcal{P}) \cdot \mathcal{P}^{-1} \\
&= \tilde{C}_p \oplus B.
\end{aligned}$$

L'étape suivante dans le déchiffrement étant le décodage, cette fois de $\tilde{C}'_p$, on récupérera, non pas $M \cdot \mathcal{S} \cdot \mathcal{G}$, mais $M \cdot \mathcal{S} \cdot \mathcal{G} \oplus B$. En effet, comme le calcul du syndrome est la première étape du décodage, on obtiendra :

$$\begin{aligned}
S &= \tilde{C}'_p \cdot {}^t\mathcal{H} \\
&= (\tilde{C}_p \oplus B) \cdot {}^t\mathcal{H} \\
&= \tilde{C}_p \cdot {}^t\mathcal{H} \oplus \underbrace{B \cdot {}^t\mathcal{H}}_{=0} \\
&= \tilde{C}_p \cdot {}^t\mathcal{H}.
\end{aligned}$$

Or, $\tilde{C}_p \cdot {}^t\mathcal{H}$ était le syndrome obtenu avec la version non masquée du calcul de la permutation. Le vecteur erreur reste donc le même pour les deux versions. Il sera ainsi bien détecté et corrigé lors du décodage après utilisation de cette contre-mesure.
L'étape restante à ne pas oublier est le démasquage. Il faut donc stocker le B choisi pour pouvoir démasquer le mot de code obtenu après le décodage et ainsi pouvoir poursuivre le déchiffrement normalement : $(M \cdot \mathcal{S} \cdot \mathcal{G} \oplus B) \oplus B = M \cdot \mathcal{S} \cdot \mathcal{G}$. Si cette étape de démasquage est omise, on se retrouvera alors dans le cas où le message en clair sera encore masqué. En posant M_B tel que $M_B \cdot \mathcal{S} \cdot \mathcal{G} = B$, on obtiendrait donc $M \cdot \mathcal{S} \cdot \mathcal{G} \oplus B$ après le décodage, qui donnerait $\tilde{M} = M \cdot \mathcal{S} \oplus M_B \cdot \mathcal{S} = (M \oplus M_B) \cdot \mathcal{S}$ et enfin comme "message en clair" $M \oplus M_B$. Ce mot serait valide mais ne correspondrait pas au texte chiffré reçu.
Pour l'exemple de la DPA donné précédemment, la trace de consommation après contre-mesure est donnée dans la Figure 5.7.

FIGURE 5.7 – Exemple pour la contre-mesure de la DPA



5.4 Conclusion et perspectives

Comme poursuite sur ces travaux, voici quelques pistes envisagées :
Il serait intéressant de regarder si la contre-mesure reste efficace pour une attaque différentielle par analyse de consommation d'ordre supérieur, autrement dit en analysant la consommation de courant à différents instants, via non pas une mais plusieurs variables intermédiaires. Une possibilité serait de combiner deux attaques par consommation sur deux étapes distinctes du déchiffrement.
L'attaque présentée dans ce chapitre (et a fortiori la contre-mesure associée) ne s'applique que dans deux des quatre profils énoncés dans [HMP10] (les profils I et II). Il faudrait regarder d'un point de vue canal auxiliaire ces deux autres profils afin de compléter l'étude.
Enfin l'attaque présentée dans ce chapitre a été réalisée contre un système embarqué. L'implantation était logicielle. Il faudrait donc tester cette attaque contre une implantation matérielle, autrement dit contre une implantation sur un circuit logique programmable (*Field-Programmable Gate Array* (FPGA) en Anglais).

Chapitre 6

Attaques par analyses simple et différentielle de consommation du calcul du syndrome

Ce chapitre correspond aux travaux effectués pour [PRD+15, PRD+16b] :

Countermeasure against the SPA Attack on an Embedded McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
RadioElektronika 2015, p. 462-466, IEEE.

Low-complexity CBDPA Countermeasure for Resource-Constrained Embedded McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
soumis à RadioEngineering 2016.

Une présentation a été faite du travail en cours :

A Side-Channel Attack Against the Secret Permutation on an Embedded McEliece Cryptosystem
The 3rd Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2015), Grenoble (France), Mars 2015.

6.1 Calcul du syndrome

Dans le cryptosystème de McEliece, le calcul du syndrome intervient au début de n'importe quel algorithme de décodage lors du déchiffrement. Cette opération est une multiplication entre un vecteur et une matrice. Nous rappellons dans l'Algorithme 25 les grandes lignes du déchiffrement de McEliece (précédemment dans le Chapitre 3, Algortihme 7).

Algorithme 25 Déchiffrement de McEliece

Entrée(s): $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \Gamma)$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré, où $\tilde{C} = M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus E$.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair associé à $\tilde{C}$.
1: Calculer $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$.
2: Décoder $\tilde{C}_p$ pour retrouver $M \cdot \mathcal{S} \cdot \mathcal{G}$.
3: Récupérer $\tilde{M} = M \cdot \mathcal{S}$ à partir de $M \cdot \mathcal{S} \cdot \mathcal{G}$.
4: Calculer $M = \tilde{M} \cdot \mathcal{S}^{-1}$.
5: **Retourner** M.

Le décodage du texte chiffré permuté $\tilde{C}_p$ pour retrouver le mot de code $M \cdot \mathcal{S} \cdot \mathcal{G}$ peut être fait de différentes façons. Les grandes lignes du décodage pour un code

de Goppa sont rappelées dans l'Algorithme 26 (précédemment dans le Chapitre 2, Algorithme 1).

Algorithme 26 Décodage d'un code de Goppa

Entrée(s): $\tilde{C}_p = M \cdot \mathcal{S} \cdot \mathcal{G} \oplus E \cdot \mathcal{P}^{-1}$ un texte chiffré permuté tel que $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$ un code de Goppa t-correcteur et $w_H(E \cdot \mathcal{P}^{-1}) \leqslant t$, avec $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ le support du code et G le polynôme de Goppa.

Sortie(s): $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$ le mot de code sans erreur.

1: Calculer le polynôme syndrome de $\tilde{C}_p$: $S_{\tilde{C}_p}(X)$.
2: Résoudre l'équation clé : $S_{\tilde{C}_p}(X)\sigma_{E\cdot\mathcal{P}^{-1}}(X) = \frac{d\sigma_{E\cdot\mathcal{P}^{-1}}(X)}{dX} \mod G(X)$.
3: //(afin d'obtenir le polynôme localisateur d'erreurs $\sigma_{E\cdot\mathcal{P}^{-1}}(X)$)
4: Construire le vecteur erreur E tel que $e_i = 1$ si $\sigma_{E\cdot\mathcal{P}^{-1}}(\alpha_i) = 0$ et $e_i = 0$ sinon.
5: **Retourner** $M \cdot \mathcal{S} \cdot \mathcal{G} = \tilde{C}_p \oplus E \cdot \mathcal{P}^{-1}$.

Le calcul du syndrome est un produit vecteur-matrice $S = \tilde{C}_p \cdot {}^t\mathcal{H}$. Présentons-le de manière simple dans l'Algorithme 27. Son implantation sera la cible de notre attaque.

Algorithme 27 Calcul du syndrome (version 1)

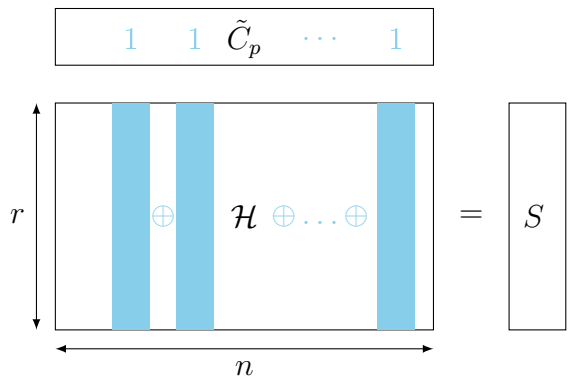
Entrée(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté, $\mathcal{H} \in \mathscr{M}_{k,n}(\mathbb{F}_2)$ une matrice de contrôle de $\Gamma(\mathscr{L}, G)$.

Sortie(s): $S \in \mathbb{F}_2^r$ le syndrome de $\tilde{C}_p$.

1: **Pour** $i = 1$ **à** n **faire**
2: **Si** $\tilde{C}_{p_i} = 1$ **Alors**
3: $S = S \oplus H_i$
4: **Fin si**
5: **Fin pour**
6: **Retourner** S

De manière schématique, voici ce que cela donne (Figure 6.1) :

Figure 6.1 – Calcul du syndrome



Pour passer du vecteur syndrome S au polynôme syndrome $S_{\tilde{C}_p}(X)$, il suffit de calculer $S_{\tilde{C}_p}(X) = S \cdot {}^t(X^{t-1}, X^{t-2}, \ldots, X, 1)$, c'est-à-dire de voire chaque coordonnée du vecteur S comme le coefficient correspondant dans $S_{\tilde{C}_p}(X)$ en partant du coefficient dominant.

Regardons à présent en quoi cet algorithme est vulnérable.

6.2 Attaques par analyse de consommation

La première attaque par analyse de consommation contre le cryptosystème de McEliece a été proposée dans [HMP10]. Les auteurs y ont énoncé quatre profils correspondants à quatre manières différentes d'envisager une implantation de la première étape du déchiffrement de McEliece (permutation du texte chiffré) et de la première étape du décodage des codes de Goppa (calcul du syndrome) selon qu'elles soient fusionnées ou non. En effet, comme ces deux étapes sont consécutives, on peut les voir comme un seul bloc ou non. Ces profils sont expliqués dans la Sous-section 4.2.2 de cette thèse. Pour l'attaque proposée ici, nous avons supposé que nous étions dans le cadre d'une implantation du type "Profile II" ou "Profile IV", c'est-à-dire que l'on calcule le syndrome en effectuant une multiplication entre un vecteur (le texte chiffré permuté ou le texte chiffré respectivement) et une matrice (une matrice de contrôle ou une matrice de contrôle permutée respectivement). Nous rappelons que la différence entre ces deux profils se fait sur la manière d'appréhender la permutation ayant lieu juste avant le calcul du syndrome et que celle-ci n'était pas l'opération cible pour cette attaque.

6.2.1 Implantation attaquée

Une implantation logicielle (en C) de l'Algorithme 27 a été réalisée sur un microcontrôleur du type STM32F103 de chez STMicroelectronics [18] . Ce microcontrôleur est un système embarqué composé d'un microprocesseur ARM Cortex-M3 de 32 bits cadencé à 72MHz. Un schéma du banc d'attaque est donné dans la Figure 6.2. Nous avons récupéré les traces de consommation d'énergie de manière instantanée en mesurant la tension aux bornes d'une résistance de 1Ω, en série entre la prise terre de la carte et celle du sol. La carte est équipée d'un port série pour les transferts de données. Le système embarqué utilisé pour l'attaque a été spécialement développé pour des attaques par canaux auxiliaires. Cela permet d'obtenir une fuite d'information nette.
Les traces de consommation ont été acquises en utilisant deux (des quatres) canaux analogiques d'un oscilloscope de chez Agilent Technologies [Agi]. Toutes les traces nécessaires à l'attaque ont été acquises au rythme de :

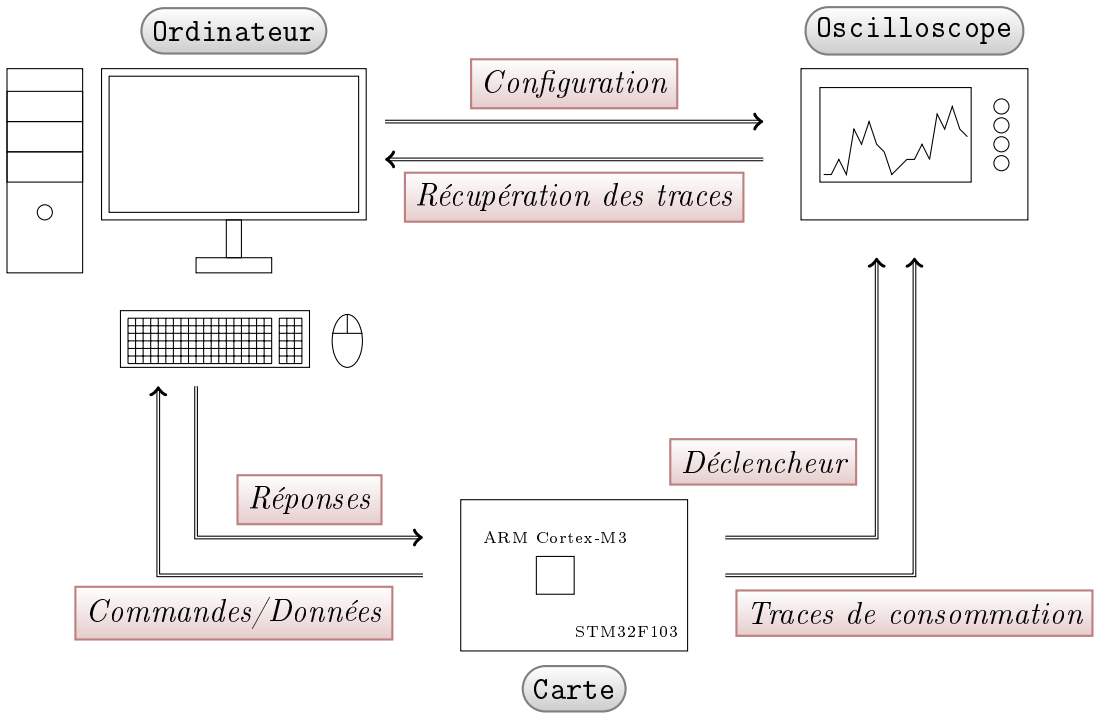
— 10^9 mesures échantillonnées par seconde pour l'analyse simple de la consommation (*Simple Power Analysis* (SPA) en Anglais),
— 100×10^6 mesures échantillonnées par seconde après la contre-mesure pour la SPA,
— et 250×10^6 mesures échantillonnées par seconde pour l'analyse différentielle de la consommation (*Differential Power Analysis* (DPA) en Anglais) et sa contre-mesure.

Deux sondes de 500MHz ont été connectées directement à la carte attaquée contenant le microcontrôleur pour effectuer ces mesures. L'acquisition des données a été contrôlée par un logiciel tournant sur l'ordinateur inclus dans l'oscilloscope. Ce logiciel a servi à la fois à envoyer les textes chiffrés au microcontrôleur et à récupérer les traces de consommation à la fin de chaque acquisition.
Le système embarqué envoyait un signal pour débuter les mesures à l'oscilloscope et

18. `http://www.st.com/web/en/catalog/mmc/FM141/SC1169`

commençait le décodage du texte chiffré permuté. L'oscilloscope mesurait la consommation d'énergie durant la première étape du décodage : le calcul du syndrome. Une fois l'acquisition pour un texte chiffré permuté terminée, la trace était envoyée à l'ordinateur qui la stocke sur son disque dur. Le processus de mesure fût répété ainsi pour le nombre de traces voulues (500 fois pour chaque texte chiffré, pour chacune des attaques qui suivent). La Figure 6.3 représente les principales étapes de l'attaque par SPA sur le calcul du syndrome du texte chiffré permuté. Ces traces furent ensuite analysées et interprétées afin de reconstruire entièrement la matrice de permutation.

FIGURE 6.2 – Schéma du banc d'attaque par analyse de consommation



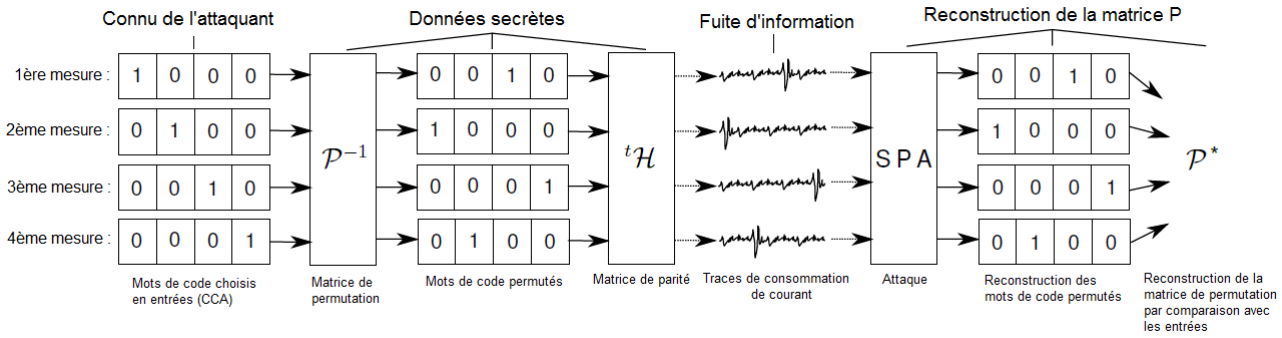
6.2.2 Attaque par SPA

Opération sensible

Un moyen classique permettant une attaque par canal auxiliaire est d'avoir un branchement non équilibré dans l'algorithme. Une branche est un choix d'opérations à effectuer ou non, en fonction d'un test. Le plus souvent ce choix se matérialise par un test **Si**. Sur une donnée privée, cela peut-être un désastre d'un point de vue sécurité du protocole, car de l'information peut fuiter via un canal auxiliaire.
On peut voir que, dans l'Algorithme 27, une branche intervient à la ligne 2. Cette branche est dépendante de la valeur du bit i du texte chiffré permuté $\tilde{C}_p$. En effet, si $\tilde{C}_{p_i} = 1$, alors on fait quelque chose, sinon on ne fait rien. Ceci est critique d'un point de vue canal auxiliaire. C'est cette partie qui nous intéresse ici et que nous allons attaquer en utilisant une analyse simple de la consommation (*Simple Power Analysis* (SPA) en Anglais) sur une implantation du calcul du syndrome. En effet, via une attaque par SPA, nous sommes capables de détecter l'instant exact où une addition modulo 2 (un XOR) est effectuée (c'est-à-dire $S = S \oplus H_i$), c'est-à-dire lorsque la valeur du syndrome est mise à jour. En connaissant l'ordre des éléments

dans le texte chiffré choisi au départ et en récupérant l'ordre des éléments au cours du calcul du syndrome, soit juste après que la permutation du texte chiffré eut été appliquée, la matrice de permutation est ainsi révélée. Notons que cette attaque avait déjà été proposée dans [HMP10, Sous-section 5.2].

FIGURE 6.3 – Exemple pour l'attaque par SPA



Analyse des traces

Commençons par rappeler le principe d'une attaque par SPA. En tant qu'attaquant, pour effectuer une analyse simple de consommation, nous n'avons besoin que d'une seule trace pour la clé privée ciblée. La méthode de récupération de celle-ci a été décrite dans la sous-section précédente. La partie de la clé privée que nous ciblons dans cette attaque est la matrice de permutation. Sur une courbe de consommation, l'attaquant doit être capable de reconnaître la forme de la consommation pour une opération donnée et ainsi déterminer où en est l'exécution de l'implantation. Le succès d'une telle attaque dépend donc directement de la connaissance que l'attaquant a de l'implantation et du fonctionnement du support. Cela signifie que l'attaquant doit être capable d'analyser une trace de consommation pour révéler des détails par rapport à l'exécution de l'implantation, notamment le cas intéressant pour une attaque par canal auxiliaire de type SPA fourni par une branche conditionnelle (un test **Si**) dépendante d'une information privée.

Exemple

La longueur de code recommandée pour les implantations du cryptosystème de McEliece sont 1024 (voire 2048 ou plus). Prenons l'exemple de $n = 1024$. Notons toutefois que ce paramètre conduit à un niveau de sécurité du cryptosystème suffisant mais pour une courte durée. Des longueurs supérieures sont devenues préférables aujourd'hui. Bien entendu, ce que nous présentons ici est facilement adaptable à d'autres longueurs.
Comme expliqué dans la sous-section précédente, nous utilisons des textes chiffrés choisis (*Chosen Ciphertext Attack* (CCA) en Anglais) pour attaquer le calcul du syndrome. Nous avons mesuré 1024 traces correspondant aux exécutions de l'Algorithme 27 avec les 1024 textes chiffrés de poids "1" possibles en entrée, choisis comme ceci :

1ère mesure : $\tilde{C} = (1, \underbrace{0, \ldots, 0}_{1023})$

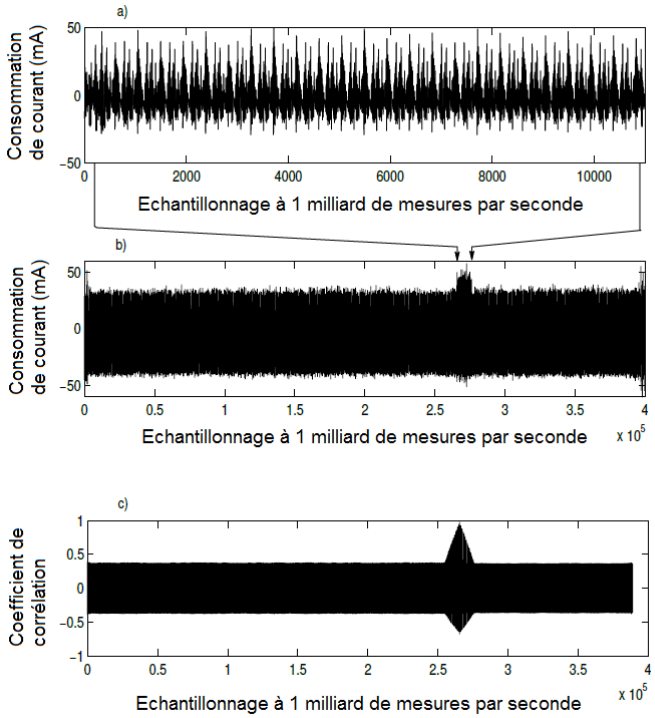
2ème mesure : $\tilde{C} = (0, 1, \underbrace{0, \ldots, 0}_{1022})$

$\vdots$

Dernière mesure : $\tilde{C} = (\underbrace{0, \ldots, 0}_{1023}, 1)$

Un exemple d'acquisition de traces pour la longueur $n = 1024$ est donné dans la Figure 6.4. La Figure 6.4b) montre un exemple typique de traces avec 400 000 mesures échantillonnées de l'ARM Cortex-M3 durant un calcul du syndrome. Dans toutes les autres traces, le motif de l'opération critique apparaît à différents instants sur la courbe. Cette apparition du motif dépend directement de la position du "1" choisie pour le texte chiffré utilisé en entrée après permutation. En effet, en regardant la matrice de permutation sur $\mathbb{F}_2$, celle-ci a exactement un seul "1" sur chaque ligne et chaque colonne. Pour la retrouver, notre but est de retrouver chaque position des "1" dans cette matrice.

FIGURE 6.4 – Résultats pour l'attaque par SPA



Les traces de corrélation ont été obtenues en calculant les coefficients de corrélation entre les motifs des additions modulo 2 (XOR) à partir de la Figure 6.4a) et d'une fenêtre dite de glissement de la trace de consommation en cours de mesure (pour avoir une légère marge d'erreur), par exemple ce qui est décrit dans la Figure 6.4b). Comme on peut l'observer sur cette figure, il est possible de localiser le motif correspondant à l'addition modulo 2 aux alentours de la 270 000ème mesure échantillonnée, pour laquelle le coefficient de corrélation est presque égal à 1.
Un logiciel a été développé pour automatiser l'évaluation des traces, calculer la corrélation entre les traces comme dans la Figure 6.4c), déterminer la position exacte à laquelle le motif de l'addition modulo 2 débute et enfin ordonner (dans le temps)

toutes les positions trouvées sur l'ensemble de la trace de consommation (les traces étant mises bout à bout) dans le but de reconstruire toute la matrice de permutation $\mathcal{P}$.
Dans une implantation simple, la matrice de permutation $\mathcal{P}$ pourrait être déduite avec succès à partir des positions des additions modulo 2 durant le calcul du syndrome pour toutes les traces mesurées (en les superposant après les avoir alignées au même instant de départ).

Contre-mesure (partie 1/2)

Notre contre-mesure est basée sur un principe simple pour éviter des branches de décision et des temps dépendants des données. Nous fournissons la contre-mesure dans l'Algorithme 28. Rappelons que la version non sécurisée a été donnée dans l'Algorithme 27.

Algorithme 28 Calcul du syndrome (version 2)

Entrée(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté, $\mathcal{H}$ la matrice de contrôle de $\Gamma(\mathscr{L}, G)$.
Sortie(s): $S \in \mathbb{F}_2^r$ le syndrome de $\tilde{C}_p$.

1: $words = r/sizeof(S)$ (nombre d'octets nécessaires pour stocker S)
2: **Pour** $i = 1$ **à** n **faire**
3: //Création d'un masque de calcul
4: $tmp = unsigned(0 - \tilde{C}_{p_i})$
5: **Pour** $j = 1$ **à** $words$ **faire**
6: $S_j = S_j \oplus h_{i,j} \& \ tmp$
7: **Fin pour**
8: **Fin pour**
9: **Retourner** S

Dans l'Algorithme 28, la première boucle **Pour** sert à parcourir tous les bits du texte chiffré permuté $\tilde{C}_p$, tandis que la deuxième sert à parcourir tous les octets du syndrome S. La variable temporaire tmp permet de remplacer le test **Si** de l'Algorithme 27. Deux cas possibles :

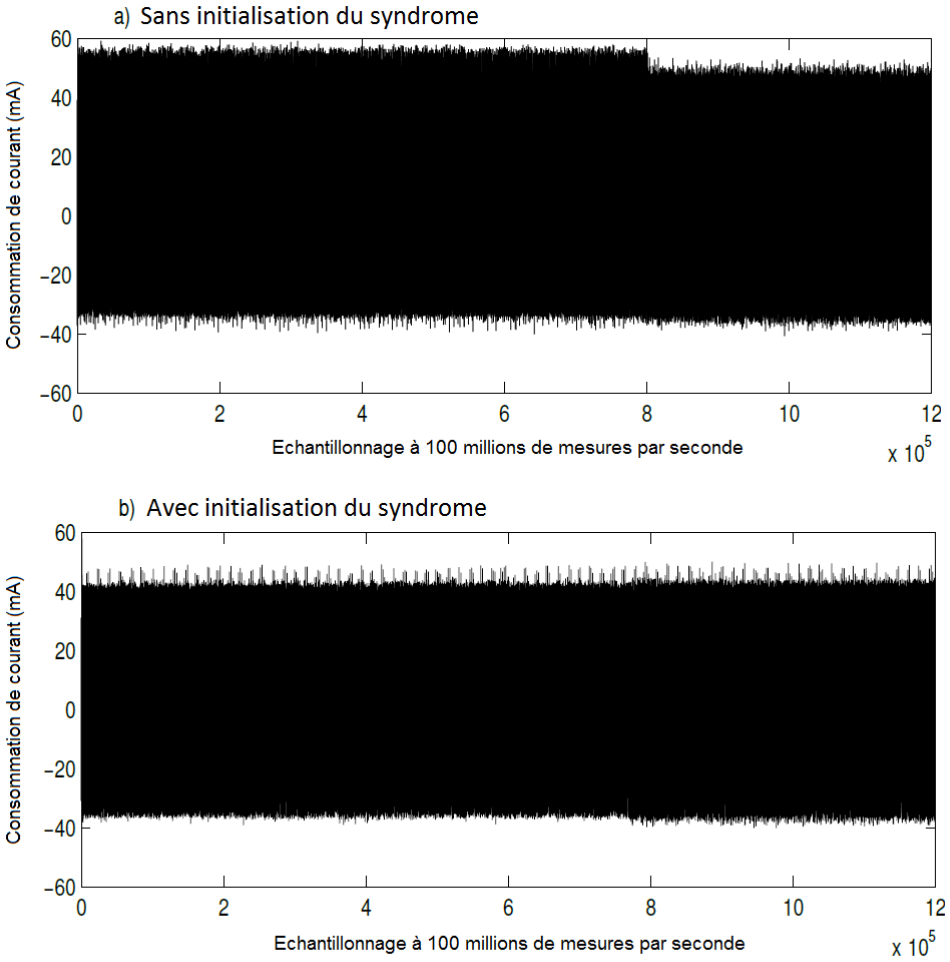
Si $\tilde{C}_{p_i} = 1$ **:** Alors tous les bits de tmp valent 1 et le XOR est à faire.

Sinon $\tilde{C}_{p_i} = 0$ **:** Tous les bits de tmp valent 0 (et on ajoute que des zéros, donc on ne change pas la valeur courante du syndrome).

Dans une simple implantation non protégée, un attaquant peut scanner tous les mots de code bit par bit. S'il trouve une valeur "1", il ajoute la colonne correspondante de la matrice de contrôle $\mathcal{H}$ au résultat (le syndrome). Ces additions (XOR) créent des motifs caractéristiques pour les traces de consommation, qui peuvent servir de base à une attaque par SPA (comme cela a été présenté précédemment). Dans notre solution, nous effectuons des étapes identiques pour chaque bit du mot de code, ce qui supprime les motifs spéciaux dans les traces de consommation tels que les motifs de la Figure 6.4b). De plus, nous avons besoin d'une initialisation spéciale du syndrome dans le but d'éviter l'effet présenté dans la Figure 6.5a), car malgré notre contre-mesure, on peut encore détecter où était le "1" dans le texte chiffré permuté, puisque la consommation de courant diminue lors du calcul du syndrome une fois le "1" passé.

Contre-mesure (partie 2/2)

FIGURE 6.5 – Traces sans et avec initialisation du syndrome



La Figure 6.5a) montre une trace de consommation de 1 200 000 mesures après avoir appliqué la contre-mesure mentionnée ci-dessus (les lignes 2 à 8 incluses dans l'Algorithme 28). Cependant, nous pouvons noter que la position du bit à "1" dans le texte chiffré permuté reste visible. On peut le voir sur le haut de la trace, aux alentours de la 800 000ème mesure, où la valeur "1" a été traitée. On passe alors de 55mA environ à 50mA. Cette diminution de la consommation est due au changement du syndrome S. En effet, dans une implantation simple, la variable S est initialisée à zéro au début de l'algorithme. Dans toutes les itérations, le syndrome est chargé puis enregistré dans la mémoire. Les registres dans le microprocesseur ARM Cortex-M3 sont pré-chargés aux valeurs 0xFFFFFFFF (matériel sur 32 bits) et les nouvelles valeurs sont ensuite écrites dedans. Si la valeur courante est écrite comme étant zéro, alors la consommation sera plus grande à cause de l'effet bascule d'une valeur à une autre.
Comme point de départ pour une nouvelle contre-mesure, nous avons utilisé le fait que les lignes et les colonnes de la matrice de contrôle $\mathcal{H}$ sur $\mathbb{F}_2$ ont approximative-

ment autant de zéros que de uns. Nous avons donc utilisé cette propriété pour dissimuler cette diminution de consommation (Figure 6.5a) en initialisant le syndrome avec le masque "1010...1010" (c'est-à-dire 0xAAAA car l'implantation réalisée était sur 16 bits) avant le calcul qui nous intéresse (lignes 3 à 5 dans l'Algorithme 29). Cette contre-mesure additionnelle est développée dans l'Algorithme 29 et sa consommation est présentée dans la Figure 6.5b), où aucune baisse remarquable de tension n'est clairement visible. Bien entendu, il ne faut pas oublier de démasquer le syndrome à la fin des calculs critiques (lignes 14 à 16 dans l'Algorithme 29).

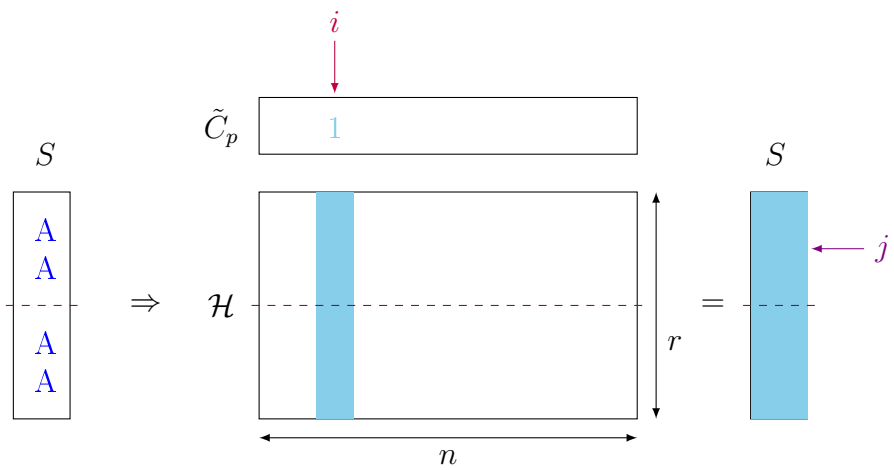
Algorithme 29 Calcul sécurisé du syndrome

Entrée(s): $\tilde{C}_p \in \mathbb{F}_2^n$ le chiffré permuté, $\mathcal{H}$ la matrice de contrôle de $\Gamma(\mathscr{L}, G)$.
Sortie(s): $S \in \mathbb{F}_2^r$ le syndrome de $\tilde{C}_p$.

1: $words = r/sizeof(S)$ (nombre d'octets nécessaires pour stocker S)
2: //Masquage du syndrome
3: **Pour** $j = 1$ **à** $words$ **faire**
4: $S_j = S_j$ & 0xAAAA
5: **Fin pour**
6: //Calcul du syndrome
7: **Pour** $i = 1$ **à** n **faire**
8: $tmp = unsigned(0 - \tilde{C}_{p_i})$
9: **Pour** $j = 1$ **à** $words$ **faire**
10: $S_j = S_j \oplus h_{i,j} \& \ tmp$
11: **Fin pour**
12: **Fin pour**
13: //Démasquage du syndrome
14: **Pour** $j = 1$ **à** $words$ **faire**
15: $S_j = S_j$ & 0xAAAA
16: **Fin pour**
17: **Retourner** S

De manière schématique, le calcul du syndrome donne (Figure 6.6) :

Figure 6.6 – Calcul sécurisé du syndrome



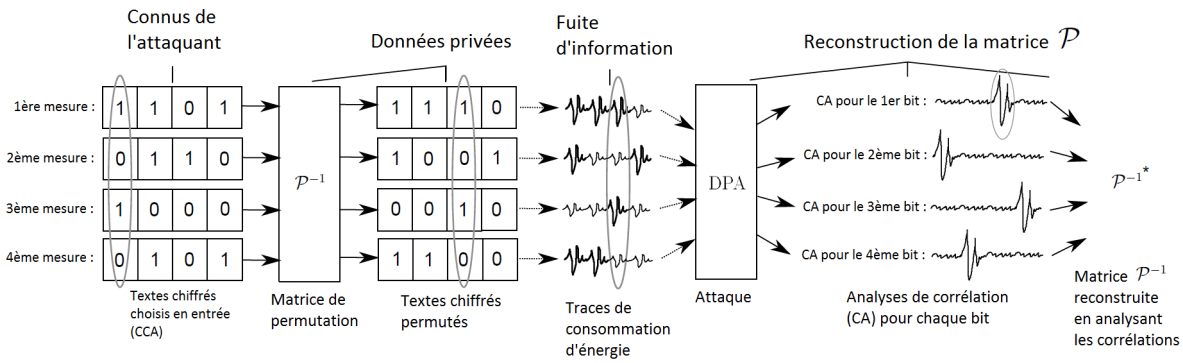
Le problème de cette double contre-mesure pour la SPA est qu'elle s'est révélée attaquable par DPA.

6.2.3 Attaque par DPA

Opération sensible

Le but de la contre-mesure précédemment présentée est d'éviter une branche dans l'algorithme. Le problème est que cette branche est cachée mais reste existante. En effet, à la ligne 8 de l'Algorithme 29, la valeur de tmp dépend directement de la valeur du bit courant de $\tilde{C}_p$. Plus précisément, si $\tilde{C}_{p_i} = 0$, alors $tmp = 0$ et si $\tilde{C}_{p_i} = 1$, alors $tmp =$ 0xFFFF. L'instant auquel la variable tmp est manipulée correspond à la position du bit courant de $\tilde{C}_p$. En comparant le texte chiffré $\tilde{C}$ choisi en entrée (CCA) et le texte chiffré permuté via une analyse différentielle de la consommation (*Differential Power Analysis* (DPA) en Anglais), il est de nouveau possible de retrouver la matrice de permutation entièrement. La matrice de permutation est la première information privée utilisée lors du déchiffrement du McEliece et le calcul du syndrome est effectué juste après la permutation du texte chiffré. C'est la raison pour laquelle nous avons attaqué la contre-mesure de la SPA, donnée dans l'Algorithme 29, par une DPA.

FIGURE 6.7 – Exemple pour l'attaque par DPA



Analyse des traces

Contrairement à la SPA, la DPA utilise des méthodes statistiques et évalue plusieurs traces de consommation pour une même clé privée. La récupération de ces traces a été décrite dans la Sous-section 6.2.1. La partie de la clé privée que nous avons ciblée était la matrice de permutation. Donc, l'idée est que, pour une même hypothèse de clé privée (et différents textes chiffrés), on effectue plusieurs mesures de la consommation, pour ensuite en faire une moyenne. Cette moyenne sert à minimiser le bruit qui dissimule l'information intéressante pour l'attaque (la consommation au cours de l'opération sensible).
Une fois les traces de consommation récupérées, nous avons utilisé le coefficient de corrélation de Pearson [BCO04] pour le texte chiffré choisi en entrée (CCA). Nous avons appliqué le poids de Hamming au modèle de fuite des bits individuellement ($H_i \in \{0, 1\}$). Pour l'analyse de corrélation [19] , nous avons ensuite utilisé l'équa-

19. Attention, les notations dans cette formule n'ont rien à voir avec les notations choisies dans cette thèse, notamment pour la théorie des codes.

tion 6.1 :

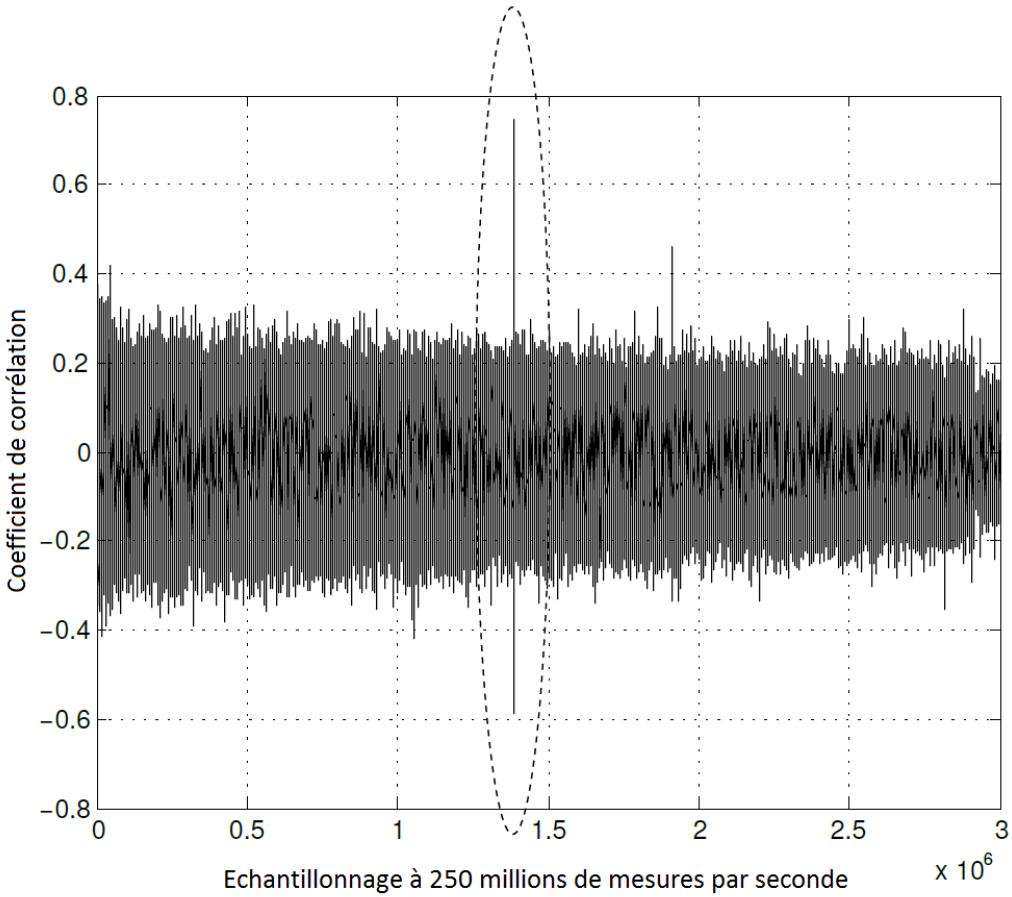
$$r_{H,X}(\eta) = \frac{\sum_{i=1}^{N}[(X_i(\eta) - \bar{X}(\eta))(H_i - \bar{H})]}{\sqrt{\sum_{i=1}^{N}[X_i(\eta) - \bar{X}(\eta)]^2 \sum_{i=1}^{N}(H_i - \bar{H})^2}}, \tag{6.1}$$

où $r_{H,X}(\eta)$ est le coefficient de corrélation de Pearson pour le $\eta^{\text{ème}}$ échantillon mesuré lors de l'exécution du calcul du syndrome du texte chiffré permuté, N est le nombre de traces mesurées, $X_i(\eta)$ est une valeur du $\eta^{\text{ème}}$ échantillon mesuré durant la $i^{\text{ème}}$ mesure ($i^{\text{ème}}$ trace), $\bar{X}(\eta)$ est la valeur moyenne des $\eta^{\text{èmes}}$ échantillons correspondants (pour toutes les traces), H_i est une hypothèse de consommation de courant pour un bit de la donnée en entrée correspondant à la $i^{\text{ème}}$ mesure ($i^{\text{ème}}$ trace) et $\bar{H}$ est une valeur moyenne de toutes les hypothèses H_i.
Nous avons effectué cette analyse différentielle de consommation basée sur la corrélation n fois, soit pour chaque bit de l'entrée (où n est la longueur du code, 1024 fois pour l'exemple qui suit). Nous avons obtenu les positions des bits permutés en cherchant des pics de corrélation durant les analyses, comme dans la Figure 6.8. En comparant les positions des bits permutés aux bits des textes chiffrés pris en entrée, nous avons pu reconstruire la matrice de permutation. Cette attaque par DPA est dite de premier ordre puisqu'elle analyse la consommation de courant pour une seule variable intermédiaire (ici tmp).

Exemple

Pour cette nouvelle attaque, gardons comme longueur de code $n = 1024$. Tout comme pour la SPA, nous avons également choisi d'utiliser tous les textes chiffrés possibles de poids "1" pour la DPA. Nous avons mesuré 500 traces par texte chiffré, afin d'obtenir une moyenne de ces traces par texte chiffré et d'éliminer les parties correspondant au bruit et ne contenant donc aucune information utile pour notre attaque. Un exemple d'acquisition de trace est donné dans la Figure 6.8. Dans toutes les traces, le motif de l'opération critique apparaît à différents instants sur la courbe. Celui-ci est entouré en pointillé dans la Figure 6.8. Cette courbe correspondant au texte chiffré $\tilde{C} = 1\underbrace{00\ldots0}_{1023}$, dont le "1" a été permuté à la position 479 donnant $\tilde{C}_p = \underbrace{00\ldots0}_{478}1\underbrace{00\ldots0}_{545}$. C'est ce texte chiffré permuté qui est utilisé lors du calcul du syndrome et qui donne cette trace de consommation. L'apparition du pic (motif de l'opération critique) dépend directement de la position du "1" dans le texte chiffré permuté utilisé en entrée du calcul du syndrome. Pour retrouver toute la matrice de permutation, il faut retrouver où chaque "1" des textes chiffrés de poids 1 a été permuté en utilisant sa nouvelle position dans le texte chiffré permuté utilisé lors du calcul du syndrome. On peut ainsi reconstruire au fur et à mesure une table de correspondance et ainsi découvrir la matrice de permutation qui était privée.

FIGURE 6.8 – Résultats pour l'attaque par DPA



Contre-mesure

Une technique couramment utilisée comme contre-mesure à une attaque par DPA est le masquage [ZPG13]. Les deux principaux désavantages à l'utilisation de masque comme contre-mesure sont : la nécessité de bits aléatoires et l'augmentation de la complexité calculatoire.
Afin de remédier à la fuite d'information exploitée ici par DPA, nous avons proposé une contre-mesure basée sur les propriétés des codes linéaires. Cela permet d'éviter l'utilisation supplémentaire d'un générateur d'aléa et la complexité n'est pas fondamentalement augmentée. Une augmentation de la complexité est due à la nécessité de séparer les calculs de masquage et de démasquage (dans le cas d'un masque additif de premier ordre, comme dans notre attaque). En revanche, nous avons tout de même besoin d'un masque. L'espace de stockage nécessaire est donc un peu plus grand (n bits supplémentaires).
Les propriétés des codes linéaires mises en avant ici sont les suivantes :

— Les lignes d'une matrice génératrice $\mathcal{G}$ sont des mots de code.
— Tout mot de code a un syndrome nul.

L'idée de notre contre-mesure est d'ajouter un mot de code au texte chiffré au début du déchiffrement. Le texte chiffré sera alors masqué lors de sa permutation et le texte chiffré permuté sera masqué par un mot de code permuté lors du calcul de son syndrome. L'Algorithme 30 détaille cette contre-mesure.

Algorithme 30 Calcul sécurisé de la combinaison de la permutation et du syndrome

Entrée(s): $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré, $\mathcal{P}^{-1} \in \mathscr{M}_{n,n}(\mathbb{F}_2)$ la matrice de permutation inverse privée représentée par une table de recherche $t^{\mathcal{P}^{-1}}$, $\mathcal{G}$ une matrice génératrice du code de Goppa privé $\Gamma(\mathscr{L}, G)$, $\mathcal{H}$ une matrice de contrôle de $\Gamma(\mathscr{L}, G)$ et deux masques M_1, M_2.

Sortie(s): $S \in \mathbb{F}_2^r$ le syndrome.

1: Choisir aléatoirement $B \in \Gamma(\mathscr{L}, G)$ (une combinaison linéaire des lignes de $\mathcal{G}$)
2: $B_p = B \cdot \mathcal{P}$
3: $\tilde{C}' = \tilde{C} \oplus B_p$
4: //Permutation du texte chiffré
5: $\tilde{C}'_p = M_1$
6: **Pour** $i = 1$ **à** n **faire**
7: $\quad tmp = unsigned(0 - \tilde{C}'_i)$
8: $\quad \tilde{C}'_p = \tilde{C}'_p \oplus (tmp \ \& \ (1 \ll t_i^{\mathcal{P}^{-1}}))$
9: **Fin pour**
10: $\tilde{C}'_p = \tilde{C}'_p \oplus M_1$
11: //Syndrome du texte chiffré permuté
12: $S = M_2$
13: **Pour** $i = 1$ **à** n **faire**
14: $\quad tmp = unsigned(0 - \tilde{C}'_{p_i})$
15: $\quad S = S \oplus (tmp \ \& \ H_i^T)$
16: **Fin pour**
17: $S = S \oplus M_2$
18: **Retourner** S

Pour choisir B dans $\Gamma(\mathscr{L}, G)$ via $\mathcal{G}$ (ligne 1), il suffit de prendre une ligne ou une somme de lignes, puisqu'une combinaison linéaire de lignes de $\mathcal{G}$ revient à une simple somme dans $\mathbb{F}_2$. Notons aussi que, comme les lignes de $\mathcal{G}$ forment une base du code linéaire (ici $\Gamma(\mathscr{L}, G)$), B est donc bien un mot de code. De plus, lorsque l'on dit, à la première ligne, "choisir aléatoirement", ce qu'il faut retenir c'est que la construction de B doit varier d'un déchiffrement à l'autre. Alors, bien entendu, une manière plus rapide de le faire est d'utiliser un générateur d'aléa, mais ceci est un autre problème.

À la fin de cet algorithme, on obtient le syndrome d'un texte chiffré permuté masqué :

$$\begin{aligned}
S &= \left(\tilde{C}'_p\right) \cdot {}^t\mathcal{H} \\
&= \left(\tilde{C}' \cdot \mathcal{P}^{-1}\right) \cdot {}^t\mathcal{H} \\
&= \left(\underbrace{(\tilde{C} \oplus B_p)}_{=\tilde{C}'} \cdot \mathcal{P}^{-1}\right) \cdot {}^t\mathcal{H} \\
&= \left(\tilde{C} \cdot \mathcal{P}^{-1} \oplus \underbrace{(B \cdot \mathcal{P})}_{=B_p} \cdot \mathcal{P}^{-1}\right) \cdot {}^t\mathcal{H} \\
&= \left(\underbrace{\tilde{C} \cdot \mathcal{P}^{-1}}_{=\tilde{C}_p} \oplus B\right) \cdot {}^t\mathcal{H} \\
&= \underbrace{\left(\tilde{C}_p \oplus B\right)}_{=\tilde{C}'_p} \cdot {}^t\mathcal{H} \\
&= (\tilde{C}_p \cdot {}^t\mathcal{H}) \oplus \underbrace{(B \cdot {}^t\mathcal{H})}_{=0} \\
&= (\tilde{C}_p \cdot {}^t\mathcal{H}).
\end{aligned}$$

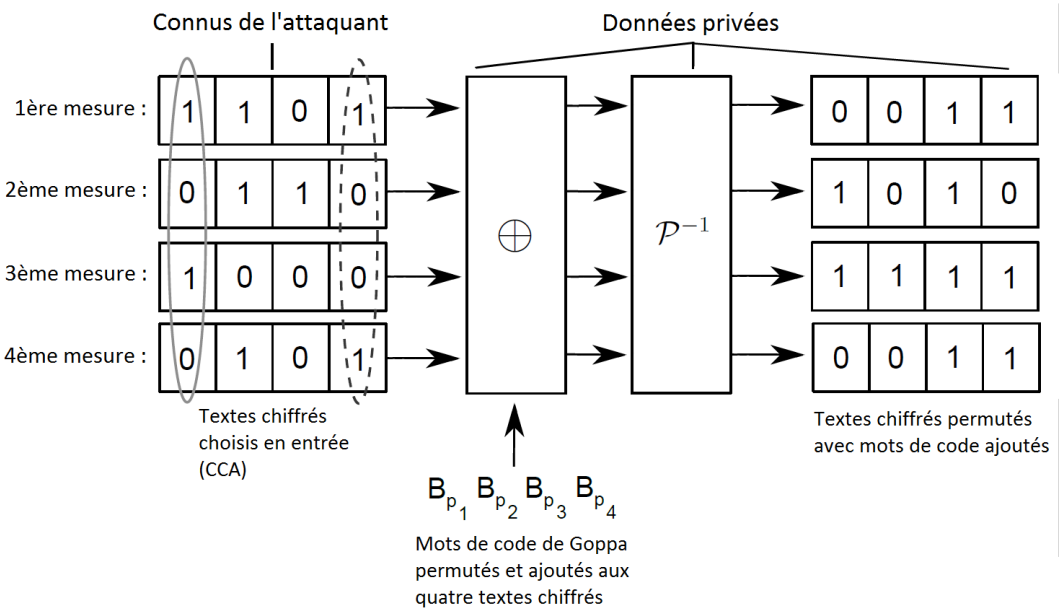
Comme on peut le constater, ce syndrome est le même que pour le texte chiffré permuté non masqué. Ceci intervient au début du décodage du texte chiffré masqué $\tilde{C}'$. A la fin du décodage, de ce dernier, on récupérera, non pas $M \cdot \mathcal{S} \cdot \mathcal{G}$, mais $M \cdot \mathcal{S} \cdot \mathcal{G} \oplus B$. En effet, comme $\tilde{C}' = \tilde{C} \oplus B_p$ et comme $\exists M_B \in \mathbb{F}_2^k$ tel que $B = M_B \cdot \mathcal{S} \cdot \mathcal{G}$ car $B \in \Gamma(\mathscr{L}, G)$, on a plus exactement :

$$\begin{aligned}
\tilde{C}' &= \tilde{C} \oplus B_p \\
&= (M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus E) \oplus (B \cdot \mathcal{P}) \\
&= (M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus B \cdot \mathcal{P}) \oplus E \\
&= (M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus M_B \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P}) \oplus E \\
&= \underbrace{((M \oplus M_B) \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P})}_{\text{mot de code de } \Gamma(\mathscr{L},G)} \oplus \underbrace{E}_{\substack{\text{vecteur erreur} \\ \text{de poids au plus } t}}
\end{aligned}$$

Ce qui signifie qu'avec le texte chiffré masqué, on arrive à corriger l'erreur E tout comme le texte chiffré non masqué. Le vecteur erreur reste donc le même pour les deux versions. Il sera ainsi bien détecté et corrigé lors du décodage après utilisation de cette contre-mesure.
L'étape restante à ne pas oublier est le démasquage. Il faut donc stocker le B choisi pour pouvoir démasquer le mot de code obtenu après le décodage et ainsi pouvoir poursuivre le déchiffrement normalement : $(M \cdot \mathcal{S} \cdot \mathcal{G} \oplus B) \oplus B = M \cdot \mathcal{S} \cdot \mathcal{G}$.
Si cette étape de démasquage est omise, on se retrouvera alors dans le cas où le message en clair sera encore masqué. Avec $B = M_B \cdot \mathcal{S} \cdot \mathcal{G}$, on obtiendrait donc $(M \oplus M_B) \cdot \mathcal{S} \cdot \mathcal{G}$ après le décodage, qui donnerait $\tilde{M} = (M \oplus M_B) \cdot \mathcal{S}$ et enfin comme "message en clair" $M \oplus M_B$. Ce mot serait valide mais ne correspondrait pas au texte chiffré reçu. Le principe de cette contre-mesure est représenté schématiquement dans la Figure 6.9. Du point de vue d'un attaquant, nous ajoutons des valeurs

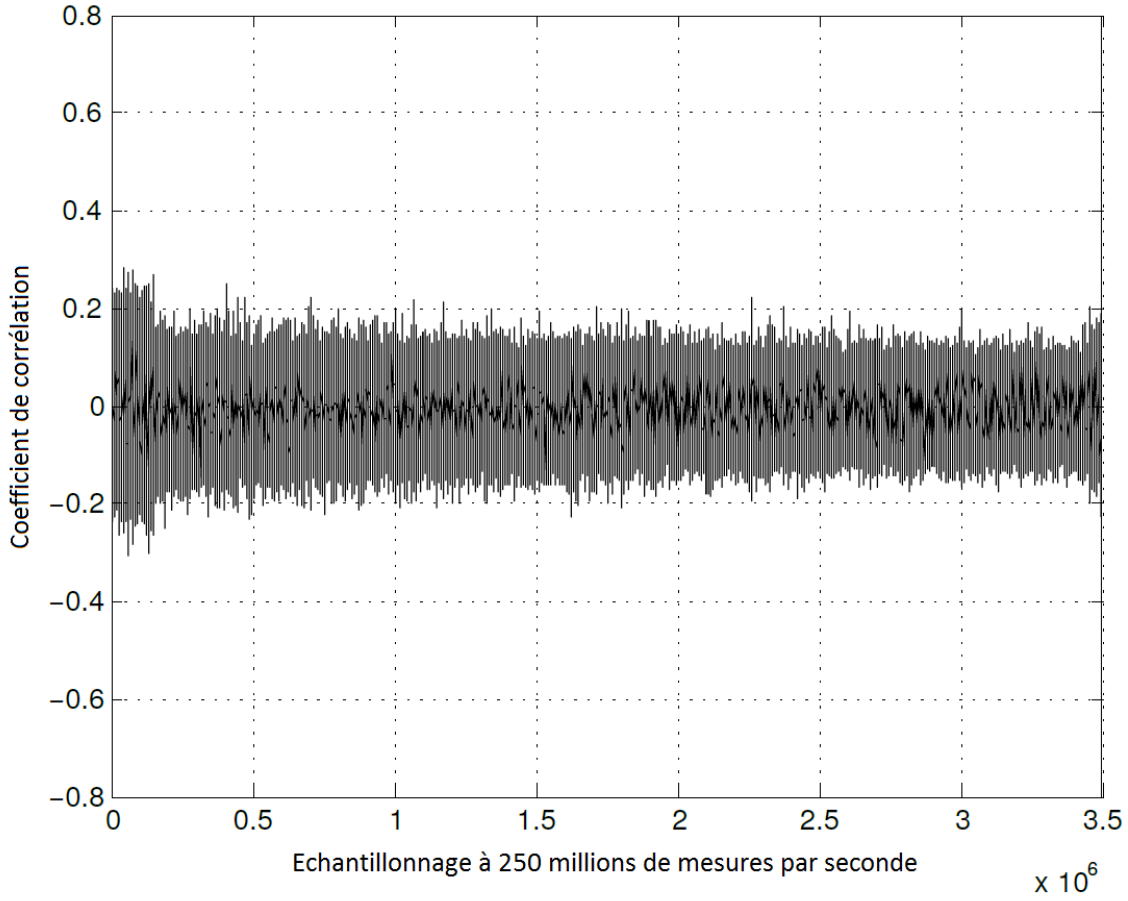
aléatoires aux textes chiffrés avec les additions modulo 2. L'attaquant ne peut donc pas distinguer les différentes valeurs de *tmp* dans l'Algorithme 30. La raison est que les hypothèses que l'attaquant pourrait faire sont complètement altérées par l'addition du masque B_p au texte chiffré $\tilde{C}$, comme dans la Figure 6.9.

FIGURE 6.9 – Principe de la contre-mesure de la DPA



Pour finir sur l'Algorithme 30, les deux masques servent à éviter l'effet présenté dans la partie 2 de la contre-mesure pour la SPA. Comme le calcul de la permutation du texte chiffré et le calcul du syndrome du texte chiffré permuté sont en fait deux produits matrice-vecteur, cela explique pourquoi les deux boucles **Pour** se ressemblent tant et pourquoi il faut deux masques différents (un par calcul). Pour l'exemple de la DPA donné précédemment, la trace de consommation après contre-mesure est donnée dans la Figure 6.10.

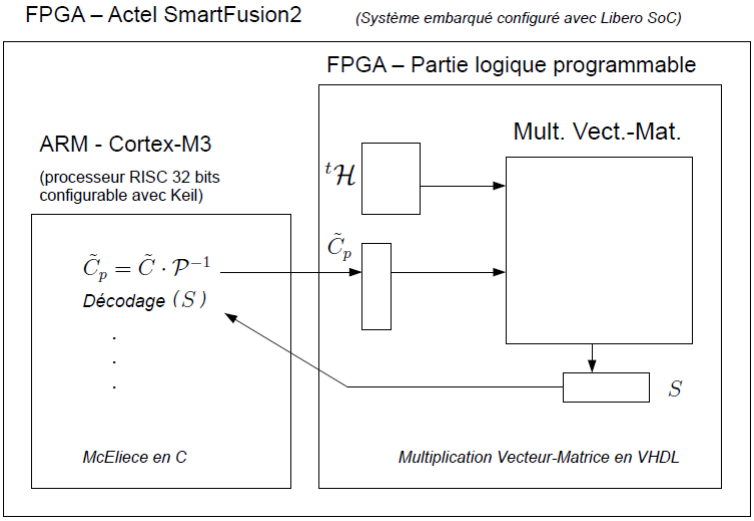
FIGURE 6.10 – Exemple pour la contre-mesure de la DPA



6.3 Conclusion et perspectives

Nous pensions au départ que la contre-mesure proposée dans [STM+08, Algorithme 3] pour une attaque sur la mémoire cache pouvait également être appliquée dans ce contexte, mais nous avons montré (c.f. Chapitre 5) que cet algorithme était lui-même la cible d'une attaque par canal auxiliaire.
Comme poursuite de ces travaux, voici quelques pistes envisagées :
Une poursuite de ce travail, pour passer d'une SPA à une DPA (Differential Power Analysis) a d'abord été envisagée sur la carte SmartFusion2, comme présenté sur la Figure 6.11. Mais il s'est avéré que cette carte n'était pas prévue pour ce type d'analyse, puisque la consommation de courant que l'on voulait observer était noyée dans le bruit causé par la consommation de la carte dans sa globalité. Une attaque par DPA a ensuite pu être réalisée sur le même matériel que pour la SPA (comme présenté dans la Sous-section 6.2.3).

FIGURE 6.11 – Schéma de la carte SmartFusion2 pour les tests de DPA



Les attaques présentées ici (et a fortiori leurs contre-mesures associées) ne s'appliquent que dans deux des quatre profils énoncés dans [HMP10] (les profils II et IV). Il faudrait regarder d'un point de vue canal auxiliaire ces deux autres profils afin de compléter l'étude.
De plus, l'attaque par SPA proposée ici, tout comme celle dans [HMP10], repose sur une CCA : les textes chiffrés pris en entrée sont tous les mots possibles de poids "1". Qu'en est-il pour les mots de poids supérieur ?
Même remarque que dans le Chapitre 5, il serait intéressant de regarder si la contre-mesure pour la DPA reste efficace pour une attaque différentielle par analyse de consommation d'ordre supérieur, autrement dit en analysant la consommation de courant à différents instants, via non pas une mais plusieurs variables intermédiaires. Une possibilité serait de combiner deux attaques par consommation sur deux étapes distinctes du déchiffrement.
Enfin l'attaque présentée de ce chapitre a été réalisée contre un système embarqué. L'implantation était logicielle. Il faudrait donc tester cette attaque contre une implantation matérielle, autrement dit contre une implantation sur un circuit logique programmable (*Field-Programmable Gate Array* (FPGA) en Anglais).

Chapitre 7

Attaque temporelle contre l'algorithme d'Euclide étendu

Ce chapitre correspond aux travaux effectués pour un article en cours de rédaction :

Improved Timing Attacks against the Secret Permutation in the McEliece PKC
avec Vlad Dragoi, Pierre-Louis Cayrel et Viktor Fischer
en préparation.

7.1 Double appel de l'algorithme d'Euclide étendu

Dans ce chapitre nous présentons une attaque temporelle dont la cible est le double appel à l'algorithme d'Euclide étendu (*Extended Euclidean Algorithm* (EEA) en Anglais) effectué lors du décodage par l'algorithme de Patterson. L'idée générale consiste à utiliser simultanément deux attaques proposées dans [Str10b, Str13b] ne permettant pas de récupérer toute la permutation lorsqu'elles sont employées individuellement. À l'aide d'une nouvelle relation sur le nombre d'étapes de l'algorithme d'Euclide étendu lors de ces deux appels, nous montrons qu'il est possible d'étendre le système d'équations obtenu dans les travaux précédents jusqu'à pouvoir récupérer l'information privée recherchée : la matrice de permutation.
Avant de présenter l'attaque, nous rappellons les grandes lignes du déchiffrement de McEliece dans l'Algorithme 31.

Algorithme 31 Déchiffrement de McEliece

Entrée(s): $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \Gamma)$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré, où $\tilde{C} = M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus E$.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair associé à $\tilde{C}$.
1: Calculer $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$.
2: Décoder $\tilde{C}_p$ pour retrouver $M \cdot \mathcal{S} \cdot \mathcal{G}$.
3: Récupérer $\tilde{M} = M \cdot \mathcal{S}$ à partir de $M \cdot \mathcal{S} \cdot \mathcal{G}$.
4: Calculer $M = \tilde{M} \cdot \mathcal{S}^{-1}$.
5: **Retourner** M.

Le décodage du texte chiffré permuté $\tilde{C}_p$ pour retrouver le mot de code $M \cdot \mathcal{S} \cdot \mathcal{G}$ peut être fait de différentes façons. Celle qui nous intéresse a été proposée par Nicholas Patterson en 1975 [Pat75], présentée dans l'Algorithme 32. De manière générale, cet algorithme prend en entrée un mot de code erroné, avec au plus t erreurs, où t est la capacité de correction du code de Goppa binaire. Cependant, comme nous faisons appel à cet algorithme dans le déchiffrement de McEliece, notre mot de code erroné est de la forme :

$$\tilde{C}_p = \underbrace{M \cdot \mathcal{S} \cdot \mathcal{G}}_{\text{mot de code}} \oplus \underbrace{E \cdot \mathcal{P}^{-1}}_{\text{erreur de poids } t} .$$

Ce mot de code erroné est le texte chiffré permuté tel que $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$, avec $w_H(E \cdot \mathcal{P}^{-1}) \leqslant t$ et $\Gamma(\mathscr{L}, G)$ le code de Goppa t-correcteur, où $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ est le support du code et G est le polynôme de Goppa.

Algorithme 32 Décodage d'un code de Goppa par Patterson

Entrée(s): $\tilde{C}_p$ un mot de code erroné avec au plus t erreurs et le code de Goppa $\Gamma(\mathscr{L}, G)$, avec $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ le support du code et G le polynôme de Goppa de degré t.

Sortie(s): $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$ le mot de code sans erreur.

1: Calculer le polynôme syndrome de $\tilde{C}_p$:
$S_{\tilde{C}_p}(X) = \tilde{C}_p \cdot {}^t\mathcal{H} \cdot {}^t(X^{t-1}, X^{t-2}, \ldots, X, 1)$.
2: **Si** $S_{\tilde{C}_p}(X) = 0$ **Alors**
3: $\sigma(X) = 1$
4: **Sinon**
5: Inverser le polynôme syndrome : $T(X) = S^{-1}_{\tilde{C}_p}(X) \mod G(X)$. (EEA)
6: **Si** $T(X) = X$ *c'est-à-dire* $R(X) = 0$ **Alors**
7: $\sigma(X) = X$
8: **Sinon**
9: Calculer $s(X) = \sqrt{X} \mod G(X)$.
(En posant $s(X) = X^{2^{m-1}} \mod G(X)$*.)*
10: Calculer $R(X) = \sqrt{T(X) + X} \mod G(X)$.
En posant $h(X) = T(X) + X$*, on utilise la formule suivante pour obtenir* $R(X) = \sqrt{h(X)} \mod G(X)$ *:*
$R(X) = \sum_{i=0}^{(t-1)/2} h_{2i}^{2^{m-1}} \cdot X^i + \sum_{i=0}^{t/2-1} h_{2i+1}^{2^{m-1}} \cdot X^i \cdot s(X)$,
où $s(X) = \sqrt{X} \mod G(X)$.
11: Trouver $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ tels que $\mathfrak{a}(X) = \mathfrak{b}(X) \cdot R(X) \mod G(X)$, (EEA)
avec $\deg(\mathfrak{a}) \leqslant \frac{t}{2}$ et $\deg(\mathfrak{b}) \leqslant \frac{t-1}{2}$.
12: Construire le polynôme $\sigma(X)$ tel que : $\sigma(X) = \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X)$.
13: **Fin si**
14: **Fin si**
15: Construire le vecteur erreur E_p tel que $e_{p_i} = 1$ si $\sigma_{E \cdot \mathcal{P}^{-1}}(\alpha_i) = 0$ et $e_{p_i} = 0$ sinon.
16: **Retourner** $M \cdot \mathcal{S} \cdot \mathcal{G} = \tilde{C}_p \oplus E_p$.

Nous présentons l'algorithme d'Euclide étendu (EEA), cible de notre attaque, dans l'Algorithme 33.

Algorithme 33 Algorithme d'Euclide étendu

Entrée(s): Les polynômes $a(X)$ et $b(X)$ appartenant à $\mathbb{F}_{2^m}[X]$ avec $b(X) \neq 0$ et $\deg(a) \geqslant \deg(b)$.

Sortie(s): Le pgcd $d(X)$ de $a(X)$ et $b(X)$ et deux polynômes $u(X)$ et $v(X)$ tels que $d(X) = u(X) \cdot a(X) + v(X) \cdot b(X)$ avec $\deg(d) < \deg(b)$.

$r_{-1} \leftarrow a(X)$
$r_0 \leftarrow b(X)$
$u_0 \leftarrow 1$
$u_1 \leftarrow 0$
$v_0 \leftarrow 0$
$v_1 \leftarrow 1$
$i \leftarrow 1$
Tant que $r_i \neq 0$ **faire**
  $q_{i+1}(X) \leftarrow quotient(r_{i-1}(X), r_i(X))$
  $r_{i+1}(X) \leftarrow r_{i-1}(X) - q_{i+1}(X).r_i(X)$
  $u_{i+1}(X) \leftarrow u_{i-1}(X) - q_{i+1}(X).u_i(X)$
  $v_{i+1}(X) \leftarrow v_{i-1}(X) - q_{i+1}(X).v_i(X)$
  $i \leftarrow i + 1$
Fin tant que
$i \leftarrow i - 1$
Retourner $(r_i(X), u_i(X), v_i(X))$.

Nous présentons dans les sous-sections suivantes les deux étapes où EEA est utilisé (en rouge dans l'Algorithme 32), afin de mettre en avant leurs éventuelles vulnérabilités.

7.1.1 Inversion modulaire du polynôme syndrome

La correction d'erreurs en utilisant le syndrome est une technique assez courante en théorie des codes. Pour les codes de Goppa binaires, on peut remarquer qu'en utilisant l'algorithme de Patterson (Algorithme 32), il faut inverser le polynôme syndrome modulo le polynôme de Goppa si des erreurs sont à corriger (autrement dit si le syndrome est non nul).
Rappelons que le polynôme de Goppa $G(X) \in \mathbb{F}_{2^m}[X]$ est irréductible et de degré t. Le polynôme syndrome $S_{\tilde{C}_p}(X) \in \mathbb{F}_{2^m}[X]$ est, quant à lui, de degré au plus $t - 1$. Cela implique que ces deux polynômes sont premiers entre eux et que l'on peut donc obtenir l'inverse de $S_{\tilde{C}_p}(X)$ modulo $G(X)$ grâce à l'algorithme d'Euclide étendu. Ceci est illustré dans l'Algorithme 34.

Algorithme 34 Algorithme d'Euclide étendu version binaire pour calculer l'inverse [Str13b, Algo 3]

Entrée(s): $S_{\tilde{C}_p}(X)$ le polynôme syndrome de $\tilde{C}_p$ et $G(X)$ le polynôme de Goppa.
Sortie(s): $T(X) = S^{-1}_{\tilde{C}_p}(X) \mod G(X)$ l'inverse du polynôme syndrome modulo le polynôme de Goppa.

1: $d \leftarrow \deg(S_{\tilde{C}_p})$
2: $\mathfrak{b}_{-1} \leftarrow 0$
3: $\mathfrak{b}_0 \leftarrow 1$
4: $r_{-1} \leftarrow G(X)$
5: $r_0 \leftarrow S_{\tilde{C}_p}(X)$
6: $i \leftarrow 0$
7: **Tant que** $\deg(r_i(X)) > d$ **faire**
8: $\quad i \leftarrow i + 1$
9: $\quad (q_i(X), r_i(X)) \leftarrow r_{i-2}(X)/r_{i-1}(X)$
10: $\quad$ //(division polynomiale avec comme quotient q_i et reste r_i)
11: $\quad \mathfrak{b}_i(X) \leftarrow \mathfrak{b}_{i-2}(X) + q_i(X) \cdot \mathfrak{b}_{i-1}(X)$
12: **Fin tant que**
13: $T(X) \leftarrow \mathfrak{b}_i(X)$
14: **Retourner** $T(X)$.

D'après l'Équation (2.9), nous savons que :

$$S_{\tilde{C}}(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \mod G(X).$$

Cela signifie qu'en connaissance du nombre d'erreurs, les degrés des polynômes $\sigma(X)$ et $\eta(X)$ peuvent respectivement être déterminés. De plus, le nombre d'itérations dans l'Algorithme 34 est majoré par le nombre d'erreurs [SKHN75, Cor. 1]. On peut d'ores et déjà en déduire que si un attaquant choisit une erreur de poids plus petite que t, l'exécution de l'Algorithme 34 s'effectuera plus rapidement.

7.1.2 Détermination du polynôme localisateur d'erreurs

On souhaite construire le polynôme localisateur d'erreurs (PLE) en utilisant la formule suivante :

$$\sigma(X) = \mathfrak{a}^2(X) \oplus X \cdot \mathfrak{b}^2(X) \tag{7.1}$$

Étant donné que le code est t-correcteur, on aura $\deg(\sigma) = \varepsilon \leqslant t$. Deux cas de figure sont possibles :

Si $\deg(\sigma) = \varepsilon$ **est pair :**

$$\begin{aligned}
\mathfrak{a}(X) &= \sigma_0 \oplus \sigma_2 \cdot X \oplus \ldots \oplus \sigma_\varepsilon \cdot X^{\varepsilon/2} \\
\mathfrak{b}(X) &= \sigma_1 \oplus \sigma_3 \cdot X \oplus \ldots \oplus \sigma_{\varepsilon-1} \cdot X^{\lfloor(\varepsilon-1)/2\rfloor} \\
\sigma(X) &= \sigma_0 \oplus \sigma_1 \cdot X \oplus \ldots \oplus \sigma_{\varepsilon-1} \cdot X^{\varepsilon-1} \oplus \sigma_\varepsilon \cdot X^\varepsilon
\end{aligned}$$

Sinon, $\deg(\sigma) = \varepsilon$ **est impair :**

$$\begin{aligned}
\mathfrak{a}(X) &= \sigma_0 \oplus \sigma_2 \cdot X \oplus \ldots \oplus \sigma_\varepsilon \cdot X^{(\varepsilon-1)/2} \\
\mathfrak{b}(X) &= \sigma_1 \oplus \sigma_3 \cdot X \oplus \ldots \oplus \sigma_\varepsilon \cdot X^{\lfloor\varepsilon/2\rfloor} \\
\sigma(X) &= \sigma_0 \oplus \sigma_1 \cdot X \oplus \ldots \oplus \sigma_{\varepsilon-1} \cdot X^{\varepsilon-1} \oplus \sigma_\varepsilon \cdot X^\varepsilon
\end{aligned}$$

On obtient les polynômes $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ donnant l'Équation (7.1) en appliquant l'Algorithme 33 à $G(X)$ et $R(X)$. Voyons la version proposée dans [Str13b, Algo 3], rappelée ici dans l'Algorithme 35.

Algorithme 35 Algorithme d'Euclide étendu pour calculer les deux parties du PLE [Str10b, Algo 3]

Entrée(s): Le polynôme $R(X)$ et le polynôme de Goppa privé $G(X)$, avec $\deg(R(X)) < \deg(G(X))$.

Sortie(s): Deux polynômes $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ satisfaisant $\mathfrak{a}(X) = \mathfrak{b}(X) \cdot R(X) \bmod G(X)$ et $\deg(\alpha) \leqslant \lfloor \deg(G)/2 \rfloor$.

1: $d \leftarrow \lfloor \deg(G)/2 \rfloor = \lfloor t/2 \rfloor$
2: $\mathfrak{b}_{-1} \leftarrow 0$
3: $\mathfrak{b}_0 \leftarrow 1$
4: $r_{-1} \leftarrow G(X)$
5: $r_0 \leftarrow R(X)$
6: $i \leftarrow 0$
7: **Tant que** $\deg(r_i(X)) > d$ **faire**
8: $\quad i \leftarrow i + 1$
9: $\quad (q_i(X), r_i(X)) \leftarrow r_{i-2}(X)/r_{i-1}(X)$
10: $\quad$ //(division polynomiale avec comme quotient q_i et reste r_i)
11: $\quad \mathfrak{b}_i(X) \leftarrow \mathfrak{b}_{i-2}(X) + q_i(X) \cdot \mathfrak{b}_{i-1}(X)$
12: **Fin tant que**
13: $\mathfrak{a}(X) \leftarrow r_i(X)$
14: $\mathfrak{b}(X) \leftarrow \mathfrak{b}_i(X)$
15: **Retourner** $(\mathfrak{a}(X), \mathfrak{b}(X))$

On peut d'ores et déjà remarquer que le nombre d'itérations dans l'Algorithme 35 dépend clairement de la condition de la boucle **Tant que**, et donc a fortiori de la limite sur les degrés des polynômes dans la suite des restes.

7.2 Attaque temporelle

Commençons par rappeler que l'objectif est de retrouver la matrice de permutation $\mathcal{P}$. Il a été montré dans divers travaux antérieurs qu'il était possible d'obtenir des informations sur les éléments du support $\mathscr{L}$ à partir du nombre d'itérations dans EEA. Falko Strenzke a en effet proposé d'attaquer de façon indépendante chacun des deux appels à l'algorithme d'Euclide étendu dans [Str10b, Str13b]. Cependant, une telle approche ne permet de récuperer la matrice de permutation que lorsque le poids de Hamming de l'erreur est petit (2 ou 4 dans [Str10b] et 2, 4 ou 6 dans [Str13b]). Après avoir rappelé les grands principes et limitations de ces attaques, nous verrons comment l'étendre jusqu'au cas où $w_H(E) = t/2$, où $t = \deg(G)$.

7.2.1 Attaques de Strenzke

1ère attaque [Str10b]

Principe : Cette attaque repose sur la capacité d'un attaquant à produire des textes chiffrés lui-même (attaque à chiffrés choisis), ce qui ne pose aucun problème puisque nous sommes dans le cadre d'un chiffrement asymétrique. L'attaquant a donc un contrôle total sur le nombre d'erreurs (et donc le vecteur erreur ajouté au

mot de code durant la phase de chiffrement). En particulier, les erreurs seront de poids ε pour cette attaque, où $\varepsilon < t$. En particulier, ε sera fixé à 4 si rien n'est précisé. Pour $\varepsilon = 4$, on a $\deg(\sigma) = 4$, donc $\deg(\mathfrak{a}) = 2$ et $\deg(\mathfrak{b}) \leqslant 1$. Comme $\varepsilon = 4$ est pair, $\mathfrak{a}(X)$ contient le coefficient dominant de $\sigma(X)$. De plus, puisque $\deg(\mathfrak{b}) \leqslant 1$, deux cas sont possibles :

Si $\deg(\mathfrak{b}) = 0$ **:** alors le nombre d'itérations dans EEA est 0.

Si $\deg(\mathfrak{b}) = 1$ **:** alors le nombre d'itérations dans EEA est 1.

Cette différence dans le nombre d'itérations implique une différence dans les temps d'exécution de l'algorithme.

Attaque : Cette fuite d'information peut être exploitée par un attaquant. Rappelons que, pour $\varepsilon = 4$, on a :

$$\sigma(X) = \sigma_4 \cdot X^4 \oplus \sigma_3 \cdot X^3 \oplus \sigma_2 \cdot X^2 \oplus \sigma_1 \cdot X \oplus \sigma_0,$$

avec :

$$\begin{aligned} \mathfrak{a}(X) &= \sigma_4 \cdot X^2 \oplus \sigma_2 \cdot X \oplus \sigma_0 \\ \mathfrak{b}(X) &= \sigma_3 \cdot X \oplus \sigma_1 \end{aligned} \tag{7.2}$$

Revenons sur les deux cas possibles. Cela signifie que :

Si $\deg(\mathfrak{b}) = 0$ **:** alors $\sigma_3 = 0$.

Si $\deg(\mathfrak{b}) = 1$ **:** alors $\sigma_3 \neq 0$.

Le canal auxiliaire provient donc directement du coefficient σ_3. Or, le coefficient σ_3 n'est ni plus ni moins que la somme des éléments dans $\mathscr{L}$ correspondant aux positions des erreurs dans E. En d'autres termes, si $e_i = 1$ pour $i \in [\![1, n]\!]$, alors α_i est une racine de $\sigma(X)$. Et comme σ_3 est égal à la somme des racines de $\sigma(X)$, on peut en déduire une relation sur les éléments du support correspondant aux erreurs. On sait déjà qu'il y a 4 erreurs par rapport au degré du PLE. Posons $i_1, i_2, i_3, i_4 \in [\![1, n]\!]$ les indices des erreurs, deux à deux distincts. Comme les positions des erreurs choisies par l'attaquant sont dans un ordre dépendant de la permutation privée $\mathcal{P}$, on a alors :

Si $\deg(\mathfrak{b}) = 0$ **:** $\sigma_3 = \alpha_{\mathcal{P}(i_1)} \oplus \alpha_{\mathcal{P}(i_2)} \oplus \alpha_{\mathcal{P}(i_3)} \oplus \alpha_{\mathcal{P}(i_4)} = 0$.

Si $\deg(\mathfrak{b}) = 1$ **:** $\sigma_3 = \alpha_{\mathcal{P}(i_1)} \oplus \alpha_{\mathcal{P}(i_2)} \oplus \alpha_{\mathcal{P}(i_3)} \oplus \alpha_{\mathcal{P}(i_4)} \neq 0$.

En générant aléatoirement des textes chiffrés ayant une erreur de poids 4, l'attaquant peut ainsi obtenir une liste d'équations impliquant la permutation privée avec les temps de déchiffrement correspondants afin d'en déduire dans quel cas il se trouve à chaque essai. Notons que les équations qui l'intéressent le plus sont celles où la somme est nulle, car elles apportent davantage d'informations exploitables par la suite. Après quoi, l'attaquant peut appliquer l'algorithme du pivot de Gauss sur son système. Si, comme supposé dans cette attaque, l'ordre des éléments à l'origine est connu, l'attaquant peut plus facilement retrouver la permutation privée.
Par contre, une question ne semble pas avoir été traitée dans cet article : Pourquoi le rang maximal de ce système semble être donné par $n - m - 1$, où n est la longueur du code et m le degré de l'extension du corps ?

Contre-mesure : Comme l'attaque dépend des différents cas possibles pour les degrés des polynômes $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$, sorties de l'Algorithme 35, l'idée est de rendre l'entrée $R(X)$ "constante", puisque le polynôme $G(X)$ lui ne change pas d'un déchiffrement à l'autre pour un même couple de clés privée/publique. La proposition faite par l'auteur est donc de manipuler le degré du polynôme $R(X)$ afin que la suite des restes n'atteigne pas trop vite la condition de la boucle **Tant que**. Cela permet d'éviter qu'il n'y ait aucune itération, origine de l'attaque proposée dans [Str10b]. Le test "**Si** $\deg(R) < d$" doit être fait juste après que $R(X)$ ait été calculé dans l'algorithme de Patterson (étape 10 de l'Algorithme 32). Si tel est le cas, l'auteur suggère de "manipuler" $R(X)$ de sorte que $\deg(R) = t-1$. Après demande de détails auprès de celui-ci, il a rajouté que tout polynôme r_i satisfaisant $\deg(r_i) = \deg(r_i - 1) - 1$ ferait l'affaire, puisque l'Algorithme 35 continuerait son déroulement et l'effet critique du temps d'exécution ne serait plus observable. Cela repose sur le fait que même pour les cas non dégénérés produits par un attaquant, l'événement $\deg(R) < d$ ne se produit jamais. Une recommandation est que cette manipulation ne doit pas se faire en utilisant un vrai générateur de nombres aléatoires, mais un générateur de nombres pseudo-aléatoires ayant pour source le texte chiffré afin que l'attaquant ne puisse pas déterminer lorsque la contre-mesure est appliquée ou non.

Améliorations : Il est intéressant de noter que pour $\varepsilon = 4$, en notant $i_1, i_2, i_3, i_4 \in [\![1, n]\!]$ les indices des erreurs, deux à deux distincts, la somme des éléments du support $\mathscr{L}$ ayant ces indices s'annule avec une probabilité $1/(n-3)$ en caractéristique 2. Ce type de réflexion est présenté dans le Chapitre 8.
L'auteur énonce également une possibilité d'adaptation de cette attaque à une analyse de consommation en utilisant le fait que pour $\varepsilon = 4$, si $\deg(\mathfrak{b}) = 0$, alors $\mathfrak{a}(X) = R(X)$, donc $\deg(R) = 2$, tandis que si $\deg(\mathfrak{b}) = 1$, alors $\deg(R) \leqslant t-1$. Ceci entraîne un plus grand nombre de changements d'états des registres dans un composant lors du calcul de $R(X)$ si les registres sont initialisés à zéro et sont ensuite mis à jour pour stocker le résultat. Cette information basée sur le poids de Hamming des registres peut permettre de distinguer lorsque peu de coefficients de $R(X)$ sont nuls (lorsque $\deg(R) = 2$) du cas où beaucoup de ses coefficients sont non nuls (lorsque $\deg(R)$ est proche de $t-1$). D'autres précisions sont à apporter. Toujours d'après l'auteur, la contre-mesure proposée dans [SSMS10] permettrait non pas de distinguer les cas 0 itération dans EEA contre 1 pour une erreur de poids 4, mais aucune itération contre t. Ceci faciliterait le travail d'un attaquant.
De plus, la conversion CCA2 [KI01, Poi00] ne permettrait pas de stopper cette attaque, car un attaquant peut mettre moins de t erreurs dans son texte chiffré en ne respectant pas les règles de sécurité du protocole. Malheureusement ce défaut du poids de l'erreur ne peut être détecter par le périphérique de déchiffrement qu'après avoir déterminé le PLE. (Toutefois, notons que si un moyen était découvert pour savoir, avant la détermination du PLE, le poids réel de l'erreur ε en tant que degré du PLE, ce serait très utile pour le schéma de signature CFS [CFS01].) L'attaquant peut donc tout de même récupérer le temps de déchiffrement grâce au message d'erreur qui lui serait renvoyé dans ce cas-là. Ce temps d'exécution dépendrait également du nombre d'itérations dans EEA. L'attaque fonctionnerait donc encore.
Enfin, on peut se poser la question de ce qu'il se passe lorsque le poids de l'erreur ε est choisi plus grand que 4 mais plus petit que t. Tout d'abord, si ε est impair, c'est $\mathfrak{b}(X)$ qui possède le coefficient dominant du PLE $\sigma(X)$ tandis que si ε est pair, c'est $\mathfrak{a}(X)$ qui possède le coefficient dominant du PLE $\sigma(X)$. Cela implique que dans le cas où ε est impair, le nombre d'itérations dans EEA, qui est lié à $\deg(\mathfrak{b})$, dépend

directement de ε et l'attaque devient donc inutilisable. Il faut donc s'intéresser aux cas où ε est pair pour mener le même type d'attaque. Cependant, les relations entre les racines qui sont exploitées pour le cas $\varepsilon = 4$ ne sont plus aussi simples. L'attaque devient donc très compliquée puisque les causes de variations dans le nombre d'itérations se multiplient. Néanmoins, si un attaquant est capable de distinguer les différentes causes de changement dans le nombre d'itérations, peut-être par une analyse de consommation, l'attaque peut encore fonctionner.

2nde attaque [Str13b]

Principe : Cette attaque repose sur l'analyse de l'*équation-clé* (Équation (2.9)) pour en déduire des relations entre les degrés des polynômes intervenant dans celle-ci, à savoir $S_{\tilde{C}}(X)$, $\sigma_E(X)$, $\eta_E(X)$ et $G(X)$, sachant que les degrés de $\sigma_E(X)$ et $\eta_E(X)$ dépendent directement du poids de l'erreur. Comme un attaquant peut produire des textes chiffrés lui-même (attaque à chiffrés choisis), il peut choisir le poids de l'erreur ajoutée au mot de code lors du chiffrement.
Rappelons que le polynôme syndrome vérifie les relations suivantes :

$$\begin{aligned} S_{\tilde{C}}(X) &\equiv \sum_{i=1}^{n} \frac{\tilde{c}_i}{X \oplus \alpha_i} \mod G(X) \\ &\equiv \frac{\eta_E(X)}{\sigma_E(X)} \mod G(X). \end{aligned}$$

Falko Strenzke, auteur de [Str13b], déclare que le nombre maximal d'itérations dans l'Algorithme 34 est majoré par $\deg(\sigma_E) + \deg(\eta_E)$. Cette majoration dépend directement du poids de l'erreur, contrôlable par un attaquant.
Cependant, cette majoration semble être large, par rapport au résultat précédemment mentionné dans [SKHN75, Cor. 1], à savoir que ce nombre est majoré par $\deg(\sigma_E)$.
Cette remarque étant faite, revenons à l'expression de $S_{\tilde{C}}(X)$ en fonction de $\sigma_E(X)$ et $\eta_E(X)$:

$$S_E(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \equiv \sum_{\substack{i=1 \\ e_i=1}}^{n} \frac{1}{X \oplus \alpha_i} \mod G(X). \tag{7.3}$$

Attaque : L'intérêt, pour cette attaque, se porte également sur le coefficient correspondant à la somme des racines. Notons $\varepsilon = w_H(E)$. Pour le cas où $\varepsilon = 4$, l'Équation (7.3) devient :

$$S_E(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \equiv \sum_{\substack{i=1 \\ e_i=1}}^{n} \frac{1}{X \oplus \alpha_i} \equiv \frac{\sigma_3 \cdot X^2 \oplus \sigma_1}{X^4 \oplus \sigma_3 \cdot X^3 \oplus \sigma_2 \cdot X^2 \oplus \sigma_1 \cdot X \oplus \sigma_0} \mod G(X).$$

Posons $i_1, i_2, i_3, i_4 \in [\![1, n]\!]$ les indices des erreurs, deux à deux distincts. On a alors :

$$\sigma_3 = \alpha_{i_1} \oplus \alpha_{i_2} \oplus \alpha_{i_3} \oplus \alpha_{i_4}.$$

Deux cas sont alors possibles :

- **Si** $\sigma_3 = 0$ **:** alors $\deg(\eta_E) = 0$, donc le nombre maximal d'itérations dans EEA vaut 4.
- **Si** $\sigma_3 \neq 0$ **:** alors $\deg(\eta_E) = 2$, donc le nombre maximal d'itérations dans EEA vaut 6.

Cette différence dans le nombre d'itérations implique une différence dans les temps d'exécution de l'algorithme, d'autant plus que cette borne supérieure est atteinte dans la plupart des cas.
La contre-mesure proposée dans [Str10b] est insuffisante pour l'attaque précédemment expliquée. La combinaison de ces deux attaques permet de distinguer le cas où $\deg(\eta_E) = 0$ de celui où $\deg(\eta_E) \neq 0$. Cela permet, comme dans le cas de l'attaque précédente, d'obtenir des équations de la forme :

$$\sigma_3 = \alpha_{i_1} \oplus \alpha_{i_2} \oplus \alpha_{i_3} \oplus \alpha_{i_4} = 0.$$

Améliorations : La remarque sur la parité de ε faite dans le cadre de l'attaque précédente est également valable dans le cas présent.

Le cas où $\varepsilon = 6$: Pour le cas où $\varepsilon = 6$, l'Équation (7.3) devient :

$$S_E(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \equiv \frac{\sigma_5 \cdot X^4 \oplus \sigma_3 \cdot X^2 \oplus \sigma_1}{X^6 \oplus \sigma_5 \cdot X^5 \oplus \sigma_4 \cdot X^4 \oplus \sigma_3 \cdot X^3 \oplus \sigma_2 \cdot X^2 \oplus \sigma_1 \cdot X \oplus \sigma_0} \mod G(X).$$

Afin d'obtenir quelque chose de semblable à l'attaque pour $\varepsilon = 4$, c'est-à-dire pouvoir distinguer le cas où $\deg(\eta_E) = 0$ de celui où $\deg(\eta_E) \neq 0$, il faut que σ_3 soit nul mais également σ_5. Cette fois-ci, c'est le coefficient σ_5 qui est la somme des racines et σ_3 est la somme des produits de trois racines. Remarquons que les cas où $\sigma_5 = 0$ et $\sigma_3 = 0$ simultanément pour l'attaque avec $\varepsilon = 6$ font partie des cas où $\sigma_3 = 0$ pour l'attaque avec $\varepsilon = 4$. L'attaquant peut donc se servir des équations qu'il a déjà obtenues afin d'en obtenir de nouvelles avec 6 racines. Posons $i_1, i_2, i_3, i_4, i_5, i_6 \in [\![1, n]\!]$ les indices des erreurs, deux à deux distincts. Ces nouvelles équations seront cubiques pour σ_3, c'est-à-dire de la forme :

$$\sigma_3 = \sum_{j=1}^{6} \sum_{\substack{k=1 \\ k \neq j}}^{6} \sum_{\substack{l=1 \\ l \neq j \\ l \neq k}}^{6} \alpha_{i_j} \cdot \alpha_{i_k} \cdot \alpha_{i_l} = 0.$$

Trouver l'élément nul dans le support grâce à $\varepsilon = 1$: Pour le cas où $\varepsilon = 1$, l'Équation (7.3) devient tout simplement :

$$S_E(X) \equiv \frac{\eta_E(X)}{\sigma_E(X)} \equiv \frac{1}{X \oplus \sigma_0} \mod G(X).$$

Cela signifie que l'inversion du polynôme syndrome se fait en une seule itération. Cependant, un cas sera notable parmi tous, à savoir le cas où l'élément nul du support est la racine.

Contre-mesure : Aucune proposition de contre-mesure n'a été faite. L'auteur a laissé cela comme problème ouvert. Toutefois l'auteur propose une réflexion sur deux possibilités. La première est de reprendre l'idée de contre-mesure de [SSMS10]. La seconde est de n'ajouter que $(t-1)$ erreurs au chiffrement, puis de changer la valeur d'un bit au début du déchiffrement. L'indice du bit changé doit toujours être le même pour un texte chiffré donné, mais doit sembler aléatoire pour l'attaquant (grâce à un générateur de nombres pseudo-aléatoires et d'une fonction de hachage). Cependant, on n'a aucun moyen a priori de forcer un attaquant à utiliser des erreurs de poids maximaux plus que minimaux et cela reste insatisfaisante d'un point de vue efficacité.

7.2.2 Amélioration de l'attaque

Nous proposons maintenant une amélioration de l'attaque en combinant les deux attaques énoncées précédemment (à savoir [Str10b, Str13b]). Rappelons que l'objectif de l'attaquant est de retrouver la permutation privée $\mathcal{P}$ grâce à une attaque par chiffrés choisis.

Identification d'une fuite : La fuite est identifiée aux lignes 5 et 11 de l'algorithme de Patterson (Algorithme 32). Ce type d'attaque a déjà été publié dans [Str10b, Str13b] comme expliqué dans la sous-section précédente. Les deux étapes utilisant EEA sont considérées comme deux parties indépendantes. Dans cette section, nous proposons de montrer la relation existante entre les deux étapes puis comment les attaquer. En effet, le principal problème des attaques précédentes est le nombre limité de cas dans lesquels elles peuvent être exploitées. Elles peuvent seulement être appliquées pour des vecteurs erreurs de poids $w_H(E) \in \{2, 4\}$ comme montré dans [Str10b], ou de poids $w_H(E) \in \{1, 4, 6\}$ comme présenté dans [Str13b]. Le problème provient du simple fait suivant : le nombre d'itérations est donné par deux conditions. L'une des conditions est que tous les quotients dans EEA doivent être des polynômes de degré 1. Donc lorsque cette condition n'est pas vérifiée, le nombre d'itérations peut être aussi grand que le souhaite l'attaquant. Nous utiliserons nb_1 et nb_2 respectivement comme notations pour le nombre d'itérations dans la $5^{\text{ème}}$ et la $11^{\text{ème}}$ lignes de l'algorithme de Patterson.

Motivations de notre attaque : Nous montrons que l'usage de la relation entre les deux étapes nous permet de contrôler entièrement le nombre d'itérations. L'autre contribution est dans la recherche de la relation entre les deux étapes et dans son utilisation pour construire un plus grand système d'équations. Nous montrons que nous sommes capables d'étendre le nombre limité d'équations du système jusqu'à $w_H(E) = \frac{\deg(G)}{2}$. Le principal intérêt est qu'au lieu de trouver des équations impliquant seulement la permutation de 1, 4 ou 6 éléments, nous pouvons étendre cette recherche autant que nécessaire dans le but de découvrir complètement la permutation privée plutôt que de réduire l'espace de recherche pour une force brute.
En termes de complexité, au lieu d'énumérer toutes les permutations possibles, c'est-à-dire $n!$ permutations, nous réduisons la complexité à l'expression suivante :

$$\sum_{i=3}^{p} \binom{n}{i}, \qquad \text{où } p \leq \frac{\deg(G)}{2} - 1$$

Donc pour de petites valeurs de p, nous avons :

$$\sum_{i=3}^{p} \binom{n}{i} \leq \binom{n}{p} \times (p-2) \leq \left(\frac{e \cdot n}{p}\right)^p \times (p-2)$$

(où $e \approx 2.72$ est la base du logarithme népérien).

Dans le but de rendre claire la différence entre l'attaque originale (par force brute) et la notre afin de retrouver la permutation privée, nous proposons de donner la borne inférieure pour la complexité de l'attaque originale :

$$\left(\frac{n}{e}\right)^n \leq n!$$

Finalement :

$$\sum_{i=3}^{p} \binom{n}{i} \leq \left(\frac{e \cdot n}{p}\right)^p \times (p-2) << \left(\frac{n}{e}\right)^n \leq n!$$

Donc l'attaque originale est exponentielle en la longueur du code alors qu'une attaque temporelle a seulement une complexité en le maximum du poids du vecteur erreur nécessaire pour l'attaque (souvent extrêmement petit en comparaison avec la longueur du code).

Scenario : L'attaquant procède selon les trois étapes suivantes :

1. Il choisit un message aléatoire M et calcule $C = M \cdot \mathcal{G}$;
2. Il choisit aléatoirement un vecteur erreur E de petit poids $w_H(E) < t$ (où t est la capacité de correction du code) et calcule $\tilde{C} = M \cdot \mathcal{G} \oplus E$;
3. Il envoie $\tilde{C}$ à un oracle ($\mathcal{O}$), qui retourne le message M et le nombre d'itérations aux lignes 5 et 11 de l'algorithme de Patterson.

Idée principale : Pour $w_H(E) = 2p$, avec $p \in \mathbb{N}$, l'attaquant trouve des équations ayant la forme suivante :

$$\sum_{i=1}^{w_H(E)} \mathcal{P}(\alpha_i) = 0.$$

Il est capable de construire ce type d'équations avec $0 < w_H(E) \leqslant \frac{\deg(G)}{2}$.

Conditions : L'hypothèse générale est que l'attaquant connaît la clé publique p_k, l'ordre des éléments dans le support $\mathscr{L}$ ($\mathscr{L}$ est supposé être public, par exemple dans l'ordre lexicographique) et a accès à un oracle $\mathcal{O}$. Ces hypothèses sont les mêmes que dans les travaux mentionnés précédemment. Nous améliorons l'attaque dans le même contexte. L'oracle $\mathcal{O}$ est aussi capable de donner des informations supplémentaires : le temps d'exécution pour l'algorithme entier ou simplement une étape particulière. Nous supposons que l'attaquant peut ne pas respecter les paramètres du protocole en ajoutant $w_H(E) < t$ erreurs. En pratique, c'est toujours possible puisque l'attaquant est capable de choisir le nombre et les positions des erreurs.

Inversion du syndrome

Il a été montré dans [Str13b] que l'inversion du syndrome laisse fuir de l'information. L'attaque est basée sur le nombre d'itérations utilisées dans l'algorithme d'Euclide étendu, dans le but de calculer l'inverse du polynôme syndrome $S(X)$ modulo le polynôme de Goppa $G(X)$. Elle utilise les propriétés suivantes :

$$nb_1 \leq \deg(\sigma) + \deg(\eta), \qquad \text{pour } w_H(E) < \frac{\deg(G)}{2}.$$

Nous ne détaillerons pas ici toutes les conditions, puisqu'elles sont bien expliquées dans [Str13b], mais nous donnons seulement certains faits importants dans le but de rendre les choses plus claires et préparer l'attaque. Considérons le PLE de la forme :

$$\sigma(X) = X^\varepsilon + S^\varepsilon_{\varepsilon-1} \cdot X^{\varepsilon-1} + S^\varepsilon_{\varepsilon-2} \cdot X^{\varepsilon-2} + \ldots + S^\varepsilon_2 \cdot X^2 + S^\varepsilon_1 \cdot X + S^\varepsilon_0,$$

avec $\varepsilon \equiv 0 \mod 2$.
Alors $\eta(X) = S^\varepsilon_{\varepsilon-1} \cdot X^{\varepsilon-2} + \ldots + S^\varepsilon_3 \cdot X^2 + S^\varepsilon_1$.

Dans ce cas, le nombre maximal d'itérations est donné par le coefficient $S^{\varepsilon}_{\varepsilon-1} = \sum_{i=1}^{\varepsilon} \alpha_i$. Donc si $S^{\varepsilon}_{\varepsilon-1} \neq 0$, nous obtenons $nb_1 = 2\varepsilon - 2$ et tous les quotients ont un degré égal à 1. Si $S^{\varepsilon}_{\varepsilon-1} = 0$ et $S^{\varepsilon}_{\varepsilon-3} \neq 0$, alors $nb_1 = 2\varepsilon - 4$ et tous les quotients ont un degré égal à 1.

Détermination du PLE

Deux observations doivent être faite dans le but de comprendre et de localiser le point de fuite. La première concerne le nombre d'itérations. Il a été prouvé dans [SSMS10] que ce nombre est (avec une forte probabilité) :

$$nb_2 = \sum_{i=1}^{d} \deg(q_i) = \deg(\mathfrak{b}),$$

où d est ici la borne utilisée sur le degré dans la condition de la boucle **Tant que** (ligne 7 de l'Algorithme 35), avec $d \leqslant t/2$ dans tous les cas. Rappelons certaines relations entre $R(X)$, $\mathfrak{b}(X)$, $\mathfrak{a}(X)$ et $\sigma(X)$ (liées à la ligne 11 de l'Algorithme 32). Puis prouvons en de nouvelles, qui nous permettent d'arriver à l'attaque juste après. Ces relations sont importantes puisqu'elles influencent chaque étape du déchiffrement et chaque forme particulière des polynômes impliqués.

Propriétés 7.1 *Les propriétés utiles pour notre approche sont :*

1. *Si* $\varepsilon \equiv 0 \mod 2$, *alors* $\deg(\mathfrak{a}) = \frac{\varepsilon}{2}$ *(voir [SSMS10]).*
2. *Si* $\varepsilon \equiv 1 \mod 2$, *alors* $\deg(\mathfrak{b}) = \frac{\varepsilon-1}{2}$ *(voir [SSMS10]).*
3. *Si* $\deg(R) \leq \lfloor \frac{\varepsilon}{2} \rfloor$, *alors* $\deg(\mathfrak{a}) = \deg(R) + \deg(\mathfrak{b})$.
4. *Si* $\deg(R) \leq \lfloor \frac{\varepsilon}{2} \rfloor$ *et* $\deg(R) \neq 0$, *alors* $w_H(E) \equiv 0 \mod 2$.

Deux cas sont possibles en fonction de la parité de $w_H(E) = \varepsilon$:

Si $w_H(E)$ est impair : Soit $\deg(G) = 2p + 1$, avec $p \in \mathbb{N}$. Pour un vecteur erreur avec un poids de Hamming $w_H(E) = 2k+1$, où $k \leq p - 1$, nous avons les relations suivantes :

$$\deg(\mathfrak{b}) = k, \deg(\mathfrak{a}) \leq k \text{ et } \deg(R) \geq 2p - k.$$

Si $w_H(E)$ est pair : Soit $\deg(G) = 2p$. Pour un vecteur erreur avec un poids de Hamming $w_H(E) = 2k$, où $k \leq p$, nous avons les relations suivantes :

$$\deg(\mathfrak{a}) = k, \deg(\mathfrak{b}) \leq k - 1 \text{ et } \deg(\mathfrak{b}) = 0 \Leftrightarrow \deg(R) = k.$$

Nombre d'itérations

Nous pouvons voir que le nombre d'itérations dans EEA est égal à $\deg(\mathfrak{b})$, donc nous nous focaliserons sur la forme du polynôme $\mathfrak{b}(X)$, plus exactement lorsque ε est pair. Soit :

$$\sigma(X) = X^{2p} + S^{2p}_{2p-1}X^{2p-1} + S^{2p}_{2p-2}X^{2p-2} + S^{2p}_{2p-3}X^{2p-3} + \ldots + S^{2p}_{2}X^{2} + S^{2p}_{1}X + S^{2p}_{0}.$$

Nous séparons les puissances impaires des puissances paires et obtenons :

$$\begin{aligned}
\sigma(X) &= (X^{2p} + S_{2p-2}^{2p} X^{2p-2} + \ldots + S_2^{2p} X^2 + S_0^{2p}) \\
&\quad +(S_{2p-1}^{2p} X^{2p-1} + S_{2p-3}^{2p} X^{2p-3} + \ldots + S_1^{2p} X) \\
\sigma(X) &= (X^p + \sqrt{S_{2p-2}^{2p}} X^{p-1} + \ldots + \sqrt{S_2^{2p}} X + \sqrt{S_0^{2p}})^2 \\
&\quad +x(\underbrace{\sqrt{S_{2p-1}^{2p}} X^{p-1} + \ldots + \sqrt{S_1^{2p}}}_{\mathfrak{b}(X)})^2
\end{aligned}$$

Donc $\deg(\mathfrak{b})$ est donné par les coefficients S_{2i-1}^{2p} avec $i \in \{1, 2, \ldots, p\}$. Par conséquent, le nombre d'itérations pourrait être donné par les mêmes coefficients sous la condition supplémentaire que tous les quotients sont de degré 1. Nous pouvons donc distinguer $p-1$ cas possibles dépendants des coefficients, si leur degré vaut 1 à chaque itération. Par conséquent :

$$\begin{cases}
nb_2 = p-1 & \text{si } S_{2p-1}^{2p} \neq 0 \\
nb_2 = p-2 & \text{si } S_{2p-1}^{2p} = 0 \text{ et } S_{2p-3}^{2p} \neq 0 \\
nb_2 = p-3 & \text{si } S_{2p-1}^{2p} = 0 \; S_{2p-3}^{2p} = 0 \text{ et } S_{2p-5}^{2p} \neq 0 \\
\quad \vdots
\end{cases}$$

Dans tous les cas, la même hypothèse est faite : le degré du quotient est égal à 1 à chaque itération. Cela signifie que nous pourrions avoir le nombre d'itérations sans condition sur les coefficients.

Attaque contre le couple (nb_1, nb_2)

Nous expliquons maintenant comment notre attaque fonctionne. Commençons par présenter la relation générale pour le couple (nb_1, nb_2). En utilisant les paragraphes sur l'inversion du syndrome et la détermination du PLE de cette sous-section, nous obtenons la propriété suivante :

Propriété 7.1 *Soit $w_H(E) = 2p < t/2$. Étant donnés nb_1 et nb_2, on a :*

$$(nb_1, nb_2) = (4p-4, p-2) \Rightarrow \sum_{i=1}^{2p} \alpha_i = 0,$$

avec probabilité $\mathbb{P}_{succès}$.

Un exemple est donné dans la Sous-Section 7.2.3.

Probabilité de succès

La probabilité de succès $\mathbb{P}_{\text{succès}}$ est décrite par l'événement suivant : *{Tous les quotients ont un degré égale à 1.}* Si nous considérons tous les éléments de notre support comme étant des variables uniformément distribuées et l'indépendance de chaque étape dans EEA, sous les hypothèses initales, nous avons :

$$\begin{aligned}
\mathbb{P}_{\text{succès}} &= \mathbb{P}(\{nb_1 = 4p-4\} \cap \{nb_2 = p-2\}) \\
&= \mathbb{P}(\{nb_1 = 4p-4\}) \cdot \mathbb{P}(\{nb_2 = p-2\}) \\
&= \left(1 - \frac{1}{n}\right)^{nb_1 + nb_2}.
\end{aligned}$$

Les résultats expérimentaux montrent que pour $n = 2048$ et $w_H(E) = 4$, dans le but de trouver les équations de la forme suivante :

$$\mathcal{P}(\alpha_1) \oplus \mathcal{P}(\alpha_2) \oplus \mathcal{P}(\alpha_3) \oplus \mathcal{P}(\alpha_4) = 0,$$

la probabilité est égale à 0.998. Cela signifie que moins de 0.2% des cas ne sont pas exploitables parmi tous les cas possibles sous la condition : $nb_1 = 4$ et $nb_2 = 0$. En d'autres termes, chaque fois que cette combinaison est révélée, la probabilité d'avoir une bonne équation pour notre attaque est égale à 0.998 pour les paramètres donnés.

Tests expérimentaux

Dans le but de valider les relations que nous présentons dans le paragraphe précédent pour (nb_1, nb_2), nous avons utilisé une implantation sur PARI/GP [20] du cryptosystème de McEliece. Nous avons calculé un couple de clés, puis chiffré et déchiffré un message donné de nombreuses fois, en vérifiant la valeur du couple (nb_1, nb_2), en cherchant pour les combinaisons valides précédemment décrites. Nous avons aussi utilisé différentes valeurs pour m, le degré de l'extension du corps fini. Nous avons exécuté l'algorithme jusqu'à ce que nous obtenions la combinaison spécifique environ cent fois. Puis, nous avons obtenu une valeur moyenne du nombre d'itérations requises pour obtenir la combinaison recherchée. Les résultats sont présentés dans le Tableau 7.1 :

TABLEAU 7.1 – Nombre d'itérations nécessaires pour obtenir la combinaison pour des vecteurs erreurs de poids de Hamming 4, 6 et 8

Combinaison :			
nb_1	4	8	12
nb_2	0	1	2
Nombre d'itérations pour $m = 7$	127	138	142
Nombre d'itérations pour $m = 8$	235	270	273

Cela signifie que pour $m = 7$, nous avons besoin d'envoyer à l'oracle en moyenne 127 textes chiffrés différents, dans le but d'obtenir la relation voulue $(\mathcal{P}(\alpha_1)+\mathcal{P}(\alpha_2)+\mathcal{P}(\alpha_3)+\mathcal{P}(\alpha_4) = 0)$. Dans ce cas, la densité de cette équation est égale en moyenne :

$$\frac{1}{127} \times \mathbb{P}_{\text{succès}} = \frac{1}{127} \times \left(1 - \frac{127}{128}\right)^4.$$

Sachant qu'une relation nous permet, en fixant l'une des positions, de réduire le nombre de textes chiffrés qui doivent être envoyés à l'oracle, cela signifie que les relations voulues sont révélées plus souvent lorsque nous progressons dans l'attaque. Cela donne aussi une première intuition sur la structure de la permutation.

Implantation de l'attaque : Dans le but de tester en pratique notre attaque, nous avons utilisé la même implantation logicielle. Afin de révéler des temps d'exécution proches des valeurs réelles, nous avons répété l'attaque plus de 10^6 fois. Nous présentons les résultats obtenus dans le Tableau 7.2.

20. `http://pari.math.u-bordeaux.fr/`

TABLEAU 7.2 – Temps (en secondes) pour le déchiffrement dans le cas où $n = 2^{11}$ et $t = 16$

$w_H(E)$	Temps pour trouver l'équation espérée de l'attaque (en secondes)	Temps pour trouver une équation de type aléatoire (en secondes)
4	30141892×10^{-6}	304856×10^{-4}
6	3072799×10^{-5}	310234×10^{-4}
8	31597171×10^{-6}	32242382×10^{-6}
10	3285724×10^{-6}	3345847×10^{-5}

Observation : Nous ne donnons pas les temps pour les valeurs impaires puisqu'elles sont constantes et indépendantes des combinaisons linéaires entre les permutations des positions d'erreurs. Dans le Tableau 7.2, nous observons qu'il y a une mince différence entre l'attaque avec notre type de combinaisons et des combinaisons aléatoires. Comme nous l'avons mentionné précédemment, celles avec le nombre maximal d'itérations apparaissent plus souvent (le cas où tous les coefficients sont différents de zéro). Donc dans ce cas, nous avons une différence de temps requise pour que notre attaque fonctionne. Dans la Section 7.3, nous expliquons comment la correction fonctionne, non seulement sur ce type d'attaque, mais aussi sur d'autres types comme les attaques par changement de valeur d'un bit.

7.2.3 Exemple

Considérons $\mathbb{F}_{2^4}[X] = \mathbb{F}_2[X]/(X^4 + X + 1)$. La matrice génératrice $\mathcal{G}$ du code de Goppa et le support $\mathscr{L} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{14}\}$ sont publics. Soient $M \in \mathbb{F}_2^k$ le message et $\mathcal{O}$ l'oracle de déchiffrement. Nous notons que si $\mathscr{L}$ est public, on peut trouver $Q(X)$ tel que $\mathscr{L} = \mathbb{F}_2[X]/(Q(X))$, avec $Q(X) = X^4 + X + 1$. La réciproque est également vraie : si $Q(X)$ est public, alors on peut facilement trouver $\mathscr{L}$. Supposons que la permutation privée soit :

$$\mathcal{P}(\mathscr{L}) = \mathscr{L}' = \{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{14}, 0, 1\} = \{\ell_i \mid i \in (1 \ldots 16)\}$$

— 1ère *étape :* $w_H(E) = 1$
  - L'attaquant demande à $\mathcal{O}$ de déchiffrer tous les $\tilde{C} = M \cdot \mathcal{G} \oplus E$ avec $w_H(E) = 1$.
  - ⋆ nb_1 et nb_2 révèle la position de $\mathcal{P}(0)$: ℓ_{15}.
  - ∘ Cela est principalement dû à : $\sigma(X) = X$. (Nous avons $R(X) = 0$ et $S^{-1}(X) = X$.)

— 2ème *étape :* $w_H(E) = 4$ *et 0 est une racine*
  - L'attaquant demande à $\mathcal{O}$ de déchiffrer tous les $\tilde{C} = M \cdot \mathcal{G} \oplus E$ avec $w_H(E) = 4$. (Les positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ sont les trois positions non nulles de E et $\ell_{i_4} = \ell_{15}$.)
  - ⋆ Le couple $(nb_1, nb_2) = (4, 0)$ révèle tous les $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ tels que $\ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} = 0$.
    Ici $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}) \in \{(\ell_1, \ell_4, \ell_{16}), (\ell_3, \ell_{14}, \ell_{16}), \ldots\}$.
  - ∘ $\deg(\sigma) = 4$ et $\deg(\eta) = \begin{cases} 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \neq 0 \\ 0 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} = 0 \end{cases}$
  - ∘ $\deg(\mathfrak{b}) = \begin{cases} 1 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \neq 0 \\ 0 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} = 0 \end{cases}$

— 3ème *étape :* $w_H(E) = 4$ *et 0 n'est pas une racine*
- L'attaquant demande à $\mathcal{O}$ de déchiffrer tous les $\tilde{C} = M \cdot \mathcal{G} \oplus E$ avec $w_H(E) = 4$. (Les positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4})$ sont les quatre positions non nulles de E.)
- ⋆ Le couple $(nb_1, nb_2) = (4, 0)$ révèle tous les $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4})$ tels que $\ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} = 0$.
  Ici $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}) \in \{(\ell_1, \ell_2, \ell_{10}, \ell_{16}), (\ell_2, \ell_3, \ell_{13}, \ell_{16}), \ldots\}$.
- ∘ $\deg(\sigma) = 4$ et $\deg(\eta) = \begin{cases} 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \neq 0 \\ 0 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} = 0 \end{cases}$
- ∘ $\deg(\mathfrak{b}) = \begin{cases} 1 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \neq 0 \\ 0 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} = 0 \end{cases}$

— 4ème *étape :* $w_H(E) = 6$ *et 0 est une racine*
- L'attaquant demande à $\mathcal{O}$ de déchiffrer tous les $\tilde{C} = M \cdot \mathcal{G} \oplus E$ avec $w_H(E) = 6$. (Les positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5})$ sont les cinq positions non nulles de E et $\ell_{i_6} = \ell_{15}$.)
- ⋆ Le couple $(nb_1, nb_2) = (8, 1)$ révèle tous les $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5})$ tels que $\ell_{i_1} \oplus \ell_{i_2} \oplus \cdots \oplus \ell_{i_5} = 0$.
  Ici $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5}) \in \{(\ell_1, \ell_2, \ell_3, \ell_{12}, \ell_{16}), (\ell_3, \ell_4, \ell_8, \ell_{12}, \ell_{16}), \ldots\}$.
- ∘ $\deg(\sigma) = 4$ et $\deg(\eta) = \begin{cases} 4 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \oplus \ell_{i_5} \neq 0 \\ 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \oplus \ell_{i_5} = 0 \end{cases}$
- ∘ $\deg(\mathfrak{b}) = \begin{cases} 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \oplus \ell_{i_5} \neq 0 \\ 1 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \ell_{i_4} \oplus \ell_{i_5} = 0 \end{cases}$

— 5ème *étape :* $w_H(E) = 6$ *et 0 n'est pas une racine*
- L'attaquant demande à $\mathcal{O}$ de déchiffrer tous les $\tilde{C} = M \cdot \mathcal{G} \oplus E$ avec $w_H(E) = 6$. (Les positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5}, \ell_{i_6})$ sont les six positions non nulles de E.)
- ⋆ Le couple $(nb_1, nb_2) = (8, 1)$ révèle tous les $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ldots, \ell_{i_6})$ tels que $\ell_{i_1} \oplus \ell_{i_2} \oplus \cdots \oplus \ell_{i_6} = 0$.
  Ici $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ldots, \ell_{i_6}) \in \{(\ell_1, \ell_2, \ell_3, \ell_4, \ell_6, \ell_{16}), \ldots\}$.
- ∘ $\deg(\sigma) = 4$ et $\deg(\eta) = \begin{cases} 4 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \cdots \oplus \ell_{i_6} \neq 0 \\ 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \cdots \oplus \ell_{i_6} = 0 \end{cases}$
- ∘ $\deg(\mathfrak{b}) = \begin{cases} 2 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \cdots \oplus \ell_{i_6} \neq 0 \\ 1 & \text{si } \ell_{i_1} \oplus \ell_{i_2} \oplus \ell_{i_3} \oplus \cdots \oplus \ell_{i_6} = 0 \end{cases}$

— $\vdots$

— *Dernière étape :* L'attaquant doit résoudre le système suivant dans le but de trouver la permutation privée :

$$\begin{cases} 1^{ère} \text{ étape : } \ell_{15} = \mathcal{P}(0) \\ 2^{ème} \text{ étape : } \begin{cases} \ell_1 \oplus \ell_4 \oplus \ell_{16} = 0 \\ \ell_3 \oplus \ell_{14} \oplus \ell_{16} = 0 \\ \cdots = 0 \end{cases} \\ 3^{ème} \text{ étape : } \begin{cases} \ell_1 \oplus \ell_2 \oplus \ell_{10} \oplus \ell_{16} = 0 \\ \ell_2 \oplus \ell_3 \oplus \ell_{13} \oplus \ell_{16} = 0 \\ \cdots = 0 \end{cases} \\ 4^{ème} \text{ étape : } \begin{cases} \ell_1 \oplus \ell_2 \oplus \ell_3 \oplus \ell_{12} \oplus \ell_{16} = 0 \\ \ell_3 \oplus \ell_4 \oplus \ell_8 \oplus \ell_{12} \oplus \ell_{16} = 0 \\ \cdots = 0 \end{cases} \\ 5^{ème} \text{ étape : } \begin{cases} \ell_1 \oplus \ell_2 \oplus \ell_3 \oplus \ell_4 \oplus \ell_6 \oplus \ell_{16} = 0 \\ \cdots = 0 \end{cases} \\ \vdots \\ \text{et ainsi de suite jusqu'aux relations correspondant à } \deg(G)/2 \text{ erreurs.} \end{cases}$$

La résolution du système permet de déterminer complètement la permutation privée, à savoir :

$$\mathcal{P}(\mathcal{L}) = \mathcal{L}' = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \ldots, 0, 1\}.$$

7.3 Contre-mesure

Nous avons vu qu'il est possible d'attaquer le cryptosystème en sachant combien de fois EEA est répété. Le nombre d'itérations peut aller de 0 à $t-1$ dans l'inversion du polynôme syndrome et de 0 à $t/2$ dans la détermination du PLE. Dans le but d'éviter une recherche de corrélation à partir du nombre d'itérations, nous proposons d'introduire des itérations supplémentaires dans EEA. Le nombre d'itérations supplémentaires devrait être choisi entre 0 et une valeur que nous appelons *extra*. La valeur *extra* est soit $t/2$ soit $t-1$, respectivement pour l'inversion du polynôme syndrome (ligne 5 de l'Algorithme 32) ou la détermination du PLE (ligne 11 de l'Algorithme 32). La variable i de l'Algorithme 36 contient le nombre d'itérations réalisées dans la première partie de EEA sécurisé. Nous avons choisi d'utiliser des valeurs entières dans les étapes supplémentaires de EEA, dans le but d'éviter des divisions par zéro qui pourrait se produire si nous gardons les termes précédents. L'idée est de continuer à calculer des choses qui coûtent aussi cher en termes d'opérations élémentaires que pour EEA original (Algorithme 33), donc qu'un attaquant ne puisse pas faire de distinction entre les vraies étapes et les étapes supplémentaires. Nous proposons, comme contre-mesure, d'utiliser l'Algorithme 36 à la place de l'Algorithme 35.

Algorithme 36 Algorithme d'Euclide étendu sécurisé

Entrée(s): $R(X)$, $G(X)$, d_{break} et t.
Sortie(s): $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$ tels que $\mathfrak{b}(X) \cdot R(X) \equiv \mathfrak{a}(X) \mod G(X)$.

1: $d \leftarrow d_{break}$
2: $\mathfrak{b}_{-1} \leftarrow 0$
3: $\mathfrak{b}_0 \leftarrow 1$
4: $r_{-1} \leftarrow G(X)$
5: $r_0 \leftarrow R(X)$
6: $i \leftarrow 0$
7: **Tant que** $\deg(r_i) > d$ **faire**
8: $\quad i \leftarrow i + 1$
9: $\quad r_{i-2}(X) = r_{i-1}(X) \cdot q_i(X) + r_i(X)$
10: $\quad \mathfrak{b}_i(x) \leftarrow \mathfrak{b}_{i-2}(X) + q_i(x) \cdot \mathfrak{b}_{i-1}(X)$
11: **Fin tant que**
12: $\mathfrak{a}(X) \leftarrow r_i(X)$
13: $\mathfrak{b}(X) \leftarrow \mathfrak{b}_i(X)$
14: $extra = f(t, d_{break})$
15: **Tant que** $i < extra$ **faire**
16: $\quad i \leftarrow i + 1$
17: $\quad r_{i-2}(X) = 3 \cdot q_i(X) + 5$
18: $\quad \mathfrak{b}_i(X) \leftarrow 5 + 6q_i(X)$
19: **Fin tant que**
20: **Retourner** $\mathfrak{a}(X)$ et $\mathfrak{b}(X)$.

Les nouveaux paramètres de sécurité : Habituellement, les paramètres de sécurité sont donnés sous l'hypothèse d'attaques théoriques, comme l'ISD [BJMM12]. Pour le cryptosystème de McEliece, les paramètres usuels sont :

— pour 100 bits de sécurité : $n = 2048$ et $t = 50$,
— pour 128 bits de sécurité : $n = 2960$ et $t = 56$,
— pour 256 bits de sécurité : $n = 6624$ et $t = 115$.

Pour les premiers paramètres, une attaque temporelle avec $p = 6$ révélerait la permutation privée avec une complexité inférieure à 2^{61} opérations élémentaires, ce qui est beaucoup plus faible que le niveau de sécurité de la proposition originale. Donc pour des attaques temporelles, des paramètres plus grands doivent être pris en considération dans le but de maintenir le même niveau de sécurité. Par exemple, dans le but d'atteindre un niveau de sécurité de 100 bits contre cette attaque, on devrait proposer $n = 131\,072 = 2^{17}$.
La solution habituelle n'est pas d'augmenter les valeurs des paramètres mais de proposer une variante sécurisée de l'algorithme, variante qui n'est pas vulnérable à l'attaque spécifiée. Notre proposition est plus lente que l'algorithme original (Algorithme 33), elle opère en $(t-1) \times O(1)$ pour l'inversion du polynôme syndrome et en $\frac{t}{2} \times O(1)$ pour la détermination du PLE (où $O(1)$ est la complexité usuelle pour une division).
D'un autre côté, elle est sécurisée contre les attaques temporelles décrites ci-dessus. La preuve est très simple et est basée sur le fait que ce type d'attaques temporelles en particulier est basé sur le nombre d'itérations dans EEA. Puisque notre algorithme effectue le même nombre d'itérations, peu importe quelles relations sont dissimulées entre les coefficients du polynôme, aucune de ces relations secrètes n'est révélée.
Une fois la contre-mesure appliquée, nous avons exécuté la même attaque et avons obtenu les temps donnés dans le Tableau 7.3 pour les poids de Hamming sélectionnés (les temps moyens sont présentés pour plus de 10^7 simulations).

Tableau 7.3 – Temps (en secondes) pour le déchiffrement dans le cas où $n = 2^{11}$ et $t = 10$

$w_H(E)$	Temps pour les équations du type de l'attaque (en secondes)	Temps pour une combinaison de type aléatoire (en secondes)
6	57.99	57.90
8	57.94	58.03
10	57.81	57.89

Interprétation : Nous observons que l'implantation protégée est impossible à attaquer (en utilisant les mêmes techniques). Nous soulignons que la contre-mesure proposée est aussi efficace dans le cas où un attaquant veut utiliser les techniques précentes, comme dans [Str10b, Str13b, SSMS10].

7.4 Conclusion et perspectives

Les travaux décrits dans ce chapitre ont été réalisés avant la publication d'une version du cryptosystème de McEliece en temps constant [BCS13]. Néanmoins, ce travail mérite d'être mentionné, puisqu'il améliore tout de même deux attaques déjà existantes en se basant sur les mêmes hypothèses.

La question du rang maximal du système intervenant dans l'attaque [Str10b] semble intéressante à traiter. De plus, comme poursuite des travaux sur ce même article et de manière plus générale sur ce chapitre, il faudrait essayer cette(ces) attaque(s) par une analyse de consommation.
Enfin, une remarque faite ici et pouvant servir de manière plus générale est qu'une contre-mesure à une attaque peut faciliter le travail d'un attaquant via une autre attaque. Il est donc important de considérer non seulement les étapes du déchiffrement du cryptosystème de McEliece de manière séparée afin de les sécuriser individuellement, mais également dans leur ensemble puisque, comme nous l'avons montré dans ce chapitre, cela peut avoir des conséquences là où on ne s'y attend pas.

Chapitre 8

Analyse de la structure du polynôme localisateur d'erreurs

La majorité de ce chapitre (Section 8.3) correspond aux travaux effectués pour [DCCR13], dont une version revue et étendue est en préparation :

Polynomial Structures in Code-Based Cryptography
avec Vlad Dragoi, Pierre-Louis Cayrel et Brice Colombier
IndoCrypt 2013, p. 286-296, LNCS.

Polynomial Structures in Code-Based Cryptography (extended)
avec Vlad Dragoi et Pierre-Louis Cayrel
en préparation.

Deux présentations ont été faites du travail concernant la Section 8.2 :

Towards a secure implementation of a Goppa decoder
The 11th Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices (CryptArchi 2013), Fréjus (France), Juin 2013.

Towards a secure implementation of a Goppa decoder
Journées Codes, Cryptographie et Stéganographie (JC2S 2013), Paris (France), Novembre 2013.

8.1 Évaluation du PLE

Dans le cryptosystème de McEliece, l'évaluation du polynôme localisateur d'erreurs (PLE) intervient à la fin de n'importe quel algorithme de décodage lors du déchiffrement. Celle-ci nous intéresse car elle est l'étape la plus consommatrice en temps de tout le déchiffrement de McEliece [Sen02, Chapitre 5, p. 21]. Cette étape consiste à tester toutes les valeurs du support $\mathscr{L}$ afin de déterminer lesquelles annulent le PLE. Ces valeurs correspondent en fait aux positions des bits non nuls dans le vecteur erreur ajouté au mot de code lors du chiffrement. Retrouver ces positions permet donc de corriger les erreurs insérées dans le texte chiffré. Après quoi, on obtient un mot de code sans erreur et il est alors facile de retrouver le message initial. L'attaque présentée ici a pour but de retrouver le message initial. Il faut donc recommencer cette attaque pour chaque nouveau message que l'on veut retrouver. Nous rappelons dans l'Algorithme 37 les grandes lignes du déchiffrement de McEliece, où Γ (sous-entendu $\Gamma(\mathscr{L}, G)$) est un code de Goppa décrit par son support $\mathscr{L} = \mathbb{F}_{2^m}$, son polynôme de $G \in \mathbb{F}_{2^m}[X]$ de degré t et où $\mathcal{G}$ est une matrice génératrice de ce code.

Algorithme 37 Déchiffrement de McEliece

Entrée(s): $s_k = (\mathcal{S}, \mathcal{G}, \mathcal{P}, \Gamma)$ la clé privée, $\tilde{C} \in \mathbb{F}_2^n$ le texte chiffré, où $\tilde{C} = M \cdot \mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} \oplus E$.
Sortie(s): $M \in \mathbb{F}_2^k$ le texte clair associé à $\tilde{C}$.
1: Calculer $\tilde{C}_p = \tilde{C} \cdot \mathcal{P}^{-1}$.
2: Décoder $\tilde{C}_p$ pour retrouver $M \cdot \mathcal{S} \cdot \mathcal{G}$.
3: Récupérer $\tilde{M} = M \cdot \mathcal{S}$ à partir de $M \cdot \mathcal{S} \cdot \mathcal{G}$.
4: Calculer $M = \tilde{M} \cdot \mathcal{S}^{-1}$.
5: **Retourner** M.

L'étape qui nous intéresse dans l'Algorithme 37 est celle du décodage (en rouge). Le décodage du texte chiffré permuté $\tilde{C}_p$ pour retrouver le mot de code $M \cdot \mathcal{S} \cdot \mathcal{G}$ peut être fait de différentes façons. Les grandes lignes du décodage d'un code de Goppa sont rappelées dans l'Algorithme 38. Nous avons vu que la partie résolution de l'équation clé pouvait être faite grâce à l'algorithme d'Euclide étendu, l'algorithme de Berlekamp-Massey ou à l'algorithme de Patterson (c.f. Sous-section 2.5.2). Peu importe lequel de ces trois algorithmes est utilisé, on arrive au même résultat, à savoir le PLE. C'est l'évaluation du PLE (en rouge dans l'Algorithme 38) qui nous intéresse dans ce chapitre, car elle permet la construction du vecteur erreur.

Algorithme 38 Décodage d'un code de Goppa

Entrée(s): $\tilde{C}_p = M \cdot \mathcal{S} \cdot \mathcal{G} \oplus E \cdot \mathcal{P}^{-1}$ un texte chiffré permuté tel que $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$ un code de Goppa t-correcteur et $w_H(E \cdot \mathcal{P}^{-1}) \leqslant t$, avec $\mathscr{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ le support du code et G le polynôme de Goppa.
Sortie(s): $M \cdot \mathcal{S} \cdot \mathcal{G} \in \Gamma(\mathscr{L}, G)$ le mot de code sans erreur.
1: Calculer le polynôme syndrome de $\tilde{C}_p$: $S_{\tilde{C}_p}(X)$.
2: Résoudre l'équation clé : $S_{\tilde{C}_p}(X)\sigma_{E\cdot\mathcal{P}^{-1}}(X) = \frac{d\sigma_{E\cdot\mathcal{P}^{-1}}(X)}{dX} \mod G(X)$.
3: //(afin d'obtenir le polynôme localisateur d'erreurs $\sigma_{E\cdot\mathcal{P}^{-1}}(X)$)
4: Construire le vecteur erreur E tel que $e_i = 1$ si $\sigma_{E\cdot\mathcal{P}^{-1}}(\alpha_i) = 0$ et $e_i = 0$ sinon.
5: **Retourner** $M \cdot \mathcal{S} \cdot \mathcal{G} = \tilde{C}_p \oplus E \cdot \mathcal{P}^{-1}$.

Détaillons maintenant comment cette évaluation peut être effectuée. Pour cela, plusieurs choses sont à prendre en compte. Tout d'abord, rappelons la forme du PLE, en notant $E_p = E \cdot \mathcal{P}^{-1}$ le vecteur erreur permuté afin d'alléger les notations :

$$\sigma_{E_p}(X) = \prod_{\substack{i=1 \\ E_{p_i} \neq 0}}^{n} (X - \alpha_i), \tag{8.1}$$

où E_{p_i} correspond au $i^{\text{ème}}$ bit de E_p. On peut alors remarquer que $\sigma_{E_p}(X) \in \mathbb{F}_{2^m}[X]$ et qu'il est unitaire. De plus, comme $w_H(E \cdot \mathcal{P}^{-1}) = w_H(E) = t$, on a $\deg(\sigma_{E_p}) = t$. Ce polynôme est donc à coefficients dans un corps fini (c'est-à-dire $\mathbb{F}_{2^m}$ ici) et est évalué sur les éléments d'un ensemble, qui est le même corps fini dans le cas présent (c'est-à-dire $\mathscr{L} = \mathbb{F}_{2^m}$), mais peut parfaitement être un sous-ensemble de celui-ci (puisqu'on a de manière générale $\mathscr{L} \subseteq \mathbb{F}_{2^m}$) ou encore ne pas être nécessairement le même que celui auquel appartiennent les coefficients.
Faisons une petite parenthèse sur tout cela. Pour un code de Goppa donné, le PLE appartient au même anneau de polynômes que le polynôme de Goppa et il a toutes ses racines avec multiplicité un dans le support. Pour évaluer un polynôme, plusieurs

algorithmes sont utilisables. Tout d'abord, présentons l'algorithme d'évaluation d'un polynôme qui nous vient intuitivement (Algorithme 39).

Algorithme 39 Evaluation d'un polynôme

Entrée(s): P un polynôme et X la valeur en laquelle on souhaite évaluer P.
Sortie(s): P_x l'évaluation de P en X.

1: $t = \deg(P)$
2: $P_x = 0$
3: **Pour** $i = 0$ **à** t **faire**
4: $\quad P_x = P_x + P_i \cdot X^i$
5: **Fin pour**
6: **Retourner** P_x

Pour notre cas, on souhaite évaluer le PLE et cet algorithme n'exploite ni la structure du polynôme ni les relations existantes entre deux valeurs consécutives sur lesquelles on l'évalue. Afin de ne pas calculer la $i^{\text{ème}}$ puissance de X depuis le début à chaque étape de la boucle **Pour** de l'Algorithme 39, William Horner proposa une méthode en 1819 dans [Hor19] que nous rappelons dans l'Algorithme 40.

Algorithme 40 Méthode de Horner d'évaluation d'un polynôme

Entrée(s): P un polynôme et X la valeur en laquelle on souhaite évaluer P.
Sortie(s): P_x l'évaluation de P en X.

1: $t = \deg(P)$
2: $P_x = 0$
3: **Pour** $i = t$ **à** 1 **faire**
4: $\quad P_x = (P_x + P_i) \cdot X$
5: **Fin pour**
6: $P_x = P_x + P_0$
7: **Retourner** P_x

Cet algorithme permet de calculer toutes les puissances de X au fur et à mesure et de toutes les avoir à la fin de la boucle **Pour** de l'Algorithme 40.
On peut encore aller plus loin en utilisant le fait que, toujours dans notre cas, nous évaluons le PLE sur l'ensemble des éléments d'un corps fini, puisque nous avons choisi $\mathscr{L}$ comme étant le corps $\mathbb{F}_{2^m}$ tout entier. Or, il se trouve que Robert Chien proposa en 1964 dans [Chi64] un algorithme (Algorithme 41) permettant de tester sur l'ensemble des éléments non nuls d'un corps fini $\mathbb{F}_{2^m}$ quels éléments étaient racines d'un polynôme donné, de manière plus efficace que la force brute. Son algorithme a été proposé pour le cas des codes BCH (c.f. Sous-section 2.4.2) car ils sont définis sur un support tel que les éléments soient tous les éléments non nuls d'une extension de $\mathbb{F}_2$. Les codes de Goppa étant plus généraux, si $0 \in \mathscr{L}$, il suffit de le tester à part pour savoir s'il est racine du PLE. En revanche, pour tous les autres éléments de $\mathscr{L}$, si on a choisi $\mathscr{L} = \mathbb{F}_{2^m}$ comme nous le supposons ici, la méthode de Chien est parfaitement applicable.

Avant de présenter cet algorithme à proprement parler, voyons tout d'abord quel type de relations il exploite afin de gagner en efficacité. Soit $\alpha \in \mathbb{F}_{2^m}$ un élément primitif. Comme on considère tous les éléments non nuls du corps $\mathbb{F}_{2^m}$, autrement dit le groupe multiplicatif $\mathbb{F}_{2^m}^*$, on peut donc prendre α comme générateur de ce

groupe cyclique, c'est-à-dire $\mathbb{F}_{2^m}\backslash\{0\} = \{1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$.
Soit $P(X) \in \mathbb{F}_{2^m}[X]$ un polynôme de degré t que l'on souhaite évaluer sur tous les éléments de $\mathbb{F}_{2^m}$. On a alors, $\forall i \in [\![0, 2^m - 2]\!]$ et $\forall j \in [\![0, t]\!]$, les relations suivantes :

$$\begin{aligned}
P(X) &= P_0 + P_1 \cdot X + \ldots + P_t \cdot X^t \\
P(\alpha^i) &= P_0 + P_1 \cdot \alpha^i + \ldots + P_t \cdot (\alpha^i)^t \\
P(\alpha^{i+1}) &= P_0 + P_1 \cdot \alpha^{i+1} + \ldots + P_t \cdot (\alpha^{i+1})^t \\
&= P_0 + P_1 \cdot \alpha^i \cdot \alpha^1 + \ldots + P_t \cdot (\alpha^i)^t \cdot \alpha^t.
\end{aligned}$$

On remarque qu'il est utile d'avoir la table des exponentielles jusqu'à la puissance t de l'élément primitif α dans ce cas-là. De plus, en notant $P_{i,j} = P_j \cdot (\alpha^i)^j$, on en déduit que l'image de $\alpha^i + 1$ par P peut s'exprimer en fonction de celle de α^i par P, comme ceci :

$$\begin{aligned}
P(\alpha^i) &= \sum_{j=0}^{t} P_j \cdot (\alpha^i)^j, \qquad \forall i = 0, \ldots, 2^m - 1 \\
&= \sum_{j=0}^{t} P_{i,j}, \qquad \forall i = 0, \ldots, 2^m - 1 \\
P(\alpha^{i+1}) &= \sum_{j=0}^{t} P_{i+1,j}, \qquad \forall i = 0, \ldots, 2^m - 1 \\
&= \sum_{j=0}^{t} P_j \cdot (\alpha^{i+1})^j, \qquad \forall i = 0, \ldots, 2^m - 1 \\
&= \sum_{j=0}^{t} P_j \cdot (\alpha^i)^j \cdot \alpha^j, \qquad \forall i = 0, \ldots, 2^m - 1 \\
&= \sum_{j=0}^{t} P_{i,j} \cdot \alpha^j, \qquad \forall i = 0, \ldots, 2^m - 1.
\end{aligned}$$

La formule à retenir pour un monôme donné de P est donc :

$$P_{i+1,j} = P_{i,j} \cdot \alpha^j, \qquad \forall j \in [\![0, t]\!], \forall i \in [\![0, 2^m - 2]\!].$$

Cela explique pourquoi, dans l'algorithme de Chien (ligne 9 de l'Algorithme 41), on multiplie le coefficient a_j (correspondant à $P_{i,j}$) par α^j, afin d'obtenir la nouvelle valeur de a_j (correspondant à $P_{i+1,j}$).

Algorithme 41 Méthode de Chien pour obtenir les racines d'un polynôme

Entrée(s): $P \in \mathbb{F}_{2^m}[X]$ un polynôme et α un élément primitif de $\mathbb{F}_{2^m}$.
Sortie(s): *Racines* l'ensemble des racines de P sur $\mathbb{F}_{2^m}^*$ et r le cardinal de *Racines*.

1: $Racines \leftarrow \emptyset$
2: $r \leftarrow 0$
3: **Pour** $j = 0$ à t **faire**
4: $\quad a_j \leftarrow P_j$
5: **Fin pour**
6: **Pour** $i = 0$ à $2^m - 2$ **faire**
7: $\quad sum = a_0$
8: $\quad //sum$ contient l'évaluation progressive de α^i
9: $\quad$ **Pour** $j = 1$ à t **faire**
10: $\quad\quad a_j \leftarrow a_j \cdot \alpha^j$
11: $\quad\quad sum \leftarrow sum \oplus a_j$
12: $\quad$ **Fin pour**
13: $\quad$ **Si** $sum = 0$ **Alors**
14: $\quad\quad //$(c'est-à-dire si α^i est racine de P)
15: $\quad\quad Racines \leftarrow Racines \cup \{\alpha^i\}$
16: $\quad\quad //$(On ajoute l'élément α^i à l'ensemble *Racines*.)
17: $\quad\quad r \leftarrow r + 1$
18: $\quad$ **Fin si**
19: **Fin pour**
20: **Retourner** *Racines*, r

Cet algorithme nous permet d'avoir les r racines du polynôme P sur $\mathbb{F}_{2^m}$ avec nt multiplications (notées $\cdot$) et nt additions (notées $\oplus$) sur le corps fini $\mathbb{F}_{2^m}$, par rapport aux deux boucles **Pour** imbriquées dans l'Algorithme 41.

Remarque 8.1 *Le nombre de racines de P sur $\mathbb{F}_{2^m}$ est :*

r *si l'élément 0 n'est pas racine de P ;*

$r + 1$ *sinon.*

8.2 Implantation en VHDL de l'évaluation du PLE

Notre but était d'implanter en matériel cette étape potentiellement vulnérable afin d'observer la consommation de courant dynamiquement lors de son exécution, puisqu'elle est directement reliée au message initial (supposé inconnu d'un attaquant). Afin d'implanter l'évaluation du PLE en matériel, nous avons pu remarquer, par rapport à l'Algorithme 41, que deux opérations élémentaires sont nécessaires, à savoir l'addition de deux éléments dans un corps fini donné (notée $\oplus$, dont la table de vérité est rappelée dans le Tableau 8.1) et la multiplication de deux éléments du même corps (notée $\cdot$). L'addition est un simple OU EXCLUSIF, d'où sa notation. En revanche, la multiplication est un peu plus compliquée, c'est pourquoi nous la détaillons ci-après.

TABLEAU 8.1 – Table de vérité du OU EXCLUSIF sur $\mathbb{F}_2$

$\oplus$	0	1
0	0	1
1	1	0

8.2.1 Implantation matérielle de la multiplication de deux éléments dans un corps fini (*Galois Multiplier* noté GM)

Cette opération étant indispensable pour les algorithmes de décodage, nous commençons par présenter le bloc servant à effectuer la multiplication de deux éléments dans un corps fini donné. Pour cela, nous allons voir les éléments du corps fini comme des polynômes et utiliser le schéma de Horner (présenté dans l'Algorithme 40).
Soient α et β deux éléments du corps fini $\mathbb{F}_{2^m} \cong \mathbb{F}_2[x]/Q_m(x)$, où $Q_m(x)$ est un polynôme irréductible sur $\mathbb{F}_2$ de degré m. Le produit $\alpha \cdot \beta$, que l'on appelera γ, est aussi dans $\mathbb{F}_{2^m}$. Comme $\mathbb{F}_{2^m} \xrightarrow{\sim} \mathbb{F}_2[x]/Q_m(x)$, on peut écrire α comme le polynôme $\alpha(x) = \sum_{i=0}^{m-1} \alpha_i \cdot x^i$ et β comme le polynôme $\beta(x) = \sum_{i=0}^{m-1} \beta_i \cdot x^i$. Autrement dit, sous forme de vecteurs dont les coordonnées sont les coefficients du polynôme correspondant, on obtient $\alpha = (\alpha_{m-1}, \ldots, \alpha_1, \alpha_0)$ et $\beta = (\beta_{m-1}, \ldots, \beta_1, \beta_0)$.

Algorithme 42 Multiplication de deux éléments dans un corps fini

ENTRÉE(S): m le degré de l'extension de corps, $Q_m(x)$ le polynôme irréductible utilisé pour étendre $\mathbb{F}_2$ à $\mathbb{F}_{2^m}$, $\alpha(x)$ et $\beta(x)$ deux éléments du corps fini $\mathbb{F}_{2^m}$.
SORTIE(S): $\gamma(x) \in \mathbb{F}_{2^m}$ le produit de $\alpha(x)$ et $\beta(x)$.

1: $\gamma(x) = \alpha_{m-1} \cdot \beta(x)$
2: **Pour** i de $m-1$ **à** 1 **faire**
3: $\quad \gamma(x) = \gamma(x) \cdot x \oplus \alpha_{i-1} \cdot \beta(x) \oplus r_{m-1} \cdot Q_m(x)$
4: **Fin pour**
5: **Retourner** $\gamma(x)$.

On dit que le calcul se fait en mode série-parallèle : les coefficients de α sont pris un à un (série) tandis que β est gardé tout entier (parallèle).

Exemple (d'exécution de l'Algorithme 42) *Soient* $m = 5$, $Q_m(x) = x^5 + x^3 + x^2 + x + 1$ *(irréductible sur* $\mathbb{F}_2$*),* $\alpha(x) = x^4 + x^2 + x$ *(c'est-à-dire* $\alpha = (1, 0, 1, 1, 0)$*)*

et $\beta(x) = x^4 + x$ *(c'est-à-dire* $\alpha = (1,0,0,1,0)$*). On a :*

$m = 5$			x^5	x^4	x^3	x^2	x	1
	$Q_m(x)$		1	0	1	1	1	1
	$\alpha(x)$			1	0	1	1	0
	$\beta(x)$			1	0	0	1	0
initialisation	$\gamma(x)$			1	0	0	1	0
	$\gamma(x) \cdot x$		1	0	0	1	0	0
$i = 4$	$\alpha_3 \cdot \beta(x)$	$\oplus$		0	0	0	0	0
	$\gamma_4 \cdot Q_m(x)$		1	0	1	1	1	1
	$\gamma(x)$		0	0	1	0	1	1
	$\gamma(x) \cdot x$		0	1	0	1	1	0
$i = 3$	$\alpha_2 \cdot \beta(x)$	$\oplus$		1	0	0	1	0
	$\gamma_4 \cdot Q_m(x)$		0	0	0	0	0	0
	$\gamma(x)$		0	0	0	1	0	0
	$\gamma(x) \cdot x$		0	0	1	0	0	0
$i = 2$	$\alpha_1 \cdot \beta(x)$	$\oplus$		1	0	0	1	0
	$\gamma_4 \cdot Q_m(x)$		0	0	0	0	0	0
	$\gamma(x)$		0	1	1	0	1	0
	$\gamma(x) \cdot x$		1	1	0	1	0	0
$i = 1$	$\alpha_0 \cdot \beta(x)$	$\oplus$		0	0	0	0	0
	$\gamma_4 \cdot Q_m(x)$		1	0	1	1	1	1
fin	$\gamma(x)$		0	1	1	0	1	1

On obtient alors $\gamma(x) = \alpha(x) \cdot \beta(x) \equiv x^4 + x^3 + x + 1 \mod Q_m(x)$.

Deux versions de cette multiplication ont été implantées en VHDL pour les besoins de cette thèse. L'une considère le polynôme $Q_m(x)$ comme un paramètre supplémentaire et est donc générale tandis que l'autre possède un polynôme $Q_m(x)$ fixé, ce qui permet d'optimiser les calculs pour un corps fini donné. Ces deux versions ont été implantées sur le module Evariste II [Eva13] développé par l'équipe, contenant un FPGA de la famille Cyclone III de chez Altera [Alt12] (la référence exacte est EP3C25F256-C8). Les résultats donnés ci-après (Tableau 8.2) ont été obtenu grâce au logiciel Quartus II (version 9.1).

TABLEAU 8.2 – Résultats d'implantation des deux versions : Version 1 (avec Q_m comme variable) vs Version 2 (avec Q_m fixé)

Version 1 (avec Q_m comme variable)			Version 2 (avec Q_m fixé)		
m	Cellules logigues (LCELLs)	Fréquence (MHz)	m	Cellules logigues (LCELLs)	Fréquence (MHz)
5	35	348	5	21	453
6	54	275	6	29	405
7	68	207	7	37	455
8	90	175	8	56	355
9	117	172	9	59	414
10	140	135	10	83	321
11	178	140	11	87	411
12	209	118	12	123	276

Comme on peut le constater dans le Tableau 8.2, la solution la plus flexible (utilisant Q_m comme variable est toujours plus lente et utilise plus d'espace (nombre de cellules logiques) que l'autre version. Une implantation matérielle étant toujours un compromis temps-espace, le choix de la version dépendra de l'utilisation faite par la suite.

Maintenant que nous avons un bloc pour effectuer le produit de deux éléments dans un corps fini et que le problème de l'addition n'en était pas un, nous pouvons passer à un bloc plus gros, englobant ces opérations, afin d'effectuer l'évaluation d'un polynôme dans une implantation matérielle.

8.2.2 Implantation matérielle de l'évaluation du PLE

Pour l'implantation matérielle de l'évaluation d'un polynôme (dans notre cas le PLE), deux versions ont là encore été proposées. La première version correspond à l'Algorithme 39, aussi connu sous le nom de multiplication "scolaire". Quant à la deuxième, elle correspond à l'Algorithme 40 (sans combinaison avec l'Algorithme 41). Le choix d'utiliser un compteur a été fait pour parcourir les différentes valeurs du corps fini, puisque les éléments du support sont tous les éléments de $\mathbb{F}_{2^m}$ et peuvent donc être représentés comme tous les vecteurs possibles de longueur m sur $\mathbb{F}_2$ (comme vu précédemment). L'avantage d'exploiter la relation entre deux évaluations d'éléments consécutifs (au sens puissances consécutives d'un élément primitif) pour tous les éléments non nuls d'un corps fini (comme décrite pour l'Algorithme 41) n'est pas utilisé et cela reste donc une amélioration possible de ce travail.
Deux architectures différentes ont donc été décrites pour l'évaluation du PLE, noté $\sigma(X)$. La première, correspondant à l'Algorithme 39, est dite "de droite à gauche" (*Right to Left* [R2L] en Anglais), puisqu'elle correspond à une lecture dans le sens indiqué du polynôme :

$$\sigma(X) = \sigma_t X^t + \sigma_{t-1} X^{t-1} + \ldots + \sigma_1 X + \sigma_0 \qquad [R2L]$$

La seconde, correspondant à l'Algorithme 40, est dite "de gauche à droite" (*Left to Right* [L2R] en Anglais), puisqu'elle correspond à une lecture dans le sens indiqué du polynôme :

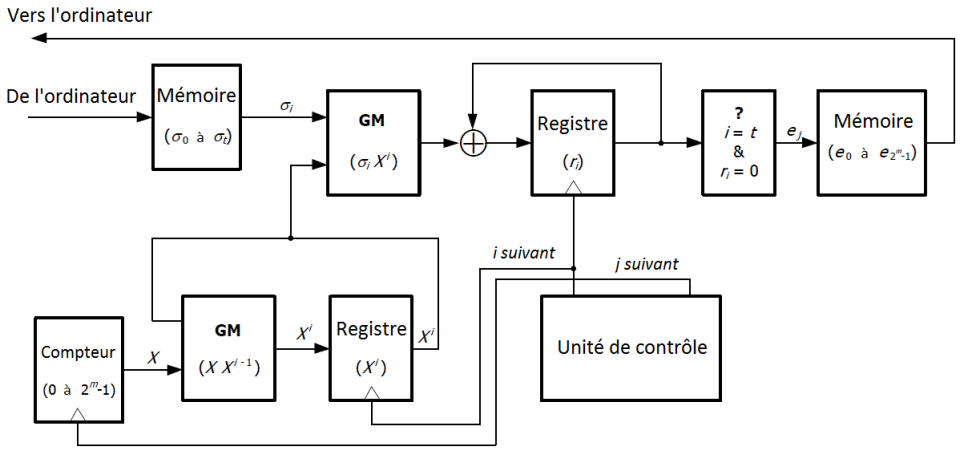
$$\sigma(X) = (((\sigma_t X + \sigma_{t-1})X + \ldots)X + \sigma_1)X + \sigma_0 \qquad [L2R]$$

Bien entendu, les deux architectures présentées ci-après, correspondant aux deux versions différentes, fournissent le même résultat, à savoir les bits du vecteur erreur $E \cdot \mathcal{P}^{-1}$. Ce vecteur erreur sera nommé E sur les schémas suivants, afin de ne pas alourdir les notations. Mais nous ne perdons pas de généralité puisque, à la permutation $\mathcal{P}$ près, E et $E \cdot \mathcal{P}^{-1}$ sont les mêmes vecteurs.

Évaluation [R2L]

La première architecture proposée pour évaluer un polynôme "de droite à gauche" [R2L] est présentée dans la Figure 8.1 et les résultats obtenus dans le Tableau 8.3 (où m est le degré de l'extension de corps et t le degré du polynôme de Goppa, autrement dit le nombre maximal d'erreurs corrigeables).

FIGURE 8.1 – Architecture de l'évaluation [R2L]



Comme on peut le constater sur la Figure 8.1, deux blocs de multiplications de deux éléments dans un corps fini (voir la Sous-section 8.2.1) sont nécessaires. De plus, deux registres sont utilisés afin de stocker les valeurs intermédiaires. Les coefficients de $\sigma(X)$ et les bits du vecteur erreur E sont enregistrés dans des mémoires embarquées séparées. La fréquence maximale de ce bloc est beaucoup plus faible que celle d'un bloc de multiplication seul à cause des accès mémoire. En effet, les coefficients σ_j sont lus à la volée à partir de la mémoire.

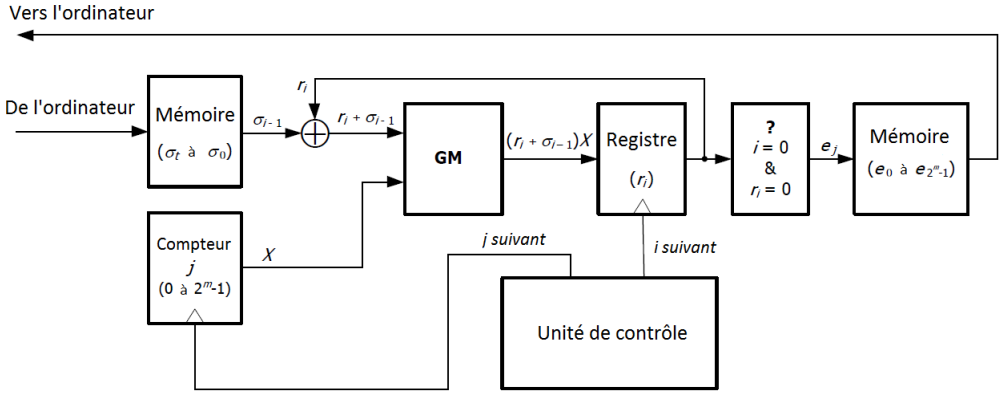
TABLEAU 8.3 – Résultats de l'implantation de l'évaluation [R2L]

m	t	Cellules logigues (LCELLs)	Fréquence (MHz)	Temps (μs)
5	3	287	120	1.07
6	5	311	118	3.25
7	10	341	95	14.82
11	50	511	78	1339.08

Évaluation [L2R]

La seconde architecture proposée pour évaluer un polynôme "de gauche à droite" [L2R] est présentée dans la Figure 8.2 et les résultats obtenus dans le Tableau 8.4 (où m est le degré de l'extension de corps et t le degré du polynôme de Goppa, autrement dit le nombre maximal d'erreurs corrigeables).

FIGURE 8.2 – Architecture de l'évaluation [L2R]



Comme on peut le constater sur la Figure 8.2, un seul bloc de multiplications de deux éléments dans un corps fini (voir la Sous-section 8.2.1) est ici nécessaire. En revanche, les mémoires sont les mêmes que dans la version précédente, mais les valeurs intermédiaires enregistrées dans les registres sont différentes.

TABLEAU 8.4 – Résultats de l'implantation de l'évaluation [L2R]

m	t	Cellules logigues (LCELLs)	Fréquence (MHz)	Temps (μs)
5	3	274	110	1.16
6	5	293	110	3.49
7	10	311	107	13.16
11	50	414	85	1228.80

Tentatives d'attaque

Notre but initial, avec une implantation matérielle de l'évaluation du PLE, était de reproduire l'attaque temporelle proposée dans [STM+08], puis améliorée dans [AHPT11], mais par analyse de consommation.
En proposant deux architectures différentes pour l'évaluation du PLE, nous espérions obtenir des fuites d'information différentes. Nous supposions que la méthode "scolaire" de multiplication serait attaquable tandis que la méthode de Horner pourrait nous servir de contre-mesure par exemple. Néanmoins, malgré des tentatives variées d'attaque sur ces implantations, nous n'avons pas réussi à récupérer de l'information secrète. Il semblerait donc qu'il y ait deux propositions d'implantation matérielle sécurisée pour l'évaluation du PLE.
Les différentes tentatives d'attaques que nous avions essayées sont les suivantes : Tout d'abord, l'acquisition multiple de traces et la synchronisation de celles-ci, pour une valeur en entrée donnée, afin de les superposer et de regarder la trace moyenne

(pour éliminer les bruits parasites). Seulement aucun motif n'est apparu. Nous avons ensuite testé le poids de Hamming de nos vecteurs. En effet, nous sommes partis de l'hypothèse que, comme le PLE s'annulait pour les éléments du support correspondant aux positions des bits non nuls dans le vecteur erreur, cela nous aiderait à déterminer les instants où un vecteur nul est manipulé. Mais là encore, ce fût un échec. Enfin, le dernier test effectué consistait à calculer la distance de Hamming entre deux valeurs consécutives lors de l'évaluation du PLE, mais là encore sans succès.
En cherchant une fuite d'information de la sorte, nous nous sommes retrouvés face au problème suivant : comment distinguer une évaluation nulle d'un polynôme d'une valeur intermédiaire (lors de l'évaluation) nulle. C'est ainsi qu'est venue la question sur la probabilité qu'un coefficient du PLE soit nul (que nous aborderons dans la section suivante). Bien sûr nous avons pensé à compter les cycles de l'évaluation afin de regarder le poids ou la distance de Hamming, mais la fin d'une évaluation s'effectuait en même temps que le début de la suivante, c'est pourquoi nos hypothèses d'attaque sont restées infructueuses.
Pour tester la robustesse de l'implantation matérielle de l'évaluation du PLE, nous avons pensé à insérer la description en VHDL dans le FPGA (Altera Cyclone V [Alt15]) de la carte d'évaluation EBV SoCrates [Soc], d'y faire appel pour ce calcul et d'effectuer le reste du déchiffrement dans le processeur ARM de la carte en utilisant un programme en C. Après quelques essais, il s'est avéré que la carte n'était pas adaptée pour une analyse de consommation, car sa propre consommation était un bruit trop important pour essayer d'extraire une quelconque information.

8.3 Dépendance entre les racines du PLE et ses coefficients

8.3.1 Relations entre coefficients et racines d'un polynôme

Voici de manière générale le problème que nous nous sommes posé :

Problème 8.1 *Soit $\{x_1, x_2, \ldots, x_t\} \subset \mathbb{F}_{2^m}$ un ensemble de t valeurs distinctes. Soit $\sigma(X)$ un polynôme unitaire scindé de l'anneau $\mathbb{F}_{2^m}[X]$ de degré t, avec toutes ses racines de multiplicité un (c'est-à-dire avec uniquement des racines simples), à savoir de la forme :*

$$\sigma(X) = \prod_{j=1}^{t}(X - x_j),$$

où les t valeurs x_j sont les racines distinctes du polynôme σ. Quelle est la probabilité que tous les coefficients du polynôme σ soient différents de zéro ?

Le reste de cette sous-section est inspiré de [Szp09, Chapitre 10, Section III]. On parlera du polynôme P de degré n pour les généralités et du polynôme σ de degré t pour notre cas particulier.

Définition 8.1 (Polynômes symétriques) *Soit $\mathfrak{S}_n$ le groupe symétrique, c'est-à-dire le groupe des bijections de l'ensemble $\{1, 2, \ldots, n\}$ dans lui-même. On définit une action de $\mathfrak{S}_n$ sur l'anneau $A[X_1, X_2, \ldots, X_n]$ en posant, pour $\tau \in \mathfrak{S}_n$ et $P \in A[X_1, X_2, \ldots, X_n]$,*

$$\tau P(X_1, X_2, \ldots, X_n) = P(X_{\tau(1)}, X_{\tau(2)}, \ldots, X_{\tau(n)}).$$

On dit que P est un polynôme symétrique en n variables (ou polynôme symétrique si le nombre de variables est fixé), si, pour tout $\tau \in \mathfrak{S}_n$, on a $\tau P = P$. Les polynômes $\sigma_j(x_1, x_2, \ldots, x_n)$, où $j = 1, 2, \ldots, n$, sont appelés les polynômes symétriques élémentaires en les indéterminées $x_1, x_2, \ldots, x_n$.

Exemple (Polynômes symétriques élémentaires) *Dans $A[x_1, x_2, \ldots, x_n]$, si on pose :*

$$\begin{aligned}
\sigma_1(x_1, x_2, \ldots, x_n) &= x_1 + x_2 + \ldots + x_n, \\
\sigma_k(x_1, x_2, \ldots, x_n) &= \sum_{1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant n} x_{i_1} x_{i_2} \cdots x_{i_k}, \\
\sigma_n(x_1, x_2, \ldots, x_n) &= x_1 x_2 \cdots x_n,
\end{aligned}$$

on trouve la formule suivante :

$$\prod_{i=1}^{n} (X - x_i) = X^n - \sigma_1 X^{n-1} + \ldots + (-1)^k \sigma_k X^{n-k} + \ldots + (-1)^n \sigma_n.$$

S'il n'y a pas d'ambiguïté sur les indéterminées x_j, on note simplement σ_j à la place de $\sigma_j(x_1, x_2, \ldots, x_n)$.

Remarque 8.2 *Les polynômes symétriques (élémentaires) sont également appelés fonctions symétriques (élémentaires) dans la littérature.*

Remarque 8.3 *Le problème qui nous intéresse ici consiste en fait à déterminer la probabilité que tous les σ_j soient différents de zéro, pour $j = 1, \ldots, t$ (Problème 8.1).*

Propriété 8.1 *Pour tout $\tau_1, \tau_2 \in \mathfrak{S}_n$, on a :*

$$\tau_1(\tau_2 P) = (\tau_1 \tau_2) P.$$

Propriété 8.2 *Il est clair que la somme et le produit de deux polynômes symétriques sont encore des polynômes symétriques.*

Proposition 8.1 *Soit $\mathbb{K}$ un corps. Soit $P \in \mathbb{K}[X]$, tel que $P(X) = a_0 + a_1 X + \ldots + a_n X^n$, avec $a_n \neq 0$. Si P est scindé dans $\mathbb{K}[X]$, c'est-à-dire s'il existe $\alpha_1, \alpha_2, \ldots, \alpha_n$ dans $\mathbb{K}$ tels que*

$$P(X) = a_n \prod_{j=1}^{n} (X - \alpha_j),$$

on a alors les relations

$$\frac{a_{n-1}}{a_n} = -\sigma_1(\alpha_1, \alpha_2, \ldots, \alpha_n), \quad \frac{a_{n-k}}{a_n} = (-1)^k \sigma_k(\alpha_1, \alpha_2, \ldots, \alpha_n), \quad \frac{a_0}{a_n} = (-1)^t \sigma_n(\alpha_1, \alpha_2, \ldots, \alpha_n).$$

Remarque 8.4 *Dans notre cas, la Proposition 8.1 est toujours vraie car le polynôme σ a été choisi comme étant unitaire et scindé dans $\mathbb{F}_{2^m}[X]$, on a donc bien :*

$$\sigma(X) = X^t - \sigma_1(x_1, x_2, \ldots, x_t) X^{t-1} + \ldots + (-1)^k \sigma_k(x_1, x_2, \ldots, x_t) X^{t-k} + \ldots + (-1)^t \sigma_t(x_1, x_2, \ldots, x_t).$$

Définition 8.2 (Poids d'un polynôme) *Soit A un anneau commutatif, soient $t_1, t_2, \ldots, t_n$ des indéterminées. Le poids du monôme $t_1^{c_1} t_2^{c_2} \cdots t_n^{c_n}$ est $c_1 + 2c_2 + \ldots + nc_n$. Le poids $w(Q)$ d'un polynôme $Q(t_1, t_2, \ldots, t_n) \in A[t_1, t_2, \ldots, t_n]$ est le maximum des poids des monômes qui interviennent dans Q.*

Théorème 8.1 (de structure des polynômes symétriques) *Soit A un anneau commutatif, soit $P(x_1, x_2, \ldots, x_n) \in A[x_1, x_2, \ldots, x_n]$ un polynôme symétrique de degré d. Alors il existe un polynôme $Q(t_1, t_2, \ldots, t_n) \in A[t_1, t_2, \ldots, t_n]$ de poids $w(Q) \leqslant d$ tel que*

$$P(x_1, x_2, \ldots, x_n) = Q(\sigma_1(x_1, x_2, \ldots, x_n), \sigma_2(x_1, x_2, \ldots, x_n), \ldots, \sigma_n(x_1, x_2, \ldots, x_n)).$$

Théorème 8.2 (Indépendance des polynômes symétriques élémentaires) *Les polynômes symétriques élémentaires sont algébriquement indépendants, c'est-à-dire qu'il n'existe pas de polynôme non nul $F(t_1, t_2, \ldots, t_n) \in A[t_1, t_2, \ldots, t_n]$ tel que :*

$$F(\sigma_1(x_1, x_2, \ldots, x_n), \sigma_2(x_1, x_2, \ldots, x_n), \ldots, \sigma_n(x_1, x_2, \ldots, x_n)) = 0.$$

De plus, le polynôme Q trouvé dans le Théorème 8.1 est unique et la sous-algèbre des éléments de $A[x_1, x_2, \ldots, x_n]$, qui sont invariants sous l'action de $\mathfrak{S}_n$ (c.f. Définition 8.1), est encore une algèbre de polynômes à n variables, c'est-à-dire une A-algèbre isomorphe à $A[t_1, t_2, \ldots, t_n]$:

$$(A[x_1, x_2, \ldots, x_n])^{\mathfrak{S}_n} = A[\sigma_1, \sigma_2, \ldots, \sigma_n]$$

Remarque 8.5 *Le but du décodage, c'est-à-dire la résolution de l'équation-clé, est en fait la recherche de cet isomorphisme d'algèbre.*

Définition 8.3 (Sommes de Newton) *On appelle $k^{\text{ième}}$ somme de Newton à n variables le polynôme symétrique $S_k = \sum_{j=1}^{n} x_j^k$. La $k^{\text{ième}}$ somme de Newton est homogène de degré k.*

Remarque 8.6 *D'après le Théorème 8.1, les S_k peuvent s'exprimer en fonction des σ_j. Ces expressions sont cependant assez compliquées, c'est pourquoi on préfère en général utiliser les formules suivantes, dues à Isaac Newton. Ces formules permettent d'exprimer de proche en proche les sommes de Newton S_k en fonction des polynômes symétriques élémentaires σ_j. De façon plus précise, pour calculer S_k, on calcule successivement, pour $j = 1, 2, \ldots, k$, les S_j en fonction des $\sigma_1, \sigma_2, \ldots, \sigma_j$.*

Théorème 8.3 *Soient k et $n \in \mathbb{N}$. En convenant que $S_0 = n$, on a les formules suivantes, pour $k \geqslant n$:*

$$S_k - \sigma_1 S_{k-1} + \sigma_2 S_{k-2} + \ldots + (-1)^n \sigma_n S_{k-n} = 0,$$

et pour $k \leqslant n$:

$$S_k - \sigma_1 S_{k-1} + \sigma_2 S_{k-2} + \ldots + (-1)^{k-1} \sigma_{k-1} S_1 + (-1)^k k \sigma_k = 0.$$

Exemples *On suppose que $n \geqslant 2$.*
La formule de Newton pour $k = 1$ est :

$$S_1 - \sigma_1 = 0. \tag{8.2}$$

La formule de Newton pour $k = 2$ est :

$$S_2 - \sigma_1 S_1 + 2\sigma_2 = 0. \tag{8.3}$$

En utilisant les Équations (8.5) *et* (8.6)*, on obtient les sommes de Newton suivantes :*

$$S_1 = \sigma_1 \text{ et } S_2 = \sigma_1^2 - 2\sigma_2. \tag{8.4}$$

Remarque 8.7 *Le but du décodage, c'est-à-dire la résolution de l'équation-clé, est de retrouver les polynômes symétriques élémentaires (les σ_j) à partir des sommes de Newton (les S_k).*

Exemples *Les exemples présentés ici sont les quatre tests [Szp09, p. 562].*

Test 1 : *On suppose que $n \geqslant 2$.*
La formule de Newton pour $k = 1$ est :

$$S_1 - \sigma_1 = 0. \tag{8.5}$$

La formule de Newton pour $k = 2$ est :

$$S_2 - \sigma_1 S_1 + 2\sigma_2 = 0. \tag{8.6}$$

En utilisant les Équations (8.5) *et* (8.6)*, on obtient :*

$$S_1 = \sigma_1 \text{ et } S_2 = \sigma_1 S_1 - 2\sigma_2 = \sigma_1^2 - 2\sigma_2. \tag{8.7}$$

Test 2 : *On suppose que $n \geqslant 4$.*
La formule de Newton pour $k = 3$ est :

$$S_3 - \sigma_1 S_2 + \sigma_2 S_1 - 3\sigma_3 = 0. \tag{8.8}$$

La formule de Newton pour $k = 4$ est :

$$S_4 - \sigma_1 S_3 + \sigma_2 S_2 - \sigma_3 S_1 + 4\sigma_4 = 0. \tag{8.9}$$

Test 3 : *On suppose que $n \geqslant 3$. En utilisant les Équations* (8.7) *et* (8.8)*, on obtient :*

$$\begin{aligned} S_3 &= \sigma_1 S_2 - \sigma_2 S_1 + 3\sigma_3 \\ &= \sigma_1^3 - 2\sigma_1\sigma_2 - \sigma_1\sigma_2 + 3\sigma_3 \\ &= \sigma_1^3 - 3\sigma_1\sigma_2 + 3\sigma_3. \end{aligned} \tag{8.10}$$

Test 4 : *On suppose que $n \geqslant 4$. En utilisant les Équations* (8.7) *et* (8.10)*, on obtient :*

$$\begin{aligned} S_4 &= \sigma_1 S_3 - \sigma_2 S_2 + \sigma_3 S_1 - 4\sigma_4 \\ &= \sigma_1^4 - 3\sigma_1^2\sigma_2 + 3\sigma_1\sigma_3 - \sigma_1^2\sigma_2 + 2\sigma_2^2 + \sigma_1\sigma_3 - 4\sigma_4 \\ &= \sigma_1^4 - 4\sigma_1^2\sigma_2 + 4\sigma_1\sigma_3 + 2\sigma_2^2 - 4\sigma_4. \end{aligned} \tag{8.11}$$

Maintenant que la relation entre les coefficients d'un polynôme et ses racines est posée, passons au problème qui nous intéresse afin de donner des bornes sur la probabilité qu'aucun des coefficients de ce polynôme ne soit nul.

8.3.2 Bornes de la probabilité

Les probabilités données dans cette sous-section seront notées $\mathbb{P}$. Afin de répondre à la question posée dans le Problème 8.1, étant donné le corps $\mathbb{F}_{2^m}$, on définit les univers Ω_t et Ω'_t comme étant :

$$\begin{aligned}\Omega_t &= \{\text{polynômes à coefficients dans } \mathbb{F}_{2^m}\text{, de degré } t\text{, unitaires,}\\ &\quad\ \text{ayant leurs racines dans } \mathbb{F}_{2^m} \text{ de multiplicité un}\}\end{aligned}$$

$$\Omega'_t = \{\text{parties de } \mathbb{F}_{2^m} \text{ à } t \text{ éléments}\}.$$

D'après ce que l'on a vu dans la Sous-section 8.3.1, il existe une bijection ensembliste entre Ω_t et Ω'_t, que l'on notera $\mathscr{R}_t$. En effet, la fonction

$$\begin{aligned}\mathscr{R}_t : \Omega_t &\to \Omega'_t\\ \sigma &\mapsto \mathscr{R}_t(\sigma) = \{\text{racines de } \sigma\}\end{aligned}$$

est bien bijective puisque pour toute partie $\{x_1, x_2, \ldots, x_t\}$ de $\mathbb{F}_{2^m}$ à t éléments, il existe un unique polynôme σ à coefficients dans $\mathbb{F}_{2^m}$ tel que $\{x_1, x_2, \ldots, x_t\}$ soit l'ensemble de toutes ses racines.
Comme $\Omega_t \simeq \Omega'_t$ et que la loi de probabilité $\mathbb{P}_{\Omega_t}$ sur Ω_t est une loi uniforme, on en déduit qu'il existe une loi de probabilité $\mathbb{P}_{\Omega'_t}$ sur Ω'_t qui est uniforme. Pour simplifier les notations, on notera la loi de probabilité $\mathbb{P}$ par la suite et on travaillera sur l'espace Ω'_t.

Rappels de probabilités élémentaires

Commençons par rappeler certaines propriétés de probabilités que nous utiliserons par la suite dans notre démonstration (sans y faire nécessairement référence) :

Soient A et B deux événements distincts. On a :

1. $\mathbb{P}(A|B) = 1 - \mathbb{P}(A^c|B)$,
2. $\mathbb{P}(A \cap B) = \mathbb{P}(A|B) \times \mathbb{P}(B)$,
3. $\mathbb{P}(A) \leq \mathbb{P}(B)$ pour $A \subset B$,
4. $\mathbb{P}(A \cap B\ |C) = \mathbb{P}(A|B \cap C) \times \mathbb{P}(B|C)$.

Notations

Rappelons certaines notations afin d'éviter les ambiguïtés :

polynôme symétrique élémentaire : $\sigma_j = \sigma_j(x_1, x_2, \ldots, x_t) = \sum\limits_{1 \leqslant i_1 < i_2 < \ldots < i_j \leqslant t} x_{i_1} x_{i_2} \ldots x_{i_j}$.

$k^{\text{ième}}$ **somme de Newton :** $S_k = \sum\limits_{i=1}^{t} x_i^k = x_1^k + x_2^k + \ldots + x_t^k$.

Nous noterons $n = 2^m$ la cardinalité du corps fini $\mathbb{F}_{2^m}$.

Problème des coefficients en fonction des racines

Le Problème 8.1 peut être reformuler de la façon suivante : Étant donné $\sigma(X) = \prod\limits_{j=1}^{t}(X - x_j) \in \mathbb{F}_{2^m}[X]$ un polynôme à t racines distinctes dans $\mathbb{F}_{2^m}$ écrit sous forme

factorisé, on cherche à savoir avec quelle probabilité tous ses coefficients sont non nuls sous forme développée, à savoir :

$$\sigma(X) = X^t \oplus \sigma_1^t X^{t-1} \oplus \ldots \oplus \sigma_k^t X^{t-k} \oplus \ldots \oplus \sigma_{t-1}^t X \oplus \sigma_t^t.$$

Ceci nous mène à l'énoncé suivant :

Problème 8.2 *On cherche à calculer :*

$$\mathbb{P}(\sigma_k^t \neq 0, \forall k \in [\![1,t]\!]) = \mathbb{P}(\sigma_1^t \neq 0 \cap \sigma_2^t \neq 0 \cap \ldots \cap \sigma_k^t \neq 0 \cap \ldots \cap \sigma_t^t \neq 0).$$

Pour répondre à ce problème, on cherche à étudier la distribution des polynômes symétriques élémentaires σ_j^t.

Remarque 8.8 *La probabilité recherchée dans le Problème 8.2 ne dépend pas du choix d'une base sur $\mathbb{F}_2$ et donc de la représentation des éléments du corps $\mathbb{F}_{2^m}$ comme des vecteurs de $\mathbb{F}_2^m$.*

Afin de fournir une réponse au Problème 8.2, nous proposons de le subdiviser en problèmes plus simples. Puisque nous voulons étudier une intersection de t événements, nous étudierons dans la suite de cette section ces t événements individuellement. Nous commencerons par les cas extrêmes correspondant au produit et à la somme des racines (les cas simples), puis nous traiterons les autres cas de manière plus générale.

Étude de certains coefficients particuliers

Propriété 8.3 *Soient t un entier plus petit que n et $\sigma(X) \in \Omega_t$. La probabilité que le produit de ses t racines (le coefficient constant) noté σ_t^t soit nul est :*

$$\mathbb{P}(\sigma_t^t = 0) = \frac{t}{n}.$$

Preuve *Soient t un entier plus petit que n et $\sigma(X) \in \Omega_t$. Le produit des racines, noté σ_t^t, est nul si et seulement si l'un des x_i (1 élément parmi t) est l'élément nul dans $\mathbb{F}_{2^m}$ (ensemble à n éléments). On obtient donc :*

$$\begin{aligned}
\mathbb{P}(\sigma_t^t = 0) &= \frac{\binom{n-1}{t-1}}{\binom{n}{t}} \\
&= \frac{\frac{(n-1)!}{(t-1)!(n-t)!}}{\frac{n!}{t!(n-t)!}} \\
&= \frac{\frac{(n-1)!}{(t-1)!}}{\frac{n!}{t!}} \\
&= \frac{(n-1)!}{(t-1)!} \times \frac{t!}{n!} \\
&= \frac{t}{n}.
\end{aligned}$$

□

Lemme 8.1 *Soient t un entier plus petit que n et $\sigma(X) \in \Omega_t$. La probabilité que la somme des produits de $(t-1)$ racines noté σ_{t-1}^t soit nul sachant que le produit des t racines est nul est :*

$$\mathbb{P}(\sigma_{t-1}^t = 0 \cap \sigma_t^t = 0) = 0.$$

Preuve *Soient t un entier plus petit que n et $\sigma(X) \in \Omega_t$. Supposons que $\sigma_t^t = 0$. Alors $0 \in \mathscr{R}_t(\sigma)$. Rappelons que σ_{t-1}^t est donné par :*

$$\sigma_{t-1}^t = \sum_{i=1}^{t} \prod_{\substack{j=1 \\ j \neq i}}^{t} x_j, \qquad \text{avec } x_j \in \mathscr{R}_t(\sigma). \tag{8.12}$$

Supposons que σ_t^t soit nul. Alors il existe $j \in [\![1, t]\!]$ tel que $x_j = 0$. Donc dans l'Équation 8.12, tous les produits sont nuls sauf un, celui ne contenant pas 0. En effet, le coefficient σ_{t-1}^t correspond aux t produits possibles de $(t-1)$ racines parmi les t éléments de $\mathscr{R}_t(\sigma)$. Sans perdre de généralité, supposons que $x_t = 0$. Donc σ_{t-1}^t devient de la forme $\prod_{i=1}^{t-1} x_i$, avec $x_i \in \mathscr{R}_t(\sigma) \backslash \{0\}$. Or, $\prod_{i=1}^{t-1} x_i$ est un élément de $\mathbb{F}_{2^m}$, donc si $\prod_{i=1}^{t-1} x_i = 0$, alors cela implique que 0 est une racine double de $\sigma(X)$, ce qui contredit l'hypothèse que $\sigma(X)$ n'a que des racines simples car $\sigma(X) \in \Omega_t$.
□

Propriété 8.4 *Soit $\sigma(X) \in \Omega_t$. La probabilité que la somme des produits de $(t-1)$ racines noté σ_{t-1}^t soit nul est :*

$$\begin{aligned} \mathbb{P}(\sigma_{t-1}^t = 0) &= \underbrace{\mathbb{P}(\sigma_{t-1}^t = 0 \cap \sigma_t^t = 0)}_{=0} + \mathbb{P}(\sigma_{t-1}^t = 0 \cap \sigma_t^t \neq 0) \\ &= \mathbb{P}(\sigma_{t-1}^t = 0 \cap \sigma_t^t \neq 0). \end{aligned}$$

Preuve *En utilisant le lemme précédent, le calcul se simplifie naturellement.*
□

Propriété 8.5 *Soit $\sigma(X) \in \Omega_t$. Soit $k \in [\![1, t]\!]$. En séparant les termes dans σ_k^t, on obtient la relation suivante sur les coefficients :*

$$\begin{aligned} \sigma_k^t &= \sum_{1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant t} x_{i_1} x_{i_2} \cdots x_{i_k} \\ &= \sum_{1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant t-1} x_{i_1} x_{i_2} \cdots x_{i_k} + \sum_{1 \leqslant i_1 < i_2 < \ldots < i_k = t} x_{i_1} x_{i_2} \cdots x_{i_k}. \end{aligned}$$

En d'autres termes, on a la relation :

$$\forall 1 \leqslant k \leqslant t, \qquad \sigma_k^t = \sigma_k^{t-1} + x_t \cdot \sigma_{k-1}^{t-1}. \tag{8.13}$$

Approche théorique

Donnons maintenant un lemme général qui nous fournit une relation de récurrence entre deux distributions consécutives de variables. Le résultat nous donnera une allusion directe sur comment la distribution des sommes sera résolue.

Chaque fois que nous l'utiliserons, nous donnerons seulement les paramètres impliqués dans la relation.

Lemme 8.2 *Soient $(x_1, x_2, \ldots, x_t)$ les t racines dans $\mathbb{F}_{2^m}$ et $\alpha_i \in \mathbb{F}_{2^m}$, avec $1 \leqslant i \leqslant n$. Alors on a :*

$$\mathbb{P}(x_t = \alpha_i) = \frac{1}{n-t+1} \times \Big[1 - (t-1) \cdot \mathbb{P}(x_{t-1} = \alpha_i)\Big].$$

Preuve *L'idée est que lorsque un α_i a déjà été tiré de l'urne (c'est-à-dire qu'il existe une variable x_j qui, déjà choisie, est égale à α_i), la probabilité s'avérera être nulle puisque les variables x_j sont toutes différentes les unes des autres.*
Commençons notre preuve par séparer la probabilité de l'événement $x_t = \alpha_i$ en deux événements complémentaires, à savoir d'une part α_i a déjà été choisi pour un x_j avec $1 \leqslant j < t$, et d'autre part α_i n'a pas encore été choisi. Cela nous donne la probabilité suivante :

$$\begin{aligned}\mathbb{P}(x_t = \alpha_i) &= \underbrace{\mathbb{P}(x_t = \alpha_i \cap \exists j \in \{1, \ldots, t-1\},\ x_j = \alpha_i)}_{=0 \text{ car les racines sont distinctes}} \\ &\quad + \mathbb{P}(x_t = \alpha_i \cap \forall\, j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i)\end{aligned}$$

En utilisant la propriété de l'intersection de deux événements sachant un troisième événement, nous avons :

$$\begin{aligned}\mathbb{P}(x_t = \alpha_i) &= \mathbb{P}(x_t = \alpha_i \cap \forall\, j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i) \\ &= \mathbb{P}(x_t = \alpha_i |\ \forall j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i) \\ &\quad \times \mathbb{P}(\forall j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i)\end{aligned}$$

Comme x_t doit être différent de tous les x_j, avec $1 \leqslant j \leqslant t-1$, il y a $n-t+1$ choix possibles pour x_t, sachant que le bon est $x_t = \alpha_i$. Nous obtenons donc pour le premier terme du produite des probabilités :

$$\mathbb{P}(x_t = \alpha_i |\ \forall j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i) = \frac{1}{n-t+1}.$$

Regardons maintenant le second terme du produit des probabilités. Il est donné par :

$$\mathbb{P}(\forall j \in \{1, \ldots, t-1\},\ x_j \neq \alpha_i) = 1 - \mathbb{P}(\exists j \in \{1, \ldots, t-1\},\ x_j = \alpha_i).$$

Comme la distribution est uniforme sur les variables x_j, la probabilité que n'importe quel x_j soit égal à α_i, pour $1 \leqslant j \leqslant t-1$, vaut $(t-1)$ fois la probabilité que $x_{t-1} = \alpha_i$. Nous obtenons donc le résultat suivant :

$$\mathbb{P}\left(\exists j \in \{1, \ldots, t-1\},\ x_j = \alpha_i\right) = (t-1) \times \mathbb{P}(x_{t-1} = \alpha_i).$$

Finalement le produit de ces deux termes nous donne bien :

$$\mathbb{P}\left(x_t = \alpha_i\right) = \frac{1}{n-t+1} \times \Big[1 - (t-1) \cdot \mathbb{P}(x_{t-1} = \alpha_i)\Big].$$

□

En utilisant ce lemme et les propriétés énoncées précédemment, nous donnons ci-après la probabilité exacte que le coefficient σ_1^t soit nul en fonction de la parité du nombre de racines t, puis bornons celles pour les $t-2$ autres coefficients σ_j^t via les trois propositions suivantes.

Trois propositions importantes Commençons par le premier coefficient : la somme des racines.

Proposition 8.2 *Soient t éléments deux à deux distincts de $\mathbb{F}_{2^m}$, notés $(x_1, x_2, \ldots, x_t)$. La probabilité que le coefficient σ_1^t correspondant à la somme des t racines soit nul est :*

Pour t pair :

$$\mathbb{P}(\sigma_1^t = 0) = \sum_{i=0}^{\frac{t}{2}-2} (-1)^i \times \frac{1}{t-(2i+1)} \times \prod_{j=0}^{i} \frac{t-(2j+1)}{n-t+2j+1}.$$

Pour t impair :

$$\mathbb{P}(\sigma_1^t = 0) = \frac{1}{n}.$$

Preuve *En utilisant l'Équation (8.13), on a :*

$$\sigma_1^t = 0 \Leftrightarrow \sigma_1^{t-1} = x_t \cdot \underbrace{\sigma_0^{t-1}}_{\substack{=1 \\ \textit{par construction}}}.$$

Comme $\sigma_1^t = 0 \Leftrightarrow \sigma_1^{t-1} = x_t$, le calcul de la probabilité $\mathbb{P}(\sigma_1^t = 0)$ revient à calculer la probabilité $\mathbb{P}(\sigma_1^{t-1} = x_t)$. De plus, comme σ_1^{t-1} est un élément de $\mathbb{F}_{2^m}$, nous pouvons appliquer le Lemme 8.2. Nous obtenons alors la relation suivante :

$$\mathbb{P}(\sigma_1^{t-1} = x_t) = \frac{1}{n-t+1} \times \left[1 - (t-1) \cdot \mathbb{P}(\sigma_1^{t-1} = x_{t-1})\right].$$

Ce résultat nous permet de donner la distribution exacte pour $\mathbb{P}(\sigma_1^t = 0)$ en utilisation l'Équation (8.13) pour $k = 1$:

$$\mathbb{P}(\sigma_1^t = 0) = \frac{1}{n-t+1} \times \left[1 - (t-1) \times \mathbb{P}(\sigma_1^{t-2} = 0)\right]$$

La preuve sera donnée en trois étapes :

1. *Jetons un œil aux premières valeurs pour $\mathbb{P}(\sigma_1^t = 0)$. Nous observons comment la récurrence fonctionne et découvrons que pour les t impairs, le résultat est le même.*
2. *Les premières étapes :*
   - *Pour $t = 1$, nous avons : $\mathbb{P}(\sigma_1^1 = 0) = \frac{1}{n}$.*
   - *Pour $t = 3$, nous avons : $\mathbb{P}(\sigma_1^3 = 0) = \frac{1}{n-t+1} \times \left[1 - \frac{-1}{n}\right] = \frac{1}{n}$.*
   - $\ldots$
   - *Pour $t = 2p+1$, où p est un entier pair, nous avons : $\mathbb{P}(\sigma_1^{2p+1} = 0) = \frac{1}{n}$.*
   - *Pour $t = 2$, nous avons : $\mathbb{P}(\sigma_1^2 = 0) = 0$ à cause de la condition $\mathscr{R}_t$ (x_1 et x_2 ne peuvent pas être égaux).*
   - *Pour $t = 4$, nous avons : $\mathbb{P}(\sigma_1^4 = 0) = \frac{1}{n-3}$.*
   - *Pour $t = 6$, nous avons : $\mathbb{P}(\sigma_1^6 = 0) = \frac{1}{n-5} - \frac{1}{n-5} \times \frac{5}{n-3}$.*

3. *Preuve de la formule :*

$$\mathbb{P}(\sigma_1^t = 0) - \frac{1}{n-t+1} + \frac{t-1}{n-t+1}\mathbb{P}(\sigma_1^{t-2} = 0) = 0$$

Pour les valeurs impaires de t, nous verrons que la preuve est très facile : Pour t impair : $\mathbb{P}(\sigma_1^t = 0) = \frac{1}{n}$.

$$\mathbb{P}(\sigma_1^t = 0) - \frac{1}{n-t+1} + \frac{t-1}{n-t+1}\mathbb{P}(\sigma_1^{t-2} = 0) = \frac{1}{n} - \frac{1}{n-t+1} + \frac{t-1}{n-t+1} \times \frac{1}{n} = 0.$$

Pour t pair : $\mathbb{P}(\sigma_1^t = 0) = \sum_{i=0}^{\frac{t}{2}-2} (-1)^i \frac{1}{t-(2i+1)} \times \prod_{j=0}^{i} \frac{t-(2j+1)}{n-t+2j+1}$

Donc : $\mathbb{P}(\sigma_1^t = 0) = \frac{1}{n-1} \times \frac{t-1}{n-t+1} + \sum_{i=1}^{\frac{t}{2}-2} (-1)^i \frac{1}{t-(2i+1)} \times \prod_{j=0}^{i} \frac{t-(2j+1)}{n-t+2j+1}$.

Nous obtenons :

$$\mathbb{P}(\sigma_1^t = 0) - \frac{1}{n-t+1} = (-1) \times \frac{t-1}{n-t+1} \sum_{i=1}^{\frac{t}{2}-2} (-1)^{i-1} \frac{1}{t-(2i+1)} \times \prod_{j=1}^{i} \frac{t-(2j+1)}{n-t+2j+1}.$$

Nous appliquons deux changements de variables : d'abord nous appliquons $j \to j'+1$, puis $i \to i'+1$. Le premier changement mène à ceci :

$$\prod_{j=1}^{i} \frac{t-(2j+1)}{n-t+2j+1} = \prod_{j'=0}^{i-1} \frac{t-(2j'+3)}{n-t+2j'+3}$$

Après le second, nous obtenons :

$$\sum_{i=1}^{\frac{t}{2}-2} \frac{(-1)^{i-1}}{t-(2i+1)} \times \prod_{j'=0}^{i-1} \frac{t-(2j'+3)}{n-t+2j'+3} = \sum_{i'=0}^{\frac{t}{2}-3} \frac{(-1)^{i'}}{t-(2i'+3)} \times \prod_{j'=0}^{i'} \frac{t-(2j'+3)}{n-t+2j'+3}$$

Finalement, après ré-écriture, nous obtenons la relation voulue :

$$\mathbb{P}(\sigma_1^{t-2} = 0) = \sum_{i'=0}^{\frac{t}{2}-3} \frac{(-1)^{i'}}{t-(2i'+3)} \times \prod_{j'=0}^{i'} \frac{t-(2j'+3)}{n-t+2j'+3}$$

□

Nous avons donc la formule exacte pour le premier coefficient. Pour les autres coefficients, nous donnerons des bornes supérieure et inférieure. Dans le but d'obtenir ces bornes, nous détaillons deux importants lemmes que nous appelons Lemme 8.3 et Lemme 8.4. Le premier donne la distribution de deux coefficients consécutifs. Nous utilisons les notations suivantes dans cette sous-section :

Étude de manière générale des coefficients

Notations 8.1 1.

$$\mathcal{E}_{k,t}^{k-1,t} = \mathbb{P}(\sigma_k^t = 0 \cap \sigma_{k-1}^t = 0)$$

2.

$$\mathcal{E}_{k,t}^{\overline{k-1,t-1}} = \mathbb{P}(\sigma_k^t = 0 \cap \sigma_{k-1}^{t-1} \neq 0)$$

3.

$$\mathbb{P} = \mathbb{P}(\forall i \in [\![1,t]\!],\ \sigma_i^t \neq 0)$$

La preuve commence avec la première notation et utilise des relations basiques de probabilité ainsi que certaines propriétés sur le corps fini.

Lemme 8.3 *On a :*

$$\mathcal{E}_{k,t}^{k-1,t} \leq 2 \times \frac{k-1}{(n-t+1)^2}.$$

Preuve *Commençons par donner la relation de récurrence pour chaque variable impliquée dans l'équation :*

$$\begin{aligned} \sigma_{k-1}^t = 0 &\Leftrightarrow \sigma_{k-2}^{t-1} \times x_t = \sigma_{k-1}^{t-1} \\ \sigma_k^t = 0 &\Leftrightarrow \sigma_{k-1}^{t-1} \times x_t = \sigma_k^{t-1} \end{aligned}$$

1. *Trouver la récurrence : Nous commençons par séparer en deux probabilités différentes :*

$$\begin{aligned} \mathcal{E}_{k,t}^{k-1,t} &= \underbrace{\mathbb{P}\left(\sigma_{k-1}^t = 0 \cap \sigma_k^t = 0 \cap \sigma_{k-2}^{t-1} = 0\right)}_{\leq \mathcal{E}_{k-1,t-1}^{k-2,t-1}} \\ &\quad + \mathbb{P}\left(\sigma_{k-1}^t = 0 \cap \sigma_k^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0\right) \end{aligned}$$

Nous obtenons la probabilité suivante :

$$\begin{aligned} \mathcal{E}_{k,t}^{k-1,t} - \mathcal{E}_{k-1,t-1}^{k-2,t-1} &\leq \mathbb{P}\left(\sigma_{k-1}^t = 0 \cap \sigma_k^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0\right) \\ &\leq \mathbb{P}\left(\sigma_{k-1}^t = 0 \cap \sigma_k^t = 0 | \sigma_{k-2}^{t-1} \neq 0\right) \times \mathbb{P}\left(\sigma_{k-2}^{t-1} \neq 0\right) \\ &\leq \mathbb{P}\left(\sigma_k^t = 0 | \sigma_{k-1}^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0\right) \\ &\quad \times \mathbb{P}\left(\sigma_{k-1}^t = 0 | \sigma_{k-2}^{t-1} \neq 0\right) \times \left(1 - \mathbb{P}(\sigma_{k-2}^{t-1} = 0)\right) \end{aligned}$$

Nous détaillons chaque partie et donnons l'inégalité correspondante dans chaque cas. Nous commençons par considérer la probabilité :

$$\mathbb{P}\left(\sigma_{k-1}^t = 0 | \sigma_{k-2}^{t-1} \neq 0\right) = \mathbb{P}\left(x_t = \frac{\sigma_{k-1}^{t-1}}{\sigma_{k-2}^{t-1}} | \sigma_{k-2}^{t-1} \neq 0\right)$$

Nous pouvons ici appliquer le Lemme 1 à $x_t = \frac{\sigma_{k-1}^{t-1}}{\sigma_{k-2}^{t-1}}$, *puisque nous sommes sous la condition* $\sigma_{k-2}^{t-1} \neq 0$. *Nous obtenons donc :*

$$\mathbb{P}\left(x_t = \frac{\sigma_{k-1}^{t-1}}{\sigma_{k-2}^{t-1}} | \sigma_{k-2}^{t-1} \neq 0\right) = \frac{1}{n-t+1} \times \left[1 - (t-1) \times \mathbb{P}\left(\frac{\sigma_{k-1}^{t-1}}{\sigma_{k-2}^{t-1}} = x_{t-1} | \sigma_{k-2}^{t-1} \neq 0\right)\right]$$

Nous ne chercherons pas à calculer l'autre partie, mais simplement à la borner :

$$\mathbb{P}\left(\sigma_k^t = 0 | \sigma_{k-1}^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0\right) = \mathbb{P}\left(\left(\sigma_{k-1}^{t-1}\right)^2 = \sigma_{k-2}^{t-1} \times \sigma_k^{t-1} | \sigma_{k-1}^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0\right)$$

Étant donnée l'intersection $\sigma_{k-1}^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0$,
nous avons $\left(\sigma_{k-1}^{t-1}\right)^2 = \sigma_{k-2}^{t-1} \times \sigma_k^{t-1}$, *qui est équivalente à :*

$$\left[\left(\sigma_{k-2}^{t-2}\right)^2 + \sigma_{k-3}^{t-2} \times \sigma_{k-1}^{t-2}\right] x_{t-1}^2 + \left[\sigma_{k-3}^{t-2} \times \sigma_k^{t-2} + \sigma_{k-1}^{t-2} \times \sigma_{k-2}^{t-2}\right] x_{t-1} + \left(\sigma_{k-1}^{t-2}\right)^2 + \sigma_{k-2}^{t-2} \times \sigma_k^{t-2} = 0.$$

Cela représente une équation du second degré à coefficients dans $\mathbb{F}_{2^m}$. Donc le nombre maximal de bons choix pour x_t vaut deux. Cela signifie que la probabilité peut être bornée par :

$$\mathbb{P}(\sigma_k^t = 0 | \sigma_{k-1}^t = 0 \cap \sigma_{k-2}^{t-1} \neq 0) \leq \frac{2}{n-t+1}$$

Finalement nous avons :

$$\begin{aligned}\mathcal{E}_{k,t}^{k-1,t} - \mathcal{E}_{k-1,t-1}^{k-2,t-1} &\leq \frac{2}{n-t+1} \times \frac{1}{n-t+1} \\ &\times \left[1 - (t-1) \times \mathbb{P}\left(\frac{\sigma_{k-1}^{t-1}}{\sigma_{k-2}^{t-1}} = x_{t-1} | \sigma_{k-2}^{t-1} \neq 0\right)\right] \times (1 - \mathbb{P}(\sigma_{k-2}^{t-1} = 0))\end{aligned}$$

Donc nous obtenons :

$$\mathcal{E}_{k,t}^{k-1,t} - \mathcal{E}_{k-1,t-1}^{k-2,t-1} \leq \frac{2}{(n-t+1)^2}$$

2. *Dans le but de finir la preuve du Lemme 8.3, la dernière étape est de donner la relation finale en utilisant une récurrence : Comme $\sigma_0^t = 1$, nous avons $\mathcal{E}_{1,t}^{0,t} = 0, \quad \forall\ t < n$. Nous avons aussi prouvé que $\mathcal{E}_{t,t}^{t-1,t} = 0$. Poser ces relations nous permettra de donner le résultat final.*

$$\begin{aligned}\mathcal{E}_{k,t}^{k-1,t} - \mathcal{E}_{k-1,t-1}^{k-2,t-1} &\leq \frac{2}{(n-t+1)^2} \\ &\vdots \\ \mathcal{E}_{t,t-k+2}^{t-1,t-k+2} - \mathcal{E}_{1,t-k+1}^{0,t-k+1} &\leq \frac{2}{(n-t+k-1)^2} \leq \frac{2}{(n-t+1)^2}\end{aligned}$$

Nous obtenons donc l'inégalité suivante :

$$\mathcal{E}_{k,t}^{k-1,t} \leq \frac{2}{(n-t+1)^2} \times (k-1), \qquad \forall k \in \{2, \ldots, t-1\}.$$

□

Continuons et donnons la distribution pour la seconde.

Lemme 8.4 *Avec les notations précédentes, nous avons :*

$$\frac{1}{n-t+1} \times \left[1 - \frac{2}{n-t+1} \times (t-1)\right] \times \left(1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0)\right) \leq \mathcal{E}_{k,t}^{\overline{k-1,t-1}} \leq \frac{1}{n-t+1}.$$

Preuve

$$\begin{aligned}\mathcal{E}_{k,t}^{\overline{k-1,t-1}} &= \mathbb{P}\left(\sigma_k^t = 0 \cap \sigma_{k-1}^{t-1} \neq 0\right) \\ &= \mathbb{P}\left(\sigma_k^t = 0 | \sigma_{k-1}^{t-1} \neq 0\right) \times \mathbb{P}\left(\sigma_{k-1}^{t-1} \neq 0\right) \\ &= \mathbb{P}\left(\sigma_k^t = 0 | \sigma_{k-1}^{t-1} \neq 0\right) \times \left(1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0)\right) \\ &= \mathbb{P}\left(x_t = \frac{\sigma_k^{t-1}}{\sigma_{k-1}^{t-1}} | \sigma_{k-1}^{t-1} \neq 0\right) \times \left(1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0)\right)\end{aligned}$$

Nous utilisons ici le Lemme 8.2 avec $x_t = \frac{\sigma_k^{t-1}}{\sigma_{k-1}^{t-1}}$ *puisque* $\sigma_{k-1}^{t-1} \neq 0$. *Nous obtenons le résultat suivant :*

$$\mathbb{P}\left(x_t = \frac{\sigma_k^{t-1}}{\sigma_{k-1}^{t-1}} | \sigma_{k-1}^{t-1} \neq 0\right) = \frac{1}{n-t+1} \times \left[1 - (t-1) \times \mathbb{P}\left(\frac{\sigma_k^{t-1}}{\sigma_{k-1}^{t-1}} = x_{t-1} | \sigma_{k-1}^{t-1} \neq 0\right)\right]$$

Bornons la probabilité :

$$\mathbb{P}\left(\frac{\sigma_k^{t-1}}{\sigma_{k-1}^{t-1}} = x_{t-1} | \sigma_{k-1}^{t-1} \neq 0\right) = \mathbb{P}\left(\left(\sigma_{k-2}^{t-2}\right)^2 x_{t-1}^2 = \sigma_k^{t-2} | \sigma_{k-1}^{t-1} \neq 0\right)$$

Comme vu précédemment, cette équation du second degré admet au plus deux solutions sur $\mathbb{F}_{2^m}$. *Nous obtenons finalement que :*

$$\frac{1}{n-t+1} \geq \mathcal{E}_{k,t}^{\overline{k-1,t-1}} \geq \frac{1}{n-t+1} \times \left[1 - \frac{2}{n-t+1} \times (t-1)\right] \times (1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0))$$

□

Proposition 8.3 *Soient* t *un entier plus petit que* n *et* $\sigma(X) \in \Omega_t$. *On a alors :*

$$\forall k \in \{2, \ldots, t-1\}, \qquad \left|\mathbb{P}(\sigma_k^t = 0) - \frac{1}{n-t+1}\right| \leq \frac{2t}{(n-t+1)^2}.$$

Preuve *En utilisant la Prorpiété 8.5, nous obtenons :*

$$\forall\ 0 < i < t, \qquad \sigma_k^t = 0 \Leftrightarrow \sigma_k^{t-1} = x_t \cdot \sigma_{k-1}^{t-1}.$$

Commençons par séparer la probabilité en deux probabilités différentes :

$$\mathbb{P}(\sigma_k^t = 0) = \underbrace{\mathbb{P}(\sigma_k^t = 0 \cap \sigma_{k-1}^{t-1} = 0)}_{\mathcal{E}_{k,t}^{k-1,t-1}} + \underbrace{\mathbb{P}(\sigma_k^t = 0 \cap \sigma_{k-1}^{t-1} \neq 0)}_{\mathcal{E}_{k,t}^{\overline{k-1,t-1}}}$$

La première est égale à :

$$\begin{aligned} \mathcal{E}_{k,t}^{k-1,t-1} &= \mathbb{P}(\sigma_k^{t-1} = x_t \cdot \sigma_{k-1}^{t-1} \cap \sigma_{k-1}^{t-1} = 0) \\ &= \mathbb{P}(\sigma_k^{t-1} = 0 \cap \sigma_{k-1}^{t-1} = 0) \end{aligned}$$

Du Lemme 8.3, nous avons :

$$\mathcal{E}_{k,t}^{k-1,t-1} \leq 2 \times \frac{k-1}{(n-t+1)^2}$$

Du Lemme 8.4, nous avons :

$$\frac{1}{n-t+1} \times \left[1 - \frac{2}{n-t+1} \times (t-1)\right] \times (1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0) \leq \mathcal{E}_{k,t}^{\overline{k-1,t-1}} \leq \frac{1}{n-t+1}$$

L'idée est de borner inférieurement la somme des deux termes par seulement un des termes et de poser la borne inférieure pour notre probabilité initiale comme la borne inférieure de ce terme en particulier. Pour la borne supérieure, il n'y a pas d'autre solution que de borner supérieurement les deux termes de la probabilité. Nous donnons la première inégalité :

$$\mathbb{P}(\sigma_k^t = 0) \leq \frac{1}{n-t+1} + 2 \times \frac{k-1}{(n-t+1)^2}$$

Nous avons donc la borne supérieure qui suit :

$$\mathbb{P}(\sigma_k^t = 0) \leq \frac{1}{n-t+1} + \frac{2t}{(n-t+1)^2}$$

Pour la seconde inégalité, nous avons :

$$\mathbb{P}(\sigma_k^t = 0) \geq \mathcal{E}_{k,t}^{\overline{k-1,t-1}} \geq \frac{1}{n-t+1} \times \left[1 - \frac{2}{n-t+1} \times (t-1)\right] \times (1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0))$$

Utilisons la première inégalité dans le but de donner une borne inférieure pour :

$$1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0) \geq 1 - \frac{1}{n-t+2} - 2 \times \frac{t-1}{(n-t+2)^2}$$

$$1 - \mathbb{P}(\sigma_{k-1}^{t-1} = 0) \geq 1 - \frac{1}{n-t+1} - 2 \times \frac{t-1}{(n-t+1)^2}$$

$$\mathbb{P}(\sigma_k^t = 0) \geq \frac{1}{n-t+1} \times \left[1 - \frac{2}{n-t+1} \times (t-1)\right] \times \left[1 - \frac{1}{n-t+1} - 2 \times \frac{t-1}{(n-t+1)^2}\right]$$

$$\mathbb{P}(\sigma_k^t = 0) \geq \frac{1}{n-t+1} - \frac{2 \times (t-1)}{(n-t+1)^2} - \frac{1}{(n-t+1)^2} - \frac{2 \times (t-1)}{(n-t+1)^3} + \frac{2 \times (t-1)}{(n-t+1)^3} + \frac{4 \times (t-1)^2}{(n-t+1)^4}$$

$$\mathbb{P}(\sigma_k^t = 0) \geq \frac{1}{n-t+1} - \frac{2t-1}{(n-t+1)^2} + \frac{4 \times (t-1)^2}{(n-t+1)^4}$$

Nous avons finalement :

$$\mathbb{P}(\sigma_k^t = 0) \geq \frac{1}{n-t+1} - \frac{2t}{(n-t+1)^2}$$

□

Récapitulons les bornes obtenues sur les probabilités dans le Tableau 8.5.

TABLEAU 8.5 – Bornes sur certaines probabilités

Probabilité	Formule	Référence
$\mathbb{P}(\sigma_1^t = 0)$	$\sum_{i=0}^{\frac{t}{2}-2}(-1)^i \frac{1}{t-(2i+1)} \times \prod_{j=0}^{i} \frac{t-(2j+1)}{n-t+2j+1}$ *; pour t pair* $\frac{1}{n}$ *; pour t impair*	Proposition 8.2
$\mathbb{P}(\sigma_k^t = 0)$	$\geq \frac{1}{n-t+1} - \frac{2t}{(n-t+1)^2}$ $\leq \frac{1}{n-t+1} + \frac{2t}{(n-t+1)^2}$	Proposition 8.3
$\mathbb{P}(\sigma_t^t = 0)$	$\frac{t}{n}$	Propriété 8.3
$\mathcal{E}_{k,t}^{k-1,t}$	$\leq 2 \times \frac{k-1}{(n-t+1)^2}$	Lemme 8.3

Donnons maintenant une réponse au Problème 8.2 énoncé au début de la Sous-section 8.3.2.

Proposition 8.4 *Soit $\sigma(X) \in \Omega_t$, avec $\sigma(X) = \prod_{i=1}^{t}(X \oplus x_i)$.*
Alors :

$$1 - \left[\frac{t}{n} + \frac{(t-2)}{n-t+1} + \frac{2t\,(t-2)}{(n-t+1)^2} + \mathbb{P}(\sigma_1^t = 0)\right] \leq \mathbb{P}$$

$$\mathbb{P} \leq 1 - \left[\frac{t}{n} + \frac{t-2}{n-t+1} - \frac{2t\,(t-2)}{(n-t+1)^2} + \mathbb{P}(\sigma_1^t = 0)\right] + \frac{2t\,(t-2)}{(n-t+1)^2}$$

Preuve *Nous pouvons facilement observer que :*

$$\mathbb{P}\left(\forall i \in [\![1,t]\!],\ \sigma_i^t \neq 0\right) = 1 - \mathbb{P}(\exists i \in [\![1,t]\!],\ \sigma_i^t = 0)$$

$$\mathbb{P}(\exists i \in [\![1,t]\!],\ \sigma_i^t = 0) = \sum_{i=1}^{t} \mathbb{P}(\sigma_i^t = 0) + \sum_{k=2}^{t} (-1)^k\, \mathbb{P}(\sigma_{i_1}^t = 0 \cap \sigma_{i_2}^t = 0 \cap \cdots \cap \sigma_{i_k}^t = 0)$$

Dans le but de donner des bornes supérieure et inférieure pour cette probabilité, nous utiliserons deux inégalités bien connues : la borne de l'union et l'inégalité de Bonferroni.

- *La première nous permettra de déterminer la borne inférieure :*

La borne de l'union :

$$\mathbb{P}(\cup \sigma_i^t = 0) \leq \sum_{i=1}^{t} \mathbb{P}(\sigma_i^t = 0)$$

Dans notre cas, cela devient :

$$\mathbb{P} \geq 1 - \sum_{i=1}^{t} \mathbb{P}(\sigma_i^t = 0)$$

Puisque nous avons

$$\mathbb{P}(\sigma_t^t = 0) = \frac{t}{n}; \qquad \mathbb{P}(\sigma_k^t = 0) \leq \frac{1}{n-t+1} + \frac{2t}{(n-t+1)^2}$$

et comme nous connaissons la formule pour la somme, nous obtenons :

$$\mathbb{P} \geq 1 - \left[\frac{t}{n} + \frac{t-2}{n-t+1} + \frac{2t\,(t-2)}{(n-t+1)^2} + \mathbb{P}(\sigma_1^t = 0)\right].$$

- *Pour la seconde, nous utiliserons l'autre inégalité :*

L'inégalité de Bonferroni :

$$\mathbb{P}\left(\cup\sigma_i^t = 0\right) \geq \sum_{i=1}^{t} \mathbb{P}\left(\sigma_i^t = 0\right) - \sum_{1\leq j<k\leq t} \mathbb{P}\left(\sigma_j^t = 0 \cap \sigma_k^t = 0\right)$$

Dans notre cas, cela devient :

$$\mathbb{P} \leq 1 - \sum_{i=1}^{t} \mathbb{P}\left(\sigma_i^t = 0\right) + \sum_{1\leq j<k\leq t} \mathbb{P}\left(\sigma_j^t = 0 \cap \sigma_k^t = 0\right)$$

Donc :

$$\mathbb{P} \leq 1 - \sum_{i=1}^{t} \mathbb{P}\left(\sigma_i^t = 0\right) + \sum_{2\leq k\leq t} \mathcal{E}_{k,t}^{k-1,t} + \sum_{2\leq j+1<k\leq t} \mathcal{E}_{j,t}^{k,t}$$

La quantité :

$$g_{n,t} = \sum_{2\leq j+1<k\leq t} \mathcal{E}_{j,t}^{k,t}$$

ne sera pas bornée puisque nous verrons via nos expérimentations que $g_{n,t}$ est négligeable.
Puisque nous avons :

$$\mathbb{P}(\sigma_k^t = 0) \geq \frac{1}{n-t+1} - \frac{2t}{(n-t+1)^2}; \mathcal{E}_{k,t}^{k-1,t} \leq \frac{2t}{(n-t+1)^2} \quad \forall k \in \{2, \ldots, t-1\}$$

Nous obtenons la borne suivante :

$$\mathbb{P} \leq 1 - \left[\frac{t}{n} + \frac{t-2}{n-t+1} - \frac{2t\,(t-2)}{(n-t+1)^2} + \mathbb{P}(\sigma_1^t = 0)\right] + \frac{2t\,(t-2)}{(n-t+1)^2} + g_{n,t}.$$

□

Distingueur sur zéro

Dans ce paragraphe, nous détaillons comment ce problème révèle la présence de zéro parmi les éléments du support. Considérons le problème initial : Soient un ensemble à t éléments $\{x_1, \ldots, x_t\} \in \Omega'_t$ et $\sigma(X) = \prod_{j=1}^{t} (X \oplus x_j)$. Maintenant considérons la probabilité que tous les coefficients soient différents de zéro : $\mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0\right)$. Notre but est de distinguer si zéro pourrait faire partie des racines ou non. Ré-écrivons notre probabilité comme ceci :

$$\mathbb{P}\left(\cap\, \sigma_i^t \neq 0\right) = \underbrace{\mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0 \cap \exists j,\ x_j = 0\right)}_{=0} + \mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0 \cap \forall j,\ x_j \neq 0\right)$$

La première vaut zéro puisque nous ne pouvons pas avoir tous les coefficients différents de zéro et une racine égale à zéro dans le même temps (avoir une racine égale à zéro implique que le dernier coefficient est aussi égal à zéro). Nous obtenons donc la relation suivante :

$$\mathbb{P}\left(\cap\, \sigma_i^t \neq 0\right) = \mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0 \cap \forall j,\ x_j \neq 0\right)$$

Cela implique que :

$$\begin{aligned}
\mathbb{P}\left(\cap\, \sigma_i^t \neq 0\right) &= \mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0 | \forall j,\ x_j \neq 0\right) \times \mathbb{P}\left(\forall j,\ x_j \neq 0\right) \\
&= \mathbb{P}\left(\forall i,\ \sigma_i^t \neq 0 | \forall j,\ x_j \neq 0\right) \times \frac{n-t}{n}
\end{aligned}$$

Conséquences : Le dernier résultat nous donne une information importante sur la présence de zéro dans le support. Lorsque zéro est l'un des éléments de notre support, nous aurons moins de chances d'avoir tous les coefficients différents de zéro si nous choisissons uniformément nos racines. Cela signifie qu'en moyenne un support contenant zéro donnera des structures moins denses pour ce type de polynômes. Un attaquant qui est capable de détecter à partir de l'évaluation du polynôme le fait que le polynôme est moins dense sera alors capable de déterminer que zéro est un élément du support.

Un attaquant qui est capable de trouver exactement combien de coefficients sont égaux à zéro devrait être capable de révéler, avec une certaine probabilité, la position de zéro comme l'une des racines du polynôme. Comme contre-mesure, nous proposons simplement de choisir un support qui ne contient pas zéro.

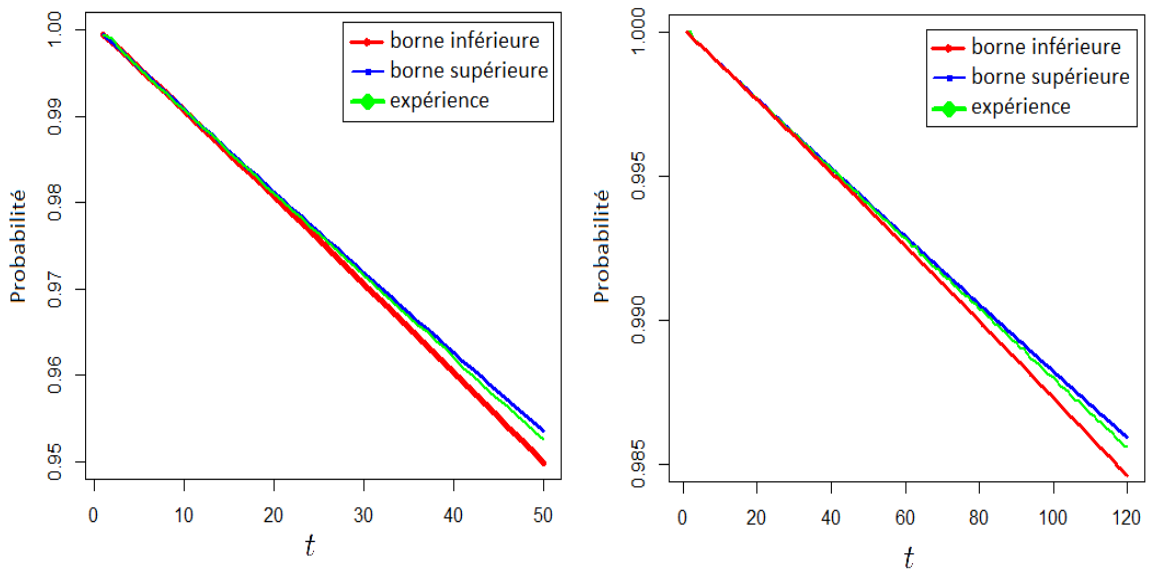
Approche expérimentale

Simulation avec PARI/GP Les simulations ont été faites en utilisant PARI/GP[21] (un logiciel libre). Pour l'approche expérimentale, nous avons utilisé la méthode de Monte-Carlo. Elle utilise le théorème central limite dans le but de donner la distribution asymptotique à moyen terme et est appliquée dans notre cas pour estimer le nombre de coefficients égaux à zéro pour un polynôme donné. Nous détaillons, dans le paragraphe suivant, la procédure utilisée dans le but d'obtenir les résultats. Après quoi, nous donnons la représentation graphique des variables simulées et les bornes théoriques. Nous verrons que la distribution est très proche de l'une des bornes.

21. `http://pari.math.u-bordeaux.fr/`

FIGURE 8.3 – Bornes expérimentales et théoriques pour $\mathbb{P}\left(\bigcap_{i=1}^{t} \sigma_i^t \neq 0\right)$

(a) $n = 2048$ (b) $n = 8192$



Tout d'abord, nous avons simulé pour un nombre donné t de racines le polynôme correspondant. Ensuite nous avons compté le nombre de coefficients égaux à zéro. Nous avons répété la simulation 3 000 000 de fois pour chaque t. Dans notre cas, la méthode de Monte-Carlo a été appliquée à la variable : *nombre de coefficients égaux à zéro*. La Figure 8.3 représente la probabilité que tous les coefficients soient différents de zéro, pour $n = 2048$ dans la Figure 8.3a et pour $n = 8192$ dans la Figure 8.3b. Pour $n = 2^{13}$, nous avons choisi aléatoirement jusqu'à 120 racines différentes pour notre polynôme.

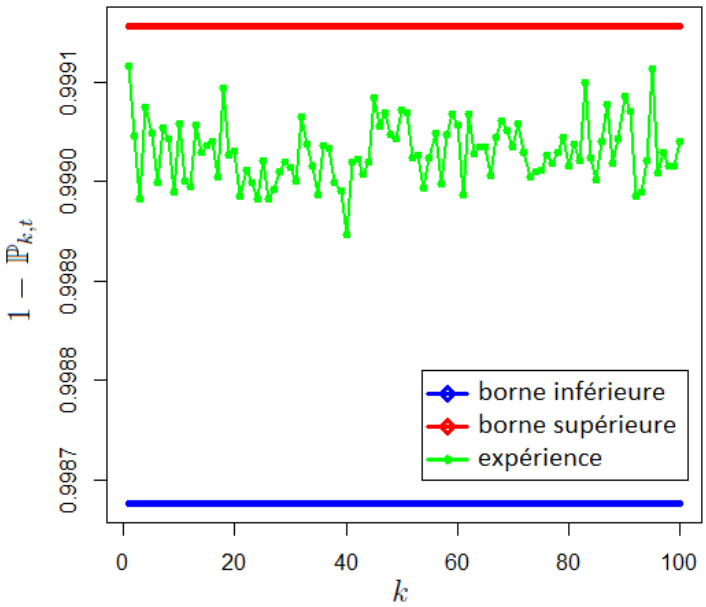
Interprétation La Figure 8.3 illustre l'importance de la borne inférieure. Comme nous voulons avoir le moins de coefficients égaux à zéro possible, la borne inférieure donne les valeurs suivantes :

$$\begin{aligned}
&100 \text{ bits de sécurité}, && n = 2048 \text{ et } t \leq 50 && \Rightarrow 0.953 \leq \mathbb{P} \leq 0.955 \\
&128 \text{ bits de sécurité}, && n = 2960 \text{ et } t = 56 && \Rightarrow 0.9635 \leq \mathbb{P} \leq 0.9642 \\
&256 \text{ bits de sécurité}, && n = 6624 \text{ et } t = 115 && \Rightarrow 0.9658 \leq \mathbb{P} \leq 0.9664
\end{aligned}$$

Nous observons que la probabilité va en grandissant avec la taille du corps choisi. Il est aussi expliqué pourquoi l'augmentation de la sécurité du cryptosystème impose directement une probabilité plus grande pour notre problème.

Probabilité qu'un coefficient particulier soit différent de zéro Pour la Figure 8.4, nous avons utilisé la même technique (la méthode de Monte Carlo) dans le but d'estimer la probabilité qu'un coefficient particulier soit différent de zéro. Les bornes supérieure et inférieure sont les mêmes que dans la Proposition 8.3. Nous pouvons observer que dans un ensemble de 100 coefficients, ils ont tous le même comportement.

FIGURE 8.4 – La probabilité que $\sigma_k^t \neq 0$



Distingueur sur zéro La Figure 8.5 représente le distingueur sur zéro. Nous avons représenté ici les bornes supérieure et inférieur dans deux cas : lorsque zéro fait parti du support et lorsqu'il n'est pas dedans. Nous observons que la probabilité d'avoir tous les coefficients différents de zéro est plus faible lorsque zéro fait parti du support que lorsqu'il n'en fait pas parti. La différence entre les deux cas, comme nous l'avons fait remarquer précédemment, pourrait être exploitée par un attaquant. L'intervalle entre les deux bornes grandit avec le degré du polynôme. Le travail d'un attaquant devient donc plus facile avec un polynôme de degré élevé (c'est-à-dire avec un grand nombre d'erreurs dans le texte chiffré pour le cryptosystème de McEliece). Cela signifie qu'imposer un support où zéro n'est pas un élément augmente notre sécurité contre ce type d'attaques (comme nous l'avons déjà noté). Par exemple, si nous posons $n = 2048$ et choisissons moins de $t = 90$ racines, nous aurons une probabilité qui sera plus grande que 0.99 d'avoir tous les coefficients différents de zéro.

8.3.3 Application à une attaque par canal auxiliaire

Plusieurs articles sur les attaques par canaux auxiliaires contre le cryptosystème de McEliece sont déjà parus (c.f. Chapitre 4). La plupart de ces attaques ciblent l'algorithme de décodage de Patterson et exploitent plusieurs faiblesses, comme nous pouvons le voir dans le Tableau 8.6.

FIGURE 8.5 – $\mathbb{P}(\bigcap_{i=1}^{t} \sigma_i^t \neq 0)$ dans deux cas : $0 \in \mathcal{L}$ et $0 \notin \mathcal{L}$

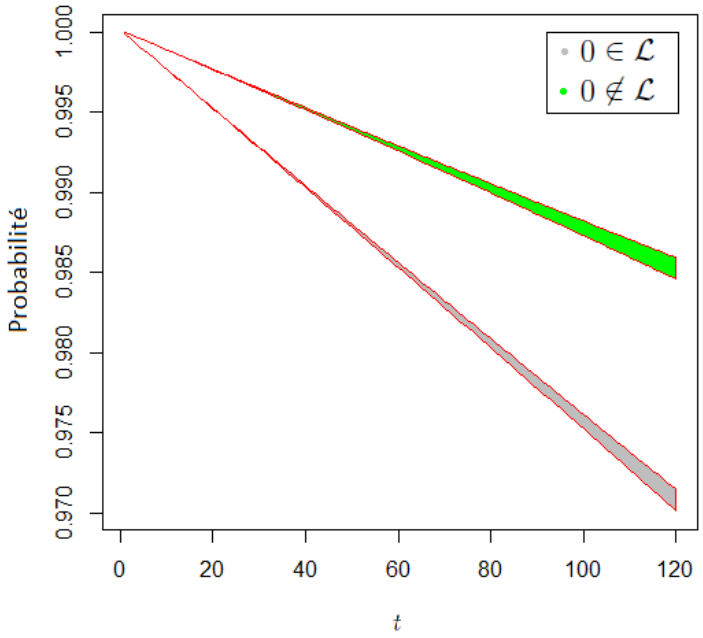


TABLEAU 8.6 – Décodage par Patterson : attaques temporelles existantes et contre-mesures

Étape	Réf.	Contre-mesure
❶ $\tilde{C}' = \tilde{C} \cdot \mathcal{P}^{-1}$		
❷ $S_{\tilde{C}'}(X) = \mathcal{H}' \cdot \tilde{C}'(X^{t-1}, \ldots, X^2, X, 1)^T$		
❸ $S_{\tilde{C}'}(X)^{-1} \mod G(X)$ *via EEA*	[Str13b]	flux de contrôle
❹ $R(X) = \sqrt{X + S_{\tilde{C}'}(X)^{-1}}$		
❺ $\mathfrak{b}(X) \cdot R(X) \equiv \mathfrak{a}(X) \mod G(X)$ $\deg(\mathfrak{a}) \leq \lfloor \frac{t}{2} \rfloor$; $\deg(\mathfrak{b}) \leq \lfloor \frac{t-1}{2} \rfloor$ *via EEA*	[SSMS10, Str10b]	dans EEA, s'assurer que $\deg(r_i) = \deg(r_{i-1}) - 1$ et $\deg(R) = t - 1$
❻ $\sigma(X) = \mathfrak{a}^2(X) + X \cdot \mathfrak{b}^2(X)$		
❼ $E = (\sigma(\alpha_1), \sigma(\alpha_2), \ldots, \sigma(\alpha_n)) \oplus (1, 1, \ldots, 1)$	[AHPT11, Str11, STM+08]	le non-support ou s'assurer que $\deg(\sigma) = t$
❽ $E' = E \cdot \mathcal{P}$		
❾ $\tilde{C} = \tilde{C}' \oplus E'$		

Il y a essentiellement deux types d'attaques classées selon leurs objectifs :

1. Attaques retrouvant le message secret M [AHPT11, Str11, STM+08] ;
2. Attaques retrouvant (totalement ou partiellement) la clé privée s_k [SSMS10, Str10b, Str11, Str13b].

Les attaques sur les étapes ❸ et ❺ permettent de déterminer certaines relations sur les éléments du support en comptant le nombre d'itérations dans EEA (c.f. Chapitre 7).

Les relations entre les éléments du support permettent à un attaquant de poser un système d'équations dont la solution est la permutation privée. Nous analysons l'avantage d'une telle attaque contre l'attaque naïve. En termes de complexité, au lieu d'énumérer toutes les permutations possibles, c'est-à-dire $n!$ permutations, nous réduisons la complexité à l'expression suivante :

$$\sum_{i=3}^{p} \binom{n}{i}, \qquad \text{où } p \leq \frac{\deg(g)}{2} - 1.$$

Donc pour de petites valeurs de p, nous avons :

$$\sum_{i=3}^{p} \binom{n}{i} \leq \binom{n}{p} \times (p-2) \leq (\frac{en}{p})^p \times (p-2),$$

(où $e \simeq 2.72$ est la base du logarithme népérien). Pour rendre plus évidente la différence entre l'attaque naïve et une attaque temporelle sur la permutation privée, nous donnons une borne inférieure pour la complexité de l'attaque naïve :

$$(\frac{n}{e})^n \leq n!$$

Finalement :

$$\sum_{i=3}^{p} \binom{n}{i} \leq \left(\frac{en}{p}\right)^p \times (p-2) << \left(\frac{n}{e}\right)^n \leq n!$$

Donc l'attaque naïve est exponentielle en la longueur du code alors qu'une attaque temporelle a seulement une complexité en le maximum des poids des vecteurs erreurs nécessaires à l'attaque (souvent extrêmement petits en comparaison avec la longueur du code).
Les autres attaques, sur l'étape ❼, révèlent les positions des erreurs en utilisant des différences de temps dans l'évaluation du polynôme localisateur d'erreurs. L'attaquant est capable de trouver le vecteur erreur avec une probabilité non négligeable. L'idée de base est que deux polynômes donnés de degrés différents ne sont pas évalués dans le même temps. Donc la différence de temps donne une information sur le vecteur erreur. Ici, en termes de complexité, l'avantage est même plus grand.
Notre analyse sur les polynômes à racines simples démontre son intérêt pour une attaque. Nous savons que le polynôme localisateur d'erreurs σ est un polynôme à racines simples. Nous savons aussi qu'un type d'attaque pour retrouver le message secret exploite des différences de temps entre deux degrés consécutifs, disons $t-1$ et t.
Nous rappelons le fait que dans [STM+08] et [AHPT11], les contre-mesures proposées consistent à manipuler $\sigma(X)$ de sorte que si $\deg(\sigma) < t$, on le transforme d'une manière ou d'une autre :

1. ajouter de façon déterministe des coefficients de sorte que $\deg(\sigma) = t$ et que tous les coefficients soient non nuls.
2. utiliser des coefficients à partir du non-support de sorte que $\deg(\sigma) = t$ et que tous les coefficients soient non nuls.

Contre-mesure : L'idée est que la seconde partie de la condition *s'assurer que tous les coefficients soient non nuls* soit déjà vérifiée par le problème de racines simples (Sous-section 8.3.2). Donc nous devons seulement manipuler le degré de σ.

Alors la probabilité d'avoir au moins un coefficient égal à zéro dans σ est extrêmement faible. Plus que cela, nous ne choisissons pas zéro comme élément du support, nous assurons aussi une contre-mesure à un éventuel distingueur sur cet élément particulier. Si cela est fait, la probabilité d'avoir au moins un coefficient égal à zéro pour $n = 2048$ et $t < 80$ est plus petite que $0,009\%$.

Schéma de signature de CFS

Dans le schéma de signature de CFS, un petit nombre t est utilisé dû à la densité des codes de Goppa. Il a été prouvé dans [CFS01] que l'algorithme de décodage doit être répété en moyenne $t!$ fois. Le décodage des codes de Goppa pour CFS avec les valeurs recommandées donne le résultat suivant : pour $n = 2^{16}$ et $t \leq 10$, nous obtenons $\mathbb{P} > 0,999$.

8.4 Attaque temporelle contre l'évaluation du PLE

L'objectif de l'attaquant est de trouver la permutation privée $\mathcal{P}$ par une attaque temporelle sur l'évaluation du PLE, en cherchant les éléments 0 et 1 permutés puis le reste par force brute.

Identification de la fuite : Une fuite est identifiée à la ligne 4 de l'algorithme de Patterson (Algorithme 38), sur l'évaluation du PLE. Nous rappelons que le PLE est noté σ. L'attaque est basée sur le fait que la forme du polynôme diffère en fonction de l'élément à décoder. Nous prouvons que la complexité de l'algorithme est fortement liée aux coefficients de $\sigma(X)$. Nous effectuons ensuite une attaque temporelle sur l'évaluation du PLE et contrôlons les valeurs des coefficients de $\sigma(X)$ (pour les codes de Goppa : Euclide étendu, Berlekamp-Massey, Patterson).

Motivations de notre attaque : L'une des principales motivations de notre attaque est qu'elle peut opérer sur toutes les implantations existantes d'un décodeur alternant général. Elle opère sur l'évaluation du PLE, étape qui doit être calculée dans tout algorithme de décodage résolvant l'équation-clé.
Nous donnons deux algorithmes de base pour l'évaluation du PLE avec quelques améliorations et montrons que même avec les améliorations publiées, notre attaque est un succès. Nous choisissons l'évaluation polynomiale de droite à gauche (l'algorithme simple) et de gauche à droite (le schéma de Ruffini-Horner), comme présenté dans la Section 8.2. Supposons que le polynôme soit de degré t. Le premier algorithme calcule le résultat en $3t - 1$ opérations (t additions et $2t - 1$ multiplications) tandis que le second le calcule en $2t$ opérations (t additions et t multiplications). Il a été prouvé en 1966 [Pan66] que le schéma de Ruffini-Horner [Hor19] est optimal en terme de complexité.
Notre attaque fonctionne dans le cas du premier algorithme. Nous donnons une amélioration pour un calcul plus rapide de cet algorithme mais il reste vulnérable à notre attaque. Pour le second algorithme, l'attaque est encore un succès avec une condition supplémentaire : l'attaquant doit être capable de détecter si à chaque étape l'algorithme calcule le même nombre d'opérations ou moins. Si cette condition est vérifiée, l'attaque fonctionne comme dans le premier cas.
L'idée principale de l'amélioration est d'utiliser le fait que certains éléments du support ont des propriétés particulières (e.g. 0 et 1). Sachant qu'un coefficient égal à zéro accélère l'algorithme pour des opérations comme la multiplication ou que la

somme a une valeur fixe si zéro est pris comme l'un des termes, la même chose se passe avec la multiplication par un. Donc nous exploitons ces propriétés dans le but d'améliorer notre implantation. Chaque fois qu'un coefficient vaut 1 ou 0, il sera stocké dans une table spéciale utilisée après par la multiplication ou l'addition. Le cas où un coefficient vaut zéro est rare et sa probabilité a été étudiée dans la Section 8.3.
Néanmoins, chaque fois qu'il y a un coefficient égale à zéro, nous ne le multiplions pas l'élément correspondant puisque le produit vaut zéro. Donc nous utilisons les tables prédéfinies pour nous débarrasser des opérations inutiles. Nous procédons exactement de la même manière lorsque la multiplication d'un élément doit être faite lorsqu'un terme vaut 1. Donc chaque fois que nous avons un terme valant 0, en utilisant nos tables prédéfinies, nous nous débarrassons de deux opérations (une addition et une multiplication).

Scenario : Le scenario de l'attaque est le même que dans l'attaque présentée au Chapitre 7, excepté pour la dernière étape. En effet, l'attaquant obtient le temps d'exécution pour l'évaluation du PLE dans le cas présent, à la place de celui de la détermination du PLE.

Idée : Pour $w_H(E) = 2$, l'attaquant trouve les positions de $\mathcal{P}(0)$ et $\mathcal{P}(1)$ (les permutations de 0 et 1). Après suffisamment d'itérations, il fixe ces deux positions et répète cette attaque avec $w_H(E) = 3$ et trouve ensuite la permutation privée $\mathcal{P}$ (en utilisant la recherche exhaustive pour les positions restantes).

Conditions : Les hypothèses sont les mêmes que dans l'attaque développée dans le Chapitre 7, excepté que l'attaquant ne connaît pas l'ordre des éléments du support.

8.4.1 Probabilité de succès

Comme nous l'avons dit, dans cette attaque nous considérons seulement les polynômes avec un degré plus petit que trois. Pour le cas $w_H(E) = 3$, nous donnons la table complète des probabilités. Nous commençons avec le problème général suivant :

Problème : Soit $\sigma(X)$ un polynôme unitaire de degré t, avec t racines distinctes sur $\mathbb{F}_{2^m}$. Quelle est la probabilité que tous ses coefficients soient différents de zéro ? Les résultats de ce problème, traité dans la Section 8.3, montrent que la probabilité peut être bornée. Pour les paramètres classiques du cryptosystème de McEliece, c'est-à-dire pour $n = 2048$ et $t \leq 50$, on obtient :

$$\mathbb{P} \geq 0.95.$$

Le cas $t = 3$:

Réponse : Soit $\sigma(X)$ un polynôme unitaire de degré 3 avec trois racines distinctes sur $\mathbb{F}_{2^m}$ et $m \equiv 1 \mod 2$.

La probabilité $\mathbb{P}_{t=3}$ que tous ses coefficients soient différents de zéro satisfait :

$$\mathbb{P}_{t=3} = 1 - \frac{5}{2^m}.$$

8.4.2 Recherche de la permutation des éléments zéro et un du support

1. Considérons les vecteurs erreurs E_i avec $w_H(E_i) = 1$.
   Dans ce cas, le polynôme localisateur d'erreurs a la forme suivante :
   $$\sigma(X) = X \oplus \alpha_i, \text{ avec } \alpha_i \in \mathscr{L} = \{0, 1, \alpha, \ldots, \alpha^{n-2}\}.$$
   Si $\alpha_i \neq 0$, il y a une addition ($\oplus$) dans l'évaluation de $\sigma(X)$.
2. Considérons les vecteurs erreurs E_i avec $w_H(E_i) = 2$.
   Dans ce cas, le polynôme localisateur d'erreurs a la forme suivante :
   $$\sigma(X) = X^2 \oplus \sigma_1^2 \cdot X \oplus \sigma_2^2, \text{ avec } \sigma_1^2 = \alpha_i \oplus \alpha_j \text{ et } \sigma_2^2 = \alpha_i \cdot \alpha_j.$$
   Nous distinguons deux cas possibles :
   (a) $\sigma(X) = X^2 \oplus \sigma_1^2 \cdot X \oplus \sigma_2^2$ si $\alpha_i \cdot \alpha_j \neq 0$
   (b) $\sigma(X) = X^2 \oplus \sigma_1^2 \cdot X$ si $\alpha_i \cdot \alpha_j = 0$
   Le cas (b) ci-dessus mène au calcul de l'évaluation polynomiale avec une addition supplémentaire $\oplus$ et les temps révèlent tous les couples $(\alpha_i, 0)$. Nous pouvons supposer maintenant que la position de $\mathcal{P}(0)$ est connue.
3. Nous fixons sa position et nous cherchons la position de $\mathcal{P}(1)$. Comme le polynôme $\sigma(X) = X^2 \oplus \sigma_1^2 \cdot X$, l'évaluation la plus rapide est obtenue pour le couple $(\mathcal{P}(0), \mathcal{P}(1))$ puisqu'il y a seulement une addition $\oplus$ et une élévation au carré.

8.4.3 Scenario de l'attaque lorsque $t = 3$

Nous considérons les vecteurs erreurs de poids de Hamming valant 3. Le polynôme $\sigma(X)$ correspondant a toujours l'une des huit représentations suivantes :

1	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3$	si $\sigma_1^3 \cdot \sigma_2^3 \cdot \sigma_0^3 \neq 0$
2	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X$	si $\sigma_3^3 = 0$ et $\sigma_1^3 \cdot \sigma_2^3 \neq 0$
3	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_3^3$	si $\sigma_2^3 = 0$ et $\sigma_1^3 \cdot \sigma_3^3 \neq 0$
4	$\sigma(X) = X^3 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3$	si $\sigma_1^3 = 0$ et $\sigma_2^3 \cdot \sigma_3^3 \neq 0$
5	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2$	si $\sigma_1^3 \neq 0$ et $\sigma_2^3 = 0$ et $\sigma_3^3 = 0$
6	$\sigma(X) = X^3 \oplus \sigma_2^3 \cdot X$	si $\sigma_2^3 \neq 0$ et $\sigma_1^3 = 0$ et $\sigma_3^3 = 0$
7	$\sigma(X) = X^3 \oplus \sigma_3^3$	si $\sigma_3^3 \neq 0$ et $\sigma_1^3 = 0$ et $\sigma_2^3 = 0$
8	$\sigma(X) = X^3$	si $\sigma_1^3 = 0$ et $\sigma_2^3 = 0$ et $\sigma_3^3 = 0$

Nous obtenons alors les probabilités suivantes :

(a)	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3$	si $\sigma_1^3 \cdot \sigma_2^3 \cdot \sigma_3^3 \neq 0$ et $\mathbb{P} = \frac{n-5}{n}$
(b)	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X$	si $\sigma_3^3 = 0$ et $\sigma_1^3 \cdot \sigma_2^3 \neq 0$ et $\mathbb{P} = \frac{3}{n}$
(c)	$\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_3^3$	si $\sigma_2^3 = 0$ et $\sigma_1^3 \cdot \sigma_3^3 \neq 0$ et $\mathbb{P} = \frac{1}{n}$
(d)	$\sigma(X) = X^3 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3$	si $\sigma_1^3 = 0$ et $\sigma_2^3 \cdot \sigma_3^3 \neq 0$ et $\mathbb{P} = \frac{1}{n}$

Plusieurs cas peuvent être éliminés en considérant le fait que nous accomplissons la première étape de l'attaque et que nous connaissons donc la position de $\mathcal{P}(0)$. Si nous considérons tous les vecteurs erreurs où $\alpha_i \neq 0, \quad \forall i \in \{1, 2, \ldots, n-1\}$ (c'est-à-dire que 0 n'est pas l'une des racines de $\sigma(X)$), nous réduisons les possibilités pour $\sigma(X)$. La nouvelle forme du système est la suivante :

$$\begin{cases} \sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3 & \text{si } \sigma_1^3 \cdot \sigma_2^3 \cdot \sigma_0^3 \neq 0 \\ \sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_3^3 & \text{si } \sigma_2^3 = 0 \text{ et } \sigma_1^3 \cdot \sigma_3^3 \neq 0 \\ \sigma(X) = X^3 \oplus \sigma_2^3 \cdot X \oplus \sigma_3^3 & \text{si } \sigma_1^3 = 0 \text{ et } \sigma_2^3 \cdot \sigma_3^3 \neq 0 \end{cases}$$

Dans tous les cas, X^3 doit être calculé donc nous ne considérons pas cette partie dans les différences de temps. Dans la structure qui calcule l'évaluation du polynôme, la plus rapide est la dernière. Mais ce cas est effectué seulement lorsque $\sigma_1^3 = 0$.

8.4.4 Recherche des positions de deux éléments tels que $\mathcal{P}(\alpha_j)\mathcal{P}(\alpha_k) = 1$

Dans le but d'augmenter le nombre d'équations dans notre système, nous exploitons le fait que $(\mathbb{F}_{2^m})^*$ est cyclique.

Rappel : nous connaissons les positions de $\mathcal{P}(0), \mathcal{P}(1)$ et $\mathcal{P}(\alpha_1) \oplus \mathcal{P}(\alpha_2) \oplus \Pi(\alpha_3) = 0$. Sans perdre de généralité, nous choisissons de fixer "$\mathcal{P}(0)$" sur la première position et choisissons deux autres positions telles que la somme soit différente de 1.

Nous sommes capables de faire cela parce que nous connaissons la position de "$\mathcal{P}(1)$" et les couples (α_1, α_2) tels que $1 \oplus \alpha_1 \oplus \alpha_2 = 0$. Nous obtenons deux nouvelles positions b_1 et b_2 telles que $b_1 + b_2 \neq 1$. Le polynôme évaluateur d'erreurs est : $\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus \sigma_2^3 \cdot X$. Pour $b_1 b_2 = 1$, nous obtenons $\sigma(X) = X^3 \oplus \sigma_1^3 \cdot X^2 \oplus X$. Cette forme est la plus rapide à être calculée comme il y a moins de multiplications comparé aux autres cas.

8.4.5 Résolution du système

Nous donnons les nombres d'équations linéaires et quadratiques obtenues par l'attaquant.

La recherche des positions de $\mathcal{P}(0)$ et $\mathcal{P}(1)$ réduit l'ensemble de recherche à $(n-2)$ éléments.

• Le premier ensemble d'équations linéaires :

$$\text{Type d'équation (1)} : \mathcal{P}(\alpha_j)\mathcal{P}(\alpha_k) = 1 \Rightarrow \sharp eq. = \tfrac{n-2}{2}$$

La dernière équation est déterminée par toutes les autres parce que pour le dernier couple seulement une solution possible reste valable. Par exemple, si l'attaquant trouve $(\frac{n-2}{2} - 1)$ équations différentes, alors la dernière équation peut être déterminée directement.

• Le deuxième ensemble d'équations linéaires :

$$\text{Type d'équation (2)} : \mathcal{P}(\alpha_j) + \mathcal{P}(\alpha_k) = 1 \Rightarrow \sharp eq. = \tfrac{n-2}{2}.$$

Comme le premier ensemble, la dernière peut être déterminée par toutes les autres. Cela vient du fait que pour les trois positions, nous avons fixé la position de $\mathcal{P}(1)$ comme la première. Donc nous avons $(n-2)$ possibilités sur la seconde position. Mais il y a deux répétitions pour chaque vecteur $(\mathcal{P}(1), \mathcal{P}(\alpha_j), \mathcal{P}(\alpha_k))$.

• Le troisième ensemble d'équations quadratiques :

$$\text{Type d'équation (3)} : \mathcal{P}(\alpha_i) + \mathcal{P}(\alpha_j) + \mathcal{P}(\alpha_k) = 0 \Rightarrow \sharp eq. = \tfrac{(n-2)(n-4)}{6}$$

Le nombre total d'équations pour $\mathcal{P}(\alpha_i) + \mathcal{P}(\alpha_j) + \mathcal{P}(\alpha_k) = 0$ incluant le second ensemble est égal à $(n-1)(n-2)$, comme la troisième position est fixée et les deux autres sont libres et différentes. Ici, le nombre de répétitions est égal à six. Donc nous obtenons $\left(\frac{(n-2)(n-1)}{6} - \frac{n-2}{2}\right)$ équations.

8.5 Conclusion et perspectives

En ce qui concerne l'implantation matérielle il faut, soit repenser l'attaque afin qu'elle devienne un succès, soit la proposer comme solution efficace de contre-mesure à une attaque temporelle sur l'évaluation du polynôme localisateur d'erreurs. Une amélioration envisagée est de combiner l'algorithme de Horner à la recherche de Chien (et pas seulement Horner pour la version "[L2R]").
Nous avons pour objectif de reprendre les travaux correspondants à [DCCR13] afin de proposer une version étendue de cet article, en espérant éclaircir certaines zones d'ombre. Ce problème peut-être intéressant également pour d'autres applications que la cryptographie.
Enfin, l'attaque temporelle n'étant plus forcément d'actualité par rapport à [BCS13], il serait intéressant de tester cette attaque par analyse de consommation.

Chapitre 9

De la connaissance de la matrice de permutation à la découverte du polynôme de Goppa

---

Ce chapitre correspond aux travaux effectués pour la sous-section IV.C de [PRD+15], puis la sous-section 3.3 de [PRD+16b]. La première proposition provient d'un échange avec Alain Couvreur. La seconde fait suite aux commentaires de Jean-Pierre Tillich et Pascal Véron.

Countermeasure against the SPA Attack on an Embedded McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
RadioElektronika 2015, p. 462-466, IEEE.

Low-complexity CBDPA Countermeasure for Resource-Constrained Embedded McEliece Cryptosystem
avec Martin Petrvalský, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer
en préparation pour RadioEngineering 2016.

Un article est en préparation, suite à de nombreuses discussions sur ce sujet avec Pascal Véron :

Goppa polynomial recovering in the McEliece PKC
avec Pascal Véron et Pierre-Louis Cayrel
en préparation.

9.1 Recherche du polynôme de Goppa

Dans ce chapitre, nous allons nous intéresser à différentes méthodes permettant de trouver les coefficients du polynôme de Goppa sous certaines conditions. Commençons par rappeler la définition d'un code de Goppa classique binaire (donnée dans la Définition 2.37 pour le cas q-aire).

Définition 9.1 (Code de Goppa classique binaire) *Soient $m > 0$, $n \leqslant 2^m$, $\mathscr{L} \triangleq \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{2^m}$ et $G(X) \in \mathbb{F}_{2^m}[X]$ un polynôme de degré t tel que $\forall i \in \{1, 2, \ldots, n\}$, $G(\alpha_i) \neq 0$. Le code de Goppa de longueur n est défini par :*

$$\Gamma(\mathscr{L}, G) \triangleq \left\{ C = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_2^n ; \sum_{i=1}^{n} \frac{c_i}{X \oplus \alpha_i} \equiv 0 \bmod G(X) \right\}.$$

On dit que $\mathscr{L}$ est le support et G est le polynôme de Goppa de $\Gamma(\mathscr{L}, G)$.

Nous supposerons dans ce chapitre que le polynôme de Goppa est irréductible et unitaire. D'après la Propriété 2.6, le nombre de polynômes irréductibles, unitaires, de degré t et à coefficients dans $\mathbb{F}_{2^m}$ est d'environ $2^{mt}/t$.

Pour son cryptosystème basé sur les codes correcteurs d'erreurs [McE78], McEliece a proposé d'utiliser un code de Goppa comme code privé.

9.1.1 Clé privée dans le cryptosystème de McEliece

Rappelons que, dans le cryptosystème de McEliece [McE78], la clé privée est $s_k = \big(\mathcal{S}, \mathcal{G}, \mathcal{P}, \mathscr{L}, G(X)\big)$, où :

- $\mathcal{S}$ est une matrice inversible dans $\mathscr{M}_{k,k}(\mathbb{F}_2)$,
- $\mathcal{G}$ est une matrice génératrice de taille $\mathscr{M}_{k,n}(\mathbb{F}_2)$ d'un code de Goppa binaire de longueur n et de dimension k,
- $\mathcal{P}$ est une matrice de permutation de taille $\mathscr{M}_{n,n}(\mathbb{F}_2)$,
- $\mathscr{L}$ est un ensemble à n éléments, sous-ensemble de l'extension de corps $\mathbb{F}_{2^m}$ et
- $G(X)$ est un polynôme de degré t appelé polynôme de Goppa, à coefficients dans $\mathbb{F}_{2^m}$.

Notons que le triplet $\big(\mathcal{G}, \mathscr{L}, G(X)\big)$ peut être remplacé par $\Gamma(\mathscr{L}, G)$, qui désigne le même code de Goppa. Ce code est t-correcteur.
L'hypothèse que l'ordre des éléments du support soit connu par l'attaquant est souvent faite dans la littérature sur les attaques par canaux auxiliaires contre le cryptosystème de McEliece, c'est pourquoi nous allons nous placer dans le même cadre. Nous allons donc supposer dans ce chapitre que l'ordre des éléments du support $\mathscr{L}$ est connu (dans l'ordre lexicographique par exemple, ou tout autre mais connu de l'attaquant), en se fixant $\mathscr{L} = \mathbb{F}_{2^m}$. Notons toutefois que cette hypothèse est forte et qu'il est facile de l'éviter (en prenant n éléments dans un corps plus grand et pas dans l'ordre lexicographique [AHPT11]).
De plus, nous supposons que l'attaquant a trouvé la matrice de permutation privée $\mathcal{P}$ à l'aide, par exemple, d'une attaque par canal auxiliaire [Str10b, PRD+15, PRD+16b]. Cela signifie que la seule partie restante de la clé privée encore inconnue de l'attaquant est alors le polynôme de Goppa $G(X)$. En effet, la connaissance de la matrice de confusion $\mathcal{S}$ n'est pas utile parce que $\mathcal{S} \cdot \mathcal{G}$ et $\mathcal{G}$ génèrent le même code (c.f. Proposition 2.3). Multiplier à gauche une matrice génératrice par une matrice inversible est l'unique moyen pour obtenir une autre matrice génératrice du même code.
Enfin nous supposerons que le polynôme de Goppa $G(X)$ est unitaire, pour avoir d'une part un coefficient en moins à chercher et d'autre part parce que cela se justifie d'un point de vue mathématique. En effet, un code de Goppa défini avec $G(X)$ et un code défini par un multiple de $G(X)$ sont les mêmes (c.f. Propriété 2.5) :

$$\forall \beta \in \mathbb{F}_{2^m}^*, \qquad \Gamma(\mathscr{L}, \beta \cdot G) = \Gamma(\mathscr{L}, G).$$

9.1.2 Réduction de l'espace de recherche des clés

Pour retrouver le polynôme de Goppa

$$G(X) = g_t \cdot X^t + g_{t-1} \cdot X^{t-1} + \ldots + g_2 \cdot X^2 + g_1 \cdot X + g_0,$$

que l'on sait de degré t, où $g_t = 1$ et où chaque $g_i \in \mathbb{F}_{2^m}$, l'attaquant peut utiliser différentes méthodes. La première idée serait de tester tous les polynômes irréductibles unitaires de degré t dans l'anneau de polynômes $\mathbb{F}_{2^m}[X]$. Il y a environ $2^{mt}/t$

vérifiant cette propriété. Par exemple, pour les paramètres donnés par McEliece dans son article, $n = 1024$, $m = 10$ et $t = 38$, cela fait $2^{380}/38 \simeq 6,48 \times 10^{112}$ possibilités. Cette méthode est une force brute sur les éléments du corps fini et peut-être implantée de manière simple par une technique de "backtracking" (retour à l'hypothèse précédente sur les coefficients du polynôme lorsque celle-ci mène à une contradiction pour passer à l'hypothèse suivante). Mais cette méthode n'est clairement pas optimale (car exponentielle en mt). On peut toutefois réduire un peu l'espace de recherche pour cette méthode en se concentrant sur les t coefficients g_{t-1} à g_0 et en remarquant que $g_0 \neq 0$ car $0 \in \mathscr{L}$ et $G(0) \neq 0$, mais les simplifications ne vont pas plus loin sans information supplémentaire sur le polynôme de Goppa (exemple : un trinôme avec des coefficients égaux à 1). Voyons plutôt une autre méthode, qui utilise les propriétés des codes de Goppa en tant que sous-code d'un code de Reed-Solomon généralisé (en Anglais *GRS* pour *generalized Reed-Solomon*).

9.2 Par la résolution d'un système linéaire suivie d'une interpolation

9.2.1 Un code de Goppa est un code alternant (sous-code d'un code GRS)

Une manière de trouver $G(X)$ à partir de $\mathcal{P}$ est de voir le code de Goppa comme un code alternant [MS77, Chapitre 12], c'est-à-dire un sous-code d'un code de Reed-Solomon généralisé (GRS) sur le sous-corps $\mathbb{F}_2$ (c.f. Sous-sections 2.4.4 et 2.4.5). Rappelons les définitions de ces codes ici :

Définition 9.2 (Code GRS) *(Rappel de la Définition 2.35.)*
Un code de Reed-Solomon généralisé $GRS_k(\bar{\alpha}, \bar{v})$ sur $\mathbb{F}_{2^m}$ de longueur $n \leqslant 2^m - 1$ et de dimension k est défini par un ensemble appelé son support $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, qui est un n-uplet d'éléments non nuls ordonnés de $\mathbb{F}_{2^m}$ deux à deux distincts, et par un vecteur appelé son multiplicateur $\bar{v} = (v_1, v_2, \ldots, v_n)$, qui est une suite d'éléments non nuls de $\mathbb{F}_{2^m}$. On a alors :

$$GRS_k(\bar{\alpha}, \bar{v}) = \Big\{(v_1 \cdot P(\alpha_1), v_2 \cdot P(\alpha_2), \ldots, v_n \cdot P(\alpha_n)); P \in \mathbb{F}_{2^m}[X], \deg(P) < k\Big\}.$$

Définition 9.3 (Code alternant) *(Rappel de la Définition 2.36.)*
Soit $GRS_{n-r}(\bar{\alpha}, \bar{v})$ un code GRS de paramètres $[n, n-r, r+1]_{q^m}$ construit sur une extension $\mathbb{F}_{q^m}$ de $\mathbb{F}_q$. Un code alternant $A_k(\bar{\alpha}, \bar{y})$ de paramètres $[n, k, d]_q$ est défini comme l'ensemble des mots du code $GRS_{n-r}(\bar{\alpha}, \bar{v})$ dont les coordonnées sont à coefficients dans le sous-corps $\mathbb{F}_q$, avec $n - mr \leqslant k \leqslant n - r$ et $d \geqslant r + 1$.

Théorème 9.1 *(Rappel du Théorème 2.5.)*
Soient $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ des éléments distincts de $\mathbb{F}_{2^m}$ et $\bar{y} = (y_1, y_2, \ldots, y_n)$ des éléments de $\mathbb{F}_{2^m}$ (non nécessairement distincts) qui satisfont $y_i \neq 0$. Soit $r \in \mathbb{N}^*$ et soit $\mathcal{Y}$ la matrice de Vandermonde à r lignes de $\bar{\alpha}$, donnée par :

$$\mathcal{Y} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_n^{r-1} \end{bmatrix}. \tag{9.1}$$

Soit $\mathcal{Z}$ la matrice diagonale de $\bar{y}$ donnée par :

$$\mathcal{Z} = \begin{bmatrix} y_1 & 0 & \dots & 0 \\ 0 & y_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & y_n \end{bmatrix}. \tag{9.2}$$

Soit $\mathcal{H} = \mathcal{Y} \cdot \mathcal{Z}$, c'est-à-dire :

$$\mathcal{H} = \begin{bmatrix} y_1 & y_2 & \dots & y_n \\ \alpha_1 \cdot y_1 & \alpha_2 \cdot y_2 & \dots & \alpha_n \cdot y_n \\ \vdots & \vdots & \dots & \vdots \\ \alpha_1^{r-1} \cdot y_1 & \alpha_2^{r-1} \cdot y_2 & \dots & \alpha_n^{r-1} \cdot y_n \end{bmatrix}. \tag{9.3}$$

Alors le code alternant de $\bar{\alpha}$ et $\bar{y}$, noté $A_k(\bar{\alpha}, \bar{y})$, est le code dont une matrice de contrôle est la matrice $\mathcal{H}$ vue sur $\mathbb{F}_2$.

Notons qu'une matrice de contrôle du code de Goppa peut être calculée comme un produit entre la matrice de Vandermonde $\mathcal{Y}$ de t lignes et la matrice diagonale $\mathcal{Z}$ avec les inverses des évaluations du polynôme G sur l'ensemble des éléments du support $\mathscr{L}$, où chaque composante de la matrice est un vecteur de m bits. Nous avons noté cette matrice $\bar{\mathcal{H}}$ et elle est donnée sur $\mathbb{F}_{2^m}$ par :

$$\begin{aligned} \bar{\mathcal{H}} &= \underbrace{\begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & \dots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \end{pmatrix}}_{\mathcal{Y}} \cdot \underbrace{\begin{pmatrix} \frac{1}{G(\alpha_1)} & 0 & \dots & 0 \\ 0 & \frac{1}{G(\alpha_2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{G(\alpha_n)} \end{pmatrix}}_{\mathcal{Z}} \\ &= \begin{pmatrix} \frac{1}{G(\alpha_1)} & \frac{1}{G(\alpha_2)} & \dots & \frac{1}{G(\alpha_n)} \\ \frac{\alpha_1}{G(\alpha_1)} & \frac{\alpha_2}{G(\alpha_2)} & \dots & \frac{\alpha_n}{G(\alpha_n)} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\alpha_1^{t-1}}{G(\alpha_1)} & \frac{\alpha_2^{t-1}}{G(\alpha_2)} & \dots & \frac{\alpha_n^{t-1}}{G(\alpha_n)} \end{pmatrix}. \end{aligned} \tag{9.4}$$

Cette forme de matrice de contrôle provient de la définition d'un code alternant. En effet, un code de Goppa est un code alternant spécifique, avec $y_i = 1/G(\alpha_i)$ et $\bar{\alpha} = \mathscr{L}$. Comme $\mathscr{L}$ est supposé connu, la matrice $\mathcal{Y}$ peut être calculée par un attaquant.

9.2.2 Résolution d'un système linéaire

L'idée de cette section avait déjà été proposée dans [Hei87, Section 5.6], et plus récemment dans [HMP10, Section 5.3].

Le problème semble simple : Comme $\deg(G) = t$, il y a $(t+1)$ coefficients g_i à trouver dans $\mathbb{F}_{2^m}$, donc il nous faut un système linéaire de $(t+1)$ équations sur $\mathbb{F}_{2^m}$ (ou de manière équivalente $m(t+1)$ équations sur $\mathbb{F}_2$) à $(t+1)$ inconnues dans $\mathbb{F}_{2^m}$ (ou de manière équivalente $m(t+1)$ inconnues sur $\mathbb{F}_2$) pour retrouver tous les g_i. Utilisons la Définition 9.3, pour voir le code de Goppa en question comme un code

alternant. Soit $y_i = 1/G(\alpha_i)$ pour tout i de 1 à n. Alors en écrivant $\bar{\mathcal{H}}$ sur $\mathbb{F}_{2^m}$, celle-ci devient pour un attaquant :

$$\bar{\mathcal{H}} = \begin{pmatrix} y_1 & y_2 & \dots & y_n \\ \alpha_1 \cdot y_1 & \alpha_2 \cdot y_2 & \dots & \alpha_n \cdot y_n \\ \vdots & \vdots & \dots & \vdots \\ \alpha_1^{t-1} \cdot y_1 & \alpha_2^{t-1} \cdot y_2 & \dots & \alpha_n^{t-1} \cdot y_n \end{pmatrix}, \tag{9.5}$$

où les y_i sont les inconnues. A noter que chaque composante de la matrice présentée dans l'Équation (9.5) est à valeur dans $\mathbb{F}_{2^m}$ et comme le code de Goppa est à valeur dans $\mathbb{F}_2$, une matrice de contrôle est la matrice $\bar{\mathcal{H}}$ vue sur $\mathbb{F}_2$, en appliquant l'isomorphisme $\mathbb{F}_{2^m} \to \mathbb{F}_2^m$.
Il faut se rendre compte que la matrice $\mathcal{Y}$ est calculable par un attaquant connaissant le support $\mathscr{L}$ et de noter les $1/G(\alpha_i)$ par y_i dans la matrice $\mathcal{Z}$ dans l'Équation (9.4) pour obtenir l'Équation (9.5).
Comme l'attaquant connaît $\mathcal{S} \cdot \mathcal{G}$, il peut appliquer à cette matrice, génératrice du code de Goppa privé, la méthode du pivot de Gauss (autrement dit mettre $\mathcal{S} \cdot \mathcal{G}$ sous forme systématique, voir Définition 2.29) afin d'obtenir une certaine matrice de contrôle $\mathcal{H} \in \mathscr{M}_{tm,n}(\mathbb{F}_2)$. Notons $\bar{\mathcal{H}}_2$ la matrice $\bar{\mathcal{H}}$ où l'on a remplacé chaque composante de $\mathbb{F}_{2^m}$ par un vecteur colonne de m bits. Alors il existe une matrice inversible $\mathcal{Q} \in \mathscr{M}_{tm,tm}(\mathbb{F}_2)$ telle que $\mathcal{Q} \cdot \mathcal{H} = \bar{\mathcal{H}}$. Le problème est que, ne connaissant pas exactement $\bar{\mathcal{H}}$, on ne connaît pas $\mathcal{Q}$ et vice-versa. On sait seulement que ces matrices existent.
Notons $\mathcal{Y}_{2,m} \in \mathscr{M}_{tm,nm}(\mathbb{F}_2)$ la matrice correspondant à $\mathcal{Y}$, où chaque élément de $\mathbb{F}_{2^m}$ est écrit comme une matrice diagonale dans $\mathscr{M}_{m,m}(\mathbb{F}_2)$ et $\mathcal{Z}_2 \in \mathscr{M}_{nm,n}(\mathbb{F}_2)$ la matrice correspondant à $\mathcal{Z}$, où chaque composante est écrite comme un vecteur de m bits. Alors on obtient la relation suivante :

$$\mathcal{Q} \cdot \mathcal{H} = \mathcal{Y}_{2,m} \cdot \mathcal{Z}_2, \tag{9.6}$$

où les $(tm \times tm)$ composantes de $\mathcal{Q}$ et nm composantes de $\mathcal{Z}_2$ (parmis les $nm \times n$) sont inconnues. Cela nous fournit un système linéaire à tmn équations (c'est-à-dire le nombre de composantes de $\bar{\mathcal{H}}_2$ sur $\mathbb{F}_2$) et $(tm)^2 \times nm$ inconnues sur $\mathbb{F}_2$. (Le système est linéaire et a plus d'équations que d'inconnues.)
Une fois la matrice $\bar{\mathcal{H}}$ retrouvée, il est évident que prendre $(t+1)$ éléments de sa diagonale nous fournit $(t+1)$ inverses d'évaluations de G sur $\mathscr{L}$, que l'on peut ensuite interpoler pour retrouver G. Ces $(t+1)$ évaluations de G seront les inverses de nos inconnues y_i dans $\mathbb{F}_{2^m}$, qui existent puisque $\mathscr{L}$ a été choisi de sorte qu'aucun de ses éléments ne soit racine de G.
Néanmoins ce système n'est pas si simple à résoudre [Hei87, Section 5.6]. En effet, outre la solution triviale nulle et la solution qui nous intéresse, il peut y en avoir d'autres et il s'avère qu'il peut y avoir beaucoup de solutions dites "parasites". En revanche, cette attaque semble mieux fonctionner pour un code alternant (sans la contrainte que les y_i sont des inverses d'évaluations d'un polynôme irréductible) pour une erreur de poids au plus $\lfloor t/2 \rfloor$. Un système peut être obtenu à partir de la matrice de contrôle $\hat{\mathcal{H}}$ (donnée ci-après dans l'Équation (9.8)), mais il est évident que celui-ci est encore plus compliqué.
Des améliorations sont peut-être à chercher du côté des relations suivantes : Comme l'attaquant connaît $\mathcal{S} \cdot \mathcal{G}$, il peut utiliser les mots de code qu'il souhaite. De plus, il faut noter que $(\mathcal{S} \cdot \mathcal{G}) \cdot {}^t\bar{\mathcal{H}} = 0_{k,r} \in \mathscr{M}_{k,r}(\mathbb{F}_2)$, par définition des matrices génératrice et de contrôle d'un code. De plus, comme G est supposé unitaire, on a directement

$g_t = 1$. On peut également obtenir g_0 en utilisant le fait que $G(0) = g_0$. Mais ceci ne suffit pas.
Une autre amélioration possible pourrait provenir du résultat suivant. Un code de Goppa généré avec G^2 à la place de G, lorsque G est irréductible, est encore le même code [MS77, Chapitre 12] : $\Gamma(\mathscr{L}, G) = \Gamma(\mathscr{L}, G^2)$. Donc, en utilisant une matrice de contrôle avec G^2 à la place de G, un attaquant peut obtenir un système linéaire de deux fois plus d'équations et espérer réussir à trouver des combinaisons linéaires entre les équations afin de réduire l'espace des solutions possibles, voir même pour des erreurs de poids jusqu'à t.

Avant d'avoir connaissance de ce rapport, nous avions fait une autre proposition dans [PRD+15], en utilisant un code de Reed-Solomon généralisé (GRS). En effet, un code alternant est un code GRS spécifique et plus particulièrement, un code de Goppa est la restriction à $\mathbb{F}_2$ d'un code GRS sur $\mathbb{F}_{2^m}$. C'est pourquoi les deux propositions sont basées sur la même idée.
Présentons tout de même cette autre proposition. Celle-ci ne fait pas intervenir le fait que $\mathcal{P}$ soit connue, hormis peut-être pour remettre $\mathscr{L}$ dans l'ordre. Le code de Goppa public, défini par $\mathcal{S} \cdot \mathcal{G} \cdot \mathcal{P} = \tilde{\mathcal{G}}$, est équivalent au code de Goppa privé, défini par $\mathcal{G}$. Le code dual d'un code GRS est un autre code GRS (voir [MS77, Chapitre 10]). L'attaquant peut donc calculer une matrice de contrôle notée $\tilde{\mathcal{H}}$ à partir de la matrice génératrice publique $\tilde{\mathcal{G}}$, d'abord en appliquant un pivot de Gauss pour mettre $\tilde{\mathcal{G}}$ sous forme systématique, puis en utilisant la Définition 2.29 pour passer à $\tilde{\mathcal{H}}$. La matrice $\tilde{\mathcal{H}}$ sera alors une matrice de contrôle du code publique, équivalente à la matrice $\bar{\mathcal{H}}$. La matrice $\tilde{\mathcal{H}}$ est une matrice génératrice du code dual, donc $\tilde{\mathcal{G}} \cdot {}^t\tilde{\mathcal{H}} = 0 \in \mathcal{M}_{k,r}(\mathbb{F}_2)$. Cette relation fournit un système linéaire de $(n - mt) \times (mt)$ équations avec n inconnues.
Le système d'équations linéaires à n inconnues donné par $\tilde{\mathcal{G}} \cdot {}^t\tilde{\mathcal{H}} = 0$ peut être résolu en utilisant des méthodes d'algèbre linéaire, comme par exemple un pivot de Gauss.

9.2.3 Interpolation polynomiale

Une fois la résolution de ce système effectuée, ces précédentes "inconnues" correspondent en fait à l'évaluation du polynôme de Goppa sur chaque élément du support $\mathscr{L}$. Par une interpolation de Lagrange par exemple, l'attaquant peut retrouver tous les coefficients du polynôme de Goppa $G(X)$.

9.2.4 Complexité de l'attaque

La complexité de la résolution du système d'équations linéaires avec n inconnues est grossièrement $\mathcal{O}(n^3)$ et celle de l'interpolation de Lagrange est environ $\mathcal{O}(n^2)$. Le coût des multiplications dans $\mathbb{F}_{2^m}$ est m^2. Cela signifie que la complexité de l'attaque présentée est de $m^2(n^3+n^2)$ opérations binaires. Par exemple, pour le cryptosystème de McEliece avec $n = 1024$, $m = 10$ et $t = 38$, la complexité peut être ramenée de 2^{62} opérations binaires (complexité originale) [BLP08] à 2^{37}. Cela représente une menace critique pour l'algorithme de déchiffrement du cryptosystème de McEliece.

9.2.5 Contre-mesure

Une première contre-mesure est de ne pas prendre le support dans l'ordre lexicographique lors de la construction du code de Goppa dans la génération de clés de

McEliece. Une solution qui augmenterait l'espace de recherche pour un attaquant est de prendre n éléments dans un corps plus grand (au moins deux fois plus grand), $n < 2^m$, équivalente à la notion de non-support exposée dans [AHPT11]. Enfin, d'après [BLM11][22] , pour contrer le problème d'interpolation, il suffit de prendre t tel que $t^2 > n$ dans le but de rendre G "non-interpolable".

9.3 Par le calcul du plus grand diviseur commun de plusieurs polynômes

Nous proposons ici une nouvelle méthode, pour laquelle il faut commencer par rappeler d'autres propriétés d'un code de Goppa. Le polynôme syndrome est donné par :

$$\mathcal{S}_C(X) = \sum_{i=1}^{n} \frac{c_i}{X \oplus \alpha_i} \tag{9.7}$$

Une autre matrice de contrôle du code de Goppa $\Gamma(\mathscr{L}, G)$, en utilisant ce polynôme syndrome, est :

$$\begin{aligned}
\hat{\mathcal{H}} &= \begin{pmatrix} \frac{g_t}{G(\alpha_1)} & \frac{g_t}{G(\alpha_2)} & \cdots & \frac{g_t}{G(\alpha_n)} \\ \frac{g_{t-1}+\alpha_1 g_t}{G(\alpha_1)} & \frac{g_{t-1}+\alpha_2 g_t}{G(\alpha_2)} & \cdots & \frac{g_{t-1}+\alpha_n g_t}{G(\alpha_n)} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\sum_{i=1}^{t} \alpha_1^{i-1} g_i}{G(\alpha_1)} & \frac{\sum_{i=1}^{t} \alpha_2^{i-1} g_i}{G(\alpha_2)} & \cdots & \frac{\sum_{i=1}^{t} \alpha_n^{i-1} g_i}{G(\alpha_n)} \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} g_t & 0 & \cdots & 0 \\ g_{t-1} & g_t & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ g_1 & \cdots & g_{t-1} & g_t \end{pmatrix}}_{\mathcal{X}} \cdot \underbrace{\begin{pmatrix} \frac{1}{G(\alpha_1)} & \frac{1}{G(\alpha_2)} & \cdots & \frac{1}{G(\alpha_n)} \\ \frac{\alpha_1}{G(\alpha_1)} & \frac{\alpha_2}{G(\alpha_2)} & \cdots & \frac{\alpha_n}{G(\alpha_n)} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\alpha_1^{t-1}}{G(\alpha_1)} & \frac{\alpha_2^{t-1}}{G(\alpha_2)} & \cdots & \frac{\alpha_n^{t-1}}{G(\alpha_n)} \end{pmatrix}}_{\mathcal{Y}\cdot\mathcal{Z}} .
\end{aligned} \tag{9.8}$$

Remarque 9.1 *Comme $g_t \neq 0$, $\mathcal{X}$ est inversible. Donc, $\hat{\mathcal{H}} = \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z}$ et $\bar{\mathcal{H}} = \mathcal{Y} \cdot \mathcal{Z}$ définissent bien le même code.*

22. La section 4 de la version disponible ici :
`https://www.cdc.informatik.tu-darmstadt.de/~rlindner/publications/monoidic.pdf`

Le syndrome est le résultat du produit entre la matrice de contrôle $\hat{\mathcal{H}}$ et un mot (de code ou non) C, c'est-à-dire $S = C \cdot {}^t\hat{\mathcal{H}}$. Ce résultat peut-être vu comme un polynôme, à savoir $S_C(X) = S \cdot {}^t[X^{t-1}, \ldots, X, 1]$, c'est-à-dire comme dans l'Équation (9.7) :

$$
\begin{aligned}
S_C(X) &= \underbrace{\begin{pmatrix} c_1, & c_2, & \ldots, & c_{n-1}, & c_n \end{pmatrix}}_{\substack{C \\ 1\times n}} \cdot \underbrace{\begin{pmatrix}
\frac{g_t}{G(\alpha_1)} & \frac{g_{t-1}+\alpha_1 g_t}{G(\alpha_1)} & \cdots & \frac{\sum_{i=2}^{t} \alpha_1^{i-2} g_i}{G(\alpha_1)} & \frac{\sum_{i=1}^{t} \alpha_1^{i-1} g_i}{G(\alpha_1)} \\
\frac{g_t}{G(\alpha_2)} & \frac{g_{t-1}+\alpha_2 g_t}{G(\alpha_2)} & \cdots & \frac{\sum_{i=2}^{t} \alpha_2^{i-2} g_i}{G(\alpha_2)} & \frac{\sum_{i=1}^{t} \alpha_2^{i-1} g_i}{G(\alpha_2)} \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\frac{g_t}{G(\alpha_{n-1})} & \frac{g_{t-1}+\alpha_{n-1} g_t}{G(\alpha_{n-1})} & \cdots & \frac{\sum_{i=2}^{t} \alpha_{n-1}^{i-2} g_i}{G(\alpha_{n-1})} & \frac{\sum_{i=1}^{t} \alpha_{n-1}^{i-1} g_i}{G(\alpha_{n-1})} \\
\frac{g_t}{G(\alpha_n)} & \frac{g_{t-1}+\alpha_n g_t}{G(\alpha_n)} & \cdots & \frac{\sum_{i=2}^{t} \alpha_n^{i-2} g_i}{G(\alpha_n)} & \frac{\sum_{i=1}^{t} \alpha_n^{i-1} g_i}{G(\alpha_n)}
\end{pmatrix}}_{\substack{{}^t\hat{\mathcal{H}} \\ n\times t}} \cdot \underbrace{\begin{pmatrix} X^{t-1} \\ X^{t-2} \\ \ldots \\ X \\ 1 \end{pmatrix}}_{t\times 1} \\
&= \underbrace{\begin{pmatrix}
\underbrace{c_1 \cdot \frac{g_t}{G(\alpha_1)} + c_2 \cdot \frac{g_t}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{g_t}{G(\alpha_{n-1})} + c_n \cdot \frac{g_t}{G(\alpha_n)}}_{S_{t-1}}, \\
\underbrace{c_1 \cdot \frac{g_{t-1}+\alpha_1 g_t}{G(\alpha_1)} + c_2 \cdot \frac{g_{t-1}+\alpha_2 g_t}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{g_{t-1}+\alpha_{n-1} g_t}{G(\alpha_{n-1})} + c_n \cdot \frac{g_{t-1}+\alpha_n g_t}{G(\alpha_n)}}_{S_{t-2}}, \\
\ldots, \\
\underbrace{c_1 \cdot \frac{\sum_{i=2}^{t} \alpha_1^{i-2} g_i}{G(\alpha_1)} + c_2 \cdot \frac{\sum_{i=2}^{t} \alpha_2^{i-2} g_i}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{\sum_{i=2}^{t} \alpha_{n-1}^{i-2} g_i}{G(\alpha_{n-1})} + c_n \cdot \frac{\sum_{i=2}^{t} \alpha_n^{i-2} g_i}{G(\alpha_n)}}_{S_1}, \\
\underbrace{c_1 \cdot \frac{\sum_{i=1}^{t} \alpha_1^{i-1} g_i}{G(\alpha_1)} + c_2 \cdot \frac{\sum_{i=1}^{t} \alpha_2^{i-1} g_i}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{\sum_{i=1}^{t} \alpha_{n-1}^{i-1} g_i}{G(\alpha_{n-1})} + c_n \cdot \frac{\sum_{i=1}^{t} \alpha_n^{i-1} g_i}{G(\alpha_n)}}_{S_0}
\end{pmatrix}}_{1\times t} \cdot \underbrace{\begin{pmatrix} X^{t-1} \\ X^{t-2} \\ \ldots \\ X \\ 1 \end{pmatrix}}_{t\times 1} \\
&= \left(c_1 \cdot \frac{g_t}{G(\alpha_1)} + c_2 \cdot \frac{g_t}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{g_t}{G(\alpha_{n-1})} + c_n \cdot \frac{g_t}{G(\alpha_n)} \right) \cdot X^{t-1} \\
&\quad + \left(c_1 \cdot \frac{g_{t-1}+\alpha_1 g_t}{G(\alpha_1)} + c_2 \cdot \frac{g_{t-1}+\alpha_2 g_t}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{g_{t-1}+\alpha_{n-1} g_t}{G(\alpha_{n-1})} + c_n \cdot \frac{g_{t-1}+\alpha_n g_t}{G(\alpha_n)} \right) \cdot X^{t-2} \\
&\quad + \ldots \\
&\quad + \left(c_1 \cdot \frac{\sum_{i=2}^{t} \alpha_1^{i-2} g_i}{G(\alpha_1)} + c_2 \cdot \frac{\sum_{i=2}^{t} \alpha_2^{i-2} g_i}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{\sum_{i=2}^{t} \alpha_{n-1}^{i-2} g_i}{G(\alpha_{n-1})} + c_n \cdot \frac{\sum_{i=2}^{t} \alpha_n^{i-2} g_i}{G(\alpha_n)} \right) \cdot X \\
&\quad + \left(c_1 \cdot \frac{\sum_{i=1}^{t} \alpha_1^{i-1} g_i}{G(\alpha_1)} + c_2 \cdot \frac{\sum_{i=1}^{t} \alpha_2^{i-1} g_i}{G(\alpha_2)} + \ldots + c_{n-1} \cdot \frac{\sum_{i=1}^{t} \alpha_{n-1}^{i-1} g_i}{G(\alpha_{n-1})} + c_n \cdot \frac{\sum_{i=1}^{t} \alpha_n^{i-1} g_i}{G(\alpha_n)} \right) \cdot 1
\end{aligned}
$$

Remarque 9.2 *Chaque ligne de ${}^t\hat{\mathcal{H}}$ correspond aux coefficients du polynôme quotient $q_i(X)$ dans $G(X) = (X - \alpha_i) \cdot q_i(X) + G(\alpha_i)$, donné par la division euclidienne de $G(X)$ par $(X - \alpha_i)$, pour obtenir $G(\alpha_i)$ (le reste).*

Introduisons maintenant deux nouveaux polynômes, qui généralisent en un sens les polynômes localisateur et évaluateur d'erreurs (Définitions 2.38 et 2.39).
Soit $C \in \mathbb{F}_2^n$ un mot de code. Notons $f_C(X)$ le polynôme associé à C qui généralise

le polynôme localisateur d'erreurs. Il est donné par :

$$f_C(X) = \prod_{i=1}^{n} c_i(X - \alpha_i).$$

Notons $f'_C(x)$ le polynôme associé à C qui généralise le polynôme évaluateur d'erreurs, la dérivée du polynôme localisateur d'erreurs. Il est donné par :

$$f'_C(X) = \sum_{i=1}^{n} c_i \prod_{\substack{j=1 \\ j\neq i}}^{n} (X - \alpha_j).$$

Propriété 9.1 *On a :*

$$S_C(X) = \sum_{i=1}^{n} \frac{c_i}{X - \alpha_i} = \frac{f'_C(X)}{f_C(X)}.$$

Preuve *Sachant que sur $\mathbb{F}_2$, les opérations $-$ et $\oplus$ sont identiques et $x^2 = x$ pour $x \in \mathbb{F}_2$, on a :*

$$\begin{aligned}
S_C(X) \times f_C(X) &= \left(\sum_{i=1}^{n} \frac{c_i}{X - \alpha_i}\right) \times \prod_{i=1}^{n} c_i(X - \alpha_i) \\
&= \sum_{i=1}^{n} \left(c_i \cdot \frac{\prod_{i=1}^{n} c_i(X - \alpha_i)}{X - \alpha_i} \right) \\
&= \sum_{i=1}^{n} \left(c_i^2 \cdot \prod_{\substack{j=1 \\ j\neq i}}^{n} (X - \alpha_j) \right) \\
&= \sum_{i=1}^{n} c_i \cdot \prod_{\substack{j=1 \\ j\neq i}}^{n} (X - \alpha_j) \\
&= f'_C(X)
\end{aligned}$$

□

9.3.1 Relation entre le polynôme de Goppa et un polynôme évaluateur d'erreurs "généralisé"

Une autre manière pour retrouver le polynôme de Goppa est de calculer le plus grand diviseur commun (PGCD) de tous les polynômes évaluateurs d'erreurs généralisés $f'_C(X)$ des lignes de $\mathcal{S} \cdot \mathcal{G}$ (mots de code). D'après [MS77, Chapitre 12], nous avons :

$$C \in \Gamma(\mathscr{L}, G) \Leftrightarrow G(X) | f'_C(X).$$

Donc la deuxième méthode semble être plus simple, parce qu'il y a $(k-1)$ PGCDs à calculer dans le pire cas, au lieu d'un système linéaire "compliqué" à résoudre.

9.3.2 Relation entre le polynôme de Goppa et les polynômes de Mattson-Solomon

Considérons maintenant le cas particulier où $0 \notin \mathscr{L}$ et où $\mathscr{L}$ correspond à l'ensemble des racines $n^{\text{ièmes}}$ de l'unité. On peut alors décrire les codes de Goppa via les polynômes de Mattson-Solomon [MS77, Chapitre 12, p. 347].

Soit $A(Z) = \sum_{i=1}^{n} a_i \cdot z^i$, avec $a_i \in \mathbb{F}_2$. Son polynôme de Mattson-Solomon est donné par :

$$\mathfrak{A}(X) = \sum_{i=1}^{n} A_{-i} X^i,$$

où $A_i = A(\alpha^i)$ et $\alpha \in \mathbb{F}_{2^m}$ est une racine primitive $n^{\text{ième}}$ de l'unité. L'opération inverse est :

$$A(Z) = \sum_{i=1}^{n} \mathfrak{A}(\alpha^i) \cdot z^i.$$

Si $\mathscr{L} = \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$, alors il existe une relation entre l'Équation (9.7) et $\mathfrak{A}(X)$. Pour la donner, nous allons utiliser la notation suivante : pour tout polynôme $P(X)$, $[P(X)]_n$ désigne le reste de la division de $P(X)$ par $(X^n - 1)$.

Théorème 9.2

$$\begin{aligned} \mathfrak{A}(X) &= [X \cdot (X^n + 1) \cdot S_C(X)]_n, \\ S_C(X) &= \sum_{i=1}^{n} \frac{\mathfrak{A}(\alpha^i)}{X \oplus \alpha^i}. \end{aligned}$$

Corollaire 9.1 *Soit $\Gamma(\mathscr{L}, G)$ un code de Goppa binaire, avec $\mathscr{L} = \{1, \alpha, \ldots, \alpha^{n-1}\}$, où $\alpha \in \mathbb{F}_{2^m}$ est une racine $n^{\text{ième}}$ de l'unité. Alors :*

$$\Gamma(\mathscr{L}, G) = \{A(Z); [X^{n-1} \cdot \mathfrak{A}(X)]_n \equiv 0 \mod G(X)\}.$$

Le corollaire nous mène à une autre façon de voir le polynôme de Goppa, qui divise les polynômes de Mattson-Solomon dans le cas binaire où $\mathscr{L} = \{1, \alpha, \ldots, \alpha^{n-1}\}$. Là encore, un calcul de PGCD entre les différents polynômes de Mattson-Solomon permet de retrouver le polynôme de Goppa, mais le théorème laisse sous-entendre que passer par ces polynômes est plus compliqué que de passer par les polynômes évaluateurs d'erreurs généralisés comme présenté juste avant. De plus, le premier cas permet de prendre en compte un support plus général que dans ce cas-ci.

9.3.3 Calcul du PGCD de plusieurs polynômes

Une première méthode simple et déterministe pour calculer le plus grand diviseur commun de k polynômes sur un corps fini, $\text{pgcd}(P_1, P_2, \ldots, P_k)$ avec $P_1, P_2, \ldots, P_k \in \mathbb{F}_q[X]$, est de calculer $(k-1)$ PGCDs de deux polynômes (itérativement sur tous les polynômes).
Une autre approche pour la recherche du PGCD de plusieurs polynômes sur un corps fini, probabiliste cette fois, a été proposée dans [Con03]. Elle consiste à choisir aléatoirement $2k$ polynômes $U_1, U_2 \ldots, U_k, V_1, V_2, \ldots, V_k \in \mathbb{F}_q[X]$ de même degré (non nul) et de calculer :

$$D = \text{pgcd}\left(\sum_{j=1}^{m} U_j \cdot P_j, \sum_{j=1}^{m} V_j \cdot P_j\right)$$

On note évidemment que D est un multiple de $\text{pgcd}(P_1, P_2, \ldots, P_k)$ et il est prouvé dans cet article qu'avec une forte probabilité on obtient même $D = \text{pgcd}(P_1, P_2, \ldots, P_k)$. La probabilité de succès de cette méthode dans le cas où le corps fini a 2 éléments est toujours plus grande que $\frac{3}{10}$ pour une exécution de l'algorithme, mais devient plus grande que $\frac{51}{100}$ pour deux exécutions.

9.3.4 Complexité de l'attaque

La complexité du calcul du PGCD de k polynômes de degré au plus $(n-d)$ sur un corps fini est de l'ordre de $\mathcal{O}(k-1)(n-d)^2$, où n est la longueur du code, k sa dimension et d sa distance minimale, en supposant que ce soit la méthode déterministe qui soit utilisée. Cela donne grossièrement $\mathcal{O}(n^2)$.

9.4 Conclusion et perspectives

Les attaques présentées dans ce chapitre sont des débuts de recherche, ouvrant la discussion sur le sujet, ces travaux ayant commencé en fin de thèse. La première méthode (Section 9.2) a déjà été traitée auparavant. En revanche, la seconde (Section 9.3) semble novatrice. De plus, la première méthode est cubique en n tandis que la seconde est quadratique en n. Des tests en Sage pour vérifier ces résultats sont prévus.
Comme poursuite sur ces travaux, voici quelques pistes envisagées :
Pour de futures recherches, il peut être intéressant d'essayer l'attaque proposée dans [Hei87] en utilisant les deux matrices (Équations (9.4) et (9.8)), avec G et G^2, puis de comparer tous ces résultats.
Il serait également intéressant de regarder ce qu'il se passe si l'on ne connaît pas tout mais seulement une partie du support. En effet, si les zéros d'un mot de code tombent aux positions dont on ne connaît pas les éléments dans le support, on reste capable de calculer a priori le polynôme localisateur d'erreurs d'un mot et a fortiori le polynôme évaluateur d'erreurs (en tant que dérivée de ce dernier).
Une autre question qui vient naturellement est : combien d'équations/de polynômes sont nécessaires pour être sûr de retrouver le "bon" polynôme de Goppa ? Une possibilité serait de faire les statistiques en faisant décroître le nombre d'éléments connus du support, en partant de n et en testant la limite (a priori n/t d'après [BLM11][23]).
Faisons également une remarque sur les codes de Goppa séparables. Nous nous sommes intéressés ici aux codes de Goppa dits irréductibles, car le polynôme de Goppa est irréductible. On appelle code de Goppa séparable un code dont le polynôme de Goppa est séparable, autrement dit il n'a que des racines simples (aucune racine multiple). En posant $z_1, \ldots, z_t$ les racines du polynôme de Goppa $G(X)$, on peut l'écrire sous la forme :

$$G(X) = (X - z_1) \cdot (X - z_2) \cdots (X - z_t).$$

Le problème de reconstruction du polynôme de Goppa revient dans ce cas-là à retrouver ses racines. Pour ce faire, on peut passer par une matrice de contrôle de forme particulière (dite matrice de Cauchy), qui ne s'applique que dans ce cas-là, et qui est dite sous forme Tzeng-Zimmermann en référence à l'article [TZ75]. Cette

23. La section 4 de la version disponible ici :
`https://www.cdc.informatik.tu-darmstadt.de/~rlindner/publications/monoidic.pdf`

matrice de contrôle est donnée par :

$$\mathcal{H} = \begin{pmatrix} \frac{1}{z_1-\alpha_1} & \frac{1}{z_1-\alpha_2} & \cdots & \frac{1}{z_1-\alpha_n} \\ \frac{1}{z_2-\alpha_1} & \frac{1}{z_2-\alpha_2} & \cdots & \frac{1}{z_2-\alpha_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{z_t-\alpha_1} & \frac{1}{z_t-\alpha_2} & \cdots & \frac{1}{z_t-\alpha_n} \end{pmatrix},$$

où les z_i sont les t racines du polynôme de Goppa G de degré t et les α_i sont les éléments du support $\mathscr{L}$.
Enfin, il faudrait vérifier si une adaptation au cryptosystème de Niederreiter de cette attaque, c'est-à-dire retrouver G en connaissant $\mathscr{L}$ et $\mathcal{P}$, n'est pas possible.

Conclusion et perspectives

Nous avons proposé dans cette thèse des attaques par canaux auxiliaires contre le système de chiffrement à clé publique de McEliece. Ces attaques varient en fonction du support ciblé et du type d'analyse réalisé. De manière générale, il en ressort que toutes les parties de la clé privée (support, polynôme de Goppa et matrice de permutation) doivent être bien protégées.

Il a été démontré que le principal inconvénient du cryptosystème de McEliece est que l'attaquant a la main sur le vecteur erreur (et en particulier son poids). Ce cryptosystème est certes basé sur un problème NP-complet, qui reste difficile malgré l'hypothétique utilisation d'un ordinateur quantique à l'heure actuelle, mais il reste vulnérable à des attaques par canaux auxiliaires. Le fait que le chiffrement puisse être utilisé par un attaquant ne respectant pas la condition sur poids de l'erreur entraîne de nombreux cas à prendre en compte et des faiblesses à combler afin de proposer une implantation sécurisée de ce protocole cryptographique.

Les contre-mesures proposées pour éviter les attaques par canaux auxiliaires doivent être adaptées au support en trouvant un bon compromis entre temps d'exécution, espace mémoire utilisé et niveau de sécurité souhaité par rapport aux choix des paramètres du cryptosystème. De plus, une contre-mesure à une attaque peut faciliter le travail d'un attaquant via une autre attaque. Il est donc important de considérer, non seulement les étapes du déchiffrement du cryptosystème de McEliece de manière séparée afin de les sécuriser individuellement, mais également dans leur ensemble puisque cela peut avoir des conséquences là où on ne s'y attend pas.

En ce qui concerne le choix de la famille de codes utilisée, un distingueur pour les codes de Goppa existe, mais pour un rendement k/n élevé. Ces codes restent donc utilisables en cryptographie puisque k peut-être choisi de l'ordre de la moitié de n. Une autre piste, que certains chercheurs ont déjà commencé à étudier ces dernières années, est de regarder les codes (QC/QD)-LDPC/MDPC pour ne pas se restreindre à une seule famille de codes.

Une idée différente encore pour poursuivre ces travaux, beaucoup moins regardée quant à elle, est d'analyser les autres types de protocoles cryptographiques basés sur les codes correcteurs d'erreurs face aux attaques par canaux auxiliaires. Il semblerait que le cryptosystème de Niederreiter soit plus adapté pour les systèmes embarqués. Les variantes McBits et QcBits sont proclamées en temps constant, mais aucune analyse de consommation n'a encore été réalisée à l'heure actuelle.

Nous avons focalisé notre attention dans cette thèse sur l'algorithme de décodage de Patterson. Celui-ci est spécifique aux codes de Goppa binaires et les codes binaires sont les plus simples à implanter dans nos systèmes de communication. Cependant, les algorithmes de Berlekamp et d'Euclide étendu peuvent également être utilisés pour des codes de Goppa mais de manière plus générale (pour des codes q-aires). De plus, comme cela est mentionné à la fin de [MS77, Chap. 12], l'algorithme de Berlekamp est plus rapide mais l'algorithme d'Euclide étendu est plus simple à comprendre. D'un point de vue recherche, et notamment pour l'étude d'attaques par canaux auxiliaires, l'algorithme de Berlekamp pourrait être un choix préférable.

Notons toutefois que deux failles peuvent être soupçonnées avec les deux tests **Si** (lignes 8 et 11 de l'Algorithme 3, Chapitre 2) et aucune opération dans le cas où les conditions ne sont pas vérifiées. Ceci est critique d'un point de vue attaque par canal auxiliaire (à commencer par une attaque temporelle) et est à étudier dans de futurs travaux.

Bibliographie de l'auteur

Publications

Revue internationale avec comité de lecture

— Martin Petrvalský, **Tania Richmond**, Miloš Drutarovský, Pierre-Louis Cayrel and Viktor Fischer. *Low-complexity CBDPA Countermeasure for Resource-Constrained Embedded McEliece Cryptosystem.* Submitted to RadioEngineering 2016.

Conférences internationales avec comité de lecture et actes

— Martin Petrvalský, **Tania Richmond**, Miloš Drutarovský, Pierre-Louis Cayrel and Viktor Fischer. *Differential Power Analysis Attack on the Secure Bit Permutation in the McEliece Cryptosystem.* In RadioElektronika 2016, pp. 132-137, IEEE.

— Martin Petrvalský, **Tania Richmond**, Miloš Drutarovský, Pierre-Louis Cayrel and Viktor Fischer. *Countermeasure against the SPA Attack on an Embedded McEliece Cryptosystem.* In RadioElektronika 2015, pp. 462-466, IEEE.

— Vlad Dragoi, Pierre-Louis Cayrel, Brice Colombier and **Tania Richmond**. *Polynomial structures in code-based cryptography.* In Progress in Cryptology (IndoCrypt 2013), pp. 286-296, Springer International Publishing.

Communications

Conférences internationales avec comité de lecture

— **DPA on the 'Secure' Bit Permutation in the McEliece PKC.**
Yet Another Conference on Cryptography (YACC 2016),
Porquerolles (France), Juin 2016.
`http://yacc.univ-tln.fr/`

— **A Side-Channel Attack Against the Secret Permutation on an Embedded McEliece Cryptosystem.**
The 3rd Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2015), Grenoble (France), Mars 2015.
`http://www.date-conference.com/date15/conference/workshop-w10`

— **Towards a secure implementation of a Goppa decoder.** (Français)
Journées Codes, Cryptographie et Stéganographie (JC2S 2013),
Paris (France), Novembre 2013.
`www.arx-arceo.fr/recherche/jc2s2013_en.php`

— **Towards a secure implementation of a Goppa decoder.**
The 11th Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices (CryptArchi 2013), Fréjus (France), Juin 2013.
`labh-curien.univ-st-etienne.fr/cryptarchi/workshop13/home.html`

Conférences nationales avec comité de lecture

— **DPA on the 'Secure' Bit Permutation in the McEliece PKC.**
Rencontres Arithmétique de l'Informatique Mathématique (RAIM 2016), Banyuls-sur-mer (France), Juin 2016.
`http://raim2016.sciencesconf.org/`

— **Le décodage des codes de Goppa appliqué à la cryptographie asymétrique.**
Forum des Jeunes Mathématicien-ne-s (FJM 2013), Lyon (France), Novembre 2013.
`forum2013.sciencesconf.org`

Colloque

— **Introduction aux codes correcteurs d'erreurs.**
Journées Scientifiques de l'Université de Toulon (JS 2016), Toulon (France), Avril 2016. `http://js2016.univ-tln.fr/`

Séminaires

— **Attaques par canaux cachés contre le cryptosystème de McEliece utilisé avec les codes de Goppa classiques.**
Séminaire Cryptologie et Sécurité du laboratoire GREYC, Caen (France), le 15 Avril 2015. `https://www.greyc.fr/fr/node/2157`

— **Introduction to code-based cryptography and side-channel attacks.** (Français)
Séminaire du laboratoire Pôle Pluridisciplinaire de Matériaux et Environnement (PPME) de l'Université de Nouvelle-Calédonie, Nouméa (Nouvelle-Calédonie), le 5 Septembre 2014. `www.ppme.univ-nc.nc`

Posters

— **Code-based cryptography : A way to secure communications.**
womENcourage, le 24 Septembre 2015 (Anglais).

— **Code-based cryptography : A way to secure communications.**
Journée de la recherche de l'École Doctorale Sciences - Ingénierie - Santé, le 16 Juin 2015.

Programmation

— **Sage Days 75.**
Journées de développement du logiciel libre de Mathématiques appelé Sage[24] à des fins d'enseignement et de recherche, à Inria Saclay, Palaiseau (France), du 22 au 26 Août 2016.

24. `http://www.sagemath.org/`

Références bibliographiques

[Agi] Agilent Technologies : DSO9404A datasheet and product information.

[AHPT11] Roberto M. Avanzi, Simon Hoerder, Dan Page et Michael Tunstall : Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. Journal of Cryptographic Engineering, 1(4):271–281, November 2011.

[Alt12] Altera : Cyclone iii, 2012.

[Alt15] Altera : Cyclone v, 2015.

[Bar11] Morgan Barbier : List decoding and application to information security. Thèse de doctorat, École Polytechnique, 2011.

[BBMR14] Felipe P. Biasi, Paulo S. L. M. Barreto, Rafael Misoczki et Wilson V. Ruggiero : Scaling efficient code-based cryptosystems for embedded platforms. Journal of Cryptographic Engineering, 4(2):123–134, 2014.

[BCO04] Eric Brier, Christophe Clavier et Francis Olivier : Correlation power analysis with a leakage model. In The 6th International Workshop on Cryptographic Hardware and Embedded Systems, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BCS13] Daniel J. Bernstein, Tung Chou et Peter Schwabe : McBits : Fast constant-time code-based cryptography. In Guido Bertoni et Jean-Sébastien Coron, éditeurs : Cryptographic Hardware and Embedded Systems (CHES 2013), volume 8086 de Lecture Notes in Computer Science (LNCS), pages 250–272. Springer, Berlin, Heidelberg, 2013.

[BDL97] Dan Boneh, Richard A. DeMillo et Richard J. Lipton : On the importance of checking cryptographic protocols for faults. In Walter Fumy, éditeur : Advances in Cryptology - EUROCRYPT '97, volume 1233 de Lecture Notes in Computer Science (LNCS), pages 37–51. Springer Berlin Heidelberg, 1997.

[Ber68] Elwyn R. Berlekamp : Algebraic coding theory, volume 111. McGraw-Hill New York, 1968.

[Ber70] Elwyn R. Berlekamp : Factoring polynomials over large finite fields. Mathematics of Computation, 24(111):713–735, 1970.

[Ber73] Elwyn R. Berlekamp : Goppa codes. IEEE Transactions on Information Theory, 19(5):590–592, September 1973.

[Ber05] Daniel J. Bernstein : The Poly1305-AES Message-Authentication Code, pages 32–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Ber08] Daniel J. Bernstein : The Salsa20 Family of Stream Ciphers, pages 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux et Emmanuel Thomé : A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen et Elisabeth Oswald, éditeurs : Advances in Cryptology (EUROCRYPT 2014), volume 8441 de Lecture Notes in Computer Science (LNCS), pages 1–16. Springer Berlin Heidelberg, 2014.

[BH09] Bhaskar Biswas et Vincent Herbert : Efficient root finding of polynomials over fields of characteristic 2. On hal.inria.fr website and presented at WEWoRC 2009 (Western European Workshop on Research in Cryptology), 2009.

[BJMM12] Anja Becker, Antoine Joux, Alexander May et Alexander Meurer : Decoding Random Binary Linear Codes in $2^{n/20}$: How $1+1=0$ Improves Information Set Decoding, volume 7237 de Lecture Notes in Computer Science, pages 520–536. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BL05] Thierry P. Berger et Pierre Loidreau : How to mask the structure of codes for a cryptographic use. Designs, Codes and Cryptography, 35(1):63–79, 2005.

[BLM11] Paulo S. L. M. Barreto, Richard Lindner et Rafael Misoczki : Monoidic codes in cryptography. In Bo-Yin Yang, éditeur : Proceedings of the Fourth international conference on Post-Quantum Cryptography (PQCrypto 2011), pages 179–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[BLP08] Daniel J. Bernstein, Tanja Lange et Christiane Peters : Attacking and defending the McEliece cryptosystem. Post-Quantum Cryptography 2008, 5299:31–46, 2008.

[BMvT78] Elwyn R. Berlekamp, Robert James McEliece et Henk C. A. van Tilborg : On the inherent intractability of certain coding problems. IEEE Transactions on Information Theory, 24(3):384–386, May 1978.

[BRC60] Raj Chandra Bose et Dwijendra Kumar Ray-Chaudhuri : On a class of error correcting binary group codes. Information and Control, 3(1):68 – 79, March 1960.

[BS08] Bhaskar Biswas et Nicolas Sendrier : McEliece cryptosystem implementation : Theory and practice. In Johannes Buchmann et Jintai Ding, éditeurs : Proceedings of the Second International Workshop on Post-Quantum Cryptography (PQCrypto 2008), volume 5299 de Lecture Notes in Computer Science (LNCS), pages 47–62. Springer, Berlin, Heidelberg, 2008.

[CD10] Pierre-Louis Cayrel et Pierre Dusart : McEliece/Niederreiter PKC : Sensitivity to fault injection. In 5th International Conference on Future Information Technology (FutureTech 2010), pages 1–6, May 2010.

[CEvMS15] Cong Chen, Thomas Eisenbarth, Ingo von Maurich et Rainer Steinwandt : Differential power analysis of a McEliece cryptosystem. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko et Michalis Polychronakis, éditeurs : Applied Cryptography and Network Security (ACNS), volume 9092 de Lecture Notes in Computer Science (LNCS), pages 538–556. Springer International Publishing, 2015.

[CEvMS16] Cong Chen, Thomas Eisenbarth, Ingo von Maurich et Rainer Steinwandt : Masking large keys in hardware : A masked implementation of McEliece. Selected Areas in Cryptography (SAC 2015), 9566:293–309, September 2016.

[CFS01] Nicolas T. Courtois, Matthieu Finiasz et Nicolas Sendrier : How to achieve a McEliece-based digital signature scheme. Advances in Cryptology (ASIACRYPT 2001), 2248:157–174, 2001. url = http ://dx.doi.org/10.1007/3-540-45682-1_10.

[CGGU+14] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani et Jean-Pierre Tillich : Distinguisher-based attacks on public-key cryptosystems using Reed-Solomon codes. Designs, Codes and Cryptography, 73(2):641–666, 2014.

[Chi64] Robert T. Chien : Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes. IEEE Transactions on Information Theory, 10(4): 357–363, 1964.

[Cho16] Tung CHOU : QcBits : Constant-Time Small-Key Code-Based Cryptography, pages 280–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[CHP12] Pierre-Louis CAYREL, Gerhard HOFFMANN et Edoardo PERSICHETTI : Efficient implementation of a CCA2-secure variant of McEliece using generalized Srivastava codes. Lecture Notes in Computer Science (LNCS), 7293:138–155, May 2012.

[CMCP16] Alain COUVREUR, Irene MÁRQUEZ-CORBELLA et Ruud PELLIKAAN : Cryptanalysis of McEliece cryptosystem based on algebraic geometry codes and their subcodes. arXiv, 2016. preprint arXiv :1401.6025.

[Con03] Alessandro CONFLITTI : On computation of the greatest common divisor of several polynomials over a finite field. Finite Fields and Their Applications, 9(4):423 – 431, 2003.

[Cov73] T. M. COVER : Enumerative source encoding. IEEE Transactions on Information Theory, 19(1):73–77, January 1973.

[CS10] Pierre-Louis CAYREL et Falko STRENZKE : Side channels attacks in code-based cryptography. First International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2010), Session 2 : Side-Channel Attacks II:24–28, 2010.

[CVEYA11] Pierre-Louis CAYREL, Pascal VÉRON et Sidi Mohamed EL YOUSFI ALAOUI : A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In Alex BIRYUKOV, Guang GONG et Douglas R. STINSON, éditeurs : Selected Areas in Cryptography (SAC 2010), volume 6544 de Lecture Notes in Computer Science (LNCS), pages 171–186. Springer, Berlin, Heidelberg, 2011.

[CZ81] David G. CANTOR et Hans ZASSENHAUS : A new algorithm for factoring polynomials over finite fields. Mathematics of Computation, 36(154):587–592, 1981.

[DCCR13] Vlad DRAGOI, Pierre-Louis CAYREL, Brice COLOMBIER et Tania RICHMOND : Polynomial structures in code-based cryptography. IndoCrypt 2013, 8250:286–296, 2013.

[DH76] Whitfield DIFFIE et Martin HELLMAN : New directions in cryptography. IEEE Transactions on Information Theory, 22(6):644–654, 1976.

[DRTV07] Jean-Guillaume DUMAS, Jean-Louis ROCH, Éric TANNIER et Sébastien VARRETTE : Théorie des codes. Dunod, 2007.

[EF09] Dave EVANS et Chief FUTURIST : Top 25 technology predictions, December 2009.

[EGHP09] Thomas EISENBARTH, Tim GÜNEYSU, Stefan HEYSE et Christof PAAR : MicroEliece : McEliece for embedded devices. In Christophe CLAVIER et Kris GAJ, éditeurs : Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009), volume 5747 de Lecture Notes in Computer Science (LNCS), pages 49–64. Springer, Berlin, Heidelberg, September 2009.

[Eva13] Evariste II - a modular hardware system for development and evaluation of cryptographic functions and random number generators. `http://labh-curien.univ-st-etienne.fr/wiki-evariste-ii/`, 2013.

[FGUO+13] Jean-Charles FAUGÈRE, Valérie GAUTHIER-UMAÑA, Ayoub OTMANI, Ludovic PERRET et Jean-Pierre TILLICH : A distinguisher for high-rate McEliece cryptosystems. IEEE Transactions on Information Theory, 59(10):6830–6844, October 2013.

[Fin04] Matthieu FINIASZ : Nouvelles constructions utilisant des codes correcteurs d'erreurs en cryptographie à clef publique. Thèse de doctorat, École Polytechnique, Octobre 2004.

[FM08] Cédric FAURE et Lorenz MINDER : Cryptanalysis of the McEliece cryptosystem over hyperelliptic codes. In Eleventh International Workshop on Algebraic and Combinatorial Coding Theory, pages 99–107, 2008.

[FT02] Sergei V. FEDORENKO et Peter V. TRIFONOV : Finding roots of polynomials over finite fields. IEEE Transactions on Communications, 50(11): 1709–1711, 2002.

[Gop70] Valerii Denisovich GOPPA : A new class of linear error-correcting codes. Problemy Peredachi Informatsii, 6(3):24–30, September 1970.

[Gop83] Valerii Denisovich GOPPA : Algebraico-geometric codes. Mathematics USSR Izvestiya, 21(1):75–91, 1983.

[Ham50] Richard W. HAMMING : Error detecting and error correcting codes. Bell System Technical Journal, 29(2):147–160, 1950.

[HC15] Jingwei HU et Ray C. C. CHEUNG : An application specific instruction set processor (ASIP) for the Niederreiter cryptosystem. Cryptology ePrint Archive, Report 2015/1172, December 2015.

[Hei87] Rafi HEIMAN : On the security of cryptosystems based on linear error correcting codes. Mémoire de D.E.A., Feinburg Craduate School of the Weitzmann Institute of Science, Rehovot, August 1987.

[Her11] Vincent HERBERT : Des codes correcteurs pour sécuriser l'information numérique. Thèse de doctorat, Université Paris 6, Décembre 2011.

[Hey10] Stefan HEYSE : Low-Reiter : Niederreiter encryption scheme for embedded microcontrollers. In Nicolas SENDRIER, éditeur : Proceedings of the Third international conference on Post-Quantum Cryptography (PQCrypto 2010), volume 6061 de Lecture Notes in Computer Science (LNCS), pages 165–181. Springer, Berlin, Heidelberg, May 2010.

[Hey11] Stefan HEYSE : Implementation of McEliece based on quasi-dyadic Goppa codes for embedded devices. In Bo-Yin YANG, éditeur : Proceedings of the 4th international conference on Post-Quantum Cryptography (PQCrypto 2011), volume 7071 de Lecture Notes in Computer Science (LNCS), pages 143–162. Springer, Berlin, Heidelberg, 2011.

[HG12] Stefan HEYSE et Tim GÜNEYSU : Towards one cycle per bit asymmetric encryption : Code-based cryptography on reconfigurable hardware. In Emmanuel PROUFF et Patrick SCHAUMONT, éditeurs : Cryptographic Hardware and Embedded Systems (CHES 2012), volume 7428 de Lecture Notes in Computer Science, pages 340–355. Springer Berlin Heidelberg, 2012.

[HG13] Stefan HEYSE et Tim GÜNEYSU : Code-based cryptography on reconfigurable hardware : tweaking Niederreiter encryption for performance. Journal of Cryptographic Engineering, 3(1):29–43, April 2013.

[HMP10] Stefan HEYSE, Amir MORADI et Christof PAAR : Practical power analysis attacks on software implementations of McEliece. In Nicolas SENDRIER, éditeur : Proceedings of the Third international conference on Post-Quantum Cryptography (PQCrypto 2010), volume 6061 de Lecture Notes in Computer Science (LNCS), pages 108–125. Springer, Berlin Heidelberg, 2010.

[Hoc59] Alexis HOCQUENGHEM : Codes correcteurs d'erreurs. Chiffres, 2(147-156):8–5, September 1959.

[Hor19] William George Horner : A new method of solving numerical equations of all orders, by continuous approximation. Philosophical Transactions of the Royal Society of London, 109:308–335, 1819.

[HvMG13] Stefan Heyse, Ingo von Maurich et Tim Güneysu : Smaller keys for code-based cryptography : QC-MDPC McEliece implementations on embedded devices. In Guido Bertoni et Jean-Sébastien Coron, éditeurs : Cryptographic Hardware and Embedded Systems (CHES 2013), volume 8086 de Lecture Notes in Computer Science (LNCS), pages 273–292. Springer Berlin Heidelberg, August 2013.

[JM96] Heeralal Janwa et Oscar Moreno : McEliece public key cryptosystems using algebraic-geometric codes. Designs, Codes and Cryptography, 8(3): 293–307, 1996.

[JMV01] Don Johnson, Alfred Menezes et Scott Vanstone : The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security, 1(1):36–63, 2001.

[Ker83] Auguste Kerckhoffs : La cryptographie militaire (military cryptography). Sciences Militaires (J. Military Science, in French), 9:5 – 38, January 1883.

[KI01] Kazukuni Kobara et Hideki Imai : Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC. In Kwangjo Kim, éditeur : Public Key Cryptography (PKC 2001), volume 1992 de Lecture Notes in Computer Science (LNCS), pages 19–35. Springer, Berlin, Heidelberg, 2001.

[KJJ99] Paul C. Kocher, Joshua Jaffe et Benjamin Jun : Differential power analysis. In Michael Wiener, éditeur : Advances in Cryptology (CRYPTO 99), volume 1666 de Lecture Notes in Computer Science (LNCS), pages 388–397. Springer Berlin Heidelberg, 1999.

[KKS97] Gregory Kabatianskii, E. Krouk et Ben J. M. Smeets : A digital signature scheme based on random error-correcting codes. In Michael Darnell, éditeur : Proceedings of the 6th IMA International Conference on Crytography and Coding, volume 1355 de Lecture Notes in Computer Science (LNCS), pages 161–167. Springer, Berlin Heidelberg, 1997.

[Koc96] Paul C. Kocher : Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, éditeur : Advances in Cryptology (CRYPTO'96), volume 1109 de Lecture Notes in Computer Science (LNCS), pages 104–113, Berlin, Heidelberg, 1996. Springer.

[LDW94] Yuan Xing Li, Robert H. Deng et Xin-Mei Wang : On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. IEEE Transactions on Information Theory, 40(1):271–273, January 1994.

[Mas69] James Massey : Shift-register synthesis and BCH decoding. IEEE Transactions on Information Theory, 15(1):122–127, January 1969.

[McE78] Robert James McEliece : A public-key cryptosystem based on algebraic coding theory. Rapport technique 44, California Inst. Technol., Pasadena, CA, January 1978.

[McE02] Robert James McEliece : The Theory of Information and Coding, volume 86 de The Encyclopedia of Mathematics and Its Applications. Cambridge University Press, second edition édition, 2002.

[MCMMP13] Irene Márquez-Corbella, Edgar Martínez-Moro et Ruud Pellikaan : The non-gap sequence of a subcode of a generalized Reed-Solomon code. Designs, Codes and Cryptography, 66(1):317–333, 2013.

[MCMMP14] Irene Márquez-Corbella, Edgar Martínez-Moro et Ruud Pellikaan : On the unique representation of very strong algebraic geometry codes. Designs, Codes and Cryptography, 70(1):215–230, 2014.

[MCMMPR14] Irene Márquez-Corbella, Edgar Martínez-Moro, Ruud Pellikaan et Diego Ruano : Computational aspects of retrieving a representation of an algebraic geometry code. Journal of Symbolic Computation, 64:67 – 87, 2014. Mathematical and computer algebra techniques in cryptology.

[MS77] Florence Jessie MacWilliams et Neil James Alexander Sloane : The theory of error-correcting codes. North-Holland, 1977.

[MS07] Lorenz Minder et Amin Shokrollahi : Cryptanalysis of the Sidelnikov cryptosystem. Advances in Cryptology (EUROCRYPT 2007), 4515:347–360, 2007. url=http ://dx.doi.org/10.1007/978-3-540-72540-4_20.

[MSSS11] H. Gregor Molter, Marc Stöttinger, Abdulhadi Shoufan et Falko Strenzke : A simple power analysis attack on a McEliece cryptoprocessor. Journal of Cryptographic Engineering, 1(1):29–36, April 2011.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier et Paulo S. L. M. Barreto : MDPC-McEliece : New McEliece variants from moderate density parity-check codes. In IEEE International Symposium on Information Theory Proceedings (ISIT 2013), pages 2069–2073, July 2013.

[MVOV96] Alfred J. Menezes, Paul C. Van Oorschot et Scott A. Vanstone : Handbook of applied cryptography. CRC press, 1996.

[Nie86] Harald Niederreiter : Knapsack-type cryptosystems and algebraic coding theory. Problems of control and information theory, 15(2):159–166, 1986. PROBLEMY UPRAVLENIYA I TEORII INFORMATSII.

[NSA15] Cryptography today, August 2015.

[OS09] Raphael Overbeck et Nicolas Sendrier : Code-based cryptography, chapitre 4, pages 95–145. Springer, Berlin, Heidelberg, 2009.

[Pan66] Victor Y. Pan : On Methods of Computing the Values of Polynomials. In UspeKhi Mathematicheskikh Nauk 21, pages 103–134, 1966.

[Par89] Chang-Seop Park : Improving code rate of McEliece's public-key cryptosystem. Electronics Letters, 25(21):1466–1467, October 1989.

[Pat75] Nicholas J. Patterson : The algebraic decoding of Goppa codes. IEEE Transactions on Information Theory, 21(2):203–207, March 1975.

[Pau10] Olga Paustjan : Post quantum cryptography on embedded devices : An efficient implementation of the McEliece public key scheme based on quasi-dyadic Goppa codes. Mémoire de D.E.A., Ruhr-University Bochum, Bochum, Germany, July 2010.

[PBGV92] Bart Preneel, Antoon Bosselaers, René Govaerts et Joos Vandewalle : A software implementation of the McEliece public-key cryptosystem. In Symposium on Information Theory in the BENELUX, pages 119–126. TECHNISCHE UNIVERSITEIT DELFT, 1992.

[Poi00] David Pointcheval : Chosen-ciphertext security for any one-way cryptosystem. In H. Imai, Y. Zheng, Hideki Imai et Yuliang Zheng, éditeurs : Public Key Cryptography (PKC 2000), volume 1751 de Lecture Notes in Computer Science (LNCS), pages 129–146. Springer, Berlin, Heidelberg, 2000.

[PRD+15] Martin Petrvalský, Tania Richmond, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer : Countermeasure against the SPA attack on an embedded McEliece cryptosystem. In Radioelektronika (RADIOELEKTRONIKA), 2015 25th International Conference, pages 462–466. IEEE, April 2015.

[PRD+16a] Martin Petrvalský, Tania Richmond, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer : Differential power analysis attack on the secure bit permutation in the McEliece cryptosystem. RadioElektronika 2016, pages 132–137, April 2016.

[PRD+16b] Martin Petrvalský, Tania Richmond, Miloš Drutarovský, Pierre-Louis Cayrel et Viktor Fischer : Low-complexity CBDPA countermeasure for resource-constrained embedded McEliece cryptosystem. Submitted to RadioEngineering journal, 2016.

[Pro95] Prometheus : goppa_code.c. http://www.eccpage.com/, 1995.

[PW95] Odile Papini et Jacques Wolfmann : Algèbre discrète et codes correcteurs, volume 20 de Mathématiques et Applications. Springer, 1995.

[Ris11] Thomas Risse : How SAGE helps to implement Goppa codes and McEliece PKCSs. In The 5th International Conference on Information Technologies (ICIT 2011), pages 1–4, May 2011.

[RS60] Irving S. Reed et Gustave Solomon : Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 8(2):300–304, 1960.

[RSA78] Ronald L. Rivest, Adi Shamir et Len Adleman : A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120–126, 1978.

[Sen02] Nicolas Sendrier : Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs. Mémoire d'habilitation à diriger des recherches, Université Paris 6, Paris, France, Mars 2002.

[Sha48] Claude Elwood Shannon : A mathematical theory of communication. The Bell System Technical Journal, 27:379–423, 623–656, July, October 1948.

[Sho94] Peter W. Shor : Algorithms for quantum computation : discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pages 124–134, November 1994.

[Sid94] Vladimir M. Sidelnikov : Public-key cryptosystem based on binary Reed-Muller codes. Discrete Mathematics and Applications, 4(3):191–207, 1994.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa et Toshihiko Namekawa : A method for solving key equation for decoding Goppa codes. IEEE on Information and Control, 27:87–99, January 1975.

[SKHN76] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa et Toshihiko Namekawa : An erasures-and-errors decoding algorithm for Goppa codes. IEEE Transactions on Information Theory, 22(2):238–241, 1976.

[Soc] Ebv socrates evaluation board.

[SS92] Vladimir M. Sidel'nikov et Sergey O. Shestakov : On insecurity of cryptosystems based on generalized Reed-Solomon codes. Discrete Mathematics and Applications, 2:439–444, 1992. Translation from "On an encoding system constructed on the basis of generalized Reed-Solomon codes", Diskretn. Mat. 4, No.3, 57-63 (1992).

[SSMS10] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter et Marc Stöttinger : A timing attack against Patterson algorithm in the McEliece PKC. In Donghoon Lee et Seokhie Hong, éditeurs : Proceedings of the 12th International Conference on Information, Security and Cryptology (ICISC 2009), volume 5984 de Lecture Notes in Computer Science (LNCS), pages 161–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Ste94] Jacques Stern : A new identification scheme based on syndrome decoding. In Douglas R. Stinson, éditeur : Proceedings of the 13th

Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'93), volume 773 de Lecture Notes in Computer Science (LNCS), pages 13–21, Berlin, Heidelberg, 1994. Springer. url = http ://www.springerlink.com/content/4a27n7223h43edkp/.

[STM+08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck et Abdulhadi Shoufan : Side channels in the McEliece PKC. In Johannes Buchmann et Jintai Ding, éditeurs : The Second International Workshop on Post-Quantum Cryptography (PQCrypto 2008), volume 5299 de Lecture Notes in Computer Science (LNCS), pages 216–229. Springer, Berlin Heidelberg, October 2008.

[Str10a] Falko Strenzke : A smart card implementation of the McEliece PKC. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis et Damien Sauveron, éditeurs : Proceedings of the 4th IFIP WG 11.2 international conference on Information Security Theory and Practices : Security and Privacy of Pervasive Systems and Smart Devices (WISTP'10), volume 6033 de Lecture Notes in Computer Science (LNCS), pages 47–59. Springer, Berlin, Heidelberg, April 2010. isbn=3-642-12367-8.

[Str10b] Falko Strenzke : A timing attack against the secret permutation in the McEliece PKC. In Nicolas Sendrier, éditeur : Proceedings of the Third international conference on Post-Quantum Cryptography (PQCrypto 2010), volume 6061 de Lecture Notes in Computer Science (LNCS), pages 95–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Str11] Falko Strenzke : Message-aimed side channel and fault attacks against public key cryptosystems with homomorphic properties. Journal of Cryptographic Engineering, 1(4):283–292, 2011.

[Str12a] Falko Strenzke : Fast and secure root finding for code-based cryptosystems. In Josef Pieprzyk, Ahmad-Reza Sadeghi et Mark Manulis, éditeurs : Cryptology and Network Security, volume 7712 de Lecture Notes in Computer Science (LNCS), pages 232–246. Springer Berlin Heidelberg, December 2012.

[Str12b] Falko Strenzke : Solutions for the storage problem of McEliece public and private keys on memory-constrained platforms. In Dieter Gollmann et Felix C. Freiling, éditeurs : Information Security, volume 7483 de Lecture Notes in Computer Science (LNCS), pages 120–135. Springer Berlin Heidelberg, September 2012.

[Str13a] Falko Strenzke : A side-channel secure and flexible platform-independent implementation of the McEliece PKC–flea version 0.1. 1–. Rapport technique, Cryptosource, 2013.

[Str13b] Falko Strenzke : Timing attacks against the syndrome inversion in code-based cryptosystems. In Philippe Gaborit, éditeur : The 5th International Workshop on Post-Quantum Cryptography (PQCrypto 2013), volume 7932 de Lecture Notes in Computer Science (LNCS), pages 217–230. Springer, Berlin Heidelberg, 2013. http://eprint.iacr.org/2011/683.

[Str14] Falko Strenzke : Botan's implementation of the McEliece PKC. Rapport technique, Cryptosource, December 2014.

[SWM+09] Abdulhadi Shoufan, Thorsten Wink, H. Gregor Molter, Sorin A. Huss et Falko Strenzke : A novel processor architecture for McEliece cryptosystem and FPGA platforms. In The 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2009), pages 98–105. IEEE, July 2009.

[Szp09] Aviva SZPIRGLAS : Mathématiques Algèbre L3 : Cours complet avec 400 tests et exercices corrigés. Pearson Education, 2009.

[TJR01] T.-K. TRUONG, J.-H. JENG et I. S. REED : Fast algorithm for computing the roots of error locator polynomials up to degree 11 in Reed-Solomon decoders. IEEE Transactions on Communications, 49(5):779–783, May 2001.

[TZ75] Kenneth K. TZENG et Kuno P. ZIMMERMANN : On extending Goppa codes to cyclic codes. IEEE Transactions on Information Theory, 21(6):712–716, November 1975.

[Vér92] Pascal VÉRON : Cryptographie et codes de Goppa. Mémoire de D.E.A., Université d'Aix-Marseille II, June 1992.

[Vér97] Pascal VÉRON : Improved identification schemes based on error-correcting codes. Applicable Algebra in Engineering, Communication and Computing, 8(1):57–69, 1997.

[vMG14a] Ingo von MAURICH et Tim GÜNEYSU : Lightweight code-based cryptography : QC-MDPC McEliece encryption on reconfigurable devices. In Design, Automation and Test in Europe Conference and Exhibition (DATE 2014), pages 1–6, 3001 Leuven, Belgium, Belgium, March 2014. European Design and Automation Association. `http://dl.acm.org/citation.cfm?id=2616606.2616654`.

[vMG14b] Ingo von MAURICH et Tim GÜNEYSU : Towards side-channel resistant implementations of QC-MDPC McEliece encryption on constrained devices. In Michele MOSCA, éditeur : Post-Quantum Cryptography, volume 8772 de Lecture Notes in Computer Science (LNCS), pages 266–282. Springer International Publishing, October 2014.

[vMHG16] Ingo von MAURICH, Lukas HEBERLE et Tim GÜNEYSU : IND-CCA secure hybrid encryption from QC-MDPC Niederreiter. In Tsuyoshi TAKAGI, éditeur : Post-Quantum Cryptography : 7th International Workshop, PQCrypto 2016, Proceedings, pages 1–17. Springer International Publishing, Cham, 2016.

[vMOG15] Ingo von MAURICH, Tobias ODER et Tim GÜNEYSU : Implementing QC-MDPC McEliece encryption. ACM Trans. Embed. Comput. Syst., 14(3): 44 :1–44 :27, April 2015.

[VO82] Lodewijk B. VRIES et Kentaro ODAKA : CIRC-the error-correcting code for the compact disc digital audio system. In Audio Engineering Society Conference : 1st International Conference : Digital Audio. Audio Engineering Society, 1982.

[Wie06] C. WIESCHEBRINK : Two NP-complete problems in coding theory with an application in code based cryptography. In IEEE International Symposium on Information Theory 2006, pages 1733–1737, July 2006.

[Wie10] Christian WIESCHEBRINK : Cryptanalysis of the Niederreiter Public Key Scheme Based on GRS Subcodes, pages 61–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[WY09] Jacques-Arthur WEIL et Alain YGER : Mathématiques L3-Mathématiques appliquées : Cours complet avec 500 tests et exercices corrigés. Pearson Education, 2009.

[Zin96] Victor A. ZINOVIEV : On the solution of equations of degree ≤ 10 over finite fields $GF(2^m)$. Rapport de recherche RR-2829, INRIA, 1996.

[ZPG13] Xiaoan ZHOU, Juan PENG et Liping GUO : An improved AES masking method smartcard implementation for resisting DPA attacks. International Journal of Computer Science Issues (IJCSI), 10(01):118–122, mars 2013.

Table des figures

Liste des tableaux

Liste des algorithmes