Contribution à la sécurisation des implémentations cryptographiques basées sur les codes correcteurs d'erreurs face aux attaques par canaux auxiliaires

Guillaume Goy

Thèse de doctorat en Informatique de l'Université de Limoges
École Doctorale Sciences et Ingénierie

Contributions à la sécurisation de la cryptographie post-quantique basée sur les codes correcteurs d'erreurs face aux attaques par canaux auxiliaires

Guillaume GOY

Membres du Jury

Nicolas Aragon, Universtié de Limoges	**Examinateur**
Philippe Gaborit, Université de Limoges	**Co-directeur de thèse**
Louis Goubin, Université de Versailles	**Rapporteur**
Pierre Loidreau, Université de Rennes	**Examinateur**
Antoine Loiseau, CEA Grenoble	**Co-directeur de thèse**
Ayoub Otmani, Université de Rouen	**Président du Jury**
Tania Richmond, Université de Nouvelle-Calédonie	**Examinatrice**
Nicolas Sendrier, INRIA Paris	**Rapporteur**

Thèse soutenue publiquement le 24 juin 2024 à Limoges

Remerciements

Tout d'abord, merci à Nicolas Sendrier et Louis Goubin d'avoir accepté de rapporter cette thèse, ainsi que pour vos précieux retours sur le manuscrit. Merci à Ayoub Otmani d'avoir endossé le rôle de président du jury. Et merci aux membres du jury, Nicolas Aragon, Pierre Loidreau et Tania Richmond, pour vos questions, remarques et votre intérêt durant la soutenance.

Merci à toi, Philippe, tu as su voir en moi le doctorant que je serais avant même que j'aie chiffré mes premiers mots. Merci pour toutes les discussions et idées sur la cryptologie basée sur les codes correcteurs d'erreurs et de m'avoir transmis la passion de cette discipline. Ton soutien est un atout inestimable qui saurait éclairer les plus sombres cavernes. Merci à Antoine pour le temps et l'attention accordés à l'encadrement de ma thèse. Merci à Tania, pour ton suivi sans faille durant ma thèse et pour tous les conseils que tu as pu me donner. Merci à Richard Monvoisin pour les débats et pour le cours de zététique, qui fut sans nul doute la formation doctorale la plus intéressante que j'ai eu l'occasion de suivre durant ma thèse.

Merci aux membres du labo d'en bas, et particulièrement à Thomas, Julien, Maxime, Antoine, Valence, pour les discussions florissantes sur les attaques side-channel. Merci pour tous les conseils techniques, merci pour les outils mis en place, qui m'ont permis de faire des expériences que je n'aurais même pas pu imaginer sans vous. Sans vous, cette thèse aurait été bien plus difficile et beaucoup plus triste. Merci à Mikael pour ta curiosité, ton humanité et ton implication dans les sujets scientifiques brûlants du labo. Je n'aurais pas pu imaginer tomber sur un chef de labo plus intéressant. Merci à Chloé d'avoir passé ces 6 mois avec nous au CEA, pour ton sérieux et ton travail. Je suis ravi de continuer à travailler avec toi après cette thèse. Je suis certain que ta thèse sera une réussite.

En arrivant en thèse, je pensais trouver des collègues avec qui travailler, j'ai eu la chance d'y trouver des amis. Merci à Ninon, Julien, Francesca, Camille, Céline, Zoé, Ivan, Gaëtan, Mike, Ulysse, Amine, Nikos, Doryan, ... et tous ceux avec qui j'ai déjeuné, partagé une pause café, discuté, rigolé durant ces trois années. Merci à vous tous pour votre bonne humeur, les pauses, les parties d'échecs, les énigmes mathématiques, les soirées jeux de société, les discussions sur tout et très souvent rien... Bref, merci pour tous ces moments qui ne se voient pas dans le manuscrit et qui sont pourtant si importants. Merci à Ninon pour ton soutien durant ces trois années, merci d'avoir été là quand ça n'allait pas et d'avoir fait de ces trois ans une belle partie de ma vie. Merci à Julien et Francesca pour votre amitié sans faille et tous les bons moments qu'on a passé et que l'on passera ensemble.

Merci aux amis qui sont loin, que je ne vois pas souvent, mais dont le soutien compte beaucoup pour moi. Merci à Emma et Léo pour tous les moments qui permettent de sortir un peu la tête de la thèse et de se reconnecter avec le monde ludique. Ce n'est que le début et j'espère noter

encore beaucoup de choses sur cette feuille iconique. Merci à Baptiste, pour ton humour, ton amitié et ta motivation pour essayer de me faire progresser en Python. Merci à Maxime, pour tous ces appels téléphoniques à des heures incongrues, pour ton soutien, les séminaires et tout simplement de croire en moi.

Merci à mes grands-parents de m'avoir donné le goût de la curiosité scientifique et artistique. And last but not least, merci à mes parents, Marilyne et Bruno, pour votre soutien constant et indéfectible tout au long de mes études. Merci à ma sœur, Camille, pour tous les moments qu'on a passés ensemble, les vacances et à cette catastrophe graphique qui orne désormais nos avant-bras. Où que j'aille, quoi que je fasse, je sais que je pourrai toujours compter sur vous trois. Soyez assurés que la réciproque est vraie. Cette thèse est aussi la vôtre.

A Rémi

TABLE DES MATIÈRES

Table des figures **XXI**

Notations

Cette petite section regroupe les notations utilisées dans ce manuscrit. Dans un but de clareté, les notations restent les mêmes dans l'ensemble du manuscrit.

pour les concepts généraux for $a \leq b$:

- $[a, b] = \{x \mid x \in \mathbb{R}\ ,\ a \leq x \leq b\}$ est un ensemble de nombres réels.
- $[\![a, b]\!] = \{x \mid x \in \mathbb{Z}\ ,\ a \leq x \leq b\}$ est un ensemble de nombres entiers.
- $\mathbb{P}(A)$ est la probabilité de l'évènement A
- Soient f et g deux polynômes de $\mathbb{A}[X]$, alors $(f - g)$ est le polynôme h défini tel que $h(x) = f(x) - g(x)$, pour tout $x \in \mathbb{A}$.
- $\mathbb{F}_q$ décrit le corps fini à q éléments. q peut être un nombre premier ou un la puissance d'un nombre premier.

pour les codes correcteurs d'erreurs

- n décrit la longeur d'un code.
- k est la dimension d'un code (on a $k \leq n$)
- d représente la distance minimale d'un code.
- t ou τ décrivent la capcité de correction d'un code.
- R est le rendement d'un code, défini comme $R := \frac{k}{n}$

pour les attaques physiques

- ζ représente un modèle de fuite théorique.

CHAPITRE 1

INTRODUCTION

Contents

1.1 La science des codes secrets

La cryptologie se décompose en deux domaines : (i) La cryptographie, du grec *kruptos* "caché" et *graphein* "écrire", est la science permettant de "cacher" des messages dans le but de les protéger. On parle alors de technique de chiffrement, pour transformer un message clair en un message chiffré, et de déchiffrement pour faire l'opération inverse. Le déchiffrement ne peut être effectué qu'en connaissant la clé de déchiffrement, conservée secrète pour la sécurité des échanges. (ii) La cryptanalyse vise à retrouver un message clair à partir d'un message chiffré, sans connaître la clé secrète de déchiffrement. On dit alors que le message chiffré est décrypté. On notera que le terme "crypter" n'existe pas en français. Bien que largement répandu dans le vocabulaire commun (notamment dans les médias, films et séries), il s'agit d'un mauvais anglicisme de "to encrypt", qui veut dire chiffrer.

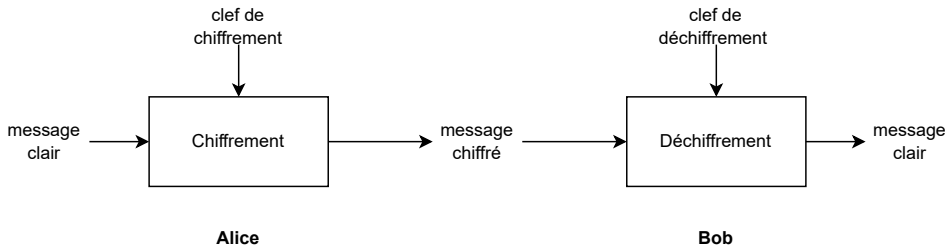


FIGURE 1.1 – Fonctionnement général d'un système de chiffrement cryptographique

La cryptologie est donc un jeu entre cryptographes, qui tentent de trouver les systèmes de chiffrement les plus robustes possibles, et cryptanalystes, dont le but est de décrypter les messages. Si les messages peuvent être décryptés en un temps raisonnable, on dit que le schéma est cassé et il faut donc le modifier ou le changer pour garantir la sécurité des messages.

1.2 Brève histoire de la cryptologie

Cette section débute par une exploration approfondie des premiers pas de la cryptologie, soulignant l'importance historique de techniques telles que le code de César, le carré de Polybe, et la machine ENIGMA. On s'intéresse aux méthodes qui ont posé les premières bases de la cryptographie, tout en reflétant les besoins et les limites technologiques de leur époque. L'accent est mis sur l'évolution des techniques de chiffrement, de la simple substitution jusqu'aux systèmes mécaniques plus complexes tels que la machine ENIGMA.

Lorsque je parle de mon sujet de thèse, on me demande souvent à quoi sert la cryptologie et dans quel contexte elle est utilisée. Je donne ici quelques

exemples non exhaustifs et une rapide description de son histoire. Historiquement, la cryptologie est surtout utilisée dans le domaine militaire afin d'assurer que les informations sensibles (position des troupes, mouvements, ...) ne soient pas connues de l'ennemi. Mais on retrouve aussi des traces dans d'autres domaines : une tablette d'argile gravée d'une recette secrète, datant du XVIème siècle av. J.-C., a été retrouvée en Irak. La modification de l'orthographe des mots et la suppression des consonnes permettrait de garantir la sécurité du secret de la recette. Il s'agit là du premier "document" chiffré dont nous ayons la trace [Kah96].

1.2.1 Systèmes de chiffrement monoalphabétique

Un chiffrement monoalphabétique, ou substitution monoalphabétique, est un système de chiffrement où chaque lettre du message clair est remplacée par un nouveau symbole (lettre, forme, nombre, ...), de telle sorte que chaque lettre soit toujours représentée par le même symbole dans le message chiffré.

Code de César Le système de chiffrement monoalphabétique le plus connu, encore utilisé aujourd'hui dans certains *escape games*, est le code de César. Il s'agit d'un chiffrement monoalphabétique pour lequel les lettres d'un message clair sont décalées d'un certain nombre de positions dans l'alphabet (voir figure 1.2). Ce nombre de lettres de décalage est conservé secret et permet de déchiffrer le message en appliquant l'opération inverse. Le code de César est peu robuste ; on se rend vite compte qu'il n'existe que 25 manières différentes de chiffrer un message (26 si l'on considère le décalage de 0 lettres, qui renvoie simplement le message en clair). Ce faible nombre de possibilités permet de toutes les tester et de finalement retrouver le message ; cela prend quelques minutes avec une feuille et un crayon, c'est instantané avec un ordinateur.

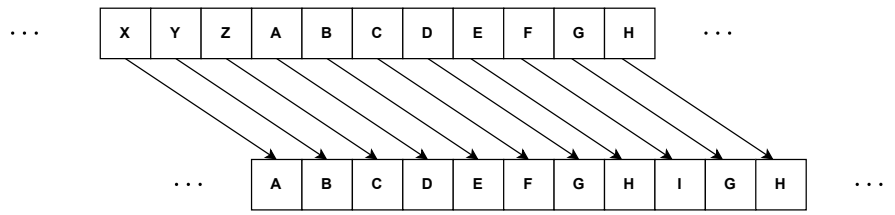


FIGURE 1.2 – Exemple de la structure du code de César pour un décalage de 3 lettres

Exemples d'autres chiffrements monoalphabétiques En présentant la cryptologie devant un public non averti (collégiens, visiteurs de la fête de la science, ...), j'aime envisager avec eux d'autres méthodes de chiffrement

monoalphabétiques. Dans l'histoire, il en existe des tas ; on peut citer par exemple : le carré de Polybe, le chiffrement des templiers (ou chiffrement de Cornelius Agrippa), le chiffrement de Pig-pen, ...

Chiffrement monoalphabétique aléatoire Supposons que j'utilise un système de chiffrement monoalphabétique sur un message écrit en français. Je dispose donc de 26 symboles différents et je dois créer une règle de chiffrement en associant chaque lettre à un symbole unique. On peut faire cela dans l'ordre alphabétique : pour le "A", j'ai 26 choix possibles parmi les symboles, 25 pour le "B", 24 pour le "C", ... etc. Au total, il existe $26! = 403291461126605635584000000 \approx 2^{88}$ manières différentes d'associer les lettres de l'alphabet et les symboles. Il existe donc 26! manières différentes de chiffrer avec ces symboles choisis. Choisir une de ces méthodes et la conserver secrète constitue le système de chiffrement. La connaissance de l'association lettre-symbole choisie est la clé de déchiffrement.

Sécurité calculatoire Dans le cas du code de César, on a vu que l'on pouvait tester toutes les possibilités pour décrypter. Ici, même en testant 10 milliards de possibilités par seconde avec un bon ordinateur, il faudrait un peu plus de 50 millions d'années pour toutes les avoir énumérées. Ce temps de calcul est bien entendu bien trop long, garantissant que ce système de chiffrement est robuste face à ce type d'attaque dite de force brute. Mais bien entendu, les cryptanalystes ne se sont pas bornés à l'énumération de toutes les possibilités et ont trouvé, au fil des siècles, d'autres méthodes permettant de décrypter les codes monoalphabétiques.

1.2.2 Analyse fréquentielle

L'analyse fréquentielle est une méthode permettant de détecter et d'inverser un chiffrement monoalphabétique. Elle a été découverte par un savant arabe, Al-Kindi, au IXème siècle. Cette méthode se base sur la fréquence d'apparition des lettres dans la langue dans laquelle le message a été écrit. Par exemple, en français, toutes les lettres de l'alphabet ne sont pas utilisées de manière équiprobable. Il est beaucoup plus fréquent de voir un "E" qu'un "W", par exemple. On peut calculer la fréquence d'apparition de chaque lettre en comptant le nombre d'occurrences dans un corpus de textes représentatif (en analysant tous les livres d'une bibliothèque, par exemple). Cette distribution moyenne obtenue servira de base pour la suite de la cryptanalyse.

Casser le chiffrement monoalphabétique Lorsque l'on chiffre avec une méthode monoalphabétique, une même lettre est toujours représentée par le même symbole. De ce fait, dans le texte chiffré, certains symboles apparaîtront plus souvent que d'autres. On peut réassocier les lettres et les

symboles en respectant la fréquence dans le corpus de référence. Cela permet de retrouver le texte clair original, à quelques erreurs près, qui peuvent être facilement corrigées. Cette méthode prend plusieurs minutes à appliquer à la main, mais peut être facilement automatisée à l'aide d'un ordinateur.

Face à cette attaque, les systèmes de chiffrement monoalphabétiques sont dits "cassés" et il faut trouver de meilleures stratégies pour protéger les messages.

1.2.3 Systèmes de chiffrement polyalphabétiques

De nouveaux systèmes de chiffrement doivent être imaginés pour parer les attaques utilisant l'analyse fréquentielle. L'idée des chiffrements polyalphabétiques est de ne pas chiffrer une même lettre par le même symbole à chaque fois.

Code de Vigenère Le plus connu des chiffrements polyalphabétiques est le code de Vigenère [Chr06], qui utilise plusieurs systèmes de chiffrement de César en même temps. Ce code a été proposé par Blaise de Vigenère en 1586 dans son *traité des chiffres*, mais il semblerait que la méthode était déjà connue par Giovan Battista Bellaso, qui publia une méthode analogue en 1553.

Dans un premier temps, le texte en clair est divisé en blocs de k lettres. Les premières lettres de chaque groupe sont chiffrées avec un code de César de décalage d_1, les deuxièmes lettres de chaque bloc avec un code de César de décalage d_2, et ainsi de suite. La clé secrète permettant le chiffrement et le déchiffrement est donc $(k, d_1, d_2, \cdots, d_k)$. À la place des valeurs numériques $(d_1, \cdots, d_n)$, on peut choisir des lettres et considérer leur position dans l'alphabet comme la valeur du décalage qui va être employé. Cette dernière stratégie permet de considérer la clé secrète comme un mot de k lettres pouvant être plus simplement retenu et transmis. Afin de chiffrer plus facilement, et à la main, avec le code de Vigenère, l'utilisation d'un tableau de chiffrement (voir figure 1.3) rend l'opération plus agréable. Cette méthode est donc résistante à l'analyse fréquentielle, étant donné qu'une même lettre sera chiffrée différemment en fonction de sa position dans un bloc de k lettres.

Cryptanalyse du chiffrement de Vigenère On remarque que si l'on connaît k, la taille des blocs, alors on peut appliquer l'analyse fréquentielle et casser chaque code de César indépendamment, comme vu dans la partie précédente. En 1863, Friedrich Kasiski publie une méthode permettant de retrouver la taille des blocs. Cette méthode repose sur l'observation de répétitions de n-grammes, qui sont des suites de n caractères se répétant dans la version chiffrée du message. La probabilité d'apparition de ces n-grammes, provenant d'un chiffrement aléatoire, est de $\frac{1}{26}^n$; il est donc plus

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

FIGURE 1.3 – Tableau de chiffrement pour le code de Vigenère. (Peut aussi être utilisé pour le code de César en choisissant une ligne).

probable que ces n-grammes proviennent de séquences de lettres identiques dans le message initial, chiffrées de la même manière. En effet, dans un texte en français, certains mots sont souvent répétés (le, de, et, être, que, pour, ...). Si un même mot (ou groupe de lettres) apparaît à la même position dans deux groupes différents, ils seront chiffrés de la même manière. Cette observation nous donne des informations sur l'espacement entre deux blocs différents, et donc sur la taille des blocs. Une fois de plus, cette méthode peut être réalisée à la main, même si cela prend du temps, et elle est facilement automatisable sur un ordinateur. Le cryptosystème de Vigenère n'est pas robuste face à ce type d'attaque ; on dit qu'il est cassé.

Exemples d'autres chiffrements polyalphabétiques Il existe d'autres méthodes de chiffrements polyalphabétiques. On peut citer, par exemple, le code ADFGVX [PK], utilisé pendant la Première Guerre mondiale, qui est une version étendue du carré de Polybe.

Machine Enigma La machine Enigma est la machine de chiffrement et de déchiffrement utilisée par les armées allemandes, notamment lors de la Seconde Guerre mondiale. Enigma marque le début de l'automatisation de la cryptographie avec le développement des machines électromécaniques portatives. Ces machines reposent sur le principe des rotors et des contacts électriques pour réaliser des substitutions polyalphabétiques de manière très efficace. La force d'Enigma réside dans la manipulation de clés secrètes de longueur considérable : là où le chiffrement de Vigenère manipule des blocs de quelques lettres, voire dizaines de lettres, la machine Enigma manipule des clés de l'ordre de centaines de millions de lettres. Cette taille de bloc rend

impossible la méthode de cryptanalyse par analyse fréquentielle présentée plus haut.

L'initialisation de la machine repose sur le positionnement de rotors et de câbles, dont l'état initial était changé quotidiennement par les forces allemandes pour garantir la sécurité des communications. Dans sa version la plus populaire, la machine disposait de 5 rotors différents et 3 d'entre eux étaient utilisés simultanément dans la machine. L'ordre dans lequel ces 3 rotors sont disposés a une influence sur le chiffrement ; choisir trois rotors parmi les cinq offre déjà $\frac{5!}{2!}$ possibilités. Chaque rotor peut être initialement positionné de 26 manières différentes ; en tout, cela représente 26^3 possibilités. Enfin, un tableau de connexion permet de permuter deux lettres à l'aide d'un câble. En tout, 10 câbles étaient positionnés sur le tableau, représentant un total de $\frac{26!}{6!\cdot 10!\cdot 2^{10}}$. Au total, la machine peut être initialisée d'environ $1.59 \times 10^{20} \approx 2^{67}$ manières différentes. Ainsi, la connaissance du système de chiffrement, autrement dit la possession de la machine et la compréhension de son fonctionnement, n'est plus suffisante pour décrypter les messages.

Toutefois, malgré ce grand nombre de possibilités, la cryptanalyse du schéma est possible et a été réalisée avant la fin de la Seconde Guerre mondiale, ce qui a vraisemblablement contribué à l'écourter. Une faille exploitée était de remarquer qu'une lettre ne pouvait pas être chiffrée par elle-même. Des équipes de cryptanalystes, dont le célèbre Alan Turing, basées à Bletchley Park, en Angleterre, s'attaquèrent à la cryptanalyse de la machine Enigma. Cette observation permet de réduire la taille de l'espace de recherche ; avec l'utilisation de calculateurs électromécaniques appelés "bombes", Enigma était dorénavant cassée.

Les cryptanalystes de Bletchley Park ne se sont pas contentés de casser Enigma. Ils ont également pris pour cible Purple, le système de chiffrement japonais, ainsi que la machine de Lorenz (variante d'Enigma), employée à partir de 1941 par le haut commandement des forces armées allemandes. Pour ce faire, ils ont dû concevoir, à la fin de 1943, le tout premier ordinateur numérique programmable de l'histoire, appelé Colossus. Malgré son importance historique, Colossus, comme tous les autres éléments de Bletchley Park, a été détruit à la fin de la guerre.

1.2.4 Cryptologie moderne

Aujourd'hui, la cryptologie ne se fait plus avec une feuille et un stylo, ni même avec des machines mécaniques comme c'était le cas pour la machine ENIGMA. Depuis les travaux de Shannon en théorie de l'information et avec la démocratisation des ordinateurs et face à leur puissance de calcul, la cryptologie informatique s'est imposée. Les systèmes de chiffrement modernes, implémentés sur du matériel informatique, reposent sur des théorèmes mathématiques et sont prouvés sécurisés. On ne choisit plus des

méthodes de chiffrement arbitraires comme cela a pu être le cas historiquement, mais on prend en compte dès la création du schéma les attaques possibles les plus fortes pour élaborer une stratégie de sécurisation efficace. On distingue deux grandes familles de cryptologie :

• **La cryptologie symétrique** Les deux protagonistes de l'échange partagent la même information secrète, dite clef secrète ou clef partagée, permettant de faire le chiffrement et le déchiffrement. Tous les schémas de chiffrement antiques et pré-modernes sont des schémas symétriques. Une limitation de ce type de schéma apparaît lorsqu'on souhaite communiquer avec plusieurs personnes. Dans ce cas, il faut générer une nouvelle clef secrète pour chaque correspondant différent afin d'assurer qu'un d'eux ne puisse pas lire les communications d'un autre. Le nombre de clefs à générer et à mémoriser croît exponentiellement avec le nombre de personnes dans le réseau de communication. Dans un réseau de communication moderne, où théoriquement tout le monde peut être en contact avec n'importe qui, ce système est extrêmement coûteux à mettre en place.

• **La cryptologie asymétrique** La cryptologie asymétrique, également connue sous le nom de cryptographie à clef publique, propose une approche novatrice pour résoudre le défi de la gestion des clefs rencontré dans la cryptologie symétrique. Contrairement à la cryptologie symétrique, où les parties impliquées partagent la même clef secrète, la cryptologie asymétrique utilise une paire de clefs distinctes pour le chiffrement et le déchiffrement, appelées clefs publique et privée.

Dans ce système, chaque utilisateur possède une paire de clefs – une clef publique qui peut être partagée ou rendue publique, et une clef privée qui doit être gardée secrète. La clef publique est utilisée pour chiffrer les données, tandis que la clef privée correspondante est utilisée pour les déchiffrer. Ce qui est unique, c'est que même si la clef publique est connue de tous, elle ne permet pas de déduire la clef privée, assurant ainsi un niveau élevé de sécurité.

Lorsqu'un utilisateur souhaite envoyer un message confidentiel à un destinataire, il utilise la clef publique du destinataire pour chiffrer le message. Une fois chiffré, seul le destinataire, qui détient la clef privée correspondante, peut déchiffrer le message. Cela élimine la nécessité de partager une clef secrète commune entre les parties.

Cette approche asymétrique résout le problème de la multiplication des clefs rencontré dans la cryptologie symétrique, car chaque utilisateur n'a besoin que de sa propre paire de clefs. De plus, elle facilite la communication sécurisée entre plusieurs parties sans la complexité de la gestion des clefs partagées.

La cryptologie asymétrique permet aussi de faire du chiffrement. Les pre-

miers schémas basés sur ce principe sont Rivest, Shamir et Adleman (RSA) [RSA78], inventé par Rivest, Shamir et Adelman en 1977. On peut aussi citer le système de chiffrement de McEliece [McE78] inventé en 1978, qui est le premier schéma de chiffrement basé sur les codes correcteurs d'erreurs et post-quantique, mais nous y reviendrons plus en détail dans le chapitre 3. Mais la cryptologie asymétrique permet également de résoudre plusieurs problèmes cryptographiques :

— L'**échange de clefs**, permet à deux protagonistes, à partir d'un couple clef secrète / clef publique, de s'échanger de manière sécurisée une clef secrète partagée (différente de la clef secrète du schéma asymétrique). Le premier schéma adressant ce problème fut proposé par Diffie et Hellman [DH76] en 1976.

— L'**authentification** permet de prouver son identité.

— La **signature**, ou signature numérique, est un protocole cryptographique qui permet de vérifier (i) l'intégrité du message qui est signé, c'est-à-dire que le message n'a pas été modifié après la signature, (ii) l'origine du message signé, autrement dit, permet d'identifier le signataire, et (iii) la non-répudiation, c'est-à-dire que le signataire du message ne peut pas prétendre qu'il n'a pas signé le message. Le premier schéma de signature est basé sur le système de chiffrement RSA.

Certaines de ces familles de schémas asymétriques peuvent être obtenues les unes à partir des autres selon des transformations plus ou moins évidentes. On a vu que connaître un schéma de signature permet d'obtenir directement un schéma d'authentification. L'inverse peut être vrai, sous certaines conditions. En effet, si un schéma d'authentification a la propriété d'être à divulgation nulle de connaissance (ou *Zero Knowledge (ZK)*), alors la transformée de Fiat-Shamir [FS86] permet d'en créer un schéma de signature. De même, la transformée de Fujisaki-Okamoto (FO) [FO99] permet de transformer un schéma de chiffrement en un protocole d'échange de clefs.

Bien que la cryptologie asymétrique offre une solution élégante à certains défis, elle peut être souvent plus gourmande en ressources, en termes de temps de calcul ou de taille de clés, par rapport à la cryptologie symétrique. C'est pourquoi une combinaison des deux approches, appelée crypto-système hybride, est souvent utilisée pour exploiter les avantages de chaque méthode. Dans un cas d'usage, deux personnes voulant communiquer de manière chiffrée peuvent utiliser un mécanisme d'échange de clefs (ou *Key Encapsulation Mechanism (KEM)*) afin de commencer par s"échanger une clef secrète partagée. Une fois la clef partagée en leur possession, ils mettent en place un système de chiffrement symétrique, moins coûteux en ressources, en utilisant leur clef partagée, pour sécuriser leurs communications.

Ces schémas modernes, symétriques et asymétriques, sont éprouvés par la communauté scientifique, notamment au travers de concours, comme ceux organisés par le National Institute of Standards and Technology (NIST),

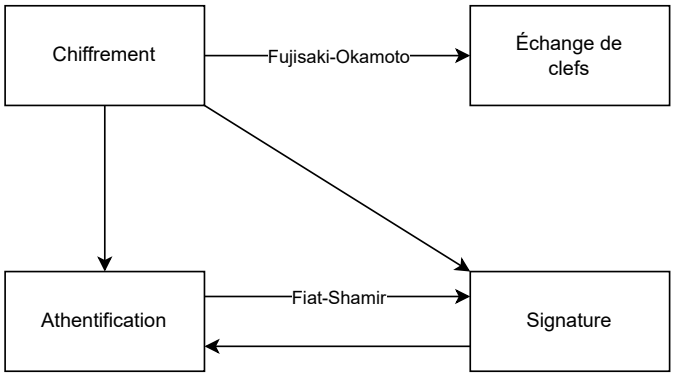


FIGURE 1.4 – Primitives de la cryptographie asymétrique et les liens entre elles.

l'agence nationale des standards américains. Un de ces concours, pour la cryptographie symétrique, s'est terminé en 2001 par la standardisation de l'Advanced Encryption Standard (AES). Cet algorithme est aujourd'hui largement déployé comme algorithme de chiffrement symétrique de référence en remplacement du Data Encryption Standard (DES), standardisé en 1977 mais devenu obsolète.

1.3 Menace quantique et crypto post-quantique

La recherche sur les ordinateurs quantiques a commencé dans les années 1980. L'idée fondamentale a été introduite par le physicien Richard Feynman [F+18] en 1982. Il a proposé l'idée d'un ordinateur qui utilise les principes de la mécanique quantique pour simuler et comprendre les phénomènes quantiques. C'est l'apparition des qubits, pour *quantum bits*, il s'agit d'une variation de la représentation binaire classique, ne pouvant prendre que les valeurs 0 ou 1. Les qubits se basent sur la physique quantique ; contrairement aux bits qui sont soit 0 soit 1, les qubits peuvent exister dans des états superposés, ce qui leur permet de représenter de multiples combinaisons de 0 et de 1 simultanément. Cela confère aux ordinateurs quantiques une puissance de calcul énorme pour certaines tâches spécifiques, leur permettant de réaliser rapidement des calculs impossibles à effectuer de manière efficace avec un ordinateur classique (utilisant des bits normaux). Les ordinateurs quantiques sont particulièrement prometteurs pour :

— La simulation de systèmes quantiques.

— L'optimisation de problèmes complexes.

— Le traitement de grandes quantités de données.

Plusieurs entreprises technologiques et institutions de recherche, telles que IBM, Google ou encore l'entreprise française Alice & Bob, travaillent

sur la construction d'ordinateurs quantiques plus puissants et fiables. Bien que les ordinateurs quantiques pleinement opérationnels ne soient pas encore une réalité, il existe des prototypes et des machines qui démontrent les principes de base de la technologie quantique. En 2019, les équipes de Google ont démontré la suprématie quantique [AAB+19] avec un ordinateur quantique de 53 qubits. La suprématie quantique est atteinte lorsqu'un ordinateur quantique réalise une tâche de calcul spécifique qui est pratiquement impossible pour un ordinateur classique. Les auteurs du papier estiment que la tâche qu'ils ont effectuée aurait demandé un temps de calcul d'environ 10,000 ans pour arriver au même résultat. Ils ont utilisé leur processeur quantique, Sycamore, pour effectuer ce calcul spécifique en 200 secondes. Ce calcul, bien que ne présentant pas d'utilité pratique immédiate, a démontré de manière pratique la capacité d'un ordinateur quantique à surpasser de loin les performances des superordinateurs classiques dans certains domaines.

1.3.1 Impact sur la Cryptographie Moderne

L'ordinateur quantique représente une menace potentielle pour la cryptographie moderne, notamment pour des systèmes comme RSA. RSA repose sur le principe que la factorisation des grands nombres est pratiquement impossible pour les ordinateurs classiques. Cependant, un ordinateur quantique pourrait réaliser cette tâche beaucoup plus rapidement, rendant les méthodes de chiffrement actuelles vulnérables. Plusieurs algorithmes ont vu le jour dans les années 1990, permettant de casser des problèmes cryptographiques de manière efficace. Parmi ces algorithmes, on peut citer :

— L'algorithme de Deutsch-Josza, qui permet de déterminer rapidement si une fonction donnée, qui renvoie soit 0 soit 1, est soit entièrement constante, soit équilibrée (c'est-à-dire renvoie un nombre égal de 0 et de 1), en utilisant un seul calcul quantique.

— L'algorithme de Bernstein-Vazirani, qui est une version restreinte de l'algorithme de Deutsh-Josza et qui permet de résoudre le problème du même nom.

— L'algorithme de Simon (ou *Simon's Algorithm*), qui résout le problème de Simon, consiste à trouver des collisions pour une certaine fonction binaire, et sert d'inspiration pour l'algorithme de Shor.

— L'algorithme de Shor [Sho94] montre que la factorisation de grands nombres et la détermination de logarithmes discrets, deux problèmes fondamentaux de la cryptographie à clef publique, peuvent être effectués en temps polynomial sur un ordinateur quantique.

— L'algorithme de Grover [Gro96] offre un avantage quadratique pour la recherche dans une base de données non structurée, ce qui a pour conséquence de réduire de moitié la sécurité de toutes les clés cryptographiques.

Ces algorithmes, qui résolvent les problèmes difficiles de la cryptographie moderne, constituent une menace pour la sécurité et les protocoles actuellement en service. Cependant, bien que ces algorithmes soient connus, ils ne sont pas encore applicables contre les schémas actuels. En effet, afin que ce genre d'algorithme puisse fonctionner, il faut un certain nombre de qubits disponibles pour faire les calculs, ainsi qu'un certain nombre de qubits dédiés à la correction des erreurs. Ainsi, les ordinateurs quantiques construits actuellement sont incapables de casser la cryptologie moderne. Des membres de la communauté cryptographique, comme Daniel J. Bernstein [Ber24], pensent qu'un ordinateur quantique sera capable de casser RSA-128 d'ici à 2032. Beaucoup d'autres sont beaucoup plus pessimistes, comme Antoine Joux, qui, dans son pari avec Bernstein, pense que RSA sera cassé par un ordinateur classique avant de l'être par un ordinateur quantique.

1.3.2 Cryptographie post-quantique (PQC)

Bien qu'une certaine incertitude règne sur la date d'apparition d'ordinateurs quantiques permettant de casser la cryptologie moderne, et dans l'éventualité où cela arriverait, il est nécessaire de trouver des solutions pour se protéger contre ce nouveau type d'attaque. Le NIST rappelle qu'il a fallu une vingtaine d'années pour déployer l'infrastructure moderne de cryptographie à clé publique, et qu'il faut donc trouver des solutions avant qu'un ordinateur quantique utilisable ne soit développé. Nous connaissons des schémas cryptographiques, utilisables avec des ordinateurs classiques, qui sont capables de résister aux algorithmes quantiques connus cités ci-dessus : c'est la cryptographie post-quantique.

La cryptographie post-quantique, aussi appelée cryptographie résistante aux ordinateurs quantiques (ou *Post-Quantum Crryptography (PQC)*), regroupe les schémas cryptographiques utilisables avec un ordinateur classique mais résistants aux algorithmes quantiques connus (voir section précédente). Elle ne doit pas être confondue avec la cryptographie quantique, qui est l'ensemble des systèmes cryptographiques qui nécessitent un ordinateur ou des propriétés quantiques. La cryptographie post-quantique vise à préparer les systèmes de sécurité informatique pour l'ère des ordinateurs quantiques. La cryptographie post-quantique a donc pour but de développer de nouveaux algorithmes qui resteront sécurisés même en présence d'ordinateurs quantiques puissants.

Dans l'optique de préparer les systèmes de sécurité d'information à pouvoir résister au calcul quantique, le NIST a lancé en 2016 un processus de sélection pour standardiser un ou plusieurs algorithmes de cryptographie post-quantique pour différentes applications : la signature numérique et l'échange de clés. Ce concours vise à identifier les algorithmes les plus sécurisés, efficaces et pratiques pour une utilisation généralisée dans l'ère post-quantique. Le processus est ouvert et collaboratif, impliquant des cryp-

tologues du monde entier, et se déroule en plusieurs tours, où les candidats sont progressivement évalués jusqu'à la sélection finale. Ce processus est crucial pour la transition vers des standards de sécurité capables de résister aux menaces potentielles posées par les ordinateurs quantiques.

Plusieurs familles de cryptosystèmes sont présentes lors du concours du NIST et garantissent une sécurité post-quantique. La sécurité des schémas qui suivent repose sur le fait que les problèmes mentionnés sont considérés comme difficiles à résoudre, même pour un ordinateur quantique.

La cryptographie basée sur les réseaux euclidiens La cryptographie basée sur les réseaux euclidiens (ou *lattice-based cryptography*) se base sur la difficulté de résoudre des problèmes dans les réseaux euclidiens, comme le problème du vecteur le plus court (ou *Shortest Vector Problem (SVP)*) ou celui du vecteur le plus proche (ou *Closest Vector Problem (CVP)*). On a surtout vu des schémas basés sur des réseaux euclidiens structurés, comme Kyber [BDK+18] et Dilithium [DKL+18] qui ont été standardisés à l'issue du troisième tour du concours, ou encore Saber [DKSRV18] et Falcon [PFH+20], mais il existe aussi des schémas basés sur des réseaux euclidiens non structurés comme FrodoKEM [NAB+19].

La cryptographie basée sur les fonctions de hachage La cryptographie basée sur les fonctions de hachage permet de créer des crypto-systèmes qui reposent entièrement sur les propriétés de sécurité des fonctions de hachage cryptographiques. Ces fonctions sont conçues de manière à être résistantes aux collisions et invulnérables aux attaques par ordinateur quantique. Le schéma représentant cette catégorie est Sphincs+ [BHK+19] qui a été standardisé à l'issue du troisième tour du concours.

La cryptographie basée sur les codes correcteurs d'erreurs La cryptographie basée sur les codes correcteurs d'erreurs (ou *code-based cryptography*) (voir chapitre 3) repose sur des problèmes de théorie des codes, comme le problème de décodage ou de décodage de syndrome (voir section 3.1) de codes linéaires aléatoires. Les schémas soumis au concours du NIST sont Hamming Quasi-Cyclic (HQC) [AMAB+17] (voir section 3.5), Bit Flipping Key Encapsulation (BIKE) [ABB+17] (voir section 3.4), et ClassicMcEliece[BCL+] (voir section 3.3), qui sont trois des quatre cryptosystèmes qui ont été conservés pour un quatrième tour dans le concours du NIST. D'autres schémas basés sur les codes, comme Rank Quasi-Cyclic (RQC) [AMAB+20], ont été soumis mais pas retenus par le NIST.

La cryptographie basée sur le multivarié La cryptographie multivariée est basée sur la difficulté de résoudre des systèmes d'équations polynomiales multivariées. Contrairement aux systèmes basés sur les nombres

entiers, les problèmes multivariés restent difficiles à résoudre même pour les ordinateurs quantiques. Cette famille est principalement utilisée pour des signatures numériques (Rainbow [DS05], GeMSS [CFMR$^+$17]) et est connue pour sa rapidité de traitement et la petite taille de ses signatures. Les schémas cryptographiques basés sur le multivarié n'ont pas été retenus par le NIST.

La cryptographie basée sur les isogénies Cette approche utilise des calculs sur des courbes elliptiques et leurs isogénies, c'est-à-dire les relations entre des courbes elliptiques. Bien que ce soit un domaine relativement nouveau en cryptographie, il offre des promesses en termes de sécurité contre les attaques quantiques, tout en ayant des clés de taille relativement petite comparées à d'autres systèmes post-quantiques. Le schéma Supersingular Isogeny Key Encapsulation (SIKE) [JAC$^+$17] présent au quatrième tour du NIST a été la cible d'une attaque algébrique le rendant complètement inutilisable. D'autres schémas basés sur les isogénies, comme Short Quaternion and Isogeny Signature (SQIsign) [DFKL$^+$20], CGL [CLG09], ou Commutative Supersingular Isogeny Diffie–Hellman (CSIDH) [CLM$^+$18], pourraient présenter des alternatives pour des schémas basés sécurisés sur les isogénies.

La cryptographie basée sur le *multi-party computation* La cryptographie basée sur le calcul multi-parties (ou *Multi-Party Computation (MPC)*) est une structure de schéma introduite par Feneuil et al. [FJR22] dans le papier "MPC is the head". Lors de l'appel du NIST pour la standardisation de schémas de signature non basés sur les réseaux euclidiens structurés, plusieurs schémas basés sur cette construction ont vu le jour : MQOM [BFR23] et Biscuit [Per], basés sur la cryptographie multivariée. PERK [ABB$^+$23a], basé sur le *Permuted Kernel Problem (PKP)* venant de la théorie des codes correcteurs d'erreurs. SDitH [AMGH$^+$23], basé sur le problème du décodage de syndrome en métrique de Hamming. RYDE [ABB$^+$23c], basé sur le problème du décodage de syndrome en métrique rang. MIRA [ABCD$^+$23] et MiRitH [ABB$^+$23b], basés sur le problème *MinRank* [FEDS13] des codes correcteurs d'erreurs en métrique rang.

1.4 Attaques par canaux auxiliaires

Les attaques par canaux auxiliaires (Side-Channel Attacks (SCA) en anglais) sont une classe d'attaques physiques exploitant des informations provenant de canaux secondaires afin de retrouver des informations secrètes manipulées par un composant matériel lors de l'exécution d'un algorithme manipulant des secrets, par exemple un schéma cryptographique. Les canaux secondaires peuvent être de différents types, tels que le temps d'exécution, la consommation de courant, le rayonnement électromagnétique, le bruit,

la chaleur, etc., générés par la machine qui exécute les algorithmes cryptographiques. Il faut attendre les travaux de Paul Kocher en 1996 [Koc96] pour montrer que ces mesures physiques permettent de déduire des informations sur des données secrètes manipulées par un algorithme de chiffrement tel que RSA. Par la suite, des travaux ont montré que ces attaques permettent souvent de contourner les défenses conventionnelles de la cryptographie. En effet, même une quantité très faible d'information qui fuit peut être exploitée pour casser la sécurité d'un algorithme cryptographique. Ces attaques posent donc la question de la sécurité matérielle des composants manipulant les données sensibles. Dans ce manuscrit, nous verrons différents types d'attaques par canaux auxiliaires contre la cryptographie basée sur les codes correcteurs d'erreurs, ainsi que différentes manières de se protéger face à ces attaques.

1.5 Mes contributions

Dans ce contexte, durant ma thèse, j'ai présenté trois attaques physiques contre HQC. Les chapitres 6, 5 et 7 sont consacrés à ces contributions.

— Dans le chapitre 5, je présente une attaque par chiffrement choisi contre HQC, dont le but est de retrouver la clef secrète statique de HQC. Cette attaque permet de retrouver la clef secrète de HQC en moins de 20,000 mesures physiques et a fait l'objet d'une publication à PQcrypto 2022 (*International Conference on Post-Quantum Cryptography*).

— Dans le chapitre 6, je présente une attaque horizontale contre HQC, visant à retrouver la clef partagée du schéma en exploitant une mesure physique unique. Cette attaque nécessite cependant 2^{96} opérations algébriques pour retrouver la clef. Bien qu'elle démontre la présence de vulnérabilités dans l'implémentation de HQC, elle est donc inutilisable en pratique. Ces travaux ont été publiés à WCC 2022 (*International Workshop on Coding and Cryptography*).

— Dans le chapitre 7, je présente une attaque contre HQC exploitant une nouvelle construction basée sur la propagation de croyance, visant à retrouver la clef partagée en une mesure unique. Cette attaque repose sur le même principe que la précédente mais a l'avantage d'être utilisable en pratique, garantissant de retrouver la clef partagée de HQC en quelques minutes avec une mesure physique unique. Cette attaque a été publiée à CHES 2024 (*Conference on Cryptographic Hardware and Embedded Systems*).

Durant ma thèse, j'ai également encadré le stage de fin d'études de Chloé Baïsse pendant 6 mois, entre mars et août 2023. L'objectif de ce stage était d'adapter des outils d'attaque physique utilisés contre le crypto-système Kyber, basé sur les réseaux euclidiens, pour en faire une attaque pratique

sur HQC. Chloé continue aujourd'hui en thèse à l'université de Limoges, et un papier [BMG+24] sur ces travaux de stage est en cours de soumission. Cette attaque, étudiée lors du stage de Chloé, n'est pas présentée dans ce manuscrit.

Première partie

État de l'art

CHAPITRE 2

CODES CORRECTEURS D'ERREURS

Contents

2.1 Description Générale

Les Codes Correcteurs d'Erreurs (CCE) (codes correcteurs d'erreurs) ont été initialement introduits dans le contexte des télécommunications. Le but des codes correcteurs est de transmettre un message à travers un canal de communication bruité, c'est-à-dire un canal où des erreurs peuvent se produire (voir figure 2.1).

Les codes correcteurs ont pour objectif de détecter et de corriger ces erreurs afin que le message reçu, après correction, soit identique à celui envoyé par l'expéditeur.

FIGURE 2.1 – Structure d'un canal de communication binaire bruité

Encoder l'information est une tâche courante, souvent inconsciente, que l'on effectue lors d'une communication téléphonique. Par exemple, lorsque l'on épèle un mot : "H comme Hôtel, E comme Éléphant, Y comme Yoyo, ...". L'idée principale est d'ajouter de la redondance dans l'information que l'on transmet, afin de détecter d'éventuelles erreurs. En effet, si l'on entend "N comme Maman", on pourra identifier qu'une erreur s'est glissée et la corriger vers la bonne lettre.

Pour les CCE (codes correcteurs d'erreurs) plus avancés, le principe reste le même, bien que les constructions soient plus complexes et reposent sur des propriétés mathématiques.

Un CCE est décrit par deux fonctions, `Encode` et `Decode`. La fonction `Encode` transforme un message en un mot codé, c'est-à-dire un mot appartenant au code. La fonction `Decode` prend un mot du code qui a été corrompu (où certains symboles ont été changés) et renvoie le message. Si le nombre d'erreurs dans le mot du code est suffisamment faible, on s'attend à ce que le message décodé corresponde au message initialement encodé (voir figure 2.2).

De manière plus formelle, un code correcteur linéaire [1] $\mathcal{C}$ est un sous-espace vectoriel de dimension k plongé dans un espace ambiant de dimension n ($n > k$) sur un alphabet $\mathcal{A}$ de taille q. On peut le représenter comme suit :

$$\mathcal{C} \subset \mathcal{A}^n \tag{2.1}$$

L'encodage consiste en une fonction injective $\texttt{Encode} : \mathcal{A}^k \to \mathcal{A}^n$, tandis que le décodage est une fonction surjective $\texttt{Decode} : \mathcal{A}^n \to \mathcal{A}^k$.

1. Certains CCE sont non linéaires, mais ceux-ci ne seront pas abordés dans ce manuscrit.

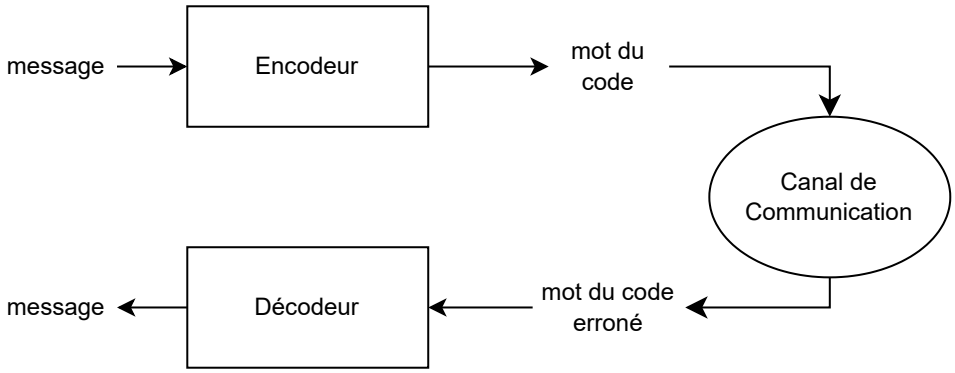


FIGURE 2.2 – Structure générale d'un CCE au-dessus d'un canal de communication bruité.

On appelle **capacité de correction**, souvent notée t, le nombre d'erreurs que le code peut corriger dans un mot du code. Le **ratio** est défini par la valeur $R = \frac{k}{n}$. Dans le contexte des télécommunications, l'objectif est de maximiser la capacité de correction tout en minimisant la redondance, c'est-à-dire corriger un maximum d'erreurs avec le minimum de redondance.

2.1.1 Exemples Simples de Codes Correcteurs

Code à Répétition Parmi les premiers codes correcteurs que l'on pourrait envisager, on trouve le code à répétition. Son principe est simple : il consiste à répéter r fois un message. On le définit comme suit :

$$\texttt{Repet}(m) = m \mid m \mid \cdots \mid m \tag{2.2}$$

où $\mid$ représente la concaténation. Pour le décodage, on prend le symbole majoritaire de chaque position dans le message répété.

Ce code correcteur permet de corriger jusqu'à $t = \lfloor \frac{r-1}{2} \rfloor$ erreurs. Cependant, cette capacité de correction s'accompagne d'un ratio de $\frac{1}{r}$, ce qui signifie que le code n'est pas très efficace en termes de longueur par rapport à la redondance ajoutée.

Codes de Hamming - Bits de Parité Les codes de Hamming ont été inventés par Richard Hamming [Ham50] dans les années 1940. Pour les codes de Hamming, la redondance ajoutée au message est représentée sous forme de bits de parité.

Les codes de Hamming forment une famille de codes où la dimension est $k = \frac{d^m-1}{d-1} - m$ et la longueur est $n = \frac{d^m-1}{d-1}$, où d est le cardinal du corps fini sur lequel le code est construit et $m > 1$ peut être choisi arbitrairement. Quel que soit le choix des paramètres, les codes de Hamming peuvent corriger une erreur.

Le code de Hamming le plus couramment utilisé est le code de Hamming binaire de dimension 4 et de longueur 7 (avec $d = 2$ et $m = 3$), aussi appelé code de Hamming $(7, 4)$. Nous allons examiner comment construire les bits de redondance pour cet exemple. Soit (m_1, m_2, m_3, m_4) les bits du message à encoder, les bits de redondance (r_1, r_2, r_3) doivent satisfaire les équations de parité suivantes :

$$\begin{aligned} r_1 &= m_1 \oplus m_2 \oplus m_3 \\ r_2 &= m_1 \oplus m_3 \oplus m_4 \\ r_3 &= m_1 \oplus m_2 \oplus m_4 \end{aligned} \tag{2.3}$$

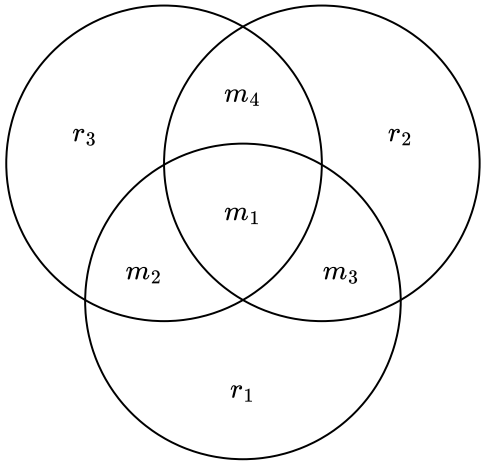


FIGURE 2.3 – Représentation des équations de parité du code de Hamming $(7, 4)$.

Si une erreur se glisse dans le mot du code, elle influence les équations décrites dans l'équation 2.3. En identifiant quelles équations restent vérifiées et lesquelles sont erronées, il est possible de localiser la position de l'erreur. Par exemple, si les deux premières équations sont incorrectes, cela signifie que le bit erroné affecte ces deux équations, donc l'erreur est soit sur m_1, soit sur m_3. Étant donné que la troisième équation est correcte, l'erreur ne peut pas être sur m_1 ; elle doit donc être sur m_3. Il est laissé au lecteur le soin de vérifier que, quelle que soit la position de l'erreur, le code peut corriger une erreur. À noter que ce code, ainsi que tout code de la famille des codes de Hamming, ne permet pas de corriger plus d'une erreur.

2.1.2 Fondements Théoriques

Plus généralement, un code linéaire est représenté par une matrice G, appelée matrice génératrice, de taille $k \times n$ définie sur $\mathbb{F}_q$. Cette matrice est

multipliée à gauche par un message m, un vecteur de longueur k, appartenant à $\mathbb{F}_q^k$. Il en résulte un mot du code de longueur n, ou longueur du code. Les propriétés de la matrice génératrice G déterminent les propriétés du code.

Forme Systématique Une matrice génératrice est dite sous la forme systématique si elle est la concaténation de la matrice identité $\mathtt{Id}_k$ de taille $k \times k$ et d'une matrice A de taille $k \times (n-k)$:

$$G = \left[\begin{array}{ccc|c} 1 & & & \\ & \ddots & & A \\ & & 1 & \end{array}\right] = [\ \mathtt{Id}_k \mid A\ ] \tag{2.4}$$

Passer une matrice sous sa forme systématique peut être effectué simplement, en temps polynomial, par une élimination gaussienne. Le code peut aussi être décrit par sa matrice de contrôle H, ou matrice de parité, de taille $(n-k) \times n$ sur $\mathbb{F}_q$, de telle sorte que :

$$G \times H^T = 0 \tag{2.5}$$

Comme les mots du code sont des combinaisons linéaires des lignes de G, il s'ensuit que pour tout mot du code c, $cH^T = mGH^T = 0$. Il existe une méthode permettant de décrire une matrice de contrôle en fonction d'une matrice génératrice sous forme systématique (voir équation 2.4), donnée par :

$$H = \left[\begin{array}{c|ccc} & 1 & & \\ -A^T & & \ddots & \\ & & & 1 \end{array}\right] = \left[\ -A^T \mid \mathtt{Id}_{n-k}\ \right] \tag{2.6}$$

Et en effet, on a bien que :

$$G \times H^T = [\ \mathtt{Id}_k \mid A\ ] \times \left[\begin{array}{c} -A \\ \mathtt{Id}_{n-k} \end{array}\right] = 0 \tag{2.7}$$

Le traitement inverse, de la matrice de parité à la matrice génératrice, peut également être réalisé. Il suffit de noter qu'une matrice de contrôle est la matrice génératrice d'un code donné.

Pour un code linéaire, l'espace $\mathbb{F}_q^n$ est muni d'une fonction de poids, à partir de laquelle est dérivée une fonction de distance $\mathtt{d}(\cdot)$. Dans la suite de ce manuscrit, et sauf mention contraire, je ferai toujours référence à des codes décrits en poids de Hamming et utilisant donc la distance de Hamming. Cependant, il existe d'autres distances permettant d'obtenir des résultats pour la théorie des codes correcteurs, notamment la métrique rang introduite par Gabidulin [Gab85] en 1985. Je ne traiterai pas cette métrique dans ce manuscrit.

Definition 2.1: (Poids de Hamming)
Soit $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbb{F}_2^n$ un vecteur binaire de longueur n, son poids de Hamming est défini comme :

$$\mathtt{HW}(\mathbf{x}) = \#\{x_i \mid x_i \neq 0\}$$

La notion de poids de Hamming peut être étendue à tout type de vecteur, pas seulement binaire, en comptant le nombre de coordonnées non nulles. Du poids de Hamming, on peut déduire une fonction de distance $\mathtt{d}$, dite distance de Hamming.

Definition 2.2: (Distance de Hamming)
La distance de Hamming $\mathtt{d}(\mathbf{x}, \mathbf{y})$ entre deux vecteurs $\mathbf{x} \in \mathbb{F}_2^n$ et $\mathbf{y} \in \mathbb{F}_2^n$ est définie par :

$$\mathtt{d}(\mathbf{x}, \mathbf{y}) = \mathtt{HW}(\mathbf{x} - \mathbf{y})$$

Pour un code correcteur et une distance $\mathtt{d}$ donnée, on parle de la distance minimale du code d comme de la plus petite distance entre deux mots du code.

$$d = \mathtt{min}\{\mathtt{d}(c_1, c_2), c_1 \neq c_2 \in \mathcal{C}\} \tag{2.8}$$

On décode un mot du code erroné par le mot du code le plus proche de l'espace selon la distance $\mathtt{d}$. Il s'ensuit que la capacité de correction t d'un code, autrement dit le poids maximal d'une erreur pouvant être corrigé ne peut pas excéder la moitié de la distance minimale :

$$t \leq \left\lfloor \frac{d-1}{2} \right\rfloor \tag{2.9}$$

Ainsi, pour un mot du code éronné $mG + e$, avec une erreur de poids de Hamming faible, $\mathtt{HW}(e) \leq t$, on décrit son syndrome s par :

$$s = (mG + e)H^T = mGH^T + eH^T = eH^T \tag{2.10}$$

Le syndrome est dépendant uniquement de l'erreur ajoutée à un mot du code, et est unique pour une erreur donnée. Cette propriété permet, dans certains cas, de faire un décodage par syndrome en retrouvant l'erreur en fonction de la valeur du syndrome calculé. Cette méthode nécessite de connaître, pour chaque erreur, le syndrome associé. Elle peut donc être utilisée pour des codes correcteurs à petits paramètres mais devient très vite inapplicable lorsque les paramètres sont trop grands (voir le problème du décodage de syndrome en sous-section 3.1.2).

Support d'un vecteur Le support $\mathsf{Supp}(\mathbf{x})$ d'un vecteur $\mathbf{x} = (x_1, \ldots, x_n)$ est l'emplacement de ses coordonnées non nulles (voir équation 2.11). Si $\mathbf{x}$ est un vecteur binaire, son support est exactement l'emplacement des uns et la connaissance du support est équivalente à la connaissance du vecteur.

$$\mathsf{Supp}(\mathbf{x}) = \{i \in \mathbb{Z} \mid x_i \neq 0\} \tag{2.11}$$

2.1.3 Bornes remarquables

Borne de Singleton La borne de Singleton concerne la distance minimale d d'un code correcteur d'erreurs de longueur n. Elle énonce une borne supérieure de la distance minimale d'un code d, dépendante de la longueur n et de la dimension k du code :

$$d \leq n - k + 1 \tag{2.12}$$

Pour un code de longueur et de dimension données, on ne peut donc pas corriger un nombre d'erreurs aussi grand qu'on le souhaite ; on est toujours limité par la borne de Singleton. Les codes pouvant corriger le plus d'erreurs (en proportion) sont donc les codes qui vérifient la borne de Singleton ($d = n - k + 1$). Ces codes dits séparable de distance maximale (ou *Maximal Distance Separable (MDS)*) sont optimaux en termes de distance minimale et donc de capacité de correction.

Les codes de Reed-Solomon (RS) (voir section 2.3) sont une des familles de CCE qui atteignent cette borne.

Borne de Hamming Pour qu'un code puisse corriger t erreurs, il faut que les boules centrées sur les mots du code de rayon t soient disjointes. Cette propriété permet d'assurer que pour toute erreur de poids plus petit ou égal à t, il existe un unique mot du code associé lors de la correction. Autrement dit, le nombre d'éléments contenus dans chacune de ces boules est inférieur ou égal au nombre d'éléments dans l'espace ambiant. Cette propriété porte le nom de borne de Hamming et, pour un code de longueur n, contenant M mots du code, la borne peut formellement s'écrire sous la forme suivante :

$$\begin{aligned} M \cdot \sum_{i=0}^{t} \binom{n}{i} (q-1)^i &\leq q^n \\ M &\leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i} (q-1)^i} \end{aligned} \tag{2.13}$$

Ainsi, pour un code binaire de dimension k, la borne s'écrit :

$$2^k \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}} \tag{2.14}$$

Dans ce contexte, on parle aussi du rayon d'empilement des sphères : il s'agit de la plus grande $\mu(e)$ telle que les boules de rayon $\mu(e)$ centrées sur les mots du code ne s'intersectent pas. On peut aussi définir le rayon de recouvrement des sphères, il s'agit de la plus grande valeur $\mu(r)$ telle que les boules de rayon $\mu(r)$ centrées sur les mots du code recouvrent tout l'espace.

On dit qu'un code est un code parfait si $\mu(e) = \mu(r)$ ou de manière équivalente si la borne de Hamming est vérifiée.

Démonstration. En effet si la borne de Hamming est vérifiée, c'est-à-dire :

$$M \cdot \sum_{i=0}^{t} \binom{n}{i} (q-1)^i = q^n \tag{2.15}$$

l'ensemble des éléments de l'espace sont contenus dans les boules de rayon $\mu(r) = t$, qui ne s'intersectent pas par définition de la capacité de correction, donc $\mu(e) = t$. □

Borne de Gilbert-Varshamov (GV) La borne de GV a été introduite indépendamment par Edgar Gilbert [Gil52] et Roman Varshamov [Var57] dans les années 1950. La borne fournit une limite théorique sur l'existence de codes correcteurs d'erreurs.

Dans le contexte de la théorie des codes, un code correcteur d'erreurs est une manière de représenter les données de telle manière que les erreurs survenues pendant la transmission ou le stockage puissent être détectées et corrigées. Ces codes sont utilisés dans divers systèmes de communication et de stockage pour assurer une transmission et un stockage de données fiables.

La borne de GV aborde spécifiquement le problème de la construction de codes capables de corriger un certain nombre d'erreurs. La borne donne une borne inférieure sur la taille d'un code avec une distance minimale spécifiée. La distance minimale d'un code est le nombre minimum de positions où deux mots de code distincts diffèrent.

Ainsi, pour un code binaire de longueur n, de distance minimale d, contenant M mots du code, la borne de GV énonce que le code existe si :

$$\frac{q^n}{\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i} \leq M \tag{2.16}$$

Pour un code binaire de dimension k, la borne s'écrit :

$$\frac{2^n}{\sum_{i=0}^{d-1} \binom{n}{i}} \leq 2^k \tag{2.17}$$

En d'autres termes, cette borne garantit qu'un code avec une distance minimale et une longueur données peut être construit tant que le nombre

de mots de code est supérieur ou égal à une certaine valeur déterminée par la borne.

Nous verrons par la suite que cette borne est également utile pour la construction de la cryptologie fondée sur les codes correcteurs d'erreurs (voir chapitre 3). Dans un contexte cryptographique, il est fréquent de trouver des codes binaires de longueur $n = 2k$, deux fois la dimension du code. Dans ce contexte, on peut approximer la valeur de borne de GV avec la fonction d'entropie binaire $H_2(\cdot)$ [Sha48, Mac03] définie comme suit :

$$H_2(x) = x\ln(x) - (1-x)\ln(1-x) \tag{2.18}$$

utilisée pour approximer la valeur des coefficient binomiaux :

$$\binom{n}{d} \approx 2^{H_2\left(n\cdot\frac{d}{n}\right)} \tag{2.19}$$

La borne de GV est vérifiée si :

$$2^k \times \sum_{i=0}^{d-1} \binom{n}{i} = 2^n \tag{2.20}$$

Or la somme centrale est majoré par $\binom{n}{d}$, que l'on peut approximer avec la fonction d'entropie binaire (voir équation 2.19), cela donne que :

$$\begin{aligned} 2^{H_2\left(n\cdot\frac{d}{n}\right)+k} &= 2^n \\ H_2\left(\frac{d}{n}\right) &= \frac{n-k}{n} \\ d &= n\cdot H_2^{-1}\left(\frac{n-k}{n}\right) \end{aligned} \tag{2.21}$$

Pour un code binaire aléatoire tel que $n = 2k$ et sachant que $H_2^{-1}(\frac{1}{2}) \approx 0.11$. La borne de GV est donc atteinte lorsque $d \approx 0.11n$.

2.1.4 Codes cycliques

Definition 2.3: (Codes cycliques)
On dit qu'un code $\mathcal{C}$ est cyclique si il vérifie la propriété suivante :

$$\forall\,(c_1,\cdots,c_{n-1},c_n) \in \mathcal{C}, (c_n,c_1,\cdots,c_{n-1}) \in \mathcal{C}$$

De nombreuses familles de codes peuvent être définies de manière cyclique, c'est le cas des codes RS, des codes Bose, Ray-Chaudhuri and Hocquenghem (BCH) et du code de Hamming. En particulier, les codes polynomiaux sont des codes cycliques. Il s'agit de codes décrits par un polynôme

générateur $g(x)$ et dont tous les mots du code sont un multiple de ce polynôme. La matrice génératrice peut s'écrire :

$$G = \begin{pmatrix} g(x) \\ Xg(X)\ [X^n - 1] \\ X^2 g(X)\ [X^n - 1] \\ \vdots \\ X^{k-1} g(x)\ [X^n - 1] \end{pmatrix} \tag{2.22}$$

La matrice génératrice du code est entièrement décrite par sa première ligne, ce qui permet de stocker le code dans un espace mémoire plus restreint. Dans le contexte de la cryptographie basée sur les codes, cette solution est fondamentale et a permis de réduire drastiquement la taille des schémas [Gab05].

2.2 Codes de Reed-Muller

En 1954, David E. Muller a publié un article [Mul54] dans lequel il présentait un nouveau type de codes de correction d'erreurs. La même année, Irving S. Reed a proposé la première méthode efficace pour décoder ces codes de correction d'erreurs [Ree54]. Les codes Reed-Muller (RM) ont été nommés en l'honneur de ces deux contributions importantes.

Les codes RM constituent une famille générale de CCE. Ils généralisent un grand nombre de codes de correction d'erreurs, y compris les codes univers, les codes de Hamming, les codes de Hadamard, les codes à répétition et d'autres. À l'origine, les codes RM sont des codes binaires linéaires, mais leur construction peut être généralisée à des alphabets plus larges. En incluant ces alphabets plus larges, les codes RM comprennent également les codes Reed-Solomon (RS) (voir Section 2.3).

2.2.1 Aperçu Mathématique

Les codes RM peuvent être décrits comme des codes récursifs.

$$\begin{aligned} \mathrm{RM}(r, m) = \{(u, u + v) \mid u \in \mathrm{RM}(r, m - 1),\ v \in \mathrm{RM}(r - 1, m - 1)\} \\ \forall m \geq 0 \text{ et } 0 \geq r \geq m \end{aligned} \tag{2.23}$$

avec

— $\mathrm{RM}(-1, m)$ est le code trivial $[2^m, 0, \infty]$; étant donné que ce code a un seul mot de code, la distance minimale est incalculable et est donc définie comme ∞.

— $\mathrm{RM}(m, m)$ est défini comme le code de l'Univers $[2^m, 2^m, 1]$; ce code est un non-code, puisque la taille de l'ensemble des messages est égale à la taille de l'ensemble des mots codés.

2.3 Codes de Reed-Solomon

Les codes Reed-Solomon (RS) ont été découverts en 1959 par Irving S. Reed et Gustave Solomon [RS60]. Historiquement, ils représentent l'une des classes de codes de correction d'erreurs les plus largement utilisées, trouvant des applications dans les CD, les DVD et les communications par satellite.

Definition 2.4: ($[n, k, t]$ Codes Reed-Solomon (RS))
Soit $(x_1, \cdots, x_n)$ des éléments distincts d'un corps fini $\mathbb{F}_q$. Pour $k \leq n$, soit $\mathbb{F}_q[X]_k$ l'ensemble de tous les polynômes de degré strictement inférieur à k sur $\mathbb{F}_q$. Un code Reed-Solomon est constitué de mots de code donnés par :

$$(f(x_1), f(x_2), \cdots, f(x_n)), f \in \mathbb{F}_q[X]_k$$

Ici, $(f_0, \cdots, f_{k-1})$, les coefficients du polynôme f, représentent le message.

Puisque le corps fini $\mathbb{F}_q$ ne contient que q éléments, il est nécessaire que $n \leq q$. Dans de nombreuses applications, $n = q - 1$ et $x_i = \alpha^{i-1}, \forall i \in [\![1, n]\!]$, avec α étant un élément primitif de $\mathbb{F}_q$.

Similairement à divers codes de correction d'erreurs largement utilisés, les codes RS exhibent la propriété de linéarité.

Proposition 2.5: (Linéarité des Codes RS)
Un code de Reed-Solomon est un code linéaire.

Démonstration. Soient deux mots de code de RS : $c_1 = (f_1(x_1), \cdots, f_1(x_n))$ et $c_2 = (f_2(x_1), \cdots, f_2(x_n))$, et soient a et b des éléments du corps fini $\mathbb{F}_q$. Alors, le mot de code $\hat{c} = ac_1 + bc_2$ est généré par le polynôme

$$f_3(x) = af_1(x) + bf_2(x)$$

Puisque $\deg(f_3) = \max(\deg(f_1), \deg(f_2)) \leq k$, f_3 appartient à $\mathbb{F}_q[X]_k$, nous conduisant à conclure que $\hat{c}$ est également un mot de code RS, établissant ainsi la linéarité. □

Proposition 2.6: (Dimension des Codes RS)
En utilisant les notations de la Définition 2.4, les codes RS possèdent une dimension de k.

Démonstration. Soient f_1 et f_2 deux polynômes distincts de $\mathbb{F}_q[X]_k$ qui génèrent le même mot de code c :

$$c = (f_1(x_1), \cdots, f_1(x_n)) = (f_2(x_1), \cdots, f_2(x_n))$$

Cela implique que $(f_1 - f_2)(x_i) = 0, \forall i \in [\![1, n]\!]$. Le polynôme $(f_1 - f_2)$ a un degré strictement inférieur à k, mais possède n racines distinctes. Comme

$k \leq n$, cela implique que $(f_1 - f_2)$ doit être le polynôme nul, conduisant à $f_1 = f_2$.

Ainsi, la dimension du code RS est la même que la dimension de $\mathbb{F}_q[X]_k$, à savoir k. □

Exploitant les propriétés de l'ensemble des polynômes, nous pouvons déduire une propriété pour la distance minimale d des codes RS.

Proposition 2.7: (Distance Minimale des Codes RS)
La distance minimale d des codes RS satisfait la borne de Singleton :

$$d = n - k + 1$$

Démonstration. Soit $c = (f(x_1), f(x_2), \cdots, f(x_n))$ un mot de code RS. Puisque f appartient à l'ensemble des polynômes $\mathbb{F}_q[X]_k$ de degré strictement inférieur à k, il a au plus $k - 1$ racines. En conséquence, $\mathtt{HW}c \geq n - (k - 1) = n - k + 1$. □

Par conséquent, la borne de Singleton est vérifiée pour les codes RS. Cela implique que ces codes sont des codes *MDS*. De plus, en raison de la propriété précédente, la distance minimale d'un code RS peut être ajustée en modifiant la valeur de k.

Les codes Reed-Solomon sont appelés codes à distance construite.

2.3.1 Décodage des Codes Reed-Solomon

Les algorithmes de Reed-Solomon sont détaillés dans "A Course in Error Correcting Codes" par Jørn Justesen et Tom Høholdt [JH04]. La section suivante est largement inspirée par ce livre.

Soit $r = c + e$ un mot de code RS reçu erroné, avec $c = (f(x_1), \cdots, f(x_n))$ étant un mot de code RS et $e = (e_1, \cdots, e_n)$ étant un vecteur ayant un poids de Hamming ne dépassant pas $t = \lfloor \frac{n-k}{2} \rfloor$, qui est la capacité de correction du code RS. Le concept central du décodage RS est de trouver un polynôme bivarié Q défini comme suit :

$$Q(x, y) = Q_0(x) + yQ_1(x) \in \mathbb{F}_q[x, y] \setminus 0 \tag{2.24}$$

Ce polynôme doit satisfaire les trois équations suivantes :

1. $Q(x_i, r_i) = 0, \forall i \in [\![1, n]\!]$
2. $l_0 := \deg(() Q_0) \leq n - 1 - t$
3. $l_1 := \deg(() Q_1) \leq n - 1 - t - (k - 1)$

Ce polynôme est appelé polynôme d'interpolation pour le mot reçu r.

Theorem 2.8: (Existence d'un Tel Polynôme)
Au moins un polynôme non nul $Q(x, y)$ dans $\mathbb{F}_q[x, y]$ satisfaisant ces trois propriétés existe.

Démonstration. La propriété 1. donne n équations linéaires homogènes sur les coefficients du polynôme. Les propriétés 2. et 3. fournissent le nombre de coefficients qui doivent être déterminés :

$$\begin{aligned}\deg(Q_0) + 1 + \deg(Q_1) + 1 &= n - 1 - t + 1 + n - 1 - t - (k-1) + 1 \\ &= 2n - 2t - k + 1 \\ &\geq 2n - (n-k)_k + 1 \\ &\geq n + 1\end{aligned}$$

Le nombre de coefficients est strictement supérieur au nombre d'équations, donc au moins un polynôme d'interpolation non nul existe. □

Theorem 2.9: (Interpolation)
Soit $r = c + e$ le mot reçu avec $c = (f(x_1), \cdots, f(x_n))$. Si le poids de l'erreur e est inférieur à $\lfloor \frac{d}{2} \rfloor$, alors

$$f(x) = \frac{-Q_0(x)}{Q_1(x)} \tag{2.25}$$

Démonstration. Soient $c = (f(x_1), \cdots, f(x_n))$, $r = +e$, et $\omega(e) \leq t$. Selon la propriété 1. du polynôme d'interpolation :

$$Q(x_i, r_i) = Q(x_i, f(x_i) + e_i) = 0, \forall i \in [1, n] \tag{2.26}$$

Cependant, $e_i = 0$ pour au moins $n - t$ coordonnées. En traitant $Q(x, f(x))$ comme un polynôme monovarié de $\mathbb{F}_q[x]$, il s'annule à au moins $n - t$ points distincts. Mais ce polynôme a un degré maximal, donné par :

$$\max(\deg(Q_0), \deg(f) + \deg(Q_1)) = n - 1 - t$$

Selon un théorème fondamental de l'algèbre, ce polynôme est nécessairement le polynôme nul. Ainsi, $Q(x, f(x)) = Q_0(x) + f(x)Q_1(x) = 0$. Cela valide le résultat désiré. □

Nous noterons que nous avons :

$$\begin{aligned}Q(x, y) &= Q_1(x)\left(y + \frac{Q_0(x)}{Q_1(x)}\right) \\ &= Q_1(x)(y - f(x))\end{aligned} \tag{2.27}$$

Et étant donné que $Q(x_i, r_i) = 0$, pour les positions, l'erreur est nulle,

$$\begin{aligned} Q_1(x_i)\left(f(x_i) + 1 - f(x_i)\right) &= 0 \\ Q_1(x_i) &= 0 \end{aligned} \tag{2.28}$$

Q_1 devient donc nul pour les positions x_i où les erreurs sont dans le message. Q_1 est appelé le polynôme localisateur d'erreurs.

2.3.2 Décodage en Liste

Dans cette section, deux algorithmes de décodage sont présentés pour les codes Reed-Solomon qui permettent de décoder des mots où les erreurs ont un poids supérieur à la capacité de correction du code. Dans ce scénario, le décodeur renvoie une liste des mots du code les plus proches du mot reçu. Cette méthode est donc appelée décodage en liste.

2.3.2.1 Décodage de Sudan

Le premier algorithme de décodage par liste a été introduit par Sudan [Sud97] en 1997. Soit $r = c + e$ le mot reçu, où $c = (f(x_1), f(x_2), \cdots, f(x_n))$ est un mot du code Reed-Solomon et e est une erreur telle que $\omega(e) \leq \tau$. L'idée reste la même que pour le décodage classique. Nous devons trouver un polynôme bivarié $Q(x, y)$ qui satisfait certaines conditions.

$$Q(x, y) = Q_0(x) + yQ_1(x) + y^2Q_2(x) + \cdots + y^lQ_l(x) \in \mathbb{F}_q[x, y] \tag{2.29}$$

et tel que :

1. $Q(x_i, r_i) = 0$, $\forall i \in [1, n]$
2. $\deg\left(()\, Q_j\right) \leq n - \tau - 1 - j(k-1)$
3. $Q(x, y) \neq 0$, Q n'est pas le polynôme nul

Pour que les n équations linéaires de la propriété 1. aient une solution non triviale, il est nécessaire que le nombre d'inconnues soit supérieur à n. Le nombre d'inconnues est le nombre de coefficients dans les polynômes Q_j, donné par la formule suivante :

$$\begin{aligned} \sum_{j=0}^{l} \left[\deg\left(Q_j\right) + 1\right] &= \sum_{j=0}^{l} \left[n - \tau - 1 - j(k-1) + 1\right] \\ &= (l+1)(n-\tau) - \frac{l(l+1)}{2}(k-1) \end{aligned} \tag{2.30}$$

Ainsi :

$$(l+1)(n-\tau) - \frac{l(l+1)}{2}(k-1) > n \tag{2.31}$$

Theorem 2.10: (Interpolation)
Si $Q(x,y)$ satisfait ces conditions et $c = (f(x_1), f(x_2), \cdots, f(x_n))$, et $\deg(f) < k$, alors $(y - f(x))|Q(x,y)$.

Démonstration. Considérons le polynôme monovarié $Q(x, f(x))$ en x dont le degré est donné par :

$$\begin{aligned}
\deg(Q) &= \max_j (\deg(Q_0), j \cdot \deg(f) + \deg(Q_j)) \\
&\leq \max_j (n - \tau - 1, j \cdot (k-1) + n - \tau - 1 - j(k-1)) \\
&\leq \max(n - \tau - 1, n - \tau - 1) \\
&\leq n - \tau - 1
\end{aligned} \tag{2.32}$$

Puisque e a un poids inférieur ou égal à τ, cela implique que $f(x_i) = r_i$ pour au moins $n - \tau$ valeurs distinctes. Pour ces $n - \tau$ valeurs, nous avons $Q(x_i, f(x_i)) = 0$. Par conséquent, le degré du polynôme $Q(x, f(x))$ est inférieur au nombre d'éléments qui l'annihilent. Par conséquent, c'est le polynôme nul. Ainsi, pour le polynôme monovarié $Q(x,y)$ en y, nous avons :

$$(y - f(x))|Q(x,y) \tag{2.33}$$

□

Ainsi, trouver les polynômes $f \in \mathbb{F}_q[x]$ qui satisfont l'équation (2.33) fournit une liste de polynômes parmi lesquels le message est à une distance inférieure à τ du mot reçu. Puisque le polynôme $Q(x, f(x))$ a un degré au plus l en y, la liste ne sera pas plus grande que la taille l. Il convient de noter que les autres éléments de la liste retournée ne sont pas nécessairement des mots de code.

Nous avons vu que l'équation (2.31) nous donne une condition primaire à respecter ; si nous la réécrivons, nous obtenons :

$$\tau < n\frac{l}{l+1} - \frac{(k-1)l}{2} \tag{2.34}$$

Une autre condition qui doit être satisfaite est que les polynômes Q_j ne sont pas de degré zéro. Cette condition peut être exprimée en disant que le polynôme de degré le plus petit, à savoir Q_l, n'est pas de degré zéro. Ainsi :

$$\begin{aligned}
n - \tau - 1 - l(k-1) &> 0 \\
n - \tau - l(k-1) &\geq 0 \\
\tau &\leq n - l(k-1)
\end{aligned} \tag{2.35}$$

Les conditions (2.34) et (2.35), nous permettent de choisir l afin de maximiser le nombre d'erreurs que nous voulons corriger.

Par exemple, si nous prenons $l = 1$, nous obtenons $\tau \leq \frac{n-k+1}{2}$, ce qui est équivalent à la capacité de correction normale d'un code Reed-Solomon.

Ce décodeur est particulièrement efficace pour améliorer la capacité de correction lorsque le rapport $\frac{k}{n}$ est faible.

2.3.2.2 Décodage de Guruswami-Sudan

Cette approche de décodage a ensuite été améliorée par Sudan et Guruswami en 1998 [GS98]. Elle améliore notamment l'efficacité du décodage pour les codes Reed-Solomon avec des rapports $\frac{k}{n}$ plus élevés.

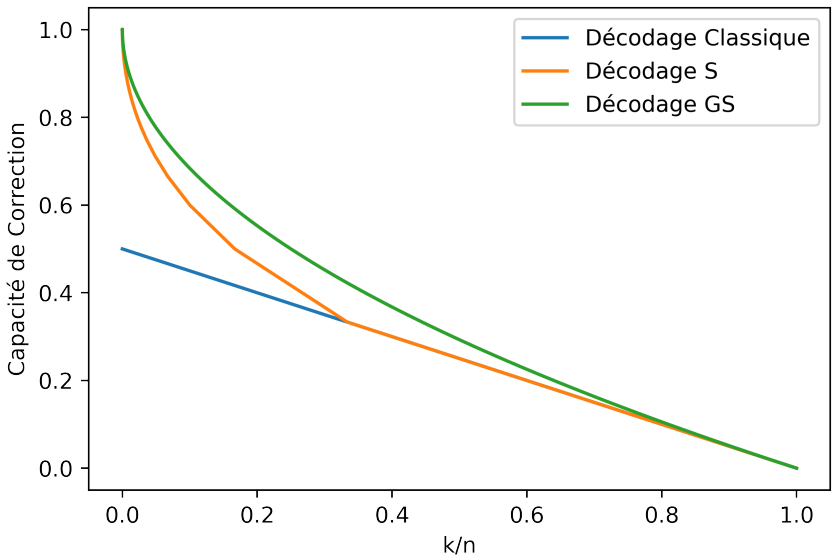


FIGURE 2.4 – Efficacité des décodeurs de codes Reed-Solomon en fonction du rapport $\frac{k}{n}$

2.4 Codes Concaténés

Les codes concaténés sont un concept puissant dans le domaine des codes de correction d'erreurs, utilisés pour assurer une transmission fiable d'informations sur des canaux de communication bruyants. Les codes concaténés sont une manière d'améliorer les capacités de correction d'erreurs des codes en combinant plusieurs couches ou étapes d'encodage et de décodage.

L'idée de base derrière les codes concaténés est d'utiliser une combinaison de deux codes ou plus en séquence. Chaque code dans la séquence est appelé

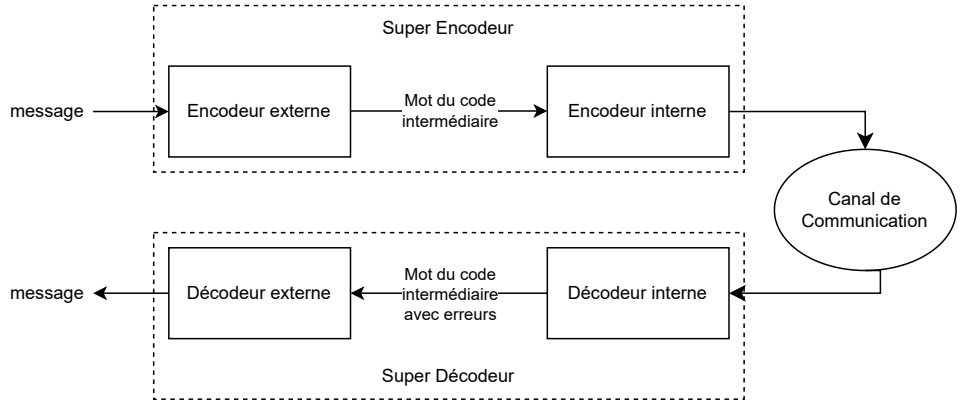


FIGURE 2.5 – Structure des codes concaténés

un "code composant". La sortie d'un code devient l'entrée du code suivant. Cette approche en couches permet une correction d'erreurs plus efficace, surtout en présence de grandes quantités d'erreurs.

Deux codes correcteurs d'erreurs ou plus, généralement avec des caractéristiques différentes, sont sélectionnés. Un code est utilisé comme code externe, et les autres sont utilisés comme codes internes. Les codes internes ont généralement de meilleures capacités de correction d'erreurs mais peuvent être moins efficaces en termes de ratio de code.

Encodage des codes concaténés Le message à transmettre est d'abord encodé en utilisant le code externe (le premier composant). Cela produit des mots de code avec une redondance qui aide à corriger les erreurs introduites pendant la transmission. Ensuite, les mots de code générés par le code externe sont encore encodés en utilisant le code interne. Cela ajoute une autre couche de redondance, fournissant une correction d'erreurs améliorée. Les codes internes traitent les mots de code externes comme des symboles, créant ainsi un nouveau code. Dans [For65], G. D. Forney décrit cette structure comme un super-code.

Les mots de code concaténés sont transmis sur le canal de communication, qui peut introduire des erreurs en raison du bruit ou d'autres facteurs.

Décodage des codes concaténés À l'extrémité de réception, le processus est inversé. Les codes internes sont d'abord décodés pour récupérer les mots de code intermédiaires. Ensuite, le code externe est utilisé pour décoder ces mots de code intermédiaires et récupérer le message d'origine. De la même manière, cette opération est appelée un super-décodeur.

L'avantage clé des codes concaténés est qu'ils peuvent fournir des performances de correction d'erreurs nettement meilleures que l'utilisation d'un seul code. Le code interne peut corriger des erreurs que le code externe ne

peut pas, et le code externe peut corriger davantage d'erreurs que le code interne ne peut pas détecter. Cette approche en couches exploite les points forts de différents codes et crée un schéma de codage global plus robuste.

Aspect Mathématique Le code externe se compose d'un code $[n_o, k_o, d_o]$ sur $\mathbb{F}_q$ avec $q = 2^{k_i}$ où k_i est un paramètre du code interne $[n_i, k_i, d_i]$ sur $\mathbb{F}_2$. Une bijection entre $\mathbb{F}_q$ et les mots de message du code interne permet de construire la transformation :

$$\mathbb{F}_q^{k_0} \xrightarrow{\text{code externe}} \mathbb{F}_q^{n_o} \xrightarrow{\text{code interne}} \mathbb{F}_2^{n_o \cdot n_i} \tag{2.36}$$

Le code concaténé est alors un code avec des paramètres $[n_o \cdot n_i, k_o \cdot k_i, D \geq d_o \cdot d_i]$ sur $\mathbb{F}_2$.

Les codes concaténés sont souvent utilisés dans des situations où une fiabilité élevée est requise, comme les communications spatiales lointaines, les liaisons par satellite et les systèmes de stockage de données. Les codes concaténés populaires utilisent des turbo codes ou des codes convolutifs comme codes internes et emploient généralement des codes de RS ou des codes BCH comme codes externes.

En résumé, les codes concaténés sont une technique sophistiquée dans la théorie des codes correcteurs d'erreurs qui implique la combinaison de plusieurs couches de codage pour obtenir des capacités de correction d'erreurs supérieures, les rendant adaptés à des environnements de communication difficiles. Ils ont également des applications en cryptographie basée sur des codes, dans le but de décoder autant d'erreurs que possible dans un temps de calcul réduit.

CHAPITRE 3

CRYPTOGRAPHIE BASÉE SUR LES CODES CORRECTEURS D'ERREURS

Contents

3.1 Problèmes difficiles pour des codes correcteurs

La sécurité de Cryptographie basée sur les codes (CBC) repose sur des problèmes difficiles. L'objectif de ce problème est de corriger une erreur de faible poids de Hamming à partir d'un mot de code erroné et de la matrice génératrice du code. Définissons la distribution suivante :

Definition 3.1: (Distribution de décodage)
Pour des entiers positifs $[n, k, t]$, soit $G \xleftarrow{\$} \mathbb{F}_2^{k \times n}$, une matrice génératrice tirée de manière aléatoire. Soit $m \xleftarrow{\$} \mathbb{F}_2^k$, un message tiré de manière aléatoire. Soit $e \xleftarrow{\$} \mathbb{F}_2^n$, t.q. $\mathrm{wt}(e) = t$, une erreur tirée de manière aléatoire de poids de Hamming t. Calculons $c = mG + e$, qui est le mot de code erroné. La distribution renvoie (G, c) et les paramètres $[n, k, t]$.

3.1.1 Problème de décodage

Definition 3.2: (Problème de décodage computationnel)
À partir d'une instance (G, c) de la distribution de décodage avec les paramètres $[n, k, t]$, le problème de décodage computationnel consiste à trouver (m', e') tels que $c = m'G + e'$ et $\mathtt{HW}e' = t$.

Ce problème peut être facile à résoudre dès que la matrice G a une structure connue. Mais pour une matrice aléatoire, et pour un bon choix de paramètres $[n, k, t]$, le problème est difficile. Ce problème a été montré Non-polynomial (NP)-complet en 1978 par Berlekamp, McEliece et Tilborg [BMVT78]. Pour les utilisations cryptographiques, nous avons également besoin d'une version décisionnelle du problème.

Definition 3.3: (Problème de décodage décisionnel)
Soit $b \xleftarrow{\$} \{0, 1\}$ tiré de manière aléatoire. Soit (G, c) provenant d'une distribution de décodage avec les paramètres $[n, k, t]$ si $b = 0$ ou (G, c) provenant d'une distribution uniforme sur $\mathbb{F}_2^{k \times n} \times \mathbb{F}_2^n$ si $b = 1$. Le problème de décodage décisionnel consiste à décider, avec un avantage non négligeable, la valeur de b, c'est-à-dire si (G, c) provient d'une distribution de décodage ou d'une distribution uniforme.

Ces deux problèmes sont équivalents et ont été prouvés NP-complets [BMVT78]. La sécurité de la cryptographie basée sur les codes repose sur la difficulté de résoudre le problème de décodage décisionnel ou ses variantes, telles que le problème de décodage par syndromes.

3.1.2 Décodage de Syndrome SD

Definition 3.4: (Distribution Problème du Décodage de Syndrome (SD))
Pour des entiers positifs $[n, k, t]$, soit $H \xleftarrow{\$} \mathbb{F}_2^{(n-k)\times n}$, une matrice de contrôle de parité tirée de manière aléatoire. Soit $e \xleftarrow{\$} \mathbb{F}_2^n$, t.q. $\mathrm{wt}(e) = t$, une erreur tirée de manière aléatoire de poids de Hamming t. Calculons $s = He^T$, qui est le syndrome de l'erreur e. La distribution renvoie (H, s) et les paramètres $[n, k, t]$.

À partir de cette distribution, nous pouvons décrire un problème qui vise à récupérer un vecteur d'erreur de faible poids à partir de la connaissance de son syndrome.

Definition 3.5: (Problème computationnel SD)
À partir d'une instance (H, s) de la distribution SD avec les paramètres $[n, k, t]$, le problème computationnel SD consiste à trouver un vecteur d'erreur e' tel que $s = He'^T$ et $\mathtt{HW}e' = t$.

Definition 3.6: (Problème décisionnel SD)
Soit $b \xleftarrow{\$} \{0, 1\}$ tiré de manière aléatoire. Soit (H, s) provenant d'une distribution SD avec les paramètres $[n, k, t]$ si $b = 0$ ou (H, s) provenant d'une distribution uniforme sur $\mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^n$ si $b = 1$. Le problème décisionnel SD consiste à décider, avec un avantage non négligeable, la valeur de b, c'est-à-dire si (H, s) provient d'une distribution SD ou d'une distribution uniforme.

3.1.2.1 Équivalence des problèmes

Il est connu que les problèmes de décodage et de décodage par syndromes sont équivalents. Nous présentons ici des preuves d'équivalence. On rappelle que pour un code dont on connaît une matrice génératrice, on peut trouver une matrice de contrôle de parité et inversement (voir équation 2.6).

Démonstration. SD $\rightarrow$ Décodage Supposons que nous puissions résoudre un problème de SD avec des paramètres $[n, n-k, t]$. Considérons une instance de décodage (G, c) avec des paramètres $[n, k, t]$. Si G n'est pas sous forme systématique (voir équation 2.4), nous devons transformer G et permuter les coordonnées de c en conséquence pour obtenir une forme systématique.

Par l'équation 2.6, nous pouvons déterminer une matrice de contrôle de parité H pour le code généré par G. Calculons le syndrome $s = Hc^T = H(mG + e)^T = He^T$.

Ensuite, en appliquant la résolution du problème SD sur l'instance (H, s), nous obtenons un vecteur d'erreur e' tel que $s = He'^T$ et $\mathtt{HW}e' = t$.

Nous définissons alors m' comme les k premières coordonnées de $c + e'$. Nous avons ainsi une solution (m', e') pour l'instance (G, c) du problème de décodage, ce qui montre que la résolution du problème SD permet de résoudre le problème de décodage.

□

Démonstration. Décodage → SD Supposons que nous puissions résoudre un problème de décodage avec des paramètres $[n, k, t]$. Considérons une instance (H, s) du problème SD avec des paramètres $[n, n-k, t]$. Si H n'est pas sous forme systématique (voir équation 2.4), nous devons transformer H pour obtenir une forme systématique.

Par l'équation 2.6, nous pouvons déterminer une matrice génératrice G pour le code généré par H. La première étape consiste à trouver un vecteur v ayant le même syndrome s que l'erreur e que nous cherchons. Étant donné que H est sous forme systématique, les $n-k$ coordonnées de s concaténées avec k zéros donnent un tel vecteur v.

Ensuite, en appliquant la résolution du problème de décodage sur l'instance (G, v), nous obtenons un couple (m', e') tel que $v = m'G + e'$ et $\mathtt{HW}e' = t$. Étant donné que v partage le syndrome s, nous avons $s = Hv^T = H(m'G + e')^T = He'^T$ et $\mathtt{HW}e' = t$, ce qui montre que nous avons résolu le problème SD.

Ainsi, la résolution du problème de décodage permet de résoudre le problème de décodage par syndrome.

□

Nombre de solutions

On note $\mathtt{CSD}\,[n, k, t]$ le nombre de solutions du problème de paramètres $[n, k, t]$. Selon la 3.4, il existe au moins une solution au problème, mais il existe des paramètres pour lesquels plusieurs solutions peuvent être trouvées. Le nombre de solutions est directement dépendant du choix des paramètres, notamment du rapport entre le poids de l'erreur et la longueur du code.

On peut évaluer la complexité de résoudre une instance d'un de ces problèmes en se basant sur un estimateur, notamment [EB22] développé par Esser et Bellini. Ce genre d'estimateur prend en compte les meilleurs décodeurs de l'état de l'art que je présente dans la section 3.2.

3.2 Résolution des problèmes basés sur les codes

Étant donné l'équivalence entre les deux problèmes de décodage D et de décodage de syndrome SD, nous ne regardons dans cette section que les stratégies permettant de résoudre le problème de décodage de syndrome. Soit (H, s) un couple provenant de la distribution du problème de décodage

de syndrome (voir 3.4) avec les paramètres $[n, k, t]$. La matrice H peut se voir comme la concaténation de ses colonnes :

$$H = \begin{bmatrix} h_1 & h_2 & h_3 & \cdots & h_n \end{bmatrix} \tag{3.1}$$

Trouver une solution au problème de décodage de syndrome, c'est donc trouver t colonnes, $(h_{j_1}, h_{j_2}, \cdots, h_{j_t})$ de H telles que :

$$\sum_{l=1}^{t} h_{j_l} = h_{j_1} + h_{j_2} + \cdots + h_{j_t} = s \tag{3.2}$$

Historiquement, la manière de résoudre le problème du décodage est le décodage par ensemble d'information (ou *Information Set Decoding (ISD)*), introduit par Prange [Pra62] en 1962. Nous verrons qu'il existe une seconde méthode permettant le décodage, introduite par Al Jabri [Jab01] en 2001, le décodage statistique (ou *Statistical Decoding*). Il est important de comprendre comment résoudre ces problèmes afin d'évaluer au mieux les paramètres d'un cryptosystème basé sur les codes correcteurs.

3.2.1 Décodage par ensemble d'information ISD

Historiquement, la manière la plus répandue de résoudre le problème de décodage est l'ISD, introduit par Prange [Pra62] en 1962. Cet algorithme a été maintes fois amélioré au cours des années, notamment par Jacques Stern [Ste89] en 1989, Dumer [Dum91] en 1991, May et al. [MMT11] en 2011, Becker et al. [BJMM12] en 2012 ou May et al. [MO15] en 2015. De plus, l'algorithme des anniversaires généralisés (ou *Generalized Birthday Algorithm (GBA)*), introduit par Wagner [Wag02] en 2002, peut être utilisé pour réduire la complexité générale de l'ISD, profitant d'un avantage de recherche quadratique garanti par le paradoxe des anniversaires. Ces améliorations sont des modifications de l'algorithme de base de Prange [Pra62] de 1962, c'est pourquoi on parle de décodage de la famille des ISD. Je ne vais pas entrer dans les détails des différentes modifications qui ont été faites au cours du temps sur l'ISD, mais faire une présentation globale du schéma, pour donner les grandes idées derrière l'ISD.

Soit (H, s) une instance du problème du décodage de syndrome avec les paramètres $[n, k, t]$, tel que décrit dans la Pour rappel, il existe un vecteur e de petit poids t qui vérifie $s = He^T$. L'ISD repose sur un constat simple : si l'on prend P une permutation quelconque, alors $s = HP(eP)^T$, où HP est la permutation P appliquée sur les colonnes de H. De plus, si le vecteur eP a son support exclusivement contenu dans ses k premières coordonnées, on peut alors réécrire :

$$s = HP(eP)^T = (H_1 \mid H_2) \times (e_1 \mid 0 \cdots 0)^T = H_1 e_1^T \tag{3.3}$$

Si la matrice H_1 est inversible, on a donc que $e_1 = (H_1^{-1}s)^T$, et donc que $e = (e_1 \mid 0 \cdots 0)\, P^{-1}$. L'algorithme de l'ISD est donc basé sur un pari : le pari que pour une permutation aléatoire donnée P, le support de e soit entièrement contenu dans les k premières coordonnées du vecteur. On peut calculer p, la probabilité que cet événement arrive :

$$p = \frac{\binom{k}{t}}{\binom{n}{t}} \tag{3.4}$$

Le coût de l'ISD est alors le coût d'une inversion matricielle, qu'il faut répéter jusqu'à ce que l'on trouve une permutation qui satisfait la propriété ci-dessus. L'inversion d'une matrice de taille $k \times k$ peut être réalisée naïvement pour un coût de $O(k^3)$ opérations. Cependant, on connaît d'autres algorithmes permettant de réduire ce coût : (i) l'algorithme de Strassen [S$^+$69], permettant de réaliser cette inversion en $O(k^{2.8})$ opérations, et (ii) l'algorithme de Coppersmith-Winograd [CW87], réduisant le coût à $O(k^{2.3})$. Ce dernier est malheureusement un algorithme galactique, c'est-à-dire que sa complexité théorique est asymptotiquement meilleure que celle de l'algorithme de Strassen, mais qu'en pratique, pour voir un gain, il faut que la taille de la matrice soit très grande. C'est Nicolas Aragon qui m'a parlé pour la première fois de ce type d'algorithme, et il en a fait selon moi la meilleure description possible, que je reprends ici : "Si tu as assez de place sur ton ordinateur pour stocker la matrice que tu veux inverser, alors elle est trop petite pour que l'algorithme galactique soit le plus performant."

Le coût final de l'ISD de Prange, noté τ_{ISD}, peut donc être estimé par :

$$\tau_{ISD} = \frac{k^{2.8}}{p} \tag{3.5}$$

3.2.2 Décodage statistique

Le décodage statistique a été introduit par Al Jabri [Jab01] en 2001. Il a ensuite été amélioré par Overbeck [Ove06] et présenté dans une version itérative par Fossorier et al. [FKI06] en 2006. La complexité asymptotique du schéma a été formalisée en 2017 par Alezard et al. [DAT17], raffinant les travaux de 2006. Le décodage statistique a finalement été à nouveau amélioré en 2022 par Carrier et al. [CDAMHT22], permettant de faire mieux que les ISD pour certains paramètres de codes. Dans ce manuscrit, je ne parlerai pas de la dernière amélioration de [CDAMHT22], mais donnerai simplement une idée générale du schéma basé sur les travaux de 2017 [DAT17].

On peut définir le produit scalaire de deux vecteurs x et y dans $\mathbb{F}_2^n$, $\langle x, y \rangle = \sum_i x_i y_i$ en caractéristique 2. Nous avons les propriétés suivantes :

$\forall c \in \mathcal{G}, \forall h \in \mathcal{H}, \langle c, h \rangle = 0$. Ensuite, le calcul de $\langle y, h \rangle = \langle c + e, h \rangle = \langle c, h \rangle + \langle e, h \rangle = \langle e, h \rangle$.

Le calcul d'un grand nombre d'équations de contrôle de parité est l'idée derrière le décodage statistique. La forme de ces équations respecte $\langle y, h \rangle = \langle e, h \rangle$ où h appartient à $\mathcal{H}_\omega$, l'ensemble des mots de code de $\mathcal{H}$ avec un poids de Hamming de ω. Pour obtenir des informations sur e, la stratégie est de parier sur la valeur du i*ème* bit de e, à savoir e_i. En fait, la probabilité de $\langle e, h \rangle$ d'être égale à un dépend de la valeur de e_i, comme nous le prouverons.

Pour une question de clarté, considérons ici l'ensemble de tous les mots de poids de Hamming ω plutôt que les mots de code de $\mathcal{H}$.

Calculons les probabilités suivantes :

$$\begin{aligned} q_0(e, \omega, i) &= \mathbb{P}\left(h \leftarrow \mathcal{S}_{\omega,i} \right) \langle e, h \rangle = 1 \ : \ e_i = 0 \\ q_1(e, \omega, i) &= \mathbb{P}\left(h \leftarrow \mathcal{S}_{\omega,i} \right) \langle e, h \rangle = 1 \ : \ e_i = 1 \end{aligned} \tag{3.6}$$

On peut vérifier que :

$$\begin{aligned} q_0(e, \omega, i) &= \sum_{j \text{ even}}^{\omega} \frac{\binom{t}{j}\binom{n-t-1}{\omega-j-1}}{\binom{n-1}{\omega-1}} \\ q_0(e, \omega, i) &= \sum_{j \text{ odd}}^{\omega} \frac{\binom{t-1}{j}\binom{n-t}{\omega-j-1}}{\binom{n-1}{\omega-1}} \end{aligned} \tag{3.7}$$

Ces probabilités sont indépendantes de l'erreur et également de la position i, de telle sorte que l'on peut les réécrire q_0 and q_1 :

$$\begin{aligned} q_0 &= \frac{1}{2} + \epsilon_0 \\ q_1 &= \frac{1}{2} + \epsilon_1 \end{aligned} \tag{3.8}$$

Supposons maintenant que q_0 et q_1 soient statistiquement distinguables, c'est-à-dire $\epsilon_0 \neq \epsilon_1$. Alors, nous sommes capables de les classer en formulant l'hypothèse $\mathscr{H}_0$ où $e_i = 0$ et l'hypothèse $\mathscr{H}_1$ lorsque $e_i = 1$. Nous devons faire attention au signe de $\epsilon_1 - \epsilon_0$ qui aura une importance pour la suite des calculs : $\delta := \text{sign}(\epsilon_1 - \epsilon_0)$.

Pour différencier les deux hypothèses $\mathscr{H}_0$ et $\mathscr{H}_1$, la borne de Chernoff est utilisée :

Soient $Y_1, \cdots, Y_m$ m variables indépendantes suivant une loi de Bernoulli $\mathcal{B}(p)$ et $Z_m = \sum_{k=0}^{m} Y_k$. Alors,

$$\mathbb{P}\left(|Z_m - mp| \geq m\delta\right) \leq 2e^{-2m\delta^2} \tag{3.9}$$

On peut calculer la variable aléatoire V_m pour m tirages uniformes et indépendants de vecteurs dans $\mathcal{H}_{\omega,i}$.

$$V_m = \sum_{i=0}^{m} \delta \cdot \langle y, h_k \rangle \in \mathbb{Z} \tag{3.10}$$

Cependant, sous l'hypothèse $\mathscr{H}_l$, les $\langle y, h_k \rangle$ suivent la loi de Bernoulli $\mathcal{B}(\frac{1}{2} + \epsilon_l)$. Ainsi, l'espérance de V_m sous $\mathscr{H}_l$ est donnée par :

$$E_l = m \cdot \delta \cdot \left(\frac{1}{2} + \epsilon_l \right) \tag{3.11}$$

Étant donné le terme δ, nous avons $E_0 < E_1$.

En appliquant la borne de Chernoff de l'équation 3.9, sous l'hypothèse $\mathscr{H}_l$, nous avons :

$$\mathbb{P}\left(\left| V_m - m \cdot \delta \cdot \left(\frac{1}{2} + \epsilon_l \right) \right| \geq m \times \frac{|\epsilon_1 - \epsilon_0|}{2} \right) \leq 2 \cdot 2^{-m \cdot \frac{(\epsilon_1 - \epsilon_0)^2}{2 \ln(2)}} \tag{3.12}$$

Ensuite, nous sélectionnons $\mathscr{H}_0$ si

$$V_m < \frac{m \cdot \delta}{2} \cdot (1 + \epsilon_0 + \epsilon_1) \tag{3.13}$$

et $\mathscr{H}_1$ sinon.

Algorithm 1 Décodage Statistique

Entrée : $y = xG + e \in \mathbb{F}_2,^n$, $\omega \in \mathbb{N}$
Sortie : $e \in \mathbb{F}_{2,t}^n$

for i dans la plage $0 \to n$ **do**
  Calculer $\mathcal{S}_i$
  $V_i \leftarrow 0$
  for tous les $h \in \mathcal{S}_i$ **do**
    $V_i \leftarrow V_i + \delta \cdot \langle y, h \rangle$
  if $V_i < \frac{m \cdot \delta}{2} \cdot (1 + \epsilon_0 + \epsilon_1)$ **then**
    $e_i \leftarrow 0$
  else
    $e_i \leftarrow 1$
return e

3.3 ClassicMcEliece

3.3.1 McEliece PKE

Le cryptosystème McEliece [McE78] est un schéma de chiffrement à clé publique basé sur la difficulté de décoder le problème Problème du Décodage

(D) pour des codes linéaires aléatoires. Il a été introduit par Robert J. McEliece en 1978 et est considéré comme le schéma PQC le plus ancien et le plus étudié. L'idée principale derrière le chiffrement McEliece PKE est d'utiliser la difficulté de décoder des codes linéaires aléatoires pour protéger la clé de chiffrement.

Voici un aperçu simplifié de la manière dont fonctionne le schéma de chiffrement McEliece :

Génération de clé :

- Générer un code binaire aléatoire de Goppa $[n, k, d]$ avec une matrice génératrice G, pour lequel nous connaissons un algorithme de décodage efficace Φ.
- Tirer une matrice inversible aléatoire S.
- Tirer une matrice de permutation aléatoire P.
- Calculer la matrice génératrice G'

$$G' = S \cdot G \cdot P \tag{3.14}$$

- La clé secrète est composée de (G, S, P) et la clé publique est G'.

Chiffrement :

- Le message m est vu comme un vecteur binaire de longueur k.
- Tirer une erreur aléatoire e de poids de Hamming t, avec t plus petit que la capacité de correction d'erreur du code de Goppa généré par G.
- Générer le texte chiffré : $c = m \cdot G' + e$.

Déchiffrement :

- Calculer :

$$m' = \Phi(c \cdot P^{-1}) \cdot S^{-1} \tag{3.15}$$

L'idée derrière le cryptosystème McEliece est de cacher la structure de Goppa, pour laquelle nous connaissons un algorithme de décodage efficace Φ, en mélangeant aléatoirement la matrice génératrice. Une fois cela fait, nous obtenons une nouvelle matrice génératriceG'. Toute la preuve de sécurité repose sur le fait que la matrice G' est indiscernable d'une matrice génératrice tirée de manière aléatoire. Cette propriété est satisfaite, sous l'hypothèse de la difficulté du problème D et un bon choix de paramètres, il est impossible de récupérer m à partir de c.

Pendant le déchiffrement, la connaissance des éléments de permutation P et S permet de décoder et de récupérer $m' = m$. Cela peut être vu en développant l'Équation 3.15 :

$$
\begin{aligned}
m' &= \Phi(c \cdot P^{-1}) \cdot S^{-1} && \text{(3.16a)}\\
&= \Phi((m \cdot G' + e) \cdot P^{-1}) \cdot S^{-1} && \text{(3.16b)}\\
&= \Phi(m \cdot G' \cdot P^{-1} + e \cdot P^{-1}) \cdot S^{-1} && \text{(3.16c)}\\
&= \Phi(m \cdot S \cdot G \cdot P \cdot P^{-1} + e \cdot P^{-1}) \cdot S^{-1} && \text{(3.16d)}\\
&= m \cdot S \cdot S^{-1} && \text{(3.16e)}\\
m' &= m && \text{(3.16f)}
\end{aligned}
$$

L' équation 3.16 donne simplement la preuve que nous voulons. Le seul point que nous devons clarifier est la transition entre l' équation 3.16d et l' équation 3.16e. Cela est fait par la propriété du décodeur Φ. À gauche, nous avons $(m \cdot S \cdot G)$ qui est un mot de code. À droite, nous avons $e \cdot P^{-1}$; puisque P est une permutation, P^{-1} en est également une. Les permutations préservent le poids de Hamming ; ainsi, $e \cdot P^{-1}$ a un poids de Hamming plus petit que la capacité de correction d'erreur de Φ. Il découle de la théorie des codes correcteurs d'erreurs que le message décodé est $m \cdot S$.

3.3.2 Niederreiter PKE

Le schéma de Niederreiter [Nie86] vise à faire la même chose que le chiffrement de McEliece mais manipule les éléments dans le domaine des syndromes, avec la matrice de parité à la place de la matrice génératrice. On suppose donc que l'on connaît un décodeur de syndrome Ψ efficace pour les codes de Goppa. Ce décodeur de syndrome peut être facilement calculé à partir du résultat de l'équivalence des problèmes, tel que décrit dans la section 3.1.2.1.

3.3.3 Description du schéma

ClassicMcEliece est le protocole d'échange de clé obtenu en appliquant la transformée de Fujisaki-Okamoto sur le schéma de Niederreiter. La sécurité de ce schéma d'échange de clé bénéficie donc de quasiment 50 ans d'analyse de sécurité, depuis la publication du schéma de McEliece en 1978. La sécurité de ce schéma repose sur le problème de distinguer un code de Goppa permuté d'un code aléatoire. À ce jour, aucun distingueur ne permet de résoudre ce problème pour les paramètres d'utilisation de ClassicMcEliece.

3.4 BIKE

BIKE [ABB+17] est un schéma d'échange de clé (KEM) post-quantique basé sur les codes correcteurs d'erreurs. La version actuelle de BIKE est née de la fusion de trois schémas, Ouroboros [MAB+17], BIG QUAKE [BBB+17]

et BIKE, lors du premier tour du NIST en 2017. L'idée derrière ce schéma est d'exploiter la structure de l'erreur transmise comme message chiffré en tant que message. Il s'agit du schéma de Niederreiter instancié avec des codes Quasi-Cyclic Moderate Density Parity Check (QC-MDPC). Le message est alors un vecteur d'erreur $e = (e_0 \mid e_1)$ de petits poids de Hamming et le chiffré est son syndrome s. La sécurité du schéma se réduit à des variantes quasi-cycliques de problèmes difficiles (voir sous-section 3.1.2) issus de la théorie des codes.

De manière plus formelle, le schéma est décrit par trois algorithmes : la génération de clés, l'encapsulation et la décapsulation décrits dans les Algorithmes 2, 3 et 4. On définit les espaces suivants : $\mathcal{R} = \mathbb{F}_2[X]/(X^r - 1)$, un anneau de polynômes ; $\mathcal{R}^2_{\frac{\omega}{2}} = \left\{(h_0, h_1) \in \mathcal{R}^2 \mid \mathtt{HW}(h_0) = \mathtt{HW}(h_1) = \frac{\omega}{2}\right\}$, l'espace des clés secrètes, où $\mathtt{HW}(\cdot)$ définit toujours le poids de Hamming ; $\mathcal{M} = \mathcal{K} = \{0,1\}^l$, les espaces de messages et de clés partagées ; et $\mathcal{E}_t = \left\{(e_0, e_1) \in \mathcal{R}^2 \mid \mathtt{HW}(e_0) + \mathtt{HW}(e_1) = t\right\}$, l'espace des erreurs. Ainsi que trois fonctions de hachage $\mathbf{H} : \mathcal{M} \to \mathcal{E}_t$, $\mathbf{K} : \mathcal{M} \times \mathcal{R} \times \mathcal{M} \to \mathcal{K}$ et $\mathbf{L} : \mathcal{R}^2 \to \mathcal{M}$. On suppose aussi que l'on a un décodeur $\mathtt{BIKE\,Decode}$ qui vérifie :

$$\begin{aligned} \forall (e_0, e_1) \in \mathcal{E}_t, \forall (h_0, h_1) \in \mathcal{R}^2_{\frac{\omega}{2}} \\ (e_0, e_1) \leftarrow \mathtt{BIKE\,Decode}(e_0 h_0 + e_1 h_1, h_0, h_1) \end{aligned} \tag{3.17}$$

Algorithm 2 BIKE Génération de clefs de BIKE

Require: params
Ensure: $(\mathsf{sk}, \mathsf{pk}) = ((h_0, h_1, \sigma), h)$
1: $(h_0, h_1) \xleftarrow{\$} \mathcal{R}^2_{\frac{\omega}{2}}$
2: $h = h_1 h_0^{-1}$
3: $\sigma \xleftarrow{\$} \mathcal{M}$
4: **return** $((h_0, h_1, \sigma), h)$

Algorithm 3 Encapsulation de BIKE

Require: $\mathsf{pk} = h$
Ensure: K la clef partagée et c le chiffré
1: $m \xleftarrow{\$} \mathcal{M}$
2: $(e_0, e_1) = \mathbf{H}(m)$
3: $c_0 = e_0 + e_1 h$
4: $c_1 = m \oplus \mathbf{L}(e_0, e_1)$
5: $K = \mathbf{K}(m, c)$
6: **return** $c = (c_0, c_1)$

Algorithm 4 Décapsulation de BIKE

Require: $\mathsf{sk} = (h_0, h_1, \sigma)$ et c le message chiffré
Ensure: K la clef partagée
1: $e' = \texttt{BIKE Decode}(c_0 h_0, h_0, h_1)$
2: $m' = c_1 \oplus \mathbf{L}(e')$
3: **if** $e' = \mathbf{H}(m')$ **then**
4: $\quad K = \mathbf{K}(m', c)$
5: **else**
6: $\quad K = \mathbf{K}(\sigma, c)$
7: **return** K

3.4.1 Décodeur de BIKE

Ce schéma présente des tailles de clés vraiment très compétitives dans le concours du NIST par rapport aux autres schémas basés sur les codes correcteurs d'erreurs. Le décodeur utilisé dans l'instanciation de BIKE est un décodeur itératif dit algorithme *Black-Gray-Flip (BGF)* [DGK20], dont la structure est proche de celle des décodeurs utilisés pour les codes Low-Density Parity-Check (LDPC). Dans une utilisation classique du décodeur, celui-ci itère jusqu'à ce que l'erreur soit complètement retrouvée et corrigée. Dans un contexte où les attaques par canaux auxiliaires, et notamment les attaques par temps de calcul, sont un sujet pour la sécurité d'un cryptosystème, les auteurs de BIKE ont décidé de borner le nombre d'itérations possibles du décodeur. De plus, afin de garantir que le décodage se fasse en temps constant, le nombre maximal d'itérations est réalisé à chaque fois, et ce même si l'erreur est corrigée en un nombre d'itérations plus faible.

Clés faibles

Il se trouve que cette construction du décodeur présente un problème : certaines clés secrètes utilisées par BIKE ont la propriété de nécessiter, en moyenne, plus d'itérations que le maximum possible, afin de décoder l'erreur. Ces clés particulières sont appelées clés faibles (*weak keys*) et présentent une menace pour la sécurité de BIKE. Ces clés faibles sont mentionnées en 2019 par Drucker et al. [DGK19] et étudiées depuis 2020 par Sendrier et Vasseur [SV20, Vas22]. Leur conclusion montre que la proportion de clés faibles est suffisamment réduite pour ne pas impacter la sécurité du schéma. Toutefois, en 2023, deux nouvelles analyses de Nousouhi et al. [NSP+23] et de Wang et al. [WWW23] montrent l'existence de nouveaux types de clés faibles ayant un impact, cette fois-ci, sur la sécurité du schéma. Une protection contre ces clés passerait par leur identification et le tirage d'une nouvelle clé (*sanity check*) lors de la génération de clés de BIKE.

3.5 HQC

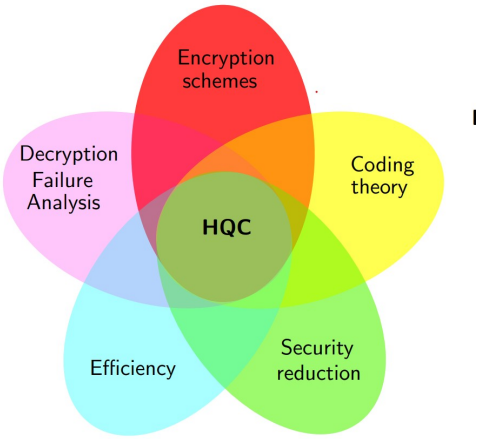


FIGURE 3.1 – HQC Logo

HQC [AMAB+17] est un KEM basé sur le code dont la sécurité repose sur la difficulté de résoudre une instance quasi-cyclique du problème de décodage du syndrome. Le schéma est construit en utilisant une variante de la transformation FO appelée Hofheinz, Hövelmanns and Kiltz (HHK) [HHK17]. Cette transformation vise à créer un KEM à partir d'un PKE.

3.5.1 Aperçu de HQC

La première étape pour décrire HQC-KEM est alors de décrire la version de HQC réalisant le chiffrement HQC-PKE.

3.5.1.1 HQC-PKE

Algorithm 5 Configuration de HQC

Require: λ le niveau de sécurité
Ensure: **param** $= (n, k, \delta, \omega, \omega_r, \omega_e)$
Ensure: G la matrice génératrice de $\mathcal{C}$, le code publiquement connu.
Ensure: $\mathcal{C}.\texttt{Decode}$ un algorithme de décodage efficace pour $\mathcal{C}$
Ensure: $\mathcal{G}, \mathcal{H}, \mathcal{K}$ trois fonctions de hachage cryptographiques.

La configuration de HQC permet de déterminer la valeur des paramètres en fonction du niveau de sécurité. Les auteurs de HQC décrivent trois ensembles de paramètres, pour trois niveaux de sécurité, I, II, III correspondant à la complexité de casser un AES avec une taille de clé de 128, 192 et 256 bits respectivement. Les paramètres sont choisis en fonction des meilleures

attaques connues contre la cryptographie basée sur les codes, des meilleurs algorithmes quantiques connus et pour permettre un compromis entre la taille des clés et les temps de calcul.

Niveau	λ	n	n_1	n_2	ω	ω_r	ω_e	μ
I	128	17.669	46	384	66	75	75	3
II	192	35.851	56	640	100	114	114	5
III	256	57.637	90	640	131	149	149	5

TABLE 3.1 – Paramètres de HQC tirés de [AMAB+17]

Presque tous les éléments de HQC se trouvent dans l'espace

$$\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$$

où n est un nombre premier, en effet, le plus petit nombre premier supérieur à $n_1 n_2$. Deux paramètres supplémentaires, n_1 et n_2, sont utilisés pour le code publiquement connu $\mathcal{C}$ de HQC. Les opérations de ce code sont effectuées sur $\mathbb{F}^{n_1 n_2}$. Pendant ces opérations, les $l = n - n_1 n_2$ bits inutiles sont tronqués.

Algorithm 6 Génération des clés de HQC

Require: paramètres : **param**
Ensure: $(\mathsf{sk}, \mathsf{pk})$
  $h \xleftarrow{\$} \mathcal{R}$
  $(x, y) \xleftarrow{\$} (\mathcal{R}_\omega)^2$
  $s = x + h \cdot y$ ▷ Calcul du syndrome
  $\mathsf{sk} := (x, y)$
  $\mathsf{pk} := (h, s)$
  return $(\mathsf{sk}, \mathsf{pk})$

Où $\mathcal{R}_\omega$ est défini comme le sous-ensemble de $\mathcal{R}$ avec des éléments de poids de Hamming ω.

$$\mathcal{R}_\omega = \{x \in \mathcal{R} | \mathtt{HW}x = \omega\} \tag{3.18}$$

La clé secrète (x, y) est protégée par la difficulté de résoudre le problème de décodage du syndrome étant donné une matrice de contrôle de parité H, générée en utilisant la quasi-cyclicité par le vecteur h et le syndrome s.

Pour le chiffrement, si une graine Θ est fournie en entrée, elle est utilisée pour toutes les générations aléatoires. Cette graine permet la reproductibilité du chiffrement : avec les mêmes entrées, elle garantit la même valeur de sortie.

Le texte chiffré est composé de deux parties, v contenant le mot de code erroné, et u contenant la relation entre l'erreur aléatoire et la clé publique. Les deux parties du texte chiffré sont nécessaires pour le déchiffrement.

Algorithm 7 Chiffrement HQC

Require: paramètres : **param**, pk, m un message
Require: optionnel : graine Θ ▷ permet la reproductibilité
Ensure: c un chiffré de m
$e \xleftarrow{\$} \mathcal{R}_{\omega_e}$
$(r_1, r_2) \xleftarrow{\$} (\mathcal{R}_{\omega_r})^2$
$u = r_1 + h \cdot r_2$
$v = mG + s \cdot r_2 + e$
return $c = (u, v)$

Algorithm 8 Déchiffrement de HQC

Require: paramètres : **param**, sk, c un texte chiffré
Ensure: m le message déchiffré à partir de c
return $m := \mathcal{C}.\mathtt{Decode}(v - u \cdot y)$

Le déchiffrement implique uniquement le décodeur du code $\mathcal{C}$. Il est clair que la correction de ce schéma dépend uniquement de la capacité de correction du code $\mathcal{C}$ choisi. En effet, pour HQC, le DFR est exactement un taux d'échec de décodage, ce sujet est abordé dans la section 3.5.2.

3.5.1.2 HQC-KEM

La configuration et la génération de clés (issues respectivement des Algorithmes 5 et 6) sont les mêmes pour HQC-KEM et PKE version of HQC (HQC-PKE).

Algorithm 9 HQC Encapsulation

Require: paramètres : **param**, pk
Ensure: K : La clef partagée, (c, d) le texte chiffré
$m \xleftarrow{\$} (\mathbb{F}_2)^k$
$\Theta := \mathcal{G}(m)$ ▷ génère une graine pour les tirages aléatoires suivants
$c := (u, v) = HQC.\mathsf{Encrypt}(\text{param}, \mathsf{pk}, m, \Theta)$
$K := \mathcal{K}(m, c)$
$d := \mathcal{H}(m)$
return (c, d)

3.5.1.3 Étape de re-chiffrement

La transformation de type Fujisaki-Okamoto utilisée dans HQC pour transformer HQC-PKE en HQC-KEM assure une sécurité sémantique. La sécurité sémantique garantit qu'un attaquant ne peut pas obtenir d'informations significatives sur le texte en clair même s'il forge des textes chiffrés

Algorithm 10 HQC Décapulation

Require: paramètres : **param**, sk, (c, d) un texte chiffré
Ensure: K : La clef partagée
  $m' := HQC.\mathsf{Decrypt}(\text{param}, \mathsf{sk}, c)$
  $\Theta' := \mathcal{G}(m')$
  $c' = (u', v') := HQC.\mathsf{Encrypt}(\text{param}, \mathsf{pk}, m', \Theta')$ ▷ re-chiffrer
  if $c \neq c'$ **ou** $d \neq \mathcal{H}(m')$ **then**
    $K :=\perp$ ▷ abandon
  else
    $K := \mathcal{K}(m, c)$
  return K

correspondant à une clé publique donnée. Cette sécurité, appelée INDistinguability under Chosen Ciphertext Attacks (IND-CCA), est garantie car le schéma initial de chiffrement est INDistinguability under Chosen Plaintext Attacks (IND-CPA). Cela est réalisé en randomisant le texte chiffré pendant le processus de chiffrement.

Dans le contexte d'un KEM, l'objectif est d'encapsuler une clé de session en utilisant la clé publique du destinataire. Le destinataire peut ensuite déchiffrer la clé de session en utilisant sa clé secrète. La connaissance de cette clé secrète peut être récupérée en effectuant diverses attaques, comme des attaques par texte chiffré choisi, pour récupérer des informations sur la clé secrète.

Le re-chiffrement du message avant de forger la clé partagée permet de s'assurer que le texte chiffré entrant était honnête. Dans un tel cas, le processus continue jusqu'à ce que la même clé partagée soit obtenue. Sinon, il est impossible de décider si le message chiffré provient d'une erreur de décodage ou d'un texte chiffré forgé par un adversaire malveillant. Dans ce cas, l'algorithme de décapsulation renvoie une valeur aléatoire, notée $\perp$, ce qui signifie qu'aucune information sur la clé secrète n'est révélée en cas d'attaque par chiffrés choisis.

3.5.2 HQC DFR

Pendant le déchiffrement de HQC, le décodeur manipule la valeur suivante :

$$\begin{aligned} v - u \cdot y &= mG + s \cdot r_2 + e - u \cdot y \\ &= mG + (x + h \cdot y) \cdot r_2 + e - (r_1 + h \cdot r_2) \cdot y \\ &= mG + x \cdot r_2 + e - y \cdot r_1 \\ &= mG + e' \end{aligned} \tag{3.19}$$

D'après la théorie des CCE :

$$\begin{aligned} \mathcal{C}.\texttt{Decode}(mG + e') = m &\iff \texttt{HW}e' \leq t \\ &\iff \texttt{HW}x \cdot r_2 + e - y \cdot r_1 \leq t \end{aligned} \tag{3.20}$$

où t est la capacité de correction d'erreur du code. Une étude de la distribution du poids de Hamming de e' permet de sélectionner des paramètres adaptés pour HQC. Dans [AGZ20], Aragon et al. montrent que le poids de Hamming de l'erreur e' peut être approximé par une distribution gaussienne.

Proposition 3.7: (Distribution du Produit Vectoriel)
Soient x et r deux vecteurs indépendants de longueur n et de poids de Hamming respectifs ω et ω_r. Le poids de Hamming de $z = x \cdot r$ suit une distribution de Bernoulli sur chaque élément $z_k, k \in \{1, \cdots, n\}$:

$$\tilde{p} := \mathbb{P}(z_k = 1) = \frac{1}{\binom{n}{\omega}\binom{n}{\omega_r}} \sum_{\substack{l=1 \\ l \text{ est impair}}}^{\min(\omega,\omega_r)} \binom{n}{l}\binom{n-l}{\omega-l}\binom{n-\omega}{\omega_r-l} \tag{3.21}$$

Pour l'analyse de e', le poids de Hamming de deux vecteurs, à savoir $x \cdot r_2$ et $y \cdot r_1$, suit la Proposition 3.7. Les deux distributions peuvent être combinées pour obtenir la distribution de la somme de ces vecteurs.

Proposition 3.8: (Distribution de la Somme de Vecteurs 1)
Soient z_1 et z_2 deux vecteurs indépendants dont les poids de Hamming suivent des lois de Bernoulli données par la Proposition 3.7. Le poids de Hamming de la somme $t = z_1 + z_2$ suit également une loi de Bernoulli avec les paramètres suivants :

$$\begin{cases} \widehat{p} & := \mathbb{P}(t_k = 1) = 2\tilde{p}(1-\tilde{p}) \\ (1-\widehat{p}) & := \mathbb{P}(t_k = 0) = (1-\tilde{p})^2 + \tilde{p}^2 \end{cases} \tag{3.22}$$

La même combinaison de distributions peut être appliquée entre $(x \cdot r_2) + (y \cdot r_1)$ et e pour obtenir une distribution de Bernoulli sur chaque élément de e'

Proposition 3.9: (Distribution de la Somme de Vecteurs 2)
Soient e et t deux vecteurs indépendants de poids de Hamming ω_e et suivant la distribution donnée par la Proposition 3.7. Le poids de Hamming de $e' = t + e$ suit une distribution de Bernoulli avec les paramètres suivants :

$$\begin{cases} p^* & := \widehat{p}(1 - \frac{\omega_e}{n}) + (1-\tilde{p})\frac{\omega_e}{n} \\ (1-p^*) & := (1-\widehat{p})(1-\frac{\omega_e}{n}) + \widehat{p}\frac{\omega_e}{n} \end{cases} \tag{3.23}$$

Les preuves des Propositions 3.7, 3.8 et 3.9 sont disponibles dans [AGZ20] et dans [AMAB+17]. Considérer chaque élément de e' comme des éléments

indépendants permet d'appliquer la formule de la distribution gaussienne discrète, qui donne la probabilité pour que e' ait un poids de Hamming donné k.

$$\mathbb{P}\left(\mathtt{HW}e' = k\right) = \binom{n}{k}(p^*)^k \times (1-p^*)^{n-k} \tag{3.24}$$

Afin de garantir un faible DFR, requis pour la réduction de sécurité de transformations comme FO, les paramètres de HQC doivent être choisis de manière à ce que :

$$\sum_{k>t} \mathbb{P}\left(\mathtt{HW}e' = k\right) \leq 2^{-\lambda} \tag{3.25}$$

où λ est le niveau de sécurité sélectionné.

3.5.3 HQC-RMRS

HQC manipule un code $\mathcal{C}$ connu publiquement. Ce code peut être choisi à la discrétion de l'utilisateur ou du développeur. Si la DFR de ce code est inférieure à la valeur attendue ou imposée par le niveau de sécurité, le choix du code n'a aucune incidence sur la sécurité globale de HQC.

Dans la version précédente de HQC, les auteurs ont sélectionné $\mathcal{C}$ comme un code BCH avec répétition. La version de HQC basée sur ce code a été la cible de différentes attaques physiques, comme nous le verrons dans la sous-section 4.7.1. Par la suite, l'équipe de HQC a remplacé ces codes BCH par des codes concaténés. Ce changement a permis de gagner en performance, en taille de clés et de paramètres, tout en maintenant la DFR au niveau souhaité.

Depuis mai 2020, les auteurs de HQC présentent une instanciation du schéma avec des codes concaténés de Codes de Reed-Muller et Reed Solomon concaténés (RMRS). Cette version présente de meilleurs paramètres que la construction initiale basée sur les codes BCH. À partir d'octobre 2020, les codes BCH sont complètement abandonnés au profit des codes Codes de Reed-Muller et Reed Solomon concaténés (RMRS) concaténés plus performants. Le message manipulé par le code $\mathcal{C}$ est d'abord encodé par un code de RS. Ce mot du code intermédiaire est découpé en blocs, et chacun de ces blocs est ensuite encodé avec un code de RM. Ces différents encodages sont parfaitement indépendants. Le décodage de ces codes s'effectue de la même manière en sens inverse. Un aperçu de cette structure est donné avec la Figure 3.2.

Les fondements théoriques à la fois du RM et du RS sont abordés dans les Sections 2.2 et 2.3, ainsi que l'explication de la construction des codes concaténés dans la Section 2.4.

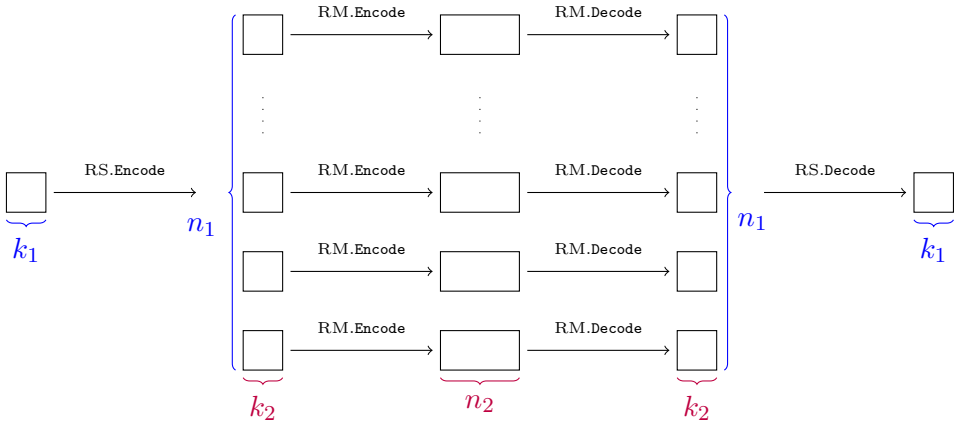


FIGURE 3.2 – Structure simplifiée des codes RMRS concatenés

DFR intermédiaire La faible probabilité d'échec dans le décodage (ou *DFR*) requis dans la réduction de sécurité de la transformée FO, implique que le code interne a aussi une probabilité très faible d'échecs. Ces probabilités sont données dans les spécifications de HQC et retranscrites dans le tableau 3.2 :

λ	DFR	RM DFR
HQC-128	2^{-128}	$2^{-10.96}$
HQC-192	2^{-192}	$2^{-14.39}$
HQC-256	2^{-256}	$2^{-11.48}$

TABLE 3.2 – Probabilités d'échecs du décodeur concaténé et du décodeur interne de HQC

Nous verrons par la suite, notamment dans les chapitre 6 et chapitre 7, que cette faible DFR nous permet de construire des attaques physiques sur le schéma.

Implémentation de HQC Dans le contexte des attaques par canaux auxiliaires, notre attention se tourne vers l'implémentation et l'algorithmie spécifique de ces codes dans le cadre de HQC. Dans le cadre du concours PQC du NIST, deux implémentations officielles de HQC sont accessibles sur le site officiel de HQC [AMAB+]. La version principale de l'implémentation est optimisée pour les architectures 64 bits, mais cette variante est inutilisable dans un contexte d'attaque physique, comme c'est le cas dans ce manuscrit, en raison du manque de prise en charge sur les architectures de microcontrôleurs. Ainsi, notre étude est centrée sur l'implémentation de

référence de HQC [AMAB+23], conçue pour être compatible avec les petites architectures. Cette implémentation de référence garantit une exécution en temps constant [AMAB+23] et reflète de manière précise l'implémentation optimisée réelle de HQC.

Je me concentre ici sur les principales opérations de correction d'erreur de code et les algorithmes impliqués dans HQC.

3.5.3.1 Encodage des codes Reed-Solomon

Soit $\mathbf{g}(x)$ le polynôme générateur d'un code RS et $\mathbf{u}(x)$ le polynôme associé à un message $\mathbf{m}$, c'est-à-dire que $(m_1, \cdots, m_k) = (u_0, \cdots, u_{k-1})$. Le mot de code $\mathbf{c}_{\mathtt{RS}} := \mathbf{c}(x)$ de RS associé au message $\mathbf{m}$ est alors donné par :

$$\mathbf{c}(x) = \mathbf{u}(x) \times x^{n-k} + \Big(\mathbf{u}(x) \times x^{n-k} \bmod \mathbf{g}(x)\Big) \tag{3.26}$$

Dans l'implémentation de référence de HQC [AMAB+23], cet encodage est réalisé par l'algorithme Algorithme 11.

Algorithm 11 Encodeur des codes de Reed-Solomon de l'implémentation de référence de HQC [AMAB+23]

Require: paramètres : k, n
Require: polynôme générateur $\mathbf{g} \in \mathbb{F}_q^{n-k}$
Require: un message $\mathbf{m} \in \mathbb{F}_q^k$
Ensure: $\mathbf{c} := \mathtt{RS.Encode}(\mathbf{m}) \in \mathbb{F}_q^n$
1: Initialiser $\mathbf{c}$ à 0^n
2: **for** i de 1 à k **do**
3: $\quad \Gamma = \mathbf{m}[k-i] \oplus \mathbf{c}[n-k]$
4: $\quad$ **for** j de 1 à $n-k$ **do**
5: $\quad\quad \mathbf{t}[j] = \mathtt{gf\_mul}\,(\Gamma, \mathbf{g}[j])$ ▷ `gf_mul` est la multiplication dans le corps de Galois
6: $\quad$ **for** l de 2 à $n-k-1$ **do**
7: $\quad\quad \mathbf{c}[l] = \mathbf{c}[l-1] \oplus \mathbf{t}[l]$
8: $\quad \mathbf{c}[1] = \mathbf{t}[1]$
9: $\mathbf{c}[n-k : n] = \mathbf{m}$
10: **return c**

3.5.3.2 Décodage des codes Reed-Solomon

Le décodeur des codes de RS utilisé dans HQC suit la théorie décrite dans [JH04]. La stratégie repose sur l'existence d'un polynôme d'interpolation unique pour le mot de code reçu. Ce polynôme permet le décodage jusqu'à $t = \lfloor \frac{n-k}{2} \rfloor$, la moitié de la distance minimale d'erreurs. Il est calculé en

fonction de la valeur du syndrome du mot reçu en entrée de l'algorithme de décodage.

La première opération est le calcul du syndrome (voir Algorithme 20), effectué avec la connaissance de la matrice de contrôle de parité $\mathbf{H} = (h_{i,j})_{\substack{1 \le i \le n-k \\ 1 \le j \le n}}$. Pour rappel, le décodeur de HQC, et donc la matrice de contrôle de parité, sont publiquement connus.

Algorithm 12 Calcul des syndromes par le décodeur RS de HQC de [AMAB+23]

Require: paramètres : k, n la dimension et la longueur du code
Require: matrice de contrôle de parité $\mathbf{H} \in \mathbb{F}_q^{(n-k,n)}$
Require: mot de code $\mathbf{c} \in \mathbb{F}_q^n$
Ensure: $\mathbf{s} := \mathbf{H}^T \cdot \mathbf{c}$ le syndrome de $\mathbf{c}$
1: Initialiser $\mathbf{s}$ à $\mathbf{c}[1]^{n-k}$
2: **for** i de 1 à $n-k$ **do**
3: **for** j de 2 à n **do**
4: $\mathbf{s}[i] = \mathbf{s}[i] \oplus \texttt{gf\_mul}\,(\mathbf{c}[j], \mathbf{H}[i, j-1])$ ▷ `gf_mul` : multiplication dans le corps de Galois
5: **return s**

3.5.3.3 Décodage des codes Reed-Muller

Quel que soit le niveau de sécurité choisi, HQC utilise le même code Reed-Muller RM(1, 7), qui est un code $[128, 8, 64]$ sur $\mathbb{F}_2$. Ajouter une multiplicité μ revient exactement à appliquer un code de répétition au vecteur. Chaque bit passe également à travers un code de répétition de paramètres $[\mu \cdot k, k, \_]$; bien que ce "code" ne soit pas décodé comme un code de répétition classique, nous le mentionnerons simplement comme une multiplicité le long des bits du vecteur. Ajouter une multiplicité μ (voir Figure 3.1) aux mots de code permet d'obtenir des codes avec les paramètres $[384, 8, 192]$ ou $[640, 8, 320]$.

Ces codes Reed-Muller d'ordre 1 sont considérés comme des codes de Hadamard (voir Section 2.2) et peuvent être décodés en utilisant la Fast Hadamard Transform (FHT). La procédure de décodage est toujours la même, composée de trois algorithmes principaux dans l'implémentation de référence HQC d'avril 2023 [AMAB+23]. Cette procédure comprend les étapes principales suivantes :

1. Suppression de la multiplicité des mots de code avec la fonction d'expansion et de somme (voir Algorithme 13).
2. Application de la Transformée de Hadamard Rapide (FHT) (voir Algorithme 14).
3. Récupération du message avec la fonction de recherche de pics (voir Algorithme 15).

L'objectif principal de la première étape est de supprimer la multiplicité du mot de code en ajoutant, bit à bit, chaque répétition (voir Algorithme 13). Le résultat est un mot de code étendu de longueur 128, laissé dans les ensembles $[\![0, 3]\!]$ ou $[\![0, 5]\!]$, selon la valeur de la multiplicité μ.

Algorithm 13 Expansion et somme

Entrée : mot de code $\mathbf{c}$ et la multiplicité μ.
Sortie : mot de code étendu $\mathbf{c}'$

1: $\mathbf{c}' = \mathbf{0} \in \mathbb{N}^{128}$
2: **for** $i \in [\![0, \mu]\!]$ **do**
3: **for** $j \in [\![0, 128]\!]$ **do**
4: $\mathbf{c}'[j]$ += $\mathbf{c}[128 \times i + j]$
5: **return c'**

3.5.3.4 Transformée de Hadamard Rapide (FHT)

La fonction FHT est une transformée de Fourier discrète généralisée, qui est appliquée pour étendre le mot de code (voir Algorithme 14). En termes pratiques, cette fonction est similaire à la multiplication du mot de code étendu par une matrice de Hadamard. Cette équivalence est vraie car MacWilliams et Sloane [MS77] ont démontré que la distribution du poids des *cosets* peut être établie en appliquant une transformée de Hadamard. Cela permet donc un décodage en utilisant une approche du maximum de vraisemblance, permettant de déterminer la distance entre un message reçu et tous les mots de code possibles. Pour un code $\mathrm{RM}(1, m)$, la matrice de Hadamard appropriée à utiliser est H_{2^m}, qui peut être définie de manière récursive (selon l'équation 3.27).

$$H_{2m} = \begin{pmatrix} H_m & H_m \\ H_m & -H_m \end{pmatrix}, \; H_1 = 1 \tag{3.27}$$

Appliquer une telle multiplication matrice vecteur nécessiterait $2^m \times 2^m$ additions et soustractions. Heureusement, H_{2^m} peut être écrit comme un produit de m matrices éparses $2^m \times 2^m$ avec seulement deux éléments non nuls par colonne (voir l'Équation 3.28). Cette observation permet de remplacer la multiplication de vecteur et de matrice par m multiplications de vecteurs et de matrices éparses.

$$H_{2^m} = M_{2^m}^{(1)} M_{2^m}^{(2)} \cdots M_{2^m}^{(m)}, \; M_{2^m}^{(i)} = I_{2^{m-i}} \otimes H_2 \otimes I_{2^{i-1}}, \; 1 \leq i \leq m \tag{3.28}$$

avec I_n la matrice identité de taille $n \times n$ et $\otimes$ le produit de Kronecker. C'est la raison pour laquelle l'Algorithme 14 est composé d'une boucle for avec l'argument $m = 7$.

Dans notre cas $\mathrm{RM}(1, 7)$, cette transformation renvoie un vecteur de longueur 128. La dernière fonction Find Peaks permet de finaliser le décodage.

Algorithm 14 Transformée de Hadamard Rapide FHT

Entrée : mot de code étendu $\mathbf{c}$ et la multiplicité μ.
Sortie : structure de mot de code étendu transformée $\mathbf{c}$

for $passe \in [\![0, 6]\!]$ **do**
  for $i \in [\![0, 63]\!]$ **do**
    $\mathbf{d}[i] = \mathbf{c}[2i] + \mathbf{c}[2i + 1]$
    $\mathbf{d}[i + 64] = \mathbf{c}[2i] - \mathbf{c}[2i + 1]$
  $\mathsf{swap}(\mathbf{d}, \mathbf{c})$ ▷ copie de $\mathbf{d}$ dans $\mathbf{c}$ et $\mathbf{c}$ dans $\mathbf{d}$
$\mathbf{c}'[0] \mathrel{-}= 64 * \mu$
return $\mathbf{c}$

Parmi ce vecteur, l'argument du maximum absolu donne les 7 bits de poids faible du message décodé. Le bit de poids fort est donné par le signe de ce maximum (voir Algorithme 15).

Algorithm 15 Trouver les pics

Entrée : structure de mot de code étendu transformée $\mathbf{c}$
Sortie : message $\mathbf{m}$

$p_v = 0$ ▷ Valeur du Pic
$p_a = 0$ ▷ Valeur Absolue du Pic
$p_p = 0$ ▷ Position du Pic
for $i \in [\![0, 123]\!]$ **do**
  $(t, t_{\text{abs}}) = (\mathbf{c}[i], \mathsf{absolute}(\mathbf{c}[i])$
  if $t_{\text{abs}} > p_a$ **then**
    $p_v = t$
    $p_a = |t|$
    $p_p = i$
$p_p \mathrel{+}= 128 * (p_v > 0)$ ▷ Mise en place du bit de poids fort
return p_p

CHAPITRE 4

ATTAQUES PAR CANAUX AUXILIAIRES

Contents

4.1 Les attaques physiques

Les attaques physiques exploite les propriété matérielles afin de contourner les mécanismes de sécurité classiques et de retrouver de l'information secrète, souvent inaccessible pour une attaque conventionnelle. Dans ce manuscrit, je intéresserai principalement aux attaques physiques ayant pour cible des algorithmes cryptographiques, cependant d'autres cas d'applications peuvent exister : l'attaque d'un algorithme vérifiant un code PIN pour ne citer qu'un exemple. Les attaques physiques sont nombreuses et variées et dépendent très souvent du modèle de l'attaquant, c'est-à-dire des ressources dont un attaquant dispose pour leur mise en œuvre.

4.1.1 Modèle de l'attaquant

Dans le contexte des attaques physiques, le modèle de l'attaquant fait référence à l'ensemble des hypothèses et des caractéristiques attribuées à un attaquant. Le modèle de l'attaquant définit les capacités, les connaissances et les ressources qu'on suppose qu'un adversaire possède lorsqu'il tente d'exploiter les vulnérabilités physiques d'un système cryptographique. Comprendre et définir le modèle de l'attaquant est crucial pour caractériser la sécurité d'un système et concevoir des contre-mesures efficaces. L'objectif de définir un modèle d'attaquant est double : *(i)* Comparer les attaques. Pour deux attaques physiques avec le même objectif, l'attaque la plus intéressante sera celle avec le modèle d'attaquant le plus faible. En règle général, il est toujours intéressant pour un attaquant de sélectionner l'ensemble d'hypothèses le plus petit afin de maximiser les cas d'applications de son attaque. *(ii)* Le second objectif est la protection des schémas cryptographiques contre les attaques physiques. Si lors de sa conception, un schéma peut-être prouvé sécurisé contre un modèle d'attaquant le plus fort possible, cela amène de grande propriété de sécurité face aux attaques physiques.

Afin de mieux catégoriser les hypothèses et les contraintes que l'on peut ajouter dans le modèle de l'attaquant, voici une liste des critères qui peuvent être considérés afin de comparer les cas d'applications :

— les **connaissances préalables** décrivent toute connaissance que l'adversaire pourrait avoir sur le système ou l'algorithme cryptographique. Alors que la sécurité des schéma ne repose plus sur l'offuscation, il est également communément admis que les algorithmes ciblés sont connus et bien compris par l'attaquant, ainsi que l'implémentation utilisée.

— Le **niveau d'accès** dont dispose l'attaquant au système cible. Par exemple, l'attaquant a-t-il un accès physique à l'appareil, ou est-il limité à des observations à distance ? Cette distinction est cruciale pour différencier les attaques actives des attaques passives.

  - **Les attaques actives** requièrent une interaction avec la cible. Les attaques par injections de fautes sont des attaques actives, l'attaquant perturbent le systèmes avec une impulsion laser ou électromagnétique. Le but est de provoquer un comportement inhabituel des opérations cryptographiques afin d'en extraire des informations secrètes. Certaines fautes sont temporaires, comme le changement de valeur d'un registre ou d'une donnée mémoire mais certaines fautes modifient irrémédiablement le comportement du système, pouvant conduire à la destruction du système. Ces erreurs créées peuvent être exploitées pour révéler des informations sensibles, par exemple en étudiant les implications de différents types de fautes sur la sortie de l'algorithme. Les attaques nécessitant le choix des entrées de l'algorithme, telles que les attaques par chiffrés choisis, sont aussi des attaques actives. L'attaquant a un rôle dans l'attaque en cours.
  - **Les attaques passives**, quant à elles, ne nécessitent pas d'interaction avec la cible, il s'agit simplement de l'observation et l'analyse des fuites d'informations involontaires émises par le système lors de son fonctionnement normal. L'attaquant observe la cible dans un contexte normal de fonctionnement et est capable retrouver des secrets manipulés par la machine. Dans la suite de ce manuscrit, nous nous concentrons sur les attaques passives.

— La **quantité** de mesure que l'attaquant peut faire sur la cible qui manipule le secret. La distinction réside dans le fait que l'attaquant se concentre sur les variations entre plusieurs mesures physiques (vertical) ou exploite les informations au sein d'une mesure physique donnée (horizontal) de la cible.

  - Les **attaques horizontales** [Wal01, CFG+10] se basent sur le fait que la même donnée sensible (ou plus généralement plusieurs valeurs intermédiaires dépendant de cette même valeur sensible) soit manipulée plusieurs fois dans la même exécution. Cette construction peut ouvrir la porte à une attaque horizontale, permettant d' exploiter la dépendance algébrique entre plusieurs résultats intermédiaires pour discriminer les hypothèses de clé. Dans certains cas, ce type d'attaque peut être réalisé en une seule trace, pouvant rendre également rendre vulnérables des schémas manipulant des secrets éphémères.
  - Dans le cas d'**attaques verticales**, on exploite les variations observées dans plusieurs mesures physiques afin des déduire des informations sur les données sensibles manipulées. Le but de l'attaquant est d'observer des modifications de comportement dans les mesures afin de déduire des informations sur une donnée commune à toutes ces exécutions, par exemple un clef secrète statique.

Ce type d'attaque nécessite parfois l'acquisition de dizaine, centaine de milliers, voire millions de traces.

— **L'accès à l'information** : ce critère prend en compte les informations auxquelles l'attaquant peut accéder. Cela peut inclure l'accès à des mesures physiques préalables, ou la possession de certains profils préalablement acquis.

- Dans une **attaque profilée**, l'attaquant dispose de connaissances préalables ou d'un accès à un profil du système cible. Le profil inclut généralement des informations sur le comportement de l'algorithme cryptographique sur le dispositif cible spécifique. Avec ce profil préalablement acquis, l'attaquant peut affiner sa stratégie d'attaque, rendant souvent l'attaque plus efficace et ciblée. La création du profil peut se faire lors d'une phase préliminaire où l'attaquant acquière des mesures lors d'une phase d'apprentissage (ou d'entrainement) sur le même dispositif ou un dispositif similaire (dit *clone device*).
- En revanche, on parle d'**attaques non profilées** lorsque l'attaquant n'a pas accès à un profil préalablement acquis du système cible. L'attaque est menée sans connaissance préalable des caractéristiques spécifiques du dispositif cible. Les attaques non profilées sont généralement plus difficiles pour l'attaquant, car il doit découvrir et exploiter des vulnérabilités en temps réel sans bénéficier d'un profil affiné. Ces attaques peuvent impliquer une approche plus générique, ciblant les faiblesses connues des algorithmes cryptographiques plutôt que d'exploiter des caractéristiques spécifiques du dispositif cible.

Définir le modèle de l'attaquant est une étape essentielle pour évaluer la sécurité d'un système cryptographique face aux attaques par canaux auxiliaires. Cela aide les chercheurs à anticiper les menaces potentielles et à concevoir des contre-mesures efficaces contre les caractéristiques et les capacités spécifiques attribuées à l'attaquant.

4.1.2 Les attaques par canaux auxiliaires

Les attaques physiques passives, aussi appelées attaques par canaux auxiliaires, secondaires ou cachés (SCA en anglais) sont des attaques exploitant le comportant physiques d'une cible afin de retrouver des informations secrète manipulées par celle-ci. Les canaux auxiliaires peuvent être de différents types tels que le temps d'exécution, la consommation de courant, le rayonnement électromagnétique, le bruit, la chaleur, etc ... générés par la machine qui exécute les algorithmes cryptographiques. Il faut attendre les travaux de Paul Kocher en 1996 [Koc96], pour montrer que ces

mesures physiques permettent de déduire des informations sur des informations secrètes manipulées par un algorithme de chiffrement tel que RSA. Par la suite, des travaux ont montré que ces attaques permettent souvent de contourner les défenses conventionnelles de la cryptologie. Ces attaques mettent en évidence la complexité croissante de la sécurisation des systèmes informatiques, nécessitant une approche dédiées pour contrer ces "nouvelles" menaces.

4.2 Simulations et Modèles de fuites

Une attaque en simulation consiste à simuler, créer des valeurs contrôlées qui correspondent à des mesures physiques que l'on pourrait obtenir dans le cas d'une vraie attaque. L'avantage de cette approche est double : *(i)* l'information contenue dans les traces peut-être choisie, dont son modèle de fuites (voir sous-section 4.2.1). Les valeurs intermédiaires d'intérêt peuvent être ciblées et ajoutées dans la mesure simulée. Et d'autre part, *(ii)* le niveau de bruit peut être contrôlé. Ainsi l'attaque peut être testée et évaluée sous différentes conditions de bruits.

Plus formellement, en reprenant les notations de [Sta10], on suppose que la fuite L, au moment de la manipulation de valeurs intermédiaires $\mathbf{v}(t)$ est décrit par l'équation suivante :

$$L(t) = \zeta(\mathbf{v}(t)) + \texttt{Noise}(t) \tag{4.1}$$

La fuite L est composée de deux parties indépendantes : *(i)* une partie déterministe décrite par le modèle de fuite $\zeta(\mathbf{v})$ où $\mathbf{v}$ est la valeur intermédiaire ciblée. L'hypothèse que les fuites physiques observées soient corrélées avec la valeur manipulée vient du fait que les bus de données sont les éléments les plus consommateurs d'un microcontrôleur [BCO04]. Un bus de donnée est un canal de communication bidirectionnel par lequel les variables intermédiaires sont transférées entre différents composants du système. Ce modèle linéaire correspond donc assez bien à la réalité. Bien sûr les bus de données ne sont pas les seuls éléments consommateurs de courant dans le microcontrôleur. On a donc *(ii)* le bruit, une partie aléatoire et indépendante de la variable intermédiaire manipulée décrivant le comportant physique de tous les autres composants. Le bruit $\texttt{Noise}(t)$ sera décrit dans la sous-section 4.2.2, le modèle ζ peut-être définis de différentes manières. Dans la sous-section suivantes, je décris 3 modèles de fuites, que je ré-utiliserai plus tard dans le manuscrit. Bien que ces trois modèles de fuites soient les principaux modèles de fuites utilisés pour les attaques physiques à ma connaissance, la liste n'est pas exhaustive.

4.2.1 Modèles de fuite

Un modèle de fuite est un cadre théorique qui décrit comment des informations sont divulguées à partir d'un composant pendant son fonctionnement normal. Le terme "fuite" fait référence à la divulgation non intentionnelle d'informations sensibles, au travers de mesures physiques.

Le modèle de fuite fait le lien entre les informations secrètes traitées par un système et les mesures physiques. L'idée de base est que certains processus physiques à l'intérieur d'un dispositif informatique présentent des variations qui dépendent des données traitées. C'est cette dépendance et la nature de cette dépendance qui vont permettre de retrouver des informations sensibles lors de l'exécution d'un algorithme cryptographique.

Modèle de fuite en poids de Hamming Dans ce modèle de fuite, on suppose que la mesure physique varie de façon corrélée avec le poids de Hamming des valeurs manipulées par le composant. On peut dire de manière très vulgarisée que la consommation de courant du système dépend directement du nombre de "1" dans le vecteur manipulé. On peut réécrire l'équation 4.1 de la façon suivante :

$$L(t) = \alpha \cdot \texttt{HW}(\mathbf{v}(t)) + \beta + \texttt{Noise}(t) \tag{4.2}$$

où $\alpha \in \mathbb{R}^*$ et $\beta \in \mathbb{R}$ sont des paramètres de ce modèle qui peuvent être dépendant de la cible choisie, des conditions de mesures ... Afin de simplifier les modèles théorique, et sans perte de généralité, on choisi généralement $(\alpha, \beta) = (1, 0)$.

On prend comme taille des variables, dont on considère le poids le Hamming, soit la taille des registres manipulé par le processeur, soit la taille des données manipulées par l'algorithme ciblé (souvent un octet). Ce modèle de fuite est le modèle de fuite le plus répandu et c'est généralement celui qui est considéré lors d'attaque réalisées en simulation. Cependant, il existe d'autres modèles de fuites.

Modèle de fuite en distance de Hamming Le modèle de fuite en distance de Hamming suppose que la fuite est dépendante du nombre de bits qui change, qui sont mis à jour, entre deux valeurs intermédiaires manipulées par le processeur. Ce modèle de fuite a été introduit par Brier et al. [BCO04] dans le papier qui présente la Correlation Power Analysis (CPA). L'argument derrière ce modèle de fuite est de dire que c'est la mise à jour des registres qui créer une variation physique, que l'on décrit avec ce modèle de fuite, autrement dit le passage d'un bit 0 vers un 1 ou inversement. Il ne s'agit donc plus de considérer uniquement la valeur intermédiaire manipulée au temps t mais également la valeur manipulée juste avant. Plus formellement, pour chaque valeur intermédiaire $\mathbf{v}(t)$, on note $\mathbf{v}'(t)$ une valeur précédemment

manipulée par le processeur, le modèle de fuite adapté de l'équation 4.1 s'écrit alors :

$$L(t) = \alpha \cdot \texttt{HW}\left(\mathbf{v}'(t) \oplus \mathbf{v}(t)\right) + \beta + \texttt{Noise}(t) \tag{4.3}$$

Dans certaines attaques, la valeur de $\mathbf{v}'(t)$ est fixe pour tout t, autrement dit, c'est toujours la même valeur qui est manipulée juste avant la valeur intermédiaire qui nous intéresse, cette valeur peut par exemple être l'instruction assembleur précédent l'appel de la variable intermédiaire. Si l'on choisit $\mathbf{v}'(t) = 0$, alors on décrit exactement le modèle en poids de Hamming.

Modèle de fuite binaire Le dernier modèle de fuite que je voudrais décrire est le modèle de fuite au niveau du bit. Ce modèle suppose que chaque bit de la valeur intermédiaire manipulée a une influence sur la fuite observée. Cette influence peut être différente en fonction de la position du bit dans la valeur. Plus formellement, chaque bit va être pondéré par une certaines valeur α_i étant un paramètre du modèle. Si l'on considère $(v_1(t), \cdots, v_m(t))$ la décomposition binaire de la valeur intermédiaire $\mathbf{v}(t)$, l'équation 4.1 peut donc être re-écrite :

$$L(t) = \sum_{i=1}^{m} (\alpha_i \cdot v_i(t)) + \beta + \texttt{Noise}(t) \tag{4.4}$$

Ce modèle de fuite est la base d'un outils que l'on verra plus loin : la LRA. On notera que le modèle de fuite en poids de Hamming est un cas particulier du modèle de fuite binaire, lorsque les coefficient de pondérations $(\alpha_i)_{1 \leq i \leq m}$ sont égaux et non-nuls.

4.2.2 Bruit simulé

Le bruit observé dans les mesures physiques vient de toutes les activités du microcontrôleur qui ne sont pas en relation avec la valeur intermédiaire et qui consomme du courant ainsi que des bruits liés à la mesure. Généralement en simulation, ce bruit est représenté par un tirage gaussien centré en zéro et indépendant pour chaque mesure. Ainsi :

$$\texttt{Noise}(t) = \mathcal{N}(0, \sigma^2) \tag{4.5}$$

On peut remarqué dans les équations 4.2, 4.3 et 4.4 que les modèles prévoient un biais β lié au bruit de mesure et au comportement de la cible. Ce biais n'est pas corrélé à la valeur intermédiaire manipulée. De ce fait, il peut être intégrer dans la description du bruit :

$$\texttt{Noise}(t) = \mathcal{N}(\beta, \sigma^2) \tag{4.6}$$

On notera toutefois que la valeur de β n'a aucune influence sur les analyses de ces traces, c'est pourquoi la valeur de β est souvent nulle dans les simulations.

Avantages des attaques simulées Les simulations présentes de nombreux avantages, et un attaquant à tout intérêt à faire son attaque en simulation afin de : *(i)* d'assurer la reproductibilité des résultats. En effet, l'ensemble des étapes de simulation étant indépendants de mesure physique ou de matériel spécifique, les scripts de simulation peuvent être facilement partageable pour reproduction. De plus les simulations reposent sur des modèles de fuite bien définies, rendant également simple une reproduction. *(ii)* comparer des attaques entre elles. On peut par exemple comparer deux attaques simulées, basées sur le même modèle de fuite en fonction du niveau de bruit maximal atteint pour lequel la probabilité de réussite de l'attaque reste supérieur à p. *(iii)* gagner du temps. La simulation permet de tester divers paramètres et outils en un temps limité. Bien entendu, il est essentiel d'éviter l'hyper-paramétrisation, mais la simulation offre la possibilité de choisir les outils les plus adaptés dans le contexte de l'attaque et du modèle de fuite. *(iv)* réduire les risques. En effet, dans le cas d'attaques actives, l'attaque peut détruire ou endommager le composant. La simulation offre un environnement contrôlé pour tester et comprendre l'impact potentiel d'une attaque sans endommager la cible de manière non-intentionnelle. *(v)* expérimenter une même attaque dans divers scénario. Un modèle de fuite n'étant pas unique, une simulation permet d'en tester plusieurs et de comparer les résultats entre eux. Ces tests permettent de comprendre la robustesse d'une stratégie d'attaque dans différentes circonstances.

En résumé, la simulation permet un gain de temps, un gain d'efficacité, et un meilleur partage avec la communauté tout en limitant les risques et les couts des attaques physiques.

4.3 Outils pour les attaques par canaux auxiliaires

Afin de traiter les mesures physiques obtenues ou les traces simulées, on dispose d'un ensemble d'outils. Dans cette section, je vais décrire les principaux outils pour les attaques physiques, cette liste n'est également pas exhaustive.

4.3.1 Évaluation des fuites

L'évaluation des fuites (ou *Leakage Assessment*) fait référence à une évaluation visant à déterminer si des informations sensibles ou secrètes peuvent être déduites à partir de mesures physiques. Elle se fait sur un ensemble de mesures (simulé ou non) dont on connait les valeurs intermédiaires manipulées pour chacune d'elles. On se retrouve alors avec deux ensembles : d'un côté *(i)* l'ensemble des mesures physiques que l'on note X et de l'autre *(ii)* l'ensemble des valeurs intermédiaires associées (ou *labels* en anglais). L'idée de ce tests statistiques est de donc de vérfier si il existe une corrélation entre les deux ensembles X et Y. Dans le cas ou une corrélation est observée,

elle peut potentiellement être exploiter afin de retrouver tout ou partie de la donnée sensible manipulée. C'est donc un bon point de départ pour savoir si une attaque a une chance d'être réaliser, identiquement, l'absence de corrélation peut être un bon signe que les traces sont "sécurisées".

Un petit avertissement doit être fait cependant : même si une corrélation est prouvée dans l'ensemble des mesures, rien ne garanti qu'une attaque soit possible. Standaert [Sta19] montre que ce type d'évaluation n'est pas suffisant pour garantir la sécurité d'un schéma ou la faisabilité d'une attaque. Il reproche a certains papiers de ne pas aller plus loin qu'une évaluation de fuites et de garantir qu'une attaque est possible ou que le schéma ou la contremesure proposée est sécurisée : c'est faux. L'analyse doit être plus poussé pour prétendre à de telles propriétés et/ou une prudence doit être de mise lors de l'interprétation de ces résultats. Il conclue en mentionnant que ces outils sont pertinent pour évaluer des fuites mais ne sont pas suffisant pour constituer une attaque ou une preuve de sécurité en tant que telle.

Parmi les outils d'évaluation de fuite, j'en présente deux, que l'on retrouvera plus tard pour des attaques que j'ai mené sur HQC : Le test t de Welch et la LRA. Une fois de plus, cette liste n'est pas exhaustive.

4.3.1.1 Test t de Welch

Le test t de Welch (ou Welch t-test en anglais) est un test statistique utilisé pour déterminer s'il existe une différence significative entre les moyennes de deux ensembles indépendants. Il s'agit d'une adaptation du test t de Student, mais il est plus robuste dans les situations où les variances des deux groupes comparés ne sont pas nécessairement égales. Le test t de Welch permet de détecter une corrélation linéaire entre deux ensembles.

Soient S_0 et S_1 deux ensemble de cardinal n_0 et n_1, de moyennes μ_0 et μ_1 et de variance σ_0 et σ_1. Le calcul de la valeur t, résultat du test t de Welch est donné par la formule suivante :

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\left(\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}\right)}} \tag{4.7}$$

Dans le cadre des attaques physiques, il est communément accepté de considérer une valeur seuil de $|t|$ fixée à 4.5. Un résultat du test t de Welch excédant cette valeur assure avec une probabilité supérieur à 99.999% que les deux ensembles S_0 et S_1 sont statistiquement distinguables [DCE16]. Si la valeur de $|t|$ est en dessous de ce seuil, il est garantit que les deux ensembles ne peuvent pas être distinguées par un classificateur linéaire.

Le test t de Welch est limitée a une utilisation ou le nombre de classe est exactement 2. Cela peut être particulièrement pertinent lorsque par exemple la valeur intermédiaire ciblée est seulement un bit ou une action déterministe

à deux sorties : par exemple, l'algorithme décode ou ne décode pas. Il peut également être utilisé dans un contexte avec plus de classes, en montrant que chaque classe peut être ou non distinguées de chacune des autres classes.

4.3.1.2 LRA

L'analyse de régression linéaire (ou *Linear Regrassion Analysis (LRA)*) est une technique statistique utilisée pour étudier la relation entre plusieurs variables dans le but de prédire la valeur d'une variable dépendante en fonction des autres. Prenons le cas ou l'on ne considère que deux variables X et Y comme énoncées au début de cette section. La relation entre ces deux variables est modélisée par la LRA comme une ligne droite, d'où le terme "linéaire". L'équation 4.8 d'une régression linéaire à deux variables peut être exprimée comme suit :

$$X = \beta_0 + \beta_1 \cdot Y + \epsilon \tag{4.8}$$

où ϵ est l'erreur résiduelle, qui représente la différence entre la valeur prédite et la valeur réelle et β_0, β_1 des paramètres. L'objectif de l'analyse de régression linéaire sera d'estimer les coefficients β_0 et β_1.

On note que cette approche peut être facilement généralisée à k variables $(Y_i)_{1 \leq i \leq k}$ de la manière suivante :

$$X = \beta_0 + \sum_{j=1}^{k} \beta_j \cdot Y_j + \epsilon \tag{4.9}$$

Si l'on prend chaque bit de la valeur intermédiaires, on retrouve exactement le modèle fuite binaire tel que décrit en section 4.2.1. Dans un contexte d'attaque physique, on veut prédire une valeur intermédiaire en fonction de mesure physiques effectuées. On suppose que les paramètres du modèles sont invariants pour chacune des n mesures physiques $(x_i)_{1 \leq i \leq n}$. Avec un bruit ϵ supposé gaussien, il existe une unique solution $\tilde{\beta} = \left(\tilde{\beta_0}, \cdots, \tilde{\beta_k}\right)$. Cette solution vise à estimer les paramètres $\beta = (\beta_0, \cdots, \beta_k)$ de manière à minimiser la somme des moindres carrés (ou *Residual Sum of Square (RSS)*) défini comme suit :

$$\begin{aligned} RSS &= \sum_{i=1}^{n} (x_i - \tilde{x}_i)^2 \\ &= \sum_{i=1}^{n} \left(x_i - \left(\tilde{\beta_0} + \sum_{j=1}^{8} \tilde{\beta_j} \cdot y_{i,j} \right) \right)^2 \end{aligned} \tag{4.10}$$

Cette approche est connue sous le nom de méthode des moindres carrés. Avec une telle approche, on peut estimer la précision du modèle, pour chaque

point de la mesure, avec un coefficient de détermination, que l'on note R^2, calculé de la manière suivante :

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (x_i - \tilde{x_i})^2}{\sum_{i=1}^{n} (x_i - E(x))^2} \tag{4.11}$$

Ce coefficient de détermination permet de faire de l'évaluation de fuite. En effet, si le modèle retrouve correctement les valeurs intermédiaires manipulées, cela veut dire qu'il existe une relation linéaire pouvant éventuellement être exploité dans la mesure physique. Les points d'intérêts dans la mesure seront donc ceux ou la valeur de R^2 est non-nul.

4.3.2 Rapport Signal/Bruit

Le rapport Signal/Bruit (ou Signal-to-Noise Ratio (SNR)), également appelé rapport signal sur bruit, est une mesure utilisée dans divers domaines, notamment les télécommunications, le traitement audio et le traitement du signal. Cet outils permet de quantifier la force relative d'un signal par rapport au bruit de fond présent dans un système. En général, le Signal-to-Noise Ratio (SNR) est calculé en divisant la puissance d'un signal par la puissance du bruit. Dans le cadre des attaques physiques, la formule du SNR est donnée par l'équation :

$$\texttt{SNR} = \frac{L}{\texttt{Noise}} \tag{4.12}$$

Une valeur de SNR plus élevée indique un signal de meilleure qualité car la puissance du signal est plus significative par rapport au bruit. À l'inverse, un SNR plus faible suggère que le signal peut être plus difficile à distinguer du bruit de fond. Dans les systèmes de communication, un SNR élevé est souhaitable car cela implique que le signal transmis est plus robuste contre les interférences et les distorsions. En revanche, si l'on veut protéger un microcontrôleur contre les attaques physiques, on souhaite que le SNR soit le plus bas possible, afin de rendre la mesure de l'information indistinguable du bruit environnant. Ceci peut être fait en ajoutant volontairement des générateurs de bruits sur les microcontrôleurs. Malheureusement, il a été montré que l'ajout de bruit de cette manière permet d'améliorer la qualité de certaines attaques basées sur des réseaux de neurones [KPH+19].

Plus formellement, on peut calculer la valeur du SNR d'une mesure en faisant quelques calculs de probabilité, selon l'équation suivante :

$$\texttt{SNR} = \frac{\texttt{Var}(\texttt{E}(X|Z))}{\texttt{E}(\texttt{Var}(X|Z))} \tag{4.13}$$

où X représente la variable aléatoire résultant de la mesure physique et Z la variable aléatoire représentant le "label", autrement dit la valeur

intermédiaire ciblée. `E` et `Var` sont l'espérance et la variance dans leur sens usuel.

4.4 Grandes familles d'attaques par canaux auxiliaires

Dans cette section, je parlerai de quelques grandes familles d'attaques par canaux auxiliaires. La plupart de ces familles sont historiquement appelées "*[...] Power Attack*". Cependant, il est établi que si la consommation de courant est *data-dependent*, c'est-à-dire dépendante des données qui circule ou sont manipulées par la cible, il en est de même pour les émanations électromagnétiques [Sta10]. Toutes les attaques présentées dans cette section exploitant la consommation de courant peuvent donc être utilisées dans une attaque mesurant des ondes électromagnétiques.

4.4.1 Attaque simple d'analyse de courant (SPA)

Une attaque simple par analyse de courant (ou *Simple Power Analysis (SPA)*) est une attaque qui implique l'examen visuel des graphiques du courant utilisé par un dispositif au fil du temps. Des variations de comportement physique se produisent lorsque le dispositif effectue différentes opérations. Par exemple, différentes instructions exécutées par un microprocesseur auront des profils de consommation de courant différents. L'exemple le plus couramment évoqué est celui des algorithmes d'exponentiation rapide, dont le but est de calculer $c := a^e \mod N$. Notamment les algorithmes de type *Square and Multiply*, qui se base sur la décomposition binaire de $e = (e_1, \cdots, e_n)$.

Algorithm 16 Exponentiation rapide *Square and Multiply* de gauche à droite

Require: $a, e = (e_1, \cdots, e_n), N$
Ensure: $c := a^e \mod N$
  $R \leftarrow 1$
  for i=1, n **do**
    $R \leftarrow R^2 \mod N$
    if $e_i = 1$ **then**
      $R = R \times e \mod N$
  return R

Si un attaquant est capable de distinguer visuellement une mesure physique provenant d'une multiplication et une mesure physique provenant d'une élévation au carré, alors il sera très simple, à partir d'une mesure physique correspondant à l'Algorithme 16 de retrouver les bits de l'exposant.

Une manière de se protéger contre ce type d'attaque est de faire des opérations inutiles (ou *Dummy operations*). Ces opérations n'ont aucune utilité dans le calcul des sorties de l'algorithme, elles font même perdre du temps d'exécution et de la performance, mais elles permettent d'avoir des mesures physiques indistinguables (en théorie), car les opérations réalisées sont toujours le même. La protection de l'Algorithme 16 est proposée dans l'Algorithme 17 :

Algorithm 17 Exponentiation rapide *Square and Multiply* avec *dummy operations* de gauche à droite

Require: $a, e = (e_1, \cdots, e_n), N$
Ensure: $c := a^e \mod N$
  $R_0 \leftarrow 1$
  $R_1 \leftarrow a$
  for i=1, n **do**
    $R_0 \leftarrow {R_0}^2 \mod N$
    if $e_i = 1$ **then**
      $R_0 = R_0 \times e \mod N$
    else
      $R_1 = R_1 \times e \mod N$
  return R_0

On sait aujourd'hui bien protéger nos implémentations contre ce type d'attaque en faisant attention de ne pas faire de branchements conditionnels dépendants d'une donnée secrète. Clavier et al. [CFG+11] présente une autre version, permettant de faire le même calcul en remplaçant les multiplication par des mise au carré en se rendant compte que :

$$\begin{aligned} x \times y &= \frac{(x+y)^2 - x^2 - y^2}{2} \\ x \times y &= \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2 \end{aligned} \tag{4.14}$$

4.4.2 *Timing Attacks*

Les *Timing attacks* sont des attaques exploitant le temps d'exécution d'un algorithme afin de retrouver des données secrètes. En observant les différences de temps de calcul de l'algorithme pour différentes entrées, un attaquant pourrait déduire des données secrètes.

Un exemple très répandu d'illustration est celui de la vérification de code PIN, on peut écrire un premier algorithme de la sorte :

Un code PIN juste devra passer par quatre branchements conditionnels avant le retour de la fonction. En revanche, un code PIN erroné sur le premier chiffre ne passera que un branchement conditionnel avant de stopper

Algorithm 18 Vérification naïve de code PIN à 4 chiffres

Require: $C = (c_1, c_2, c_3, c_4)$ le bon code
Require: $T = (t_1, t_2, t_3, t_4)$ la proposition envoyée par un utilisateur
Ensure: True si $C = T$, False sinon.

```
if c1 = t1 then
    if c2 = t2 then
        if c3 = t3 then
            if c4 = t4 then
                return Vrai
return Faux
```

l'algorithme. En testant les 10 possibilités pour le premier chiffre, et en conservant le candidat qui représentait le plus grand temps d'exécution, un attaquant peut retrouver le bon code PIN en 40 tentatives. Sans aucune autre information, le nombre de tentative moyen permettant à un attaquant de retrouver le code PIN est 5000.

Comme précédemment pour les SPA, on sait protéger les algorithmes contre les *timing attacks*. Pour cela il faut implémenter que des algorithmes dit en temps constant, c'est-à-dire que dans une utilisation normale de l'algorithme, le temps d'exécution ne soit pas dépendant d'une valeur de secret. L'absence de branchement conditionnel sur une donnée secrète et le temps constant sont les deux propriétés minimales indispensables lorsque l'on développe un schéma cryptographique.

4.4.3 Attaques différentielles (*DPA*)

Les attaques différentielles (ou *Differential Power Analysis (DPA)*) sont introduites par Kocher et al. [KJJ+98, KJJ99] et formalisées par Messerges et al. [MDS99] en 1999. Ce type d'attaque vise à retrouver l'information sur un bit précis. Selon une hypothèse de clef, on peut trier les mesures en deux groupes X_0 et X_1 de telle sorte que les mesures manipulant la même valeur de b soient regroupées :

$$X_i = \{L \mid b = i\} \tag{4.15}$$

Si on suppose que lorsque qu'un bit b d'une valeur intermédiaire est manipulé, le comportement physique de la cible est différent si $b = 0$ ou $b = 1$, comme c'est le cas pour le modèle de fuite en poids de Hamming par exemple, alors on saura distinguer X_0 et X_1 sous la bonne hypothèse de clef secrète. Cette distinction se traduit par la présence de pics lors que l'on regarde la valeur de $\bar{X}_1 - \bar{X}_0$.

Il sera noté plus tard par Brier et al. [BCO04] que ce type d'attaque présente des pics fantômes même lorsqu'une mauvaise hypothèse de clef est sélectionnée. Ce défaut présents dans les attaques différentielles sera corrigé par l'invention des attaques par corrélation.

4.4.4 Attaques par corrélation (*CPA*)

Les attaques par corrélation (ou *Correlation Power Analysis (CPA)*) sont des attaques dont le but est, pour un ensemble d'hypothèses données sur une variables aléatoire sensible, de déterminer laquelle présente le plus de corrélation avec les données mesurées. Ce type d'attaque a été introduit par Brier et al. [BCO04] en 2004.

Un outils permettant de mesurer cette corrélation est le coefficient de Pearson. Soient variables aléatoires X et Y, la corrélation de Pearson entre ces deux variables est donnée par l' équation 4.16. Dans notre cas, nous calculons la corrélation entre les mesures physiques et les valeurs intermédiaires manipulées par l'algorithme cible.

$$\rho_{\mathbf{X},\mathbf{Y}} = \frac{\operatorname{cov}(\mathbf{X},\mathbf{Y})}{\sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}} = \frac{\mathrm{E}\left[(\mathbf{X}-\mu_{\mathbf{X}})(\mathbf{Y}-\mu_{\mathbf{Y}})\right]}{\sqrt{\mathrm{E}\left[(\mathbf{X}-\mu_{\mathbf{X}})^2\right]\mathrm{E}\left[(\mathbf{Y}-\mu_{\mathbf{Y}})^2\right]}} \tag{4.16}$$

où $\mu_{\mathbf{X}}$ et $\sigma_{\mathbf{X}}$ sont, respectivement, la valeur attendue mesurée et la variance mesurée de $\mathbf{X}$.

L'équation (4.16) renvoie une valeur entre -1 et 1. Plus la valeur de $|\rho_{\mathbf{X},\mathbf{Y}}|$ est proche de 1, plus la corrélation entre les deux variables est forte.

Dans un scénario d'attaque, la première étape est l'enregistrement de mesures physiques $(x_i)_{1\leq i\leq N}$. Ces mesures correspondent à la manipulation de valeurs intermédiaires Z dépendante d'une fonction d'une donnée secrète K, i.e. $z_i = f_i(K)$. On peut donc calculer l'ensemble des possibilités pour Z en fonction de la donnée secrète $Z_j = \{z_i = f_i(k_j)\}$. Si l'attaque par corrélation est un succès, la valeur de corrélation maximale sera atteinte lors $k_j = K$, l'hypothèse est sur la bonne valeur secrète.

4.4.5 Attaque par profilage

Les attaques par profilage [CRR03, RO04] (ou *Template Attacks*) sont une famille d'attaque physique qui exploite une connaissance détaillée des caractéristiques physiques ou des schémas de fuite d'un dispositif cryptographique cible. Dans une attaque par profilage, un attaquant crée généralement un modèle ou un profil de référence basé sur une analyse approfondie des fuites observées, lors de mesure dites d'entrainement. L'attaquant recueille une quantité significative de mesure physiques du dispositif cible pendant qu'il effectue des opérations cryptographiques. Dans le cas d'un apprentissage pouvant s'effectuer sans la manipulation de la donnée secrète qui doit être retrouvé, l'attaquant peut enregistré les mesures d'entrainement sur un dispositif copie de la cible. Le but de ce dispositif copie est de présenter les même caractéristiques physiques que la cible, mais l'attaquant en a le contrôle. Il peut notamment choisir les valeurs intermédiaires ou simuler des données secrètes manipulées. Il peut également faire des études approfondies sur le comportement de la cible lors de l'exécution de l'algorithme ciblé, la

selection de Point of Interest (PoI) ... Toutes ces informations peuvent servir à renforcer le modèle d'apprentissage et dès lors renforcer l'attaque.

Ce modèle peut par la suite être interrogé lors d'une acquisition d'une nouvelle mesure, appelé mesure d'attaque, qui traite des données inconnues de l'attaquant. Le but du modèle est donc de retrouver ces données inconnues et sensibles.

Les attaques par modèle de canal auxiliaire sont particulièrement puissantes lorsque l'attaquant possède une connaissance approfondie du dispositif cible, y compris de ses caractéristiques physiques. Ces attaques nécessitent cependant souvent une quantité importante de mesures, de calcul et d'analyse.

Les techniques de profilages sont multiples et peuvent par exemple être basées sur des algorithmes d'apprentissage de motifs, tel que des algorithmes de *Machine Learning* ou *Deep Learning*. En exemple, nous retrouvons la LRA (voir sous-sous-section 4.3.1.2) ou la LDA :

4.4.5.1 LDA

L'Analyse Discriminante Linéaire (en anglais *Linear Discriminant Analysis (LDA)*) est une technique utilisée en statistiques et en apprentissage automatique pour réduire la dimension des données et effectuer une classification. Son objectif principal est de trouver une combinaison linéaire des caractéristiques (variables) d'un ensemble de données qui maximise la séparation entre les différentes classes tout en minimisant la dispersion à l'intérieur de chaque classe.

Plus formellement, soient N mesures physiques $(x_i)_{1\leq i\leq N}$ appartement à K classes d'effectifs $(n_k)_{1\leq k\leq K}$, avec $N := \sum_{k=1}^{K} n_k$. Pour chacune des k classes, on calcule le vecteur moyen m_k sur les différentes mesures effectuées et m la moyenne sur tous les échantillons.

$$\begin{aligned} m_k &= \frac{1}{n_k} \sum_{i=1}^{n_k} \mathbf{x}_i^{(k)} \\ \mathbf{m} &= \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{n_k} \mathbf{x}_i^{(k)} \end{aligned} \tag{4.17}$$

où $\mathbf{x}_i^{(k)}$est le i-ème échantillon de la classe k.

On calcul ensuite la matrice de dispersion intra-classe $\mathbf{S}_W$, qui mesure la dispersion des données au sein de chaque classe et la matrice de dispersion inter-classe $\mathbf{S}_B$, qui mesure la dispersion des moyennes de classe.

$$\begin{aligned} \mathbf{S}_W &= \sum_{k=1}^{K} \sum_{i=1}^{n_k} (\mathbf{x}_i^{(k)} - \mathbf{m}_k)(\mathbf{x}_i^{(k)} - \mathbf{m}_k)^T \\ \mathbf{S}_B &= \sum_{k=1}^{K} n_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \end{aligned} \tag{4.18}$$

Enfin, on résout le système :

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w} \tag{4.19}$$

Dans le but de retrouver les valeurs propres λ et les vecteurs propres correspondant $\mathbf{w}$ pour le produit matriciel $\mathbf{S}_W^{-1}\mathbf{S}_B$. Le but est d'extraire les plus grandes valeurs propres pour former une matrice de transformation $\mathbf{W}$.

Quand on ne connait pas les classes de certaines variables, l'objectif de la LDA est de trouver une classification, donc un transformation $\mathbf{W}$ définissant un nouvel espace. On peut alors projeter les données originales sur le nouvel espace en calculant $\mathbf{Y} = \mathbf{XW}$, où $\mathbf{X}$ est la matrice des données originales et $\mathbf{Y}$ est la matrice des données transformées. Le but est que ce changement d'espace maximise la variabilité entre les classes et minimise la variabilité intra-classe. Les matrices de dispersion $\mathbf{S}_W$ et $\mathbf{S}_B$ permettent de quantifier ces variabilités, auxquelles on peut associer un score que l'on tente de minimiser.

La LDA peut aussi être utilisée pour la réduction de dimension, en mettant en évidence les différences entre les classes. Nous n'arborerons pas cet aspect dans ce manuscrit.

4.5 Propagation de croyance

Les Attaques analytiques (ou en anglais *Soft Analytical Side-Channel Attacks (SASCA)*) sont des méthodes puissantes pour effectuer des attaques physiques. Les attaques SASCA sont principalement basés sur la théorie de la Propagation de Croyance (ou *Belief Propagation (BP)*), dont les détails peuvent être trouvés dans [Mac03], chapitre 26. la BP une approche largement utilisée dans le domaine des modèles graphiques probabilistes, en particulier dans les réseaux bayésiens et les chaines de Markov. Elle est basée sur un algorithme de transmission de messages conçu pour calculer les probabilités marginales ou faire des inférences sur les variables aléatoires au sein de ces modèles. Les attaques basées sur la propagation de croyance font partie de la famille des attaques physiques analytiques (ou *Soft Analytical Side-Channel Attacks (SASCA)*) dont je ne détaillerai que la BP ici. Dans ce contexte des SASCA, le graphe peut être alimenté avec des informations de fuite, c'est-à-dire des distributions de probabilités sur certaines valeurs intermédiaires lors du calcul d'un algorithme cible.

La propagation de croyance est appliquée sur un graphe bipartite, appelé graphe de facteurs (ou *Factor Graph*), composé de deux types de nœuds : *(i)* les nœuds de variables utilisés pour stocker les distributions de probabilités des variables intermédiaires de l'algorithme, et *(ii)* les nœuds de facteurs qui représentent les liens arithmétiques entre les nœuds de variables.

Le processus commence par une étape d'initialisation où chaque nœud de variable dans le graphe reçoit une "croyance", ou une distribution de probabilité initiale. Cette croyance initiale peut provenir d'une fuite observée lors d'une mesure physique. Le plus souvent, les attaques par BP sont basées sur des attaques de profilage, qui permet d'obtenir les croyances sur les valeurs intermédiaires. Les nœuds de variables sans connaissance préalable sont initialisés avec une distribution uniforme. Une stratégie plus fine peut être mise en place en calculant, ou en observant empiriquement, la distribution de chaque valeur intermédiaire dans une utilisation classique de l'algorithme cible. Cette distribution de référence, qui peut être différente de la distribution uniforme sur la variable intermédiaire, peut également servir à initialiser le graphe.

Une fois le graphe initialisé, on passe à l'étape de la propagation des croyances. Les nœuds se transmettent des informations, selon Kschischang et al. [KFL01], le message $\mu_{x\to f}$ envoyé du nœud de variable x au nœud de facteur f est défini comme suit :

$$\mu_{x\to f}(x) = \prod_{h\in n(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{4.20}$$

Où $n(x)$ renvoie les voisins de x dans le graphe de facteurs. De plus, les messages envoyés par un facteur f en fonction d'une variable x sont calculés avec la formule du *sum-product* comme suit :

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(X) \prod_{y\in n(f)\setminus\{x\}} \mu_{y\to f}(y) \right) \tag{4.21}$$

où X représente l'ensemble des nœuds de variables connectés à f et $\sim\{x\}$ est une notation abrégée telle que définie dans [KFL01].

Les messages sont transmis de manière itérative entre les nœuds du graphe. L'algorithme s'arrête lorsque la convergence est atteinte : on parle de convergence du graphe lorsque les 'hypothèses classées premières par les distributions de probabilités de chaque valeur intermédiaires ne varient plus. Il a été prouvé que l'algorithme de propagation de croyance est exact sur les graphes en forme d'arbre, assurant une convergence après un nombre fini d'étapes dépendant de la taille du graphe. Cependant, en pratique, les graphes contiennent souvent des cycles. Dans ces cas, il n'existe pas de preuves de convergence et l'algorithme de propagation de croyance peut alors tourner indéfiniment, sans trouver de solution stable. C'est pourquoi

il est nécessaire d'ajouter une seconde condition d'arrêt, un nombre maximal d'itérations autorisé par le graphe. Le nombre maximal d'itérations peut être configurer en fonction de la taille du graphe et la détection de la convergence se fait par le calcule d'un changement statistique maximal de toutes les distributions de variables. Plus formellement, on calcule la distance, le changement, entre deux itérations, si ce changement est inférieur à un seuil donné en paramètre de la BP, alors la propagation s'arrête. En d'autres termes, l'algorithme s'arrête si les distributions de tous les nœuds restent presque constantes entre deux (ou plusieurs) mises à jour.

Finalement, on extrait les marginales de probabilités des variables intermédiaires comme suit :

$$P(x) = \frac{1}{Z} \prod_{f \in n(x)} \mu_{f \to x}(x) \tag{4.22}$$

avec Z étant un facteur de normalisation.

4.5.1 Améliorations de la propagation de croyance

Dans la littérature, il existe plusieurs stratégies permettant d'améliorer les performances de la BP pour certaines attaques.

— Le parcours de graphe (ou *Graph schedule*) permet de choisir dans quel ordre les nœuds du graphe sont mis à jour. Si la structure de l'algorithme est bien connu, les nœuds peuvent être mis à jour dans le même ordre de l'algorithme traite les données. Cette stratégie n'a pas d'influence sur la théorie BP et ne change pas les propriétés de convergence.

— La réduction du nombre de cycle en fusionnant des nœuds facteurs. Dans la description d'un algorithme, des cycles très courts peuvent être présents entre quelques variable intermédiaires. La stratégie introduite par [HHP+21], dans le cadre de la Number Theoric Transform (NTT) de Kyber, vise a regrouper quelques nœuds facteurs ensemble dans un seul nœud afin de ne pas considérer les cycles. Le nœud regroupé crée demande alors une plus grande ressource de calcul, mais ne modifie pas les propriétés de convergence du graphe, si son influence sur les variables intermédiaires connectées est la même que les nœuds initiaux.

D'autres améliorations ont été proposées, notamment dans les travaux visant à attaquer la NTT de Kyber. Ces améliorations n'ont pas de preuves théoriques, elles sont empiriquement efficaces dans certaines attaques mais sorte du cadre théorique fixé par la BP.

— L'amortissement (ou *Damping*) est une stratégie qui modifie la mise à jours des marginales de probabilités des nœuds. au lieu d'utiliser l'équation classique (équation 4.21), les nouvelles marginales sont une

moyenne pondérée entre cette nouvelle marginale et l'ancienne marginale du nœud. Cette moyenne permettrait de réduire les effets d'oscillations lorsque le graphe présente des cycles et ne converge pas.

— l'élagage (ou *Pruning*) est introduit par Assael et al. [AEVR23] pour améliorer le temps d'exécution de l'algorithme. Le but est de tronquer les marginales de probabilités pour ne traiter que les résultats non nuls lors du calcul des mises à jour de messages. La stratégie peut être pousser plus loin en tronquant également les résultats de probabilité plus faible qu'un certain seuil.

4.5.2 Attaques basées sur la propagation de croyance

la BP a été utilisée pour la première fois comme attaque par canal auxiliaire contre la cryptographie par Veyrat-Charvillon et al. [VCGS14] en 2014, ciblant l'implémentation AES Furious. Les auteurs ont décrit une attaque pratique et ont souligné l'efficacité de SASCA par rapport aux meilleures attaques de l'état de l'art à l'époque. SASCA a également été utilisé contre la fonction de hachage cryptographique standardisée Keccak. En 2020, Kannwischer et al. [KPP20] ont décrit une attaque en une seule trace sur Secured Hash Algorithm (SHA)-3. Les auteurs ont mentionné une contre-mesure de masquage booléen pour empêcher l'attaque. Cependant, comme spécifié dans [GS18], les contre-mesures de masquage pourraient permettre de nouvelles attaques.

Enfin, SASCA a également été appliqué à la PQC, notamment contre Kyber, le KEM basé sur des réseaux euclidiens standardisé par le NIST et baptisé Module-Lattices KEM (ML-KEM) par le FIPS 203. Kyber a été la cible de quatre attaques analytiques [PPM17, PP19, HHP$^+$21, HSST23] entre 2017 et 2023. Primas et al. [PPM17] ont introduit la première attaque basée sur BP contre Kyber. Ils ont montré que SASCA pouvait être utilisé contre la cryptographie basée sur des réseaux euclidiens en ciblant la NTT qui est une stratégie d'optimisation de calcul pour la cryptographie basée sur des réseaux. De plus, ils ont ciblé une implémentation protégée par une contremesure de masquage de la NTT, conduisant malgré la protection à récupérer la clé secrète dans un scénario d'attaque réel. Les auteurs ont également réalisé l'attaque en simulation sous le modèle de fuite de poids de Hamming (voir sous-section 4.1.1) et ont obtenu un taux de réussite satisfaisant (supérieur à 0.9) jusqu'à un niveau de bruit $\sigma = 0.4$. Leur évaluation sur un dispositif réel a nécessité la construction d'environ un million de profils d'attaques (voir sous-section 4.4.5). Plus tard, Pessl et Primas [PP19] ont amélioré l'attaque en ne créant que 213 profils d'attaques en faisant le profilage sur le poids de Hamming des valeurs intermédiaires et non plus directement sur leurs valeurs. Ils ont également utilisé des techniques de fusion de nœuds afin de limiter le nombre de cycles, d'amortissement et de parcours de graphe. Les simulations ont montré un bon taux de réussite

jusqu'à $\sigma = 1.5$. En 2021, Hamburg et al. [HHP+21] ont combiné cette attaque SASCA avec une stratégie de chiffrés choisis (ou *Chosen Ciphertext Attack (CCA)*). Cette attaque combinée permet de récupérer la clé secrète jusqu'à un niveau de bruit de $\sigma = 2$ et avec un taux de réussite supérieur à 0.9. Ravi et al. [RPBC20] ont introduit des contre-mesures de mélange, a différents niveau de la NTT. Un premier mélange dit "fin" vient modifier l'ordre des entrées et des sorties de chaque opération intermédiaires de la NTT dit *butterfly* et un second mélange dit "grossier" visant à faire les opérations de chaque étage dans un ordre aléatoire. En 2023, Hemerlink et al. [HSST23] ont analysé ces contre-mesures de mélange et ont évalué leur résistance contre l'attaque de Hamburg et al. Jusqu'à présent, ces contre-mesures de mélange n'ont pas été menacées par d'autres attaques, mais les auteurs soulignent que cette situation pourrait conduire à une "fausse perception de sécurité" et encouragent la prudence. Finalement, en 2023, Assael et al. [AEVR23] étendent l'attaque sur une version optimisée de Kyber.

4.6 Grandes familles de contremesures

Les contremesures sont des techniques utilisées pour protéger les implémentations cryptographiques contre les fuites d'information à travers les canaux auxiliaires. Le but est d'introduire de la variabilité et rendre plus difficile la création de modèles précis par les attaquants. Il existe différents types de contremesures : *(i)* les contremesures physiques, *(ii)* les contremesures architecturales et *(iii)* les contremesures *software*.

Les contremesures physiques et *hardware* sont initialement intégré dans le microcontrôleur cible de l'attaque par canaux auxiliaires. Une contremesure physique regroupent les contremesures qui modifient le comportement physique de la cible. Par exemple, l'ajout d'une cage autour du microcontrôleur afin de bloquer les émanations électromagnétiques, ou d'un composant afin de garantir une consommation de courant constante ou ou contraire, d'un composant de consommation aléatoire afin d'augmenter le niveau de bruit de la cible. Les contremesures *hardware* concernent l'architecture du microcontrôleur. Parmi plus connues on peut citer l'architecture *out-of-order*, qui réalise les opérations assembleur dans un ordre aléatoire ou l'ajout de *dummy operations* durant l'exécution. Ces stratégies visent à empêcher l'attaquant de décider à quel moment de la mesure telle valeur intermédiaire est manipulée et à de-synchroniser les différentes mesures physiques.

Dans ce manuscrit, nous nous intéresserons principalement aux contremesures *software*, qui sont mis en place en changeant l'implémentation du schéma cryptographique que l'on cherche à protéger. j'ai déjà évoqué dans les sections 4.4.2 et 4.4.1 des algorithmes en temps constant ne présentant pas de branchements conditionnel dépendants d'une donnée secrète. Au delà de

ces contremesures, qui aujourd'hui sont considérées comme indispensables face aux attaques par canaux auxiliaires, il existe deux grandes familles : Le masquage (*masking*) et le mélange (*shuffling*).

4.6.1 Mélange

Le mélange (ou *shuffling*) [VCMKS12] est l'une des solutions les plus fréquemment envisagées pour renforcer la sécurité des dispositifs embarqués contre les attaques par canaux auxiliaires. Cette idée a été proposée pour la première fois en 2010 par Clavier et al. [CFG+10] pour contrer une attaque horizontale. Il consiste à réaliser les opérations d'un algorithme dans un ordre aléatoire pour chaque exécution, lorsque cela est possible. Dans le cas d'algorithmes itératif commutatifs, le mélange s'applique particulièrement bien. On peut prendre l'exemple de la NTT de Kyber, pour laquelle plusieurs stratégies de mélange ont été proposées [RPBC20].

Le mélange a pour but de modifier le comportement physique observé par un attaquant réalisant des mesures. L'attaquant ne sera pas capable, à priori, de déterminer quelle partie de la mesure correspond à telle ou telle variable intermédiaire. Ces techniques de mélange peuvent être inversé en testant toutes les permutations aléatoires possibles, ajoutant un cout combinatoire conséquent pour l'attaquant.

4.6.2 Masquage

L'idée fondamentale derrière le masquage est de diviser l'information sensible en plusieurs morceaux (aussi appelés *shares* ou masques). L'utilisation de ces masques aléatoires va introduire des comportements physiques différents d'une utilisation classique de l'algorithme. L'objectif est de garantir que même si un attaquant observe les informations des canaux auxiliaires, il ne peut pas déduire facilement la clé secrète réelle ou d'autres informations sensibles. La combinaison des masques créer une représentation masquée de la valeur intermédiaires, et toutes les opérations sont effectuées sur ces valeurs masquées.

Masquage booléen Le masquage booléen d'une variable sensible z revient à manipuler $z \oplus r$ où r est tiré de manière aléatoire et $\oplus$ est l'opération XOR ou OU exclusif bit à bit. Ce masquage peut être réaliser à l'ordre deux ou à l'ordre supérieur en ajoutant plus de masques. Le masquage à l'ordre d est donné par :

$$z \oplus \bigoplus_{i=1}^{d} r_i \tag{4.23}$$

où chaque r_i est tiré indépendamment de manière aléatoire.

Masquage arithmétique Le masquage arithmétique d'une variable sensible z revient à manipuler $z + r \mod q$ où r est tiré de manière aléatoire et $+$ est l'addition modulo q. Ce masquage peut être réaliser à l'ordre deux ou à l'ordre supérieur en ajoutant plus de masques. Le masquage à l'ordre d est donné par :

$$z + \sum_{i=1}^{d} r_i \mod q \tag{4.24}$$

où chaque r_i est tiré indépendamment de manière aléatoire.

Que ce soit pour le masque booléen ou le masquage arithmétique, le masquage d'ordre supérieur implique l'utilisation de plusieurs masques lors du calcul, le rendant plus résistant aux attaques sophistiquées par canaux auxiliaires

Masquage affine Le masquage affine [FMPR10] utilise une opération supplémentaire, en plus de l'opération XOR $\oplus$ ou de l'addition $+$. Cette opération est une opération multiplicative $\cdot$ sur $\mathbb{F}_{2^n}$. On substitut la manipulation d'une valeur intermédiaire z par :

$$r_0 \cdot z \oplus r_1 \tag{4.25}$$

où r_0 et r_1 sont des valeurs choisies aléatoirement. Le masquage affine englobe le masquage booléen ou arithmétique d'ordre 1 lorsque $r_0 = 1$.

4.7 État de l'art des attaques par canaux auxiliaires contre la cryptologie basée sur les codes correcteurs

Dans cette section, je présente les attaques physiques les plus pertinentes à étudier dans le cadre de mes travaux de thèse. C'est la raison pour laquelle, je me concentre dans un premier temps sur les attaques physiques ayant pour cible HQC. Concernant ces attaques, l'état de l'art que je présente dans la sous-section 4.7.1 est complet au moment ou je l'écris (Janvier 2024). HQC est un des candidats du quatrième tour du concours du NIST pour la cryptographie post-quantique (PQC). Je présente donc également les attaques visant les schémas directement concurrent à HQC dans ce concours. Je donne un état de l'art complet sur les attaques visant BIKE dans la sous-section 4.7.2 et je renvoie le lecteur vers un *survey* complet [DRBL18] pour les attaques contre ClassicMcEliece ; je détaille quelques unes de ces attaques qui me semblaient importante dans mon expérience doctorale. Enfin, je présente quelques attaques visant des structures cryptographiques identifiées, notamment la transformée FO. HQC et BIKE utilisant cette structure pourraient être la cible de telle attaques.

Dans le tableau 4.1, je présente les attaques physiques contre HQC et BIKE avec leurs caractéristiques importantes, avant de les présenter plus en détails dans les sections suivantes.

Schéma	Référence	Horizontale	Verticale	Supervisée	Non-Supervisée	Timing Attack	Power	EM	Caches att.
HQC	[PT19]		✓	✓		✓			
	[WTBB+20]		✓	✓		✓			
	[SRSWZ20]		✓	✓			✓		
	[HLS21]		✓	✓		✓			
	[SHR+22]		✓	✓			✓		
	[GLG22a]		✓	✓				✓	
	[GLG22b]	✓		✓				✓	
	[HSC+23]		✓	✓					✓
	[GMGL23]	✓		✓				✓	
	[SGG24]	✓		✓		✓			
	[BMG+24]	✓	✓	✓				✓	
BIKE	[RHHM17]		✓	✓			✓		
	[SKC+19]		✓	✓			✓		
	[SKC+19]	✓		✓			✓		
	[CARG23]		✓	✓			✓		
FO	[GJN20]		✓	✓		✓			
	[GHJ+22]		✓	✓		✓			

TABLE 4.1 – État de l'art des attaques physiques contre HQC et BIKE.

4.7.1 Attaques contre HQC

Cette section présente l'état de l'art des attaques physiques sur HQC. Mes contributions étant des attaques contre ce même schéma, je passerai plus de temps à détailler cet état de l'art que les attaques physiques contre les autres cryptosystèmes basés sur les codes correcteurs.

HQC a été la cible de plusieurs attaques physiques depuis 2019. La première version de HQC, basée sur les codes BCH, a été attaquée par deux *timing attacks* similaires en 2019 [WTBB+20, PT19], ainsi que par une attaque de texte chiffré choisi [SRSWZ20] par Schamberger et al. en 2020.

Les deux *timing attacks* [WTBB+20, PT19] se basent sur le fait que le décodeurs des codes de BCH n'est pas un décodeur en temps constant. L'attaque exploite une corrélation entre le temps de décodage du décodeur et le poids du mot qui est décodé. Cette corrélation permet d'avoir un Oracle pour prédire les valeurs de la clef secrète utilisée par le schéma lors de l'échange de clef. Comme évoqué dans la sous-section 4.4.2, la contremesure est d'implémenter un algorithme en temps constant. Cet algorithme est proposé dans [WTBB+20]. L'algorithme en temps constant ne donne pas de grosse pénalité sur les performances globales du schéma.

L'attaque par chiffrés choisis [SRSWZ20] introduite par Schamberger et al. repose sur un oracle de décodage capable de distinguer chaque fois que le décodeur BCH corrige une erreur. Cet Oracle permet de retrouver la clef secrète en utilisant environ 10000 traces d'attaques et une réduction algébrique est nécessaire pour finir l'attaque. Pour cet attaque, les auteurs utilise le t-test (voir sous-sous-section 4.3.1.1) et une recherche de points d'intérêt (ou *Point of Interest (PoI)*). Dans ce papier, les auteurs ne proposent pas de contremesures afin d'empêcher leur attaque.

Depuis Octobre 2020, HQC n'est plus construit sur les codes de BCH avec répétition mais sur les codes de Codes de Reed-Muller et Reed Solomon concaténés (RMRS) concaténées. Cette nouvelle construction laisse place à de nouvelles attaques physiques. En 2022, Schamberger et al. [SHR+22] adaptent leur approche sur les codes de BCH [SRSWZ20] pour construire une attaque sur la nouvelle construction de HQC. Ils se basent sur un oracle permettant de savoir si le décodeur des codes RMRS corrige ou non une erreur. Cet oracle permet de retrouver la clef secrète de HQC en 50000 traces de consommation de courant. Nous verrons que cette approche peut-être améliorer en considérant une approche spécifique contre la nouvelle structure de code utilisée. Dans le chapitre 5, je décrirai également une attaque par chiffrés choisis, permettant de retrouver la clef secrète en 20000 traces électromagnétiques. Ces deux attaques on été publiés à PQcrypto 2022.

Dans le chapitre 6, je détaillerai une première attaque par corrélation [GLG22b] permettant de retrouver la clef partagée de HQC en une seule trace d'attaque. Cette attaque a ensuite été améliorée avec des outils de propagation de croyance. Je décris cette nouvelle approche [GMGL23] dans le chapitre 7. Ces deux attaques exploitent la faible probabilité d'échecs de décodage (Decryption Failure Rate (DFR)) en ciblant le code externe de la structure concaténée.

Par la suite, dans le cadre du stage de Chloé Baïsse, que j'ai encadré au CEA, nous avons appliqué les attaques par propagation de croyance sur le code interne de HQC. Nous avons montré que cette attaque permet de retrouver soit la clef secrète, soit la clef partagée en fonction de comment est menée l'attaque. Un papier [BMG+24] décrivant l'attaque est disponible sur eprint depuis début 2024 et est en cours de soumission.

Début 2024, une nouvelle *Timing attack* [SGG24] contre HQC a été publié. Les auteurs montrent que l'opération de division polynomial utilisée dans l'implémentation optimisé de HQC n'est pas en temps constant et permet de mener une attaque afin de retrouver la clef secrète en une centaine de traces.

Pour être complet, HQC a aussi été la cible d'une attaque par cache [HSC+23] en 2023 en utilisant des techniques de *Flush+Reload*. Cette attaque se base elle aussi sur un oracle permettant de déterminer si le décodeur décode un certain chiffré en un certain message. Cette attaque nécessite un peu plus de 50000 chiffrés choisis pour retrouver la clef secrète de HQC. HQC peut également être la cible d'attaque générique [RRCB20, UXT+22, GJN20], exploitant la structure de mécanismes d'échange de clef ou sur la transformation Fujisaki-Okamoto (FO). La génération aléatoire a également été ciblée. En 2021, l'attaque [HLS21] de Hlauschek et al. se base sur le fait qu'une partie des tirages aléatoires est dépendante de la clef secrète et que le tirage aléatoire peut-être rejeté et retiré dans certains cas. Cette dépendance permet de monter une attaque pour retrouver la valeur de la clef secrète de HQC. Cette attaque a mené au changement du tirage aléatoire de HQC, le rendant temps constant, permettant d'éviter des attaques temporelles de ce type.

4.7.2 Attaques contre BIKE

L'attaque de Sim et al. [SKC+19] est une amélioration de l'attaque de Ross [RHHM17]. Ces attaques visent l'implémentation en temps constant des codes QC-MDPC, notamment utilisé dans BIKE, mais également dans d'autres schémas (comme LEDAcrypt). Plus précisément, l'opération visée est l'opération de multiplication lors du calcul du syndrome. Dans le papier est présenté deux attaques, une attaque verticale et une attaque horizontale. Dans cette version en temps constant, un masque a été ajouté pour prévenir les attaques différentielles. Ce masque peut être utilisé pour retrouver de l'information à propos de la clé secrète avec une attaque par canaux auxiliaires. Les attaques se basent sur un modèle de fuite. L'attaque verticale permet de retrouver directement la clef secrète alors que l'attaque horizontale nécessite une phase de calculs algébriques.

La seconde attaque, introduite par Cheriere et al. [CARG23] permet de retrouver la clef secrète de BIKE en exploitant une mesure physique unique. L'attaque cible une implémentation du décodeur des codes QC-MDPC qui manipule la décomposition binaire de la clef secrète. Avec une méthode de *clustering* non-supervisée, les auteurs sont capable de retrouver la clef secrète, à 100% lorsque l'implémentation C est visée, et avec une certaine probabilité lorsque le code assembleur est ciblé. Les auteurs montrent que la structure de l'information retrouvée avec l'attaque par canaux auxiliaires permet d'appliquer un ISD et retrouver l'entièreté de la clef secrète. Une

contremesure spécifique est proposée qui vient masquer l'opération exploitée pour cette attaque.

4.7.3 Quelques attaques contre ClassicMcEliece

Il existe de nombreuses attaques contre le KEM ClassicMcEliece, la plupart de ces attaques peuvent être trouvée dans le *survey* [DRBL18] par Dragoi et al.

Pour ne citer que les principales, Strenzke et al. [STM+08] présentent en 2008 une attaque temporelle contre le crypto-système McEliece. cette attaque exploite le lien entre le poids de l'erreur et le degré du polynôme localisateur d'erreur lors du décodage. L'attaque est a clair choisi sur des entré bien choisi permettant de faire varier progressivement les erreurs au différentes position. L'attaque se base sur un oracle permettant de déduire le degré du polynôme en fonction du temps d'exécution de l'algorithme de décodage ciblé. En 2019, Lahr et al. [LNPS19] adaptent cette attaque au schéma d'échange de clefs ClassicMcEliece. C'est une adaptation permettant de cibler schéma dans sa variante de Nidereiter. C'est une attaque verticale nécessitant plus de 5000 traces pour retrouver le secret. Dans ce papier, les auteurs proposent d'utiliser l'algorithme ISD pour réduire le nombre de traces nécessaires. Le développement d'algorithme de décodage en temps constant est la stratégie pour prévenir ce type d'attaques.

En 2015, Chen et al.[CEVMS15] introduisent une DPA contre le chiffrement de McEliece utilisant les codes QC-MDPC. De nouveau c'est le décodeur des codes de Goppa qui est visé. L'attaque est complété par une phase de calcul algébrique permettant de retrouver l'entièreté de la clef secrète. Pour contrer cette attaque, les auteurs propose une stratégie de mélange, basée sur les travaux de Tillich et al. [TH08] Il s'agirait de mélanger l'ordre dans lequel sont calculé les syndromes des mots du code manipulé dans l'algorithme de décodage. Cette stratégie empêcherait de labelliser efficacement les traces et rendrait l'attaque inopérante.

Une équipe de Grenoble a monté en 2020 une attaque par injection de faute contre ClassicMcEliece. Cayrel et al. [CCD+20] ont présenté a Eurocrypt 2020 une attaque ajoutant des erreurs durant le calcul de syndrome. Avec ces erreurs, le problème SD du décodage de syndrome peut être facilement résolu dans l'arithmétique de $\mathbb{N}$. L'attaque nécessite l'injection d'environ 10000 fautes pour réussir, qui peut être réduit par des stratégies présentées dans le papier. Un point intéressant est que l'attaque est plus facile à réaliser lorsque le niveau de sécurité augmente. C'est un fait intéressant que nous reverrons dans une de mes contributions (voir chapitre 6

4.7.4 Attaques visant la structure des schémas

Guo et al. [GJN20] présente en 2020 une attaque visant spécifiquement la transformation FO. Ils réalisent cette attaque sur FrodoKEM, un ancien candidat dans le concours PQC du NIST. Cette attaque est une attaque temporelle utilisant la structure de la FO. Cette attaque pourrait donc représenter une menace pour d'autres schémas tels que BIKE ou HQC. Mais comme toutes les attaques temporelles, on peut s'en protéger à condition de proposer une implémentation en temps constant. f En 2022, Guo et al. [GHJ+22] proposent une nouvelle attaque générique, exploitant également la structure de la transformation FO et plus précisément la phase de re-chiffrement. Dans l'algorithme de chiffrement (voir Algorithme 7 pour le chiffrement de HQC), le tirage des valeurs aléatoires est effectué avec une graine, afin de garantir les répétabilité des résultats. Les auteurs ont remarqué que cette graine était dépendante d'une valeur de secret. Ils ont alors monté une attaque basé sur un oracle permettant de déterminer une erreur dans le re-chiffrement. En faisant varié la valeur de chiffrés choisis, cet oracle permet de retrouver la valeur de la clef secrète du schéma ciblé. Dans le papier, HQC et BIKE sont évoqués comme cible de cette attaque. Pour HQC, cette attaque a motivé le changement du tirage aléatoire pour une fonction sans rejet et retirage.

Deuxième partie

Contributions

CHAPITRE 5

RETROUVER LA CLEF SECRÈTE DE HQC AVEC DES CHIFFRÉS CHOISIS

Contents

5.1 Introduction

Dans ce chapitre, je décris une nouvelle attaque physique contre HQC. Cette attaque est le résultat d'un travail commun avec mes encadrants, Antoine Loiseau et Philippe Gaborit. L'attaque a été publié à la conférence PQcrypto en 2022 [GLG22a], et le papier peut être retrouver ici : `https://cea.hal.science/cea-03823234/document`.

Cette attaque cible la version RMRS de HQC permetttant de retrouver la clef secrète en utilisant une stratégie basée sur des chiffrés choisis. L'idée principale de cette attaque est de choisir des chiffrés dans le but de créer des collisions avec les bits de la clef secrète. En créant des collisions, le nombre d'erreur corrigé par le décodeur de HQC est modifier, induisant des modifications dans le comportement physique mesuré. Il est possible de détecter ces modifications de comportement et alors de déduire des informations sur la clef secrète. Dans ce chapitre, j'explique comment construire un Oracle simple permettant de déduire le nombre d'erreurs corrigé en entrainant un algorithme de *machine learning*. Pour cela nous utilisons une analyse linéaire discriminante (LDA). Pour la création de l'Oracle, la connaissance de la clef secrète ciblée n'est pas requise, ce qui permet de réaliser la phase d'apprentissage hors ligne, sur un clone du microcontrolleur ciblé. Nous construisons notre Oracle avec 120000 traces d'entrainement et au final, il nous faut 50 traces d'attaques pour retrouver chaque bits de la clef secrète. Au total notre attaque nécessite 20000 traces d'attaque et permet de retrouver entièrement la clef secrète avec une précision de 100%. Cette attaque peut être éviter avec une contremesure, détaillée dans ce chapitre.

5.1.1 Structure du code concaténé de HQC

Cette attaque repose sur la structure des codes concaténés de HQC présentés dans la section 3.5. Un schéma simplifié de cette structure est donné (voir figure 5.1).

Le but de l'attaque est d'attaquer la première étape du décodage, le décodage des codes de Reed-Muller (RM) qui est la première structure à manipuler le vecteur $v - u \cdot y$. Une stratégie de chiffrés choisis (u, v) permettra de donner des informations sur la clef secrète y de HQC.

5.2 Attaques physique et avec chiffrés choisis

Dans cette section, nous présentons une nouvelle attaque pour récupérer la clé secrète $\mathbf{y} \in \mathbb{F}_2^n$. L'opération principale lors de la décapsulation de HQC est le décodage du mot de code erroné $\mathbf{v} - \mathbf{uy}$. On remarque que la connaissance de la partie $\mathbf{y}$ de la clé secrète suffit pour décapsuler. Pour un attaquant, retrouver la valeur de y est donc suffisant pour réussir une attaque de récupération de clef secrète contre HQC.

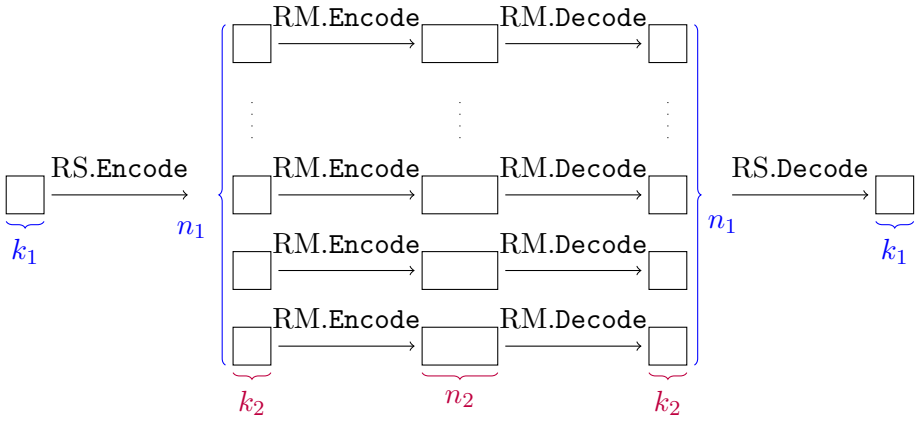


FIGURE 5.1 – Simplified HQC Concatenated RMRS Codes Framework

5.2.1 Modèle d'attaquant

Nous considérons un accès physique à un microcontrolleur pendant la phase de décapsulation de HQC avec une clé secrète statique. Nous supposons que nous pouvons soumettre n'importe quel texte chiffré à ce dispositif. Notre objectif est de récupérer cette clef pour être capable de décapsuler n'importe quel message encapsulé par la clef publique associée. Notre attaque exploite une fuite d'information observée dans les mesures physiques pour créer un Oracle capable distinguer plusieurs motifs de décodage. Les mesures électromagnétiques enregitrées pendant la décapsulation constituent des requêtes que nous pouvons donner à l'Oracle.

Nous réalisons nos mesures sur un microcontrôleur ARM Cortex M4 avec une fréquence d'horloge de 168 MHz. Nous enregistrons les ondes électromagnétiques (EM) émises avec une sonde de champs proche ICR HH 100-6 de LANGER EMV-Technik. Les mesures sont enregistrées avec un oscilloscope ROHDE & SCHWARZ RTO2014 avec une fréquence d'échantillonnage de 750M points par secondes. Pendant les acquisitions, la communication entre le microcontrôleur et l'ordinateur s'effectue via une connexion UART. Pendant les acquisitions, nous avons utilisé une horloge externe dans le but d'atténuer les effets de gigue (*jitter*) entre les traces. La cible exécute l'algorithme de la transformée de Hadamard (*FHT*), présenté dans la sous-sous-section 3.5.3.4, que nous avons au préalable extrait de l'implémentation de référence de HQC [AMAB+23] de juin 2021. Nous configurons une broche GPIO dédiée juste avant la fonction de la transformée d'Hadamard (FHT) pour déclencher l'oscilloscope et le réinitialiser après, nous appellerons trace la mesure électromagnétique obtenue.

5.3 Scénario d'attaque

Nous montrons d'abord comment construire des requêtes qui permettent de récupérer entièrement $\mathbf{y}$. Tout d'abord, remarquons que choisir le message chiffré $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$ implique que l'on décode uniquement la clef secrète $\mathbf{y}$ de HQC. Le vecteur $\mathbf{y}$ est un vecteur creux, c'est-à-dire avec un petit poids de Hamming. Dans le tableau 3.1, la longeur de $\mathbf{y}$ est donnée par n et son poids de Hamming par ω. En conséquence, lors du décodage de $\mathbf{y}$, le décodeur RM manipule presque uniquement des zéros. En fait le poids de Hamming de $\mathbf{y}$ est si petit qu'il est en dessous de la capacité de correction des codes, à ce titre, la clef secrète $\mathbf{y}$ sera décodée en 0, le mot nul étant le plus proche mot du code.

5.3.1 Problème du re-chiffrement

Dans HQC-KEM, le message chiffré est re-chiffré après le déchiffrement afin d'atteindre le niveau de sécurité IND-CCA2 et de garantir que le message chiffré reçu était honnête. Un chiffré choisi, tel que décrit plus haut, ne passera pas cette vérification et sera rejeté par le re-chiffrement. Cette sécurité aurait pu nous empêcher de réaliser notre attaque, cependant, ne pas passer la vérification n'est pas un problème dans notre cas. En effet, au moment de la vérification, la fuite observée dans les mesures physiques se sera déjà produite et aura déjà été enregistrée, indépendamment de la réponse à la vérification du re-chiffrement. Le message chiffré est manipulé par les algorithmes de déchiffrement de HQC, parmi lesquels le décodeur RM, ce qui nous donne toutes les informations nécessaires pour réussir notre attaque.

5.3.2 Oracle sur le décodeur des RM

Selon la figure 5.1, le décodeur RM est appliqué indépendamment sur n_1 blocs de mots de code de taille n_2 (revoir tableau 3.1 pour les paramètres de HQC). Nous pouvons réaliser notre attaque individuellement et indépendamment sur chacun des n_1 blocs et combiner les résultat pour retrouver la clef secrète $\mathbf{y}$. Le comportement de l'Oracle dépend du poids de Hamming du bloc à décoder. En fait, l'Oracle permet de reconnaitre le nombre d'erreurs corrigées lors du décodage d'un bloc. L'idée est de faire varier le nombre d'erreurs corrigées en choisissant judicieusement la valeur de $\mathbf{v}$. Nous montrons une stratégie pour atteindre efficacement ce résultat. Mais dans un premier temps, nous étudions la distribution des coordonnées non nulles de $\mathbf{y}$ sur chacun de ces différents blocs.

5.3.3 Distribution du support de y

Le vecteur $\mathbf{y}$ a une longueur de n bits, qui est le plus petit nombre premier supérieur à $n_1 n_2$. Le nombre premier n est utilisé pour l'espace ambiant afin

de contrecarrer les attaques structurelles. Cependant, seuls les $n_1 n_2$ premiers bits, correspondant à la longueur du code concaténé, sont utilisés pendant le décodage du code, rendant les derniers bits tronqués inutiles. Ainsi, $\mathbf{y} \in \mathbb{F}_2^n$ peut être vu comme la concaténation de deux vecteurs $\mathbf{y} = (\mathbf{y}', \mathbf{y}'')$ avec $\mathbf{y}' \in \mathbb{F}_2^{n_1 n_2}$ et $\mathbf{y}'' \in \mathbb{F}_2^l$. Cette particularité nous empêche de récupérer des informations sur le support $\mathsf{Supp}(\mathbf{y}'')$ de $\mathbf{y}''$. Heureusement, ces bits ne sont pas pertinents pour l'étape de décodage, et les fixer à $\mathbf{0}$ est suffisant pour un décodage réussi. De plus, dans presque tous les cas, $\mathtt{HW}(\mathbf{y}'') = 0$ (voir tableau 5.1), et dans les autres cas, $\mathtt{HW}(\mathbf{y}'')$ est proche de zéro avec une probabilité élevée. La probabilité $\mathbb{P}\left(\mathtt{HW}(\mathbf{y}'') = k\right)$ peut être approximée par :

$$Q_k := \mathbb{P}\left(\mathtt{HW}(\mathbf{y}'') = k\right) \cong \binom{\omega}{k} p^k (1-p)^{\omega - k} \tag{5.1}$$

où $\omega = \mathtt{HW}(\mathbf{y})$ et p est la probabilité de tirer avec remplacement un bit dans $\mathbf{y}'$ qui est égale à $p = \dfrac{|\mathbf{y}'|}{|\mathbf{y}|}$.

λ	Q_0	Q_1	Q_2	$Q_{\geq 2}$
128	$98,15\%$	$1,83\%$	$0,02\%$	$\leq 10^{-3}\%$
192	$96,98\%$	$2,98\%$	$0,05\%$	$\leq 10^{-3}\%$
256	$91,93\%$	$7,73\%$	$0,32\%$	$\leq 10^{-2}\%$

TABLE 5.1 – Distribution de probabilité [1] du support entre $\mathbf{y}'$ et $\mathbf{y}''$ suivant l'équation 5.1

Récupérer $\mathbf{y}'$ suffit pour considérer l'attaque comme réussie. Néanmoins, plusieurs stratégies existent pour déduire les derniers l bits, permettant une récupération complète de $\mathbf{y}$. Cette distribution de faible poids de Hamming permet de mener une recherche exhaustive pour les derniers $l = n - n_1 n_2$ bits de $\mathbf{y}$. D'autres part, Schamberger et al. [SRSWZ20] proposent l'utilisation de l'ISD (voir sous-section 3.2.1) dans la Section 3.3 de leur article pour récupérer $\mathsf{Supp}(\mathbf{y}'')$ connaissant $\mathsf{Supp}(\mathbf{y}')$ en temps polynomial.

5.3.4 Distribution du support de $\mathbf{y}'$

Le vecteur $\mathbf{y}'$ vit dans $\mathbb{F}_2^{n_1 n_2}$ et le décodeur externe manipule le mot de code par bloc de taille n_1, nous pouvons réécrire :

$$\mathbf{y}' = \left(\mathbf{y}'_0, \mathbf{y}'_1, \cdots, \mathbf{y}'_{n_1 - 1}\right) \text{ et pour tout } i,\ \mathbf{y}'_i \in \mathbb{F}_2^{n_2} \tag{5.2}$$

Pour notre attaque, le pire cas se produit lorsque $\mathbf{y}'$ a un poids complet, c'est-à-dire $\mathtt{HW}(\mathbf{y}') = \mathtt{HW}(\mathbf{y})$, ce qui se produit lorsque $\mathtt{HW}(\mathbf{y}'') = 0$ avec une

1. Pour chaque ligne, la somme n'est pas égale à un en raison de l'approximation choisie.

probabilité élevée (voir tableau 5.1). En effet, ce cas augmente la probabilité d'avoir des blocs avec un poids de Hamming élevé. Plus tard, nous verrons que notre Oracle ne peut distinguer que décodage ou le vecteur décodé à un poids de Hamming plus petit qu'un certain seuil κ. Étant donné que le vecteur $\mathbf{y}$ est presque toujours de poids maximal, nous ne considèrerons que ce cas là dans la suite de l'analyse.

Le décodeur RM manipule chaque bloc $\mathbf{y}'_i$ indépendamment, nous calculons la probabilité P_k qu'un bloc tiré au hasard $\mathbf{y}'_i$ ait un poids de Hamming de k (voir l'équation 5.3 et le tableau 5.2).

$$P_k := \mathbb{P}\left(\mathtt{HW}(\mathbf{y}'_i) = k \;\middle|\; \mathbf{y} \xleftarrow{\$} \mathcal{R}_\omega,\ i \xleftarrow{\$} [\![0, n_1 - 1]\!]\right) \tag{5.3}$$

λ	P_0	P_1	P_2	P_3	P_4	$P_{\geq 5}$
128	$23,44\%$	$34,38\%$	$24,83\%$	$11,77\%$	$4,12\%$	$1,45\%$
192	$16,50\%$	$30,00\%$	$27,00\%$	$16,04\%$	$7,07\%$	3.40%
256	$23,14\%$	$34,06\%$	$24,87\%$	$12,02\%$	$4,32\%$	$1,59\%$

TABLE 5.2 – Distribution de probabilité [2] du support sur les blocs de $\mathbf{y}'$.

D'après la tableau 5.2, nous observons que les blocs de faible poids de Hamming sont principalement représentés. De plus, il est relativement rare de rencontrer des blocs de poids supérieur ou égal à 5. Pour la suite, en approximation, nous considérons tous les blocs de poids $\kappa = 5$ ou plus dans la même classe.

5.3.5 Erreur de grande magnitude

Expand and Sum est la première fonction de décodage et réalise une addition classique sur $\mathbb{N}$. Ainsi, le mot de code étendu se situe dans $[\![0, \mathsf{mul}]\!]$. Appliqué à $\mathbf{y}$, dans la plupart des cas, le résultat est dans $[\![0, 1]\!]$, mais il arrive que deux erreurs partagent le même emplacement modulo 128 dans un bloc $(\mathbf{y}'_i)$. Cela donne une erreur de magnitude 2, noté HME. Ces erreurs induisent un comportement légèrement différent de la FHT au sein de la même classe, affectant le comportement de notre Oracle. Heureusement, ces erreurs de plus grande magnitude se produisent avec une faible probabilité. L'équation (5.4) donne la probabilité d'avoir une erreur de magnitude au

2. Pour chaque ligne, la somme n'est pas égale à un en raison de l'approximation choisie.

moins 2.

$$\begin{aligned}\mathbb{P}\left(\text{HME}\right) &= \sum_{k=0}^{n_2} \mathbb{P}\left(\mathtt{HW}(\mathbf{y}'_i) = k\right) \times \mathbb{P}\left(\text{HME} \mid \mathtt{HW}(\mathbf{y}'_i) = k\right) \\ &= \sum_{k=0}^{n_2} P_k \times \left(1 - \mathbb{P}\left(\overline{\text{HME}} \mid \mathtt{HW}(\mathbf{y}'_i) = k\right)\right) \\ &= \sum_{k=0}^{n_2} P_k \times \left(1 - \prod_{i=0}^{k} \frac{n_2 - (\mathsf{mul} - 1) \times i}{n_2}\right)\end{aligned} \tag{5.4}$$

Une erreur de grande magnitude se produit dans le vecteur $\mathbf{y}'$ avec des probabilités de $0,53\%$, $0,97\%$ et $0,65\%$ respectivement pour HQC-128, HQC-192 et HQC-256.

5.4 Attaque par chiffrés choisis

5.4.1 Description de l'Oracle

Nous utilisons un Oracle sur le décodage RM $\mathcal{O}_{i,b}^{\text{RM}}$ qui prend en entrée un texte chiffré HQC $(\mathbf{u}, \mathbf{v})$. Notre Oracle $\mathcal{O}_{i,b}^{\text{RM}}$ est capable de déterminer le nombre d'erreurs corrigées par le décodeur RM dans le i-ème bloc $\mathbf{y}'_i$ pour $i \in [\![0, n_1 - 1]\!]$. Notre oracle fonctionne dans une plage donnée et détermine correctement le nombre d'erreurs corrigées s'il n'excède pas le seuil donné $\kappa = 5$. L'Oracle peut être interrogé pour différentes entrées et renvoie $b \in [\![0, \kappa]\!]$. Notez que dans le cas du décodage de $\mathbf{y}$, le nombre d'erreurs décodées dans un bloc $\mathbf{y}'_i$ est presque toujours le poids de Hamming $\mathtt{HW}(\mathbf{y}'_i)$, aux erreurs de grande magnitude près (voir sous-section 5.3.5). Nous décrivons comment construire cet Oracle $\mathcal{O}_{i,b}^{RM}$ dans la section 5.5, à partir de fuites physiques observées.

5.4.2 Description de l'attaque

Concentrons-nous sur un seul bloc choisi $\mathbf{y}'_j$, l'attaque est identique pour les autres blocs de $\mathbf{y}$. Dans une première étape, l'Oracle $\mathcal{O}_{j,b}^{RM}$ est interrogé pour connaitre le nombre d'erreurs à corriger dans $\mathbf{y}'_j$, ce qui donne une valeur de référence pour les étapes suivantes. Deuxièmement, l'idée principale de l'attaque est de sélectionner de manière récursive $\mathbf{v}_j$ de poids de Hamming 1 afin de trouver une collision avec le support de $\mathbf{y}'_j$. Trouver une collision implique de modifier le nombre d'erreurs décodées par rapport à la valeur de référence et de récupérer ainsi une information sur le support de la clef $\mathsf{Supp}(\mathbf{y}'_j)$. En fait, pour un $\mathbf{v}_j$ de poids de Hamming 1 choisi, il existe deux cas que nous pouvons distinguer avec l'Oracle :

1. $\mathsf{Supp}(\mathbf{y}'_j) \cap \mathsf{Supp}(\mathbf{v}_j) = \mathsf{Supp}(\mathbf{v}_j)$. Alors $\mathtt{HW}(\mathbf{v}_j - \mathbf{y}'_j) = \mathtt{HW}(\mathbf{y}'_j) - 1$, le décodeur corrigera une erreur de moins que le décodage de référence de $\mathbf{y}'_j$.
$$\mathcal{O}^{\mathrm{RM}}_{j,b}(\mathbf{v} - \mathbf{y}) = O^{\mathrm{RM}}_{j,b}(\mathbf{y}) - 1$$
2. $\mathsf{Supp}(\mathbf{y}'_j) \cap \mathsf{Supp}(\mathbf{v}_j) = \varnothing$. Alors $\mathtt{HW}(\mathbf{v}_j - \mathbf{y}_j) = \mathtt{HW}(\mathbf{y}_j) + 1$, le décodeur corrigera une erreur de plus que le décodage de référence de $\mathbf{y}$.
$$\mathcal{O}^{\mathrm{RM}}_{j,b}(\mathbf{v} - \mathbf{y}) = O^{\mathrm{RM}}_{j,b}(\mathbf{y}) + 1$$

Ces observations permettent de déterminer le support de $\mathbf{y}'_j$ en choisissant $\mathbf{v}_j$ successivement égal à tous les vecteurs de poids de Hamming 1. En se rappelant des emplacements pour lesquels l'Oracle produit 1 de moins que la valeur de référence $O^{\mathrm{RM}}_{j,b}(\mathbf{y})$, nous sommes capables de déterminer l'ensemble complet du support $\mathsf{Supp}(\mathbf{y}'_j)$. Appliquer cette stratégie à tous les blocs de $\mathbf{y}'$ permet finalement de récupérer l'ensemble complet du support $\mathsf{Supp}(\mathbf{y}')$, et donc la clef secrète $\mathbf{y}$.

5.4.3 Diviser pour Régner

Étant donné que l'on doit sélectionner, comme chiffré choisi, tous les vecteurs de poids de Hamming 1, l'attaque nécessite autant de requêtes que le nombre de bits dans la clef $\mathbf{y}'$. Autrement dit, il faut à minima $n_1 n_2$ requêtes afin de tester tous les éléments de poids de Hamming 1. Cependant, étant donné que le décodage des blocs RM est indépendant, l'attaque peut être réalisée en parallèle sur chaque bloc. Nous interrogeons simultanément les n_1 oracles $\mathcal{O}^{\mathrm{RM}}_{i,b}$, ce qui conduit à une seule requête. Ainsi, le nombre minimal de requêtes nécessaires pour récupérer $\mathbf{y}'$ est réduit au nombre de bits dans un seul bloc, soit n_2 bits. En conséquence, cibler HQC-128 (resp. HQC-192 et HQC-256) nécessite au minimum 384 requêtes (resp. 640). Pour connaitre le nombre total de traces d'attaque nécessaires, cette valeur est multipliée par le nombre de traces d'attaque nécessaires pour déterminer, avec une bonne probabilité, un seul bit de la clef. Ce nombre de traces d'attaque nécessaire est décrit dans la sous-section 5.6.1.

5.5 Attaque Pratique

Dans cette section, nous construisons un Oracle de décodage RM qui permet d'identifier le nombre d'erreurs décodées. Cet Oracle est construit à partir de fuites physiques et permet de récupérer la clé secrète $\mathbf{y}'$, comme expliqué dans la section 5.2. Nous nous plaçons dans le contexte d'attaque décrit dans la sous-section 5.2.1. Ensuite, nous décrivons notre Oracle et effectuons une évaluation des fuites avec le test de Welch t. Enfin, nous évaluons la robustesse de notre Oracle avec un nombre différent de traces d'entrainement et donnons le cout et les performances de l'attaque pratique.

5.5.1 Construction de l'Oracle

Nous construisons l'Oracle selon les 6 classes principales identifiées avec la distribution de probabilités du support (voir tableau 5.2). Chacune de ces classes correspond à un poids de Hamming différent pour $\mathbf{y}'_i$, un bloc de $\mathbf{y}$. Chaque élément de la classe k est créé en tirant aléatoire un vecteur de poids de Hamming k. Ensuite, parmi ces classes, la proportion d'erreur de haute magnitude est la même que dans une instance HQC, suivant l'Équation (5.4). La génération aléatoire est fournie par le générateur aléatoire du microcontrôleur. Nous avons acquis un ensemble de 10.000 traces par classe utilisé pour évaluer l'Oracle. Ces acquisitions ont été effectuées dans un ordre aléatoire.

5.5.2 Évaluation des Fuites

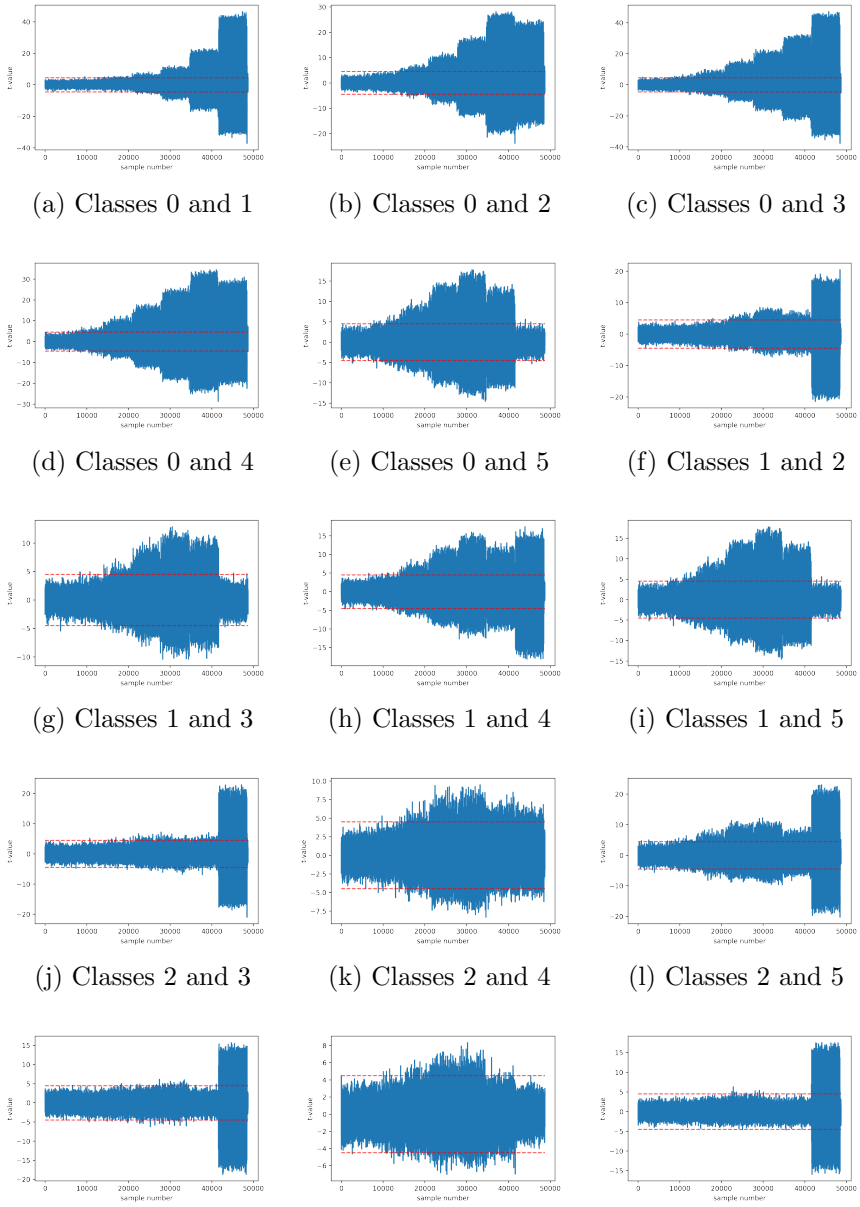
Nous utilisons un test t de Welch (voir sous-sous-section 4.3.1.1) pour mener une évaluation des fuites pour l'Oracle. La valeur t entre deux ensembles S_0 et S_1 avec leur cardinalité respective n_0 et n_1, moyenne de μ_0 et μ_1 et variance de σ_0 et σ_1 est calculée avec la formule de l'Équation (5.5). Généralement, un seuil $|t| = 4.5$ est défini, admettant une différence statistiquement significative avec un haut degré de confiance lorsque ce seuil est dépassé.

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\left(\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}\right)}} \tag{5.5}$$

Nous calculons les valeurs de t pour chaque paire de classes afin de caractériser la distinction entre elles. Les résultats sont présentés dans la figure 5.2. Pour chaque sous-figure, nous observons les 7 occurrences de la boucle *for* parcourues pendant l'éxecution de la transformée d'Hadamard (FHT) (voir l'Algorithme 14). Ce test indique un bon niveau de distinguabilité, ce qui pourrait permettre de construire un classificateur.

5.5.3 Régions d'Intérêt

En matière de sélection des Points d'Intérêt (PoI), l'évaluation des fuites nous permet de choisir des Régions d'Intérêt (ou Area of Interest (AoI)). Cette sélection permet de ne conserver que les parties pertinentes des traces et de réduire le temps de calcul de notre classifieur. Comme on peut le voir, parmi les sept occurrences de la boucle for, la dernière (points $41,000$ à $48,000$ environ) semble très informative (voir Figures 5.2a, 5.2c, 5.2f, 5.2h, 5.2j, 5.2l, 5.2m et 5.2o). Cette partie des traces est la première Area of Interest (AoI) considérée pour des études ultérieures.



(a) Classes 0 and 1 (b) Classes 0 and 2 (c) Classes 0 and 3

(d) Classes 0 and 4 (e) Classes 0 and 5 (f) Classes 1 and 2

(g) Classes 1 and 3 (h) Classes 1 and 4 (i) Classes 1 and 5

(j) Classes 2 and 3 (k) Classes 2 and 4 (l) Classes 2 and 5

(m) Classes 3 and 4 (n) Classes 3 and 5 (o) Classes 4 and 5

FIGURE 5.2 – Résultats du test t de Welch entre chaque classe en utilisant $10,000$ traces d'entrées tirées de manière aléatoire dans la classe. La trace correspond à la transformée d'Hadamard (FHT). Des lignes pointillées rouges à $-4,5$ et $4,5$ sont tracées.

Cependant, cette AoI ne semble pas suffisante pour construire un classificateur complet, en effet, certains résultats ne montrent aucune fuite dans la dernière occurrence (voir Figures 5.2e, 5.2g, 5.2i et 5.2n). En pratique, utiliser uniquement des segments de trace dans cette AoI n'est pas suffisant pour monter une attaque avec une précision suffisante. Pour combler cette lacune d'information, nous utilisons également une autre AoI. Afin de créer un classificateur complet entre toutes les classes, nous extrayons également une AoI de la cinquième occurrence de la boucle *for* (points $27,300$ à $34,200$ environ). Cette deuxième AoI permet de distinguer entre des cas non couverts par la première (voir 5.2e, 5.2g, 5.2i, 5.2k et 5.2n). Pour ces deux AoI, nous ne conservons que les 1000 premiers points, ce qui réduit significativement le temps de calcul et les besoins en mémoire.

5.6 Résultats

Nous construisons un classifieur avec une Linear Discriminant Analysis (LDA) (voir sous-sous-section 4.4.5.1), qui est un classificateur linéaire. Nous utilisons la version LDA de la bibliothèque python scikit-learn [PVG+11]. Nous effectuons plusieurs fois l'attaque avec un nombre différent de traces d'entrainement, 1000, 2500, 5000, 10000, 20000 ou 40000 traces par classe. Les traces sont découpées en fonction de la région d'intérêt (AoI) identifiée dans la sous-section 5.5.3 et réparties uniformément entre les classes cibles.

Théoriquement, avec un Oracle parfait, récupérer un seul bit dans un bloc donné ne nécessite qu'une seule trace. Pratiquement, nous constatons rapidement qu'une seule trace d'attaque n'est pas suffisante pour obtenir un taux de réussite suffisant (voir figure 5.3).

Nous construisons une attaque avec s traces afin d'augmenter le taux de réussite. Chaque trace est traitée indépendamment par l'Oracle et nous cumulons les sorties des s requêtes avec une technique de *soft-max*. Étant donné $(p_1, \cdots, p_\tau) = \sum_{i=1}^{s}(p_{1,i}, \cdots, p_{\tau,i})$ la somme des probabilités renvoyées par les s instances de l'attaque pour chacune des τ classes. La sortie de l'Oracle est donnée par $b = \mathsf{argmax}(p_1, \cdots, p_\tau)$. Nous réalisons l'attaque plusieurs fois avec s de 0 à 60, nous représentons dans la figure 5.3 les résultats de ces tests.

5.6.1 Taux de Réussite de l'Attaque et Coût

Les expériences de la figure 5.3 montrent que $s = 50$ traces d'attaque sont suffisantes pour atteindre un taux de réussite parfait sur un bit avec un ensemble d'entrainement de $40,000$ traces par classe. Nous construisons l'attaque sur notre ensemble de mesures avec $40,000$ traces d'entrainement par classe, soit $240,000$ traces d'entrainement au total. Avec cet ensemble de paramètres et notre configuration spécifique d'entrainement et d'attaque,

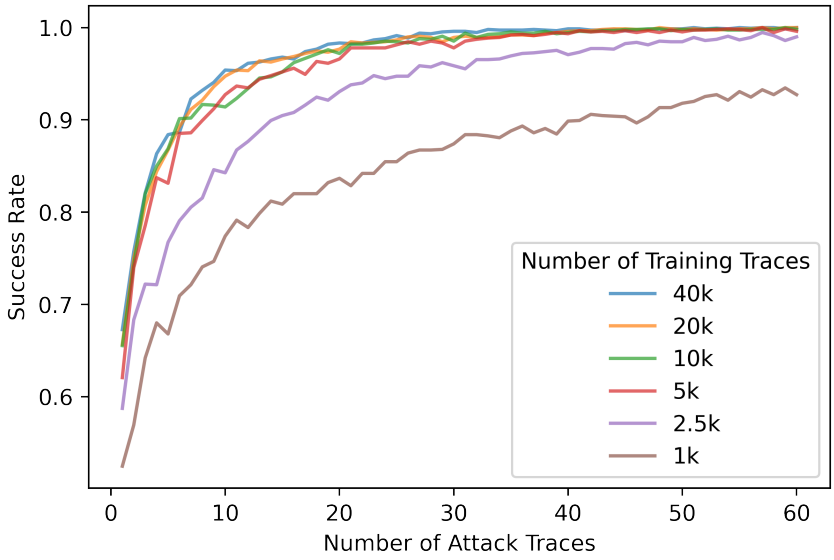


FIGURE 5.3 – Taux de réussite de récupération d'un seul bit en fonction du nombre de traces d'attaque s. Comparaison du taux de réussite entre les attaques avec un nombre différent de traces d'entraînement par classe.

nous sommes capables de récupérer tous les bits de $\mathbf{y}'$ avec $s \times n_2 = 50 \times 384 = 19,200$ traces d'attaque pour HQC-128 (voir sous-section 5.4.3).

5.7 Contremesure

Une contremesure directe contre un classificateur de décodage RM est l'utilisation d'un masque. L'idée est de cacher les données sensibles en divisant ses connaissances en plusieurs *shares*. En effet, si l'entrée $\mathbf{c}$ de la FHT satisfait l'équation 5.6,

$$\mathbf{c} = \sum_{i=0}^{n} \mathbf{c}_i \tag{5.6}$$

par linéarité de la transformée d'Hadamard (FHT), le résultat est donné par l'équation 5.7.

$$\mathsf{FHT}(\mathbf{c}) = \sum_{i=0}^{n} \mathsf{FHT}(\mathbf{c}_i) \tag{5.7}$$

Pour créer un schéma de masquage sécurisé, les $n-1$ premiers $\mathbf{c}_i$ doivent être tirés aléatoirement et de manière uniforme et le dernier élément $\mathbf{c}_n$ est choisi pour satisfaire la relation de l'équation 5.6. La contremesure de masquage d'ordre n nécessite de calculer $n+1$ fois la transformation d'Hadamard

FHT. L'algorithme de la transformée d'Hadamard (voir Algorithme 14), peut être masqué à l'ordre n. Nous donnons l'algorithme masqué à l'ordre 1 avec l' Algorithme 19.

5.7.1 Évaluation de la Contremesure

Un attaquant qui souhaiterait cibler la version masquée de la transformée d'Hadamard (FHT) devrait viser les n *shares* afin de récupérer l'ensemble de l'information. Notre scénario d'attaque ne peut pas être appliqué directement à ces *shares* étant donné que les mots de code étendus $\mathbf{c}_i$ sont tirés de manière aléatoire. Cela implique que les *shares* ne respectent pas les restrictions de poids de Hamming imposées par notre Oracle. Sans cette conditions, si les poids de Hamming des blocs manipulés par le décodeurs sont aléatoires, nous ne sommes pas capable de construire un classifieur efficace.

Malgré cela, nous évaluons la robustesse de la contremesure en répétant l'expérience comme dans la section 5.5. Nous calculons la transformée d'Hadamard (FHT) avec le masque de premier ordre (voir Algorithme 19) sur $60,000$ mots de code étendus répartis uniformément entre les classes. Ensuite, nous calculons les valeurs de t pour chaque paire de classe, ce qui donne les résultats présentés dans la figure 5.4. Nous supposons que la réduction significative du nombre de différences statistiques observées avec le test t de Welch garantit que la contremesure est efficace contre notre attaque.

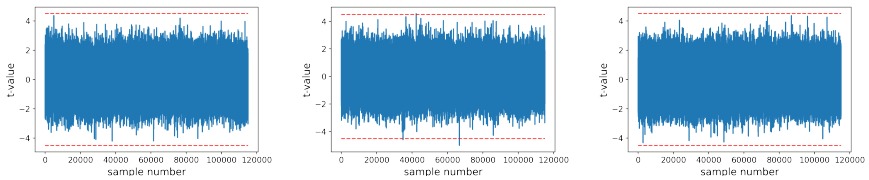
Algorithm 19 Transformée d'Hadamard FHT masquée au premier ordre.

Input : expanded codeword $\mathbf{c}$ and the multiplicity mul.
Output : expanded codeword transformed structure $\mathbf{c}$

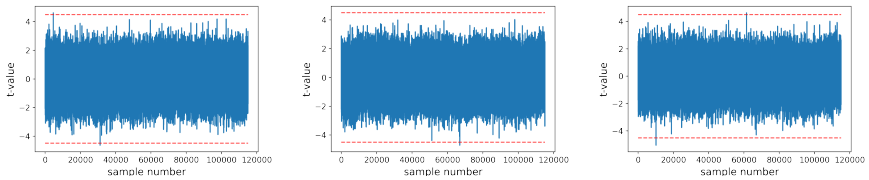
1: $\mathbf{c}_0 \xleftarrow{\$}$ expanded codeword
2: $\mathbf{c}_1 = c - c_0$
3: $\mathbf{c}_0 = \mathsf{FHT}(\mathbf{c}_0)$
4: $\mathbf{c}_1 = \mathsf{FHT}(\mathbf{c}_1)$
5: $\mathbf{c} = \mathbf{c}_0 + \mathbf{c}_1$
6: **return** $\mathbf{c}$

5.8 Conclusion et Travaux Futurs

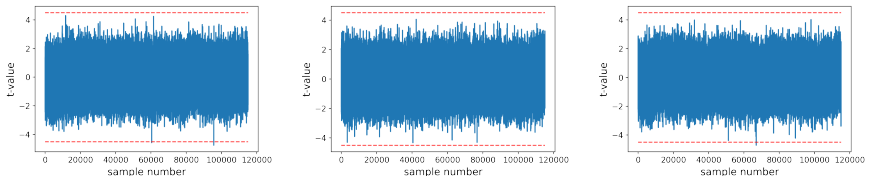
Dans ce chapitre, nous avons présentés une nouvelle attaque physique sur la version RMRS de HQC qui vise à récupérer une clef secrète statique. Nous montrons qu'en choisissant un certain texte chiffré, c'est la clef secrète seule qui est manipulée par l'algorithme de décodage des RM. Nous construisons une attaque par chiffrés choisis à partir de ce constat en modifiant légèrement le texte chiffré afin de trouver des collisions avec la clef secrète. Finalement, nous montrons une stratégie, une séquence de requêtes,



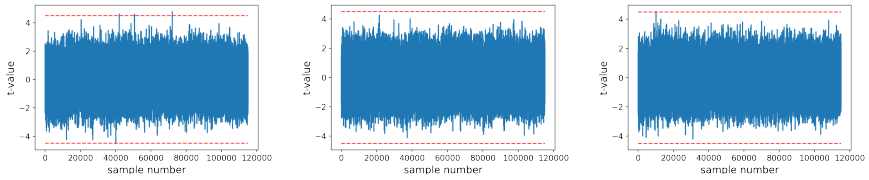
(a) Classes 0 and 1 (b) Classes 0 and 2 (c) Classes 0 and 3



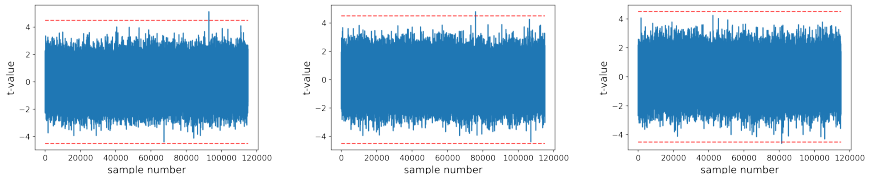
(d) Classes 0 and 4 (e) Classes 0 and 5 (f) Classes 1 and 2



(g) Classes 1 and 3 (h) Classes 1 and 4 (i) Classes 1 and 5



(j) Classes 2 and 3 (k) Classes 2 and 4 (l) Classes 2 and 5



(m) Classes 3 and 4 (n) Classes 3 and 5 (o) Classes 4 and 5

FIGURE 5.4 – Résultats du test t de Welch entre chaque classe en utilisant $10,000$ traces d'entrées tirées de manière aléatoire dans la classe. La trace correspond à la transformation d'Hadamard (FHT) avec la contre-mesure. Les lignes pointillées rouges à $-4,5$ et $4,5$ sont tracées montrant le seuil d'acceptation du t-test.

qui permet de trouver des collisions et de récupérer ensuite l'ensemble de la clef secrète.

Notre attaque repose sur un Oracle capable de distinguer plusieurs motifs de décodage à partir de mesures physiques du décodage. Nous évaluons notre Oracle avec des mesures d'ondes électromagnétiques de notre microcontrôleur Cortex M4 et montrons qu'il est facile à construire et fiable. En effet, les meilleures performances de notre Oracle ont été observées avec un compromis de $40,000$ traces d'entrainement par classe et 50 traces d'attaque pour récupérer un bit sur un bloc avec un taux de réussite de 1 sur notre ensemble de mesures de test. L'indépendance du décodage entre les différents blocs permet de paralléliser l'attaque et de récupérer tous les bits de la clé secrète avec $19,200$ traces d'attaque.

Notre attaque constitue une menace pour la sécurité de HQC et assoie la nécessité d'une contremesure efficace pour atténuer de telles attaques. Nous proposons une contremesure simple basée sur le masquage afin de contrecarrer notre attaque, doublant le temps d'exécution de la fonction cible.

En perspective, le nombre de traces d'attaque pourrait être réduit en trouvant une séquence de requêtes plus efficace. À cette fin, d'autres fonctions de HQC pourraient jouer le rôle de distingueur, permettant d'améliorer les performances ou même de construire de nouvelles attaques.

CHAPITRE 6

RETROUVER LA CLEF PARTAGÉE DE HQC PAR CORRÉLATION

Contents

6.1 Introduction

Dans cette chapitre, je présente une attaque qui a été découverte et menée durant ma première année de thèse. Le papier qui en a découlé [GLG22b] est un travail commun avec mes encadrants de thèse, Antoine Loiseau et Philippe Gaborit. Il a été publié à International Workshop on Coding and Cryptography (WCC) 2022 et est disponible en accès libre en suivant ce lien : `https://cea.hal.science/cea-04176652v1/document`.

Il s'agit d'une attaque par canaux auxiliaires horizontale par corrélation sur HQC dans le but de retrouver la clef partagée à la fin du KEM. Le but de cette attaque est de retrouver la clef partagée en une mesure physique unique. Cette restriction sur le nombre de trace provient du constat qu'une clef donnée n'est échangée qu'une seule fois. Si l'échange de clef échoue, et qu'un rejeu est nécessaire, la clef partagée est modifiée par le processus aléatoire du cryptosystème. Envisager un scénario d'attaque permettant l'utilisation de plusieurs mesures physique pour la même clef partagé serait donc aberrant dans ce contexte.

Cette attaque cible le décodage des codes de Reed-Solomon (RS) impliqués dans HQC comme code externe de la structure concaténé. Basé sur la faible probabilité d'échec du décodeur, il advient que le calcul du syndrome de RS manipule de multiple fois des données sensibles non-masquées. Cette répétition dans la manipulation permet d'extraire de l'information au travers de mesures physiques. Nous exploitons la structure des codes correcteurs pour décoder la sortie de notre attaque et corriger des erreurs dues à l'attaque.

6.1.1 Contexte

Low DFR Comme mentionnée dans la section 3.5, plus précisément dans la sous-section 3.5.2, la probabilité que le décodage du code interne, i.e. le code de RM, échoue est très faible. Dans la majorité des cas ($> 99.9\%$), le décodeur externe de RS manipule un mot du code non erroné. Après ce décodage, le message $\mathbf{m}$ est retrouvé et est utilisé de manière déterministe pour déduire la clef partagée. Retrouver le mot du code intermédiaire, décodé par le code RS, permet donc de retrouver la clef partagée du KEM.

Nous nous intéressons donc particulièrement à la manière dont ce mot du code est décodé. Le décodeur RS utilisé dans HQC suit la stratégie de décodage polynomiale de [JH04], comme évoqué dans la section 2.3. La première opération est le calcul du syndrome (voir Algorithme 20), effectuée avec la connaissance de la matrice de parité $\mathbf{H} = (h_{i,j})_{\substack{1 \leq i \leq n-k \\ 1 \leq j \leq n}}$, qui pour rappel, est connue publiquement. Comme le mot du code est sans erreur, le syndrome obtenu est le syndrome nul. Par la suite, des zéros seront propagés dans tout l'algorithme de décodage, le calcul du syndrome est donc la seule opportunité de retrouver des informations sur le mot du code in-

termédiaire. Nous allons nous intéresser plus particulièrement au calcul du syndrome. Dans l'implémentation de référence de HQC, cette opération s'effectue comme suit (voir Algorithme 20) :

Algorithm 20 Calcul de syndrome de RS de l'implémentation de référence de HQC [AMAB+23]

Require: parameters : k, n the dimension and length of the code
Require: parity check matric $\mathbf{H} \in \mathbb{F}_q^{(n-k,n)}$
Require: codeword $\mathbf{c} \in \mathbb{F}_q^n$
Ensure: $\mathbf{s} := \mathbf{H}^T \cdot \mathbf{c}$ the syndrome of $\mathbf{c}$
1: Initialize $\mathbf{s}$ to $\mathbf{c}[1]^{n-k}$
2: **for** i from 1 to $n-k$ **do**
3: **for** j from 2 to n **do**
4: $\mathbf{s}[i] = \mathbf{s}[i] \oplus \texttt{gf\_mul}(\mathbf{c}[j], \mathbf{H}[i, j-1])$ ▷ `gf_mul` : Galois field multiplication
5: **return s**

Un constat rapide est que en ligne 4 de l'algorithme, chaque octet du mot du code est manipulé plusieurs fois et indépendamment des autres. Chaque octet du mot du code est manipulé avec chacun des éléments d'une colonne de la matrice de parité $\mathbf{H}$. Cette particularité permet d'avoir des variations dans les mesures physiques observées (chaque octet ciblé n'est jamais manipulé de la même manière) mais cette variation est contrôlée car on sait *(i)* la valeur des éléments de la matrice de parité $\mathbf{H}$ qui sont publiquement connus et *(ii)* l'ordre dans lequel ces éléments sont manipulés car l'algorithme est déterministe. Ces constats amène fortement à penser qu'une attaque physique est possible contre cet algorithme, si l'on est capable d'exploiter des fuites provenant de la ligne 4 de l'algorithme. Observons toutefois que le premier octet du mot du code, $C[1]$ ou C_1, n'est pas manipulé avec l'opération en ligne 4 mais sers à l'initialisation du syndrome en ligne 1. Si l'attaque que l'on mène se focalise sur la ligne 4, on ne sera donc pas en mesure de récupérer la valeur de C_1 avec cette attaque. Il faudra donc trouver une manière de conclure l'attaque afin de retrouver aussi la valeur de cet octet particulier.

A la ligne 4, c'est l'opération `gf_mul` qui est utilisée, représentant la manipulation dans le corps de Galois $\mathbb{F}_{2^8}$. Nous nous intéressons maintenant donc plus particulièrement à de cette fonction dans l'implémentation de référence de HQC.

6.1.2 Multiplication dans le corps de Galois de HQC (Juin 2021)

En juin 2021, l'implémentation de la multiplication dans le corps de Galois `gf_mul` est basée sur une stratégie de conversion des éléments dans

leur représentation logarithmique. Soit $\mathbb{F}_q$ un corps de Galois et g un élément générateur de ce corps, alors pour chaque élément $x \in \mathbb{F}_q$ du corps de Galois, il existe un unique élément noté $\texttt{log}(x)$ entre 0 et $q-1$ tel que $x = g^{\texttt{log}(x)}$. Ainsi la multiplication dans le corps de Galois peut s'écrire :

$$a \times b = y^{\texttt{log}(a)} \times y^{\texttt{log}(b)} = y^{\texttt{log}(a)+\texttt{log}(b)} \tag{6.1}$$

Soit $\texttt{exp}(\cdot)$ l'opération inverse de `log`, qui à une représentation logarithmique donnée renvoie l'élément unique associé dans le corps de Galois, on en déduit l'Algorithme 21, permettant de faire cette multiplication.

Algorithm 21 Multiplication dans le corps de Galois. Extrait de l'implémentation de référence [AMAB+23] de HQC de Juin 2021

Require: a, b deux éléments de $\mathbb{F}_q$
Require: `exp` et `log`
Ensure: $v = a \times b$
  $l = \texttt{log}(a) + \texttt{log}(b) \mod q$
  return $v = \texttt{exp}(l)$

Si les tables `exp` et `log` sont pré-calculées, la multiplication dans le corps de Galois ne necessite plus qu'une addition et 3 appels mémoires et une réduction modulaire (voir Algorithme 21). Dans l'implémentation de référence de HQC, ces tables sont données en paramètres du schéma. Cet algorithme reste le même indépendamment du niveau de sécurité sélectionné pour HQC.

6.2 Attaque par Corrélation

Dans cette attaque, nous ciblons l'implémentation de référence de Juin 2021 et plus précisément la multiplication `gf_mul` dans le corps de Galois (voir Algorithme 21) impliqué dans le calcul du syndrome des codes de RS (voir Algorithme 20). Dans cette première attaque nous ne ciblons que HQC-128, le premier niveau de sécurité de HQC. Pour ce niveau de sécurité, la matrice de parité $\mathbf{H}$ est de taille $(n-k=30) \times (n=46)$ (voir tableau 3.1). Autrement dit, on a $46-1$ (C_1 est mis de côté) octets de mot du code à retrouver et chacun de ces octets est manipulé 30 fois de manière indépendante.

6.2.1 Modèle de l'attaquant

Nous avons ciblé l'implémentation de référence de HQC datant de juin 2021 [AMAB+23] exécuté sur un microcontrolleur ARM Cortex-M4 avec une fréquence d'horloge de 168 MHz. Nous avons mesuré les ondes électromagnétiques avec une micro-sonde *microprobe* LANGER EMV-Technik ICR

HH 100-6. Nous avons enregistré ces mesures avec un oscilloscope ROHDE-&SCHWARZ RTO2014 qui effectue un échantillonnage de 10 milliards de points par seconde. Une broche GPIO dédiée du microcontrôleur a été utilisée pour déclencher l'oscilloscope. Deux autres broches UART ont été utilisées pour la communication entre le microcontrôleur et l'ordinateur pendant les acquisitions. Nous avons acquis un ensemble de 256000 mesures électromagnétiques de l'exécution de la multiplication dans le corps de Galois, issu de l'implémentation de référence de HQC de juin 2021.

6.2.2 Résultats Simulés

Dans un premier temps, nous réalisons l'attaque en simulation. Nous simulons le comportement de plusieurs valeurs intermédiaires manipulées pendant l'algorithme avec un modèle de fuite en poids de Hamming (voir sous-section 4.1.1). Sans perte de généralité, les paramètres du modèle de fuite sont choisis comme $\alpha = 1$ et $\beta = 0$. Pour le calcul de $a \times b$ par l'Algorithme 21, où b est connu provenant de la matrice de parité et a secret que l'on cherche à retrouver, les valeurs intermédiaires ciblées sont :

— $\texttt{log}(a)$, qui correspond à l'appel en mémoire de la table pré-calculée des `log` dépendent de a.

— $\texttt{log}(a)+\texttt{log}(b) \mod q$, la valeur intermédiaires avant l'appel mémoire de la table `exp`.

— $\texttt{exp}(\texttt{log}(a) + \texttt{log}(b) \mod q)$, la sortie de la fonction `gf_mul`.

L'idée est donc de générer des traces simulées de 3 points, associées à des valeurs a et b, en faisant varier progressivement le bruit ajouté. Pour cette étude, nous avons générée environ 80000000 traces simulées reparties en 60000 ensembles de 30×45 traces. En effet, pour cette attaque, nous avons décidé de nous concentré sur une manipulation particulière du calcul du syndrome, correspondant à 30 éléments prédéfinis provenant de la matrice de parité. Ces éléments sont manipulées avec 45 éléments aléatoire d'un mot du code, que nous avons tirés aléatoirement. la dernière dimension provient de la variation du niveau de bruit sur chacune des traces simulées. Pour chacune de ces traces, la valeur du SNR est calculée selon l'équation (4.13). Ensuite, nous avons procédé à l'analyse par corrélation sur ces traces (voir sous-section 4.4.4). Cette analyse par corrélation est rendu possible par la connaissance de la seconde valeur intermédiaire b, permettant d'avoir des variations dans les traces pour une même valeur secrète a.

Nous considérons que l'attaque est un succès si la bonne hypothèse sur le byte secret est classée première par l'analyse de corrélation. Pour un niveau de SNR donné, on peut donc calcul le pourcentage moyen de succès sur les 45 octets cible de l'attaque (voir figure 6.1).

Sur cette figure, on observe qu'un SNR de plus de 8 est nécessaire pour que l'attaque soit un succès avec probabilité 100%. Or, dans un vrai scénario

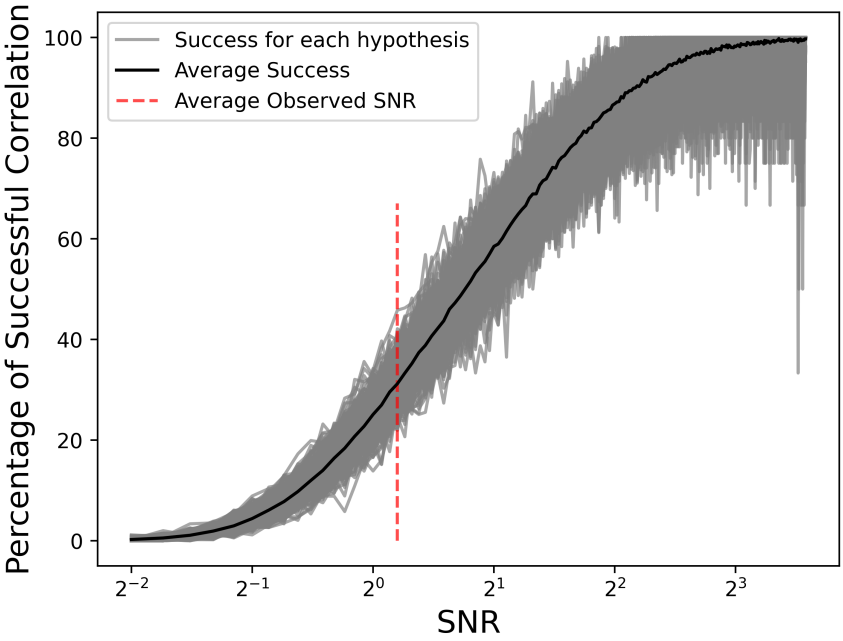


FIGURE 6.1 – Pourcentage moyen de succès de l'analyse de corrélation en fonction de la valeur du SNR sur les traces simulées.

d'attaque sur un banc, nous observons un SNR moyen de 1.15, cette valeur a été reportée en pointillés rouges sur la figure 6.1. Pour cette valeur de SNR, nous ne retrouvons correctement que 30% des bytes du mot du code en moyenne, ce qui est insuffisant pour réussir l'attaque sans plus d'efforts. En usant de quelques stratégies, nous allons voir qu'il est possible de retrouver ce mot du code sans avoir à obtenir un taux de succès de 100%.

6.2.3 Stratégie du re-décodage

Comme nous l'avons vu dans la section 3.5, le décodeur des codes de RS est connu publiquement dans le schéma. Nous pouvons donc l'utiliser pour corriger les erreurs dans le mot du code retourné par notre attaque par corrélation. Afin de réussir l'attaque, nous n'avons donc plus besoin d'un pourcentage de succès de 100% mais seulement d'un taux de $\frac{16}{46}$ donné par la capacité de correction du code des RS (voir tableau 3.1). Ce degré de liberté dans l'attaque nous permet de nous contenter d'un SNR de 2.35 pour réussir l'attaque (voir figure 6.2

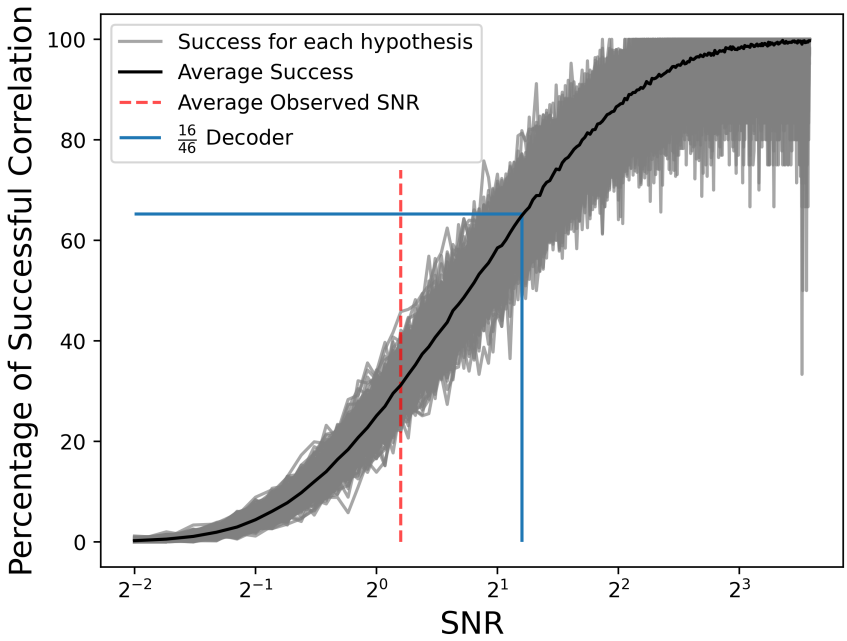


FIGURE 6.2 – Pourcentage moyen de succès de l'analyse de corrélation en fonction de la valeur du SNR sur les traces simulées avec l'influence du décodeur.

Décodage en liste Comme on l'a vu dans la sous-sous-section 3.5.3.2, les codes RS peuvent être décodés à l'aide d'algorithmes de décodage en liste. L'utilisation de ces décodeurs permet d'augmenter la capacité de correction d'erreurs. Cette première amélioration réduit la limite de rapport signal sur bruit imposée par notre analyse précédente.

Dans cette section, nous supposons que nous avons un message reçu erroné $\mathbf{r} = \mathbf{c} + \mathbf{e}$ de longueur n, généré par un mot de code RS $\mathbf{c}$ et une erreur $\mathbf{e}$ de poids de Hamming τ. Nous supposons également que l'erreur suit une distribution uniforme parmi les n coordonnées du mot de code. Nous notons que les décodeurs RS renvoient l'emplacement et la valeur des erreurs avant de renvoyer le message correspondant.

Ces résultat ne sont toujours pas suffisant pour passer sous la barre du SNR observé dans une attaque réelle. Dans les sections suivantes, des stratégies sont proposées pour réduire encore le SNR nécessaire pour réussir l'attaque.

6.3 Stratégies d'améliorations

Dans cette section, je présente des stratégies d'améliorations permettant de "rendre possible" l'attaque présentée dans la section précédente avec un SNR observé en pratique. Pour cela, il est nécessaire d'avoir recours à des stratégie permettant de corriger plus d'erreurs après la sortie de la corrélation. Nous exhibons deux types de stratégies :

— Une stratégie basée sur les propriété des codes tronqués et poinçonnés de RS. En supposant que le mot du code retourné par l'attaque en corrélation contient Δ erreurs. On montre que l'on peut corriger ces Δ erreurs, plus grands que la capacité de correction τ des codes de RS au prix de paris sur la position des erreurs dans le mot.

— Une stratégie demandant de considérer les x premiers candidats retournés par la phase de corrélation, au lieu de seulement le premier candidat.

Ces stratégies viennent avec un cout non-négligeable qui sera présenté dans la section suivante.

6.3.1 Poinçonner des positions non erronées

La première idée est d'observer que la capacité de correction des codes de RS augmente lorsque le rapport $R = \frac{k}{n}$ diminue (voir figure 2.4). On peut faire baisser ce rapport en poinçonnant le code, c'est-à-dire en retirant une coordonnée non erronée du code. Le code de RS initialement de paramètre $[n, k, t]$ devient un code de RS raccourci de paramètre $[n-1, k-1, t']$. Le rapport $\frac{k}{n}$ ayant diminué, la capacité de correction t' est plus importante

en théorie. En pratique, il faut poinçonné au moins deux fois pour que la capacité de correction $\lfloor \frac{n-k}{2} \rfloor$ diminue.

En tirant des positions au hasard, la probabilité de trouver u erreurs est donnée par :

$$\mathbb{P}\left(\text{Tirer } u \text{ erreurs}\right) = \frac{\binom{u}{\Delta}}{\binom{u}{n}} \tag{6.2}$$

Le nombre moyen de tirages T pour trouver ces u positions non erronées est donc :

$$T = \frac{\binom{u}{n}}{2\binom{u}{\Delta}} \tag{6.3}$$

6.3.2 Tronquer des positions erronées

La seconde idée est de faire l'hypothèse que l'on connait u positions erronées dans le mot du code. En retirant des positions erronées, on passe d'un code de paramètre $[n, k, \tau]$ à un code de paramètre $[n-u, k, \tau]$. Le rapport du code $R = \frac{k}{n}$ augmente, donc la capacité de correction diminue (voir figure 2.4). Cependant, la capacité de correction diminue plus doucement que le nombre d'erreurs tronquées. En effet, pour chaque groupe de 2 erreurs tronquées, on ne perds que 1 de capacité de correction selon la formule $\lfloor \frac{n-k}{2} \rfloor$.

On peut donc faire l'hypothèse que l'on connait u emplacements erronés dans le mot de code. En tirant ces u positions de manières aléatoires, la probabilités de trouver u positions sans erreurs est donnée par :

$$\mathbb{P}\left(\text{Tirer } u \text{ positions sans erreurs}\right) = \frac{\binom{u}{n-\Delta}}{\binom{u}{n}} \tag{6.4}$$

Le nombre moyen de tirages T pour trouver ces u positions non erronées est donc :

$$T = \frac{\binom{u}{n}}{2\binom{u}{n-\Delta}} \tag{6.5}$$

6.3.3 Considérer plusieurs candidats de corrélation

Cette dernière stratégie consiste à considérer plusieurs candidats plutôt qu'un seul en sortie de l'étape de corrélation. La dernière étape de l'attaque doit être effectuée pour chaque candidat retenu. La Figure 6.3 résume le nombre moyen d'attaques réussies en fonction du SNR et du nombre de candidats retenus. Cette stratégie peut être très couteuse, en effet, pour x candidats considérés, le nombre d'étapes de décodage nécessaires augmente d'un facteur x[46].

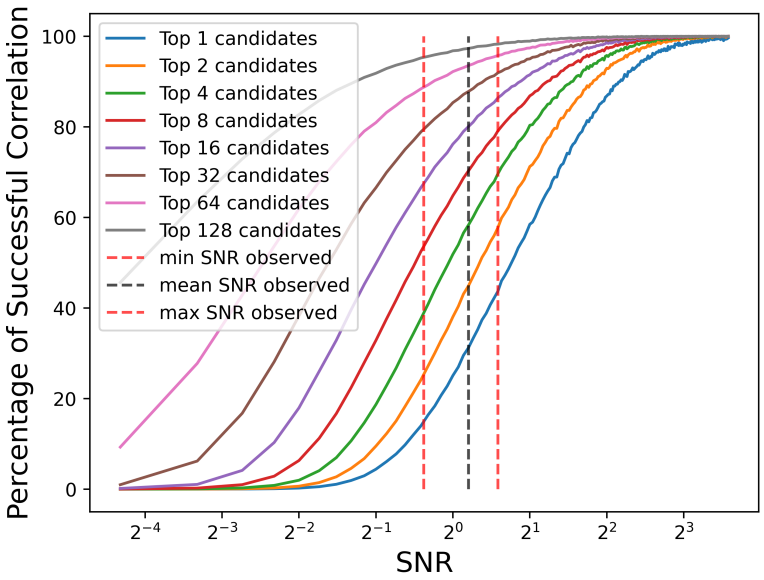


FIGURE 6.3 – Pourcentage moyen de succès de l'analyse de corrélation en fonction de la valeur du SNR et du nombre de candidats retournés.

6.4 Complexité de l'Attaque

En considérant un SNR moyen de $1,15$ et les deux meilleurs candidats retournés par la corrélation. Le pourcentage moyen de bytes corrects trouvés est de plus de 46%. Cette stratégie renvoie 2^{46} candidats à tester. D'autre part, en supposant que $u = 13$ emplacements sans erreur sont connus, nous pouvons mettre en évidence un décodeur capable de décoder 25 bytes sur les 46. Cela représente environ 54% du nombre total de bytes, pour un cout de 2^{15} étapes de décodage. Nous décodons suffisamment d'octets pour retrouver le mot de code sans erreur et réussir l'attaque. Le décodeur a une complexité maximale de $\mathcal{O}(n^7)$ opérations sur le corps [Bar11], avec n la longueur du code. Nous appliquons le décodeur sur des mots de taille $n - u = 46 - 13$, ce qui correspond à moins de 2^{35} opérations dans notre cas. La complexité de notre attaque est alors donnée par $2^{46} \times 2^{15} \times 2^{35} = 2^{96}$ opérations sur le corps de Galois. Ce nombre est gigantesque et ne permet pas de réaliser cette attaque en pratique. Cependant, la complexité de cette attaque est en dessous du niveau de sécurité requis pour HQC qui est de 2^{128} opérations nécessaires pour retrouver les valeurs secrètes.

6.5 Contremesures

Bien que cette attaque soit difficilement réalisable en pratique, elle diminue le niveau de sécurité et représente donc une vulnérabilité pour HQC. Nous proposons donc deux contremesures qui empêchent notre attaque.

6.5.1 Réaliser les opérations dans un ordre aléatoire

Comme nous l'avons vu dans la section 7.5, le mélange est une bonne manière de se protéger des attaques par canaux auxiliaires. Ici, notre attaque est basée sur le fait que l'on est capable d'associé à chaque mesure physique de l'opération `gf_mul` les valeurs qui y on été manipulées. Cela est possible car les opérations lors du calcul de syndrome (voir Algorithme 20 sont réalisées dans un ordre déterministe, prévu à l'avance et invariant. Il est donc facile d'associer les mesures physiques avec les valeurs manipulées, que ce soit les octets secrets du mot du code ou les éléments de la matrice de parité. Sans cette association, notre attaque ne peut être appliquée dans l'état.

Notre contremesure consiste donc à rendre aléatoire l'ordre d'exécution des multiplications pendant le calcul du syndrome. En reprenant le mélange de Clavier et al. [CFG+10] dans notre cas nous obtenons l'Algorithme 22. Le nombre de possibilité dans le nombre de permutation s'élève à $\delta! \cdot n_1!$. Pour réussir l'attaque telle que je viens de la présenter, il faudrait dans un premier temps, inverser la permutation en trouvant la bonne dans cette liste. Sous cette hypothèse, cette contremesure est efficace contre notre attaque.

Algorithm 22 Calcul de syndrome randomisé

Entrée : (message $\mathbf{m}$, param = (alpha, δ, n_1))
Sortie : syndrome **synd**

1: $L_1 = [1, \cdots, 2 \times \delta])$
2: $L_2 = [1, \cdots, n_1]$
3: $L = \textbf{Shuffle}(L_1 \times L_2)$
4: **for** $(i, j) \in L$ **do**
5: $\quad \textbf{synd}[i] = \textbf{synd}[i] \oplus \text{GF\_MUL}(\mathbf{m}[j], \text{alpha}[i][j-1])$
6: **for** $i \in L_1$ **do**
7: $\quad \textbf{synd}[i] = \textbf{synd}[i] \oplus \mathbf{m}[0]$
8: **return synd**

Cependant, nous n'avons pas fait l'analyse de sécurité de l'algorithme randomisé (Algorithme 22). Des attaques basées sur le profilage des comportements de chaque exécution pourraient peut-être renverser ce mélange et retrouver la clef partagée de HQC. Cependant ce type d'attaque sort du cadre de cette attaque et nous ne le traiterons pas dans ce chapitre.

6.5.2 *Codeword Masking*

La seconde contremesure dont nous parlons est le *Codeword Masking* [MSS13] introduit par Merli et al. e, 2013 dans le cadre des Physical Unclonable Function (PUF). Une variante de cette stratégie de contremesure a été utilisé par Petrvalsky et al. [PRD+16] pour protéger ClassicMcEliece en 2016. Le but de cette contremesure est de masquer l'entièreté du décodeur. Cette contremesure peut s'avérer très efficace, à la fois pour HQC mais aussi pour d'autres schémas basés sur les codes correcteurs d'erreurs. En effet, comme nous l'avons vu en section 4.7, la majorité des attaques physiques contre la cryptologie basée sur les codes correcteurs d'erreurs ciblent les étapes de décodage.

Néanmoins, le décodeur n'est pas une fonction linéaire sur l'ensemble des mots dans l'espace ambiant $\mathbb{F}_2^n$. Heureusement, la linéarité existe entre l'ensemble des messages et l'ensemble des mots de code sans erreur (voir Équations 6.6 et 6.7).

$$\mathcal{C}.\texttt{Encode}(m_1+m_2) = \mathcal{C}.\texttt{Encode}(m_1)+\mathcal{C}.\texttt{Encode}(m_2),\ m_1, m_2 \xleftarrow{\$} \mathbb{F}_2^k \quad (6.6)$$

$$\begin{aligned} \mathcal{C}.\texttt{Decode}(c_1 + c_2) = \mathcal{C}.\texttt{Decode}(c_1) + \mathcal{C}.\texttt{Decode}(c_2), \\ c_1 = \mathcal{C}.\texttt{Encode}(m_1),\ c_2 = \mathcal{C}.\texttt{Encode}(m_2),\ m_1, m_2 \xleftarrow{\$} \mathbb{F}_2^k \end{aligned} \quad (6.7)$$

De plus, pour toute erreur e dont le poids est inférieur à la capacité de correction d'erreur du code t, l'Équation (6.8) est satisfaite.

$$\mathcal{C}.\texttt{Decode}\left(\mathcal{C}.\texttt{Encode}(m) + e\right) = m,\ |e| < t,\ m \xleftarrow{\$} \mathbb{F}_2^k \quad (6.8)$$

À partir des Équations (6.7) et (6.8), nous sommes capables de construire une protection solide contre les attaques par canaux auxiliaires qui exploitent le comportement du décodeur (voir figure 6.4).

Codewrod Masking peut-être appliqué dans le cadre de HQC au moment du déchiffrement (voir Algorithme 8). L'Algorithme 23 présenté le déchiffrement de HQC masqué avec *Codeword Masking*.

Algorithm 23 Déchiffrement de HQC avec *Codeword Masking*

Entrée : $(\mathsf{sk}, \mathbf{c})$
Sortie : $\mathbf{m}$

1: $\mathbf{r} \xleftarrow{\$} \mathbb{F}_2^k$
2: $\mathbf{c}_r = \mathcal{C}.\texttt{Encode}(r)$
3: $\mathbf{m}_r = \mathcal{C}.\texttt{Decode}(\mathbf{v} - \mathbf{uy} + \mathbf{c}_r)$
4: $\mathbf{m} = \mathbf{m}_r - \mathbf{r}$

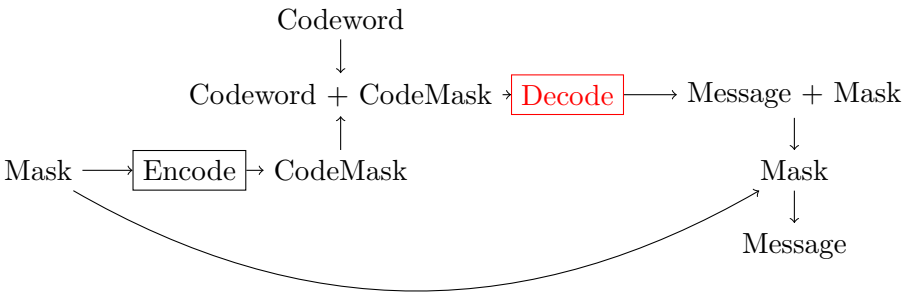


FIGURE 6.4 – Fonctionnement général de *Codeword Masking* pour masquer un décodeur sensible (en rouge)

Cout et Force Le cout de *Codeword Masking* est limité et ne nécessite que deux éléments : *(i)* un calcul d'encodeur supplémentaire et *(ii)* le tirage aléatoire d'un vecteur de longueur k bits.

D'une part, en général, pour une contremesure basée sur le masquage, la fonction à protéger doit être calculée deux fois. Avec *Codeword Masking*, le cout pour une telle stratégie est réduit car un encodeur est beaucoup plus rapide qu'un décodeur. En effet, l'étape d'encodage est simplement une multiplication matrice vecteur tandis que le décodage est une stratégie complexe. De plus, comme on l'a dit, *Codeword Masking* peut être appliqué à n'importe quel code correcteur d'erreurs linéaire.

Masque d'Ordre Supérieur Comme chaque stratégie de masquage, *Codeword Masking* peut être utilisé à l'ordre supérieur en tirant d messages aléatoires en fonction des besoins (voir Algorithme 24). L'utilisation d'un masque de degré supérieur augmente l'information qu'un attaquant doit récupérer pour mener son attaque.

Algorithm 24 Decrypt avec COMA d'ordre d

Entrée : $(\mathsf{sk}, \mathbf{c})$
Sortie : $\mathbf{m}$

1: $(\mathbf{r}_1, \cdots, \mathbf{r}_d) \xleftarrow{\$} \left(\mathbb{F}_2^k\right)^d$
2: $\mathbf{c}_r = \sum_{i=1}^{d} \mathcal{C}.\mathtt{Encode}(r_i)$
3: $\mathbf{m}_r = \mathcal{C}.\mathtt{Decode}(\mathbf{v} - \mathbf{uy} + \mathbf{c}_r)$
4: $\mathbf{m} = \mathbf{m}_r - \sum_{i=1}^{d} \mathbf{r}_i$

6.6 Conclusion

Dans ce chapitre, nous avons montré une vulnérabilité dans le schéma HQC au niveau du décodeur. Cette vulnérabilité permet de réduire le niveau de sécurité du schéma, même si l'attaque n'est pas réalisable en pratique. Il est donc nécessaire d'éliminer cette vulnérabilité en protégeant HQC avec une contremesure adaptée. Dans ce chapitre, nous en présentons deux, une stratégie de mélange et une stratégie de masquage.

CHAPITRE 7

RETROUVER LA CLEF PARTAGÉE DE HQC AVEC SASCA

Contents

7.1 Introduction

Ce troisième papier [GMGL23] est un travail en commun, réalisé avec Julien Maillard, du CEA Grenoble, et mes encadrants de thèse, Antoine Loiseau et Philippe Gaborit. Cette attaque a été accepté pour être présenté à Conference on Cryptographic Hardware and Embedded Systems (CHES) 2024 et est disponible en accès libre par le lien suivant : `https://eprint.iacr.org/2023/1590`. Il s'agit d'une attaque utilisant des algorithmes de propagation de croyance. Cette attaque reprend les fondements de l'attaque par corrélation, présentée dans le chapitre 6, en la surpassant largement en terme de résultats.

Avec ces travaux, nous introduisons les premières attaques SASCA pratiques contre un cryptosystème basé sur les codes correcteurs d'erreurs. Nous proposons des attaques par canaux auxiliaires qui sont exécutables en quelques minutes contre la cryptographie post-quantiques (PQC). Plus précisément, nous récupérons la clef partagée manipulée par le code de Reed-Solomon (RS) impliqué dans le schéma HQC-RMRS. Toutes les attaques présentées exploitent un ou deux modèles ciblant la multiplication dans le corps de Galois. Nous montrons que l'implémentation de référence de HQC peut être visée par des attaques physiques en une seule trace et reste menacée même lorsqu'elle est protégée par des contre-mesures établies dans l'état de l'art. Nos attaques sont réalisées à la fois en simulation et dans un scénario d'attaque réelle sur un STM32F407.

— Nous exploitons d'abord le point de vulnérabilité identifié par Goy et al. [GLG22b], présenté dans le chapitre 6 de ce manuscrit, et la transformons en une attaque physique réalisable en pratique avec une seule trace dans le but de retrouver clef partagée. Bien que cette attaque repose sur l'établissement de modèles préalables (voir sous-section 4.4.5), l'exigence de stratégies de propagation de croyance (BP) dépend fortement des choix d'implémentation. Nous décrivons comment construire le graphe des facteurs pour l'algorithme de décodage RS, qui manipule des mots de code sans erreur contenant des informations sur la clef partagée.

— Nous montrons que la stratégie de masquage *Codeword Masking* (voir sous-section 6.5.2, qui est applicable à HQC (voir Algorithme 23), ne fournit pas une sécurité satisfaisante contre notre attaque. Même si des contre-mesures de masquage appliqués à la NTT de Kyber ont été montrées vulnérables aux attaques SASCA, cette considération ne peut pas être appliquée telle quelle pour HQC. En effet, le masquage *Codeword Masking* sur le décodeur de HQC est effectué avec un encodeur RS à des fins de performance. Ainsi, nous réalisons une étude contre l'encodeur RS et montrons qu'aucune contre-mesure de masquage simple ne peut contrecarrer notre attaque.

— Nous étudions également la robustesse des stratégies de permutation connues dites fine et grossière (ou *fine* et *coarse* en anglais) [RPBC20], ainsi qu'une stratégie spécifique à HQC la permutation des fenêtres (ou *Window Shuffling*) contre notre attaque. Nous montrons qu'aucune de ces stratégies ne constitue une contre-mesure durable contre notre attaque dans un scénario d'attaque réelle. Enfin, nous introduisons la stratégie de double permutation du décodeur RS, qui ajoute la complexité combinatoire la plus élevée pour une permutation aléatoire, rendant l'attaque impraticable.

Comme l'attaque par corrélation, présentée dans le chapitre 6, la cible de notre attaque est le calcul du syndrome lors du décodage des codes de RS. La fonction principale lors de ce calcul de syndrome est la multiplication dans le corps de Galois. C'est également cette fonction particulière qui sera au centre de notre attaque. Cependant, depuis Avril 2023, l'implémentation de référence de HQC [AMAB$^+$23] a changé, et avec elle l'algorithme réalisant cette multiplication.

7.1.1 Multiplication dans le coprs de Galois de HQC (Avril 2023)

A partir d'Avril 2023, l'implémentation de référence de HQC est mise à jour, dans le but principal de la rendre temps constant et d'enlever les appels mémoires dépendants de valeurs de secret. Parmi les modifications, la multiplication dans le corps de Galois est ré-écrite. Elle utilise désormais un algorithme de multiplication rapide issu de [BGTZ08], basé sur un modèle de Transformée de Fourier Rapide (ou *Fast Fourier Transform (FFT)*). Avec Algorithme 25, nous décrivons cette opération utilisée depuis 2023 dans [AMAB$^+$23], depuis l'implémentation de référence de HQC en avril 2023. L'algorithme de la multiplication dans le corps de Galois, `gf_mul`, est invariant en fonction du niveau de sécurité HQC sélectionné.

Algorithme 25 effectue une multiplication $a \times b$ dans un corps de Galois $\mathbb{F}_{2^8}$. Il est important de noter que les deux entrées a et b ne sont pas traitées de manière symétrique par l'algorithme. En effet, $t[1]$ extrait les deux bits les moins significatifs de a aux lignes 8 et 16. Les lignes 25 et 26 décalent les bits de a de valeurs différentes dans une boucle *for*. Cette asymétrie entraine une manipulation plus importante de l'une des deux opérandes, et nous verrons que cela a des conséquences dans les fuites observées au travers des mesures physiques.

7.2 Attaque Simple par profilage

Dans cette section nous présentons une première attaque, uniquement basée sur une analyse de profilage du comportement physique de la multi-

Algorithm 25 Multiplication le corps de Galois $\mathbb{F}_{2^8}$ de l'implémentation de référence de HQC [AMAB+23] depuis Avril 2023

```
uint16_t gf_mul(uint16_t a, uint16_t b) {
  uint8_t c[2] = {0};
  uint16_t h = 0, l = 0, g = 0, u[4];
  u[0] = 0;
  u[1] = b & ((1UL << 7) - 1UL);
  u[2] = u[1] << 1;
  u[3] = u[2] ^ u[1];
  uint16_t tmp1 = a & 3;
  for(int i =0; i < 4; i++) {
    uint32_t tmp2 = tmp1 - i;
    g ^= (u[i] & -(1 - ((tmp2 | -tmp2) >> 31)));
  }
  l = g;
  for (uint8_t i = 2; i < 8; i+=2) {
    g = 0;
    uint16_t tmp1 = (a >> i) & 3;
    for (int j = 0; j < 4; ++j) {
      uint32_t tmp2 = tmp1 - j;
      g ^= (u[j] & -(1 - ((tmp2 | -tmp2) >> 31)));
    }
    l ^= g << i;
    h ^= g >> (8 - i);
  }
  uint16_t mask = (-((b >> 7) & 1));
  l ^= ((a << 7) & mask);
  h ^= ((a >> (1)) & mask);
  c[0] = l;
  c[1] = h;
  uint16_t tmp = (uint16_t) (c[0] ^ (c[1] << 8));
  return gf_reduce(tmp, 2*(PARAM_M-1));
}
```

plication dans le corps de Galois.

7.2.1 Modèle de l'attaquant et Configuration Expérimentale

Pour cette attaque, nous supposons avoir un accès physique sur une cible : un dispositif effectuant la décapsulation de HQC pour en calculer une clef partagée. C'est cette clef partagée que nous cherchons à retrouver au travers de notre attaque. Nous supposons également que nous avons accès à un dispositif clone de la cible, avec les mêmes caractéristiques et qui exécute la même implémentation du schéma. Dans un soucis de simplicité, pour nos attaques, nous utilisons le même dispositif pour jouer les deux rôles. Le dispositif clone permet d'enregistrer des mesures physiques de référence, d'entrainement, en ayant le contrôle des valeurs manipulées dans l'algorithme cible.

Nous avons mesuré les ondes électromagnétiques émise par le dispositif avec une sonde Langer en champs proche (*Near Field*). Nous utilisons un oscilloscope Rhode&Schwarz RT02024 pour enregistrer les mesures avec un taux d'échantillonnage de 1G points par secondes. Nous avons extrait la multiplication dans le corps de Galois (voir Algorithme 25) de l'implémentation de référence de HQC [AMAB+23]. Nous avons compilé cet algorithme avec l'option d'optimisation $-O3$, borné par un déclenchement (*trigger*) contrôlé par un pin dédié du dispositif (GPIO). Le code assembleur de cette fonction peut être consulté en annexe A.2. Cette configuration conduit à un temps de calcul de $1,3\mu s$ et à des traces de 1300 points (voir la figure 7.1 pour la trace moyenne acquise). Ces traces comprenant un faible nombre de points, nous permettent de réaliser notre attaque sur toute la longueur des traces, sans sélectionner de points ou de zones d'intérêt (PoI, AoI). Au total, un nombre de 500000 traces ont été acquises pour des entrées tirées de manière aléatoire.

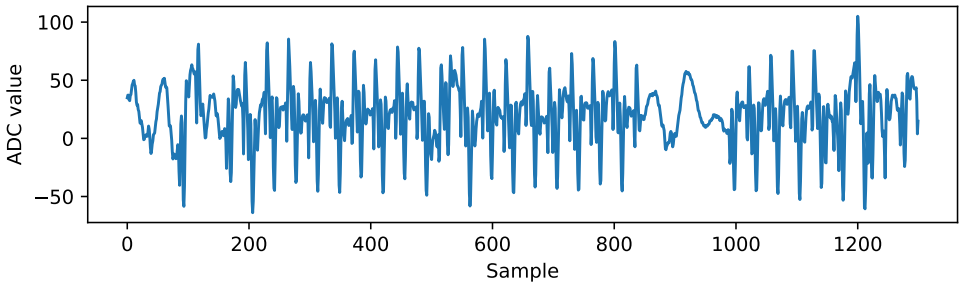


FIGURE 7.1 – Trace moyenne de l'exécution de la fonction `gf_mul`.

7.2.2 Profilage de la Multiplication dans le Corps de Galois

Au préalable, nous effectuons une évaluation des fuites sur les trois variables 8 bits impliquées dans la multiplication dans le corps de Galois `gf_mul` : les deux opérandes de la multiplication ainsi que la sortie. Nous faisons l'hypothèse d'un modèle de fuite en poids de Hamming (voir section 4.2.1) et nous nous appuyons sur une Analyse de Régression Linéaire (LRA), (voir sous-sous-section 4.3.1.2).

Pour chaque point des mesures physiques, nous appliquons l'analyse linéaire et calculons le le coefficient de détermination, tel que expliqué dans la sous-sous-section 4.3.1.2. Nous faisons ce calcul pour chacune des trois variables ciblées, nous obtenons le résultat suivant (voir figure 7.2) :

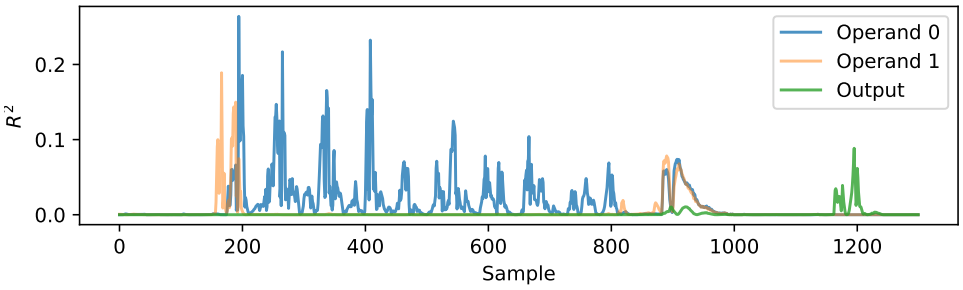


FIGURE 7.2 – Coefficients de détermination calculés pour les entrées et la sortie de la multiplication dans le corps de Galois.

En observant la sortie de la LRA dans figure 7.2, nous pouvons remarquer que *(i)* la fuite du premier opérande est à la fois importante et répartie le long du calcul de `gf_mul` : cela peut s'expliquer par les nombreuses opérations logiques effectuées sur cet opérande que j'ai décrit dans la section précédente, *(ii)* la fuite correspondant au deuxième opérande est moins importante et *(iii)* la sortie de `gf_mul` fuit uniquement à la fin de la fonction, probablement lorsqu'elle est stockée en mémoire.

Ensuite, nous mettons en place 6 modèles de profilage différents : *(i)* 3 modèles ciblent le poids de Hamming des entrées et de la sortie de la multiplication dans le corps de Galois. *(ii)* Les 3 derniers modèles visent à récupérer la valeur exacte des entrées et de la sortie impliquées dans la multiplication dans le corps de Galois. Dans le second cas, les modèles sont entrainés et validés en considérant les valeurs intermédiaires exactes. De plus le nombre de classes est plus importants, ces différences permettent à l'algorithme de faire une classification plus précise et de repérer des points de distinction que le profilage en poids de Hamming ne peut pas voir. Au final, l'entrainement a accès à des informations plus précises sur le comportement des variables et cela sert à un meilleur taux de succès dans la classification sur les jeux de validation. Je précise que tous ces modèles de profilage (variables

ou Hamming) sont construits sans connaissance préalable des deux autres variables manipulées par la multiplication dans le corps de Galois.

Les modèles sont construits avec l'Analyse Discriminante Linéaire de Fisher (LDA) comme classifieur. La précision de validation de chaque modèle est évaluée sur des ensembles de données de tailles différentes segmentés en 90% de traces d'entrainement et 10% de traces de validation. Nous avons analysé la précision en fonction du nombre sélectionné de traces d'entrainement (voir figure 7.3) et conclu que le meilleur compromis est atteint pour 300000 traces d'entrainement. Les précisions obtenues des modèles pour 300000 traces d'entrainement sont résumées dans tableau 7.1.

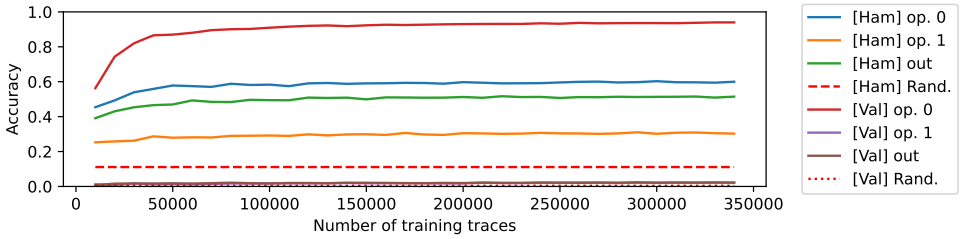


FIGURE 7.3 – Précision sur chaque variable pour les modèles en valeur et en poids de Hamming en fonction du nombre de traces d'entrainement.

	Précision du modèle en valeur	Précision du modèle en poids de Hamming
Opérande 0	**0.9389**	0.5929
Opérande 1	0.0211	**0.3035**
Sortie	0.0221	**0.5178**

TABLE 7.1 – Précisions des modèles de poids de Hamming et de valeur sur `gf_mul` et taux de réussite des attaques sur STM32F407. Les modèles ont été entraînés avec 300000 traces d'entrainement avec une segmentation 10%/90% pour la validation/l'entrainement.

Plusieurs observations peuvent être faites : *(i)* la valeur du premier opérande peut être prédite avec une précision de $93,89\%$, *(ii)* les attaques basées sur le modèle de valeur sur le deuxième opérande et la valeur de sortie ne fournissent pas de prédictions significativement meilleures que des suppositions aléatoires et *(iii)* les poids de Hamming de la sortie et du deuxième opérande donnent des résultats satisfaisants, plus informatifs qu'une supposition aléatoire.

Nous pouvons conclure que le grand nombre d'opérations logiques agissant sur le premier opérande de `gf_mul` (voir Algorithme 25) est bénéfique du point de vue d'un attaquant utilisant une stratégie de profilage. En effet, les

divers *shift* de bits permettent d'apprendre des informations sur le support de sous blocs de la valeur intermédiaires. Cela augmente la séparabilité entre les différentes classes de valeurs et de ce fait augmente le taux de succès du classifieur. Par conséquent, il est plus facile pour la LDA de discriminer les valeurs que les classes de poids de Hamming pour cet opérande particulier. Rappelons que, lors du calcul des syndromes RS (voir Algorithme 20), le message, qui est la donnée sensible de ce calcul, est utilisé comme le premier opérande de la multiplication qui est celui qui fuit le plus. De plus, le stockage de la sortie de `gf_mul` dans la mémoire principale permet à un attaquant d'atteindre des précisions de modèle exploitables.

7.2.3 Construction de Matrices de Prédiction

Dans cette sous-section, nous décrivons une structure de données appelée "matrice de prédiction", qui vise à fournir des simulations répétables proches de la réalité en stockant de multiples prédictions de profilage. Concevoir une simulation qui correspond aux prédictions d'une attaque en profilage sur une cible réelle est une tâche difficile. En effet, les sorties des modèles dépendent de plusieurs facteurs. Par exemple, les composants matériels impliqués dans l'attaque, tels que la carte cible et les outils de mesure, ainsi que les conditions de l'expérience (*e.g.*, positionnement de la sonde électromagnétique, température, etc.) ont un impact sur la précision du profilage. Le choix d'un classifieur, ainsi que son exploitation de la fuite multivariée, a également un impact considérable sur les propriétés du modèle de template.

Pour ces raisons, nos attaques de scénarios réels sont effectuées grâce à des matrices de prédiction. Celles-ci contiennent un ensemble de 100000 distributions de probabilité indépendantes prédites par les modèles décrit dans sous-section 7.2.2, ainsi que les *labels* (c'est-à-dire la valeur des variables intermédiaires) correspondants. Les avantages de telles matrices de prédiction sont doubles : *(i)* les éléments tirés de manière aléatoire à partir d'une matrice de prédiction peuvent être considérés comme une prédiction réelle du modèle. Ces matrices peuvent être utilisée dans une simulation pour tester la robustesse des attaques, qui peuvent facilement être exécutées un grand nombre de fois. *(ii)* comme toutes les attaques présentées dans cet article exploitent les fuites de l'opération `gf_mul`, l'utilisation de matrices de prédiction permet de dériver des attaques sur plusieurs fonctions et scénarios de contre-mesure.

Dans notre cas, utiliser des matrices de prédiction dans le cas de l'attaque est comparable à une attaque dans un cas réel à la condition que l'attaquant soit capable de détecter et d'isoler les différentes exécutions de la multiplication de Galois dans une mesure plus vaste. Nous pensons que cette hypothèse est raisonnable dans le cadre du modèle d'attaquant que nous considérons dans ce travail. Nous soulignons que cette détection peut être réalisée avec des algorithmes de détection de forme (*pattern matching*).

7.2.4 Combiner et Conquérir

Dans Algorithme 20, nous remarquons que chaque octet du mot de code $c[j]$ est manipulé indépendamment $n-k$ fois dans la boucle for, aux lignes 2 et 4. Dans l'implémentation de référence actuelle de HQC [AMAB+23], `gf_mul` manipule toujours les octets du mot de code avec le premier opérande de la fonction. Ce choix permet une attaque, exploitant le taux de succès p élevé de notre modèle sur le premier opérande. L'attaquant peut combiner les sorties des modèles avec une stratégie, telle que le vote majoritaire, qui fournit une borne inférieure pour le taux de réussite de l'attaque.

La probabilité que l'hypothèse correcte soit classée en première position par le classificateur peut être vue comme le résultat d'une distribution de Bernoulli de paramètre p. Étant donné que les essais sont indépendants, ils peuvent être combinés en une distribution binomiale de paramètres $n-k$ et p. Soit X la variable aléatoire suivant cette distribution pour un octet du mot de code. Comme pour l'attaque par corrélation (voir chapitre 6), on peut observer que C_1, le premier octet du mot de code, n'est pas manipulé avec `gf_mul`, et ne peut donc pas être récupéré avec notre attaque basée sur le modèle. Pour tout autre octet du mot de code, le vote majoritaire est un succès si et seulement si $X > \lfloor \frac{n}{2} \rfloor$. De plus, tous les octets du mot de code sont indépendants, ce qui donne la probabilité de réussite suivante :

$$\mathbb{P}\left(\texttt{succès}_{\setminus C_1}\right) = \mathbb{P}\left(X > \left\lfloor \frac{n}{2} \right\rfloor\right)^{n-1}, X \rightsquigarrow \mathcal{B}(n-k,p) \tag{7.1}$$

7.2.5 Stratégie de Re-Décodage

De la même manière que pour le chapitre 6, la stratégie qui consiste à re-décoder le mot de code récupéré peut être appliqué, ce qui offre plusieurs avantages : *(i)* le re-décodage permet de corriger les erreurs ou imprécisions des modèles, *(ii)* cela permet de récupérer la valeur de C_1 qui ne peut pas être trouvée par les résultats des modèles et *(iii)* l'attaquant gagne une flexibilité supplémentaire concernant la précision du modèle. Les décodeurs de liste (voir sous-sous-section 3.5.3.2) peuvent également être utilisé dans cette attaque afin d'augmenter encore d'avantage la capacité de correction. Ces décodeurs ne sont pas utilisés dans HQC pour des questions de performance, cependant, nous pouvons tirer parti de leur capacité accrue de correction d'erreurs pour améliorer l'attaque.

Avec les paramètres HQC, le décodeur en liste de Guruswami et Sudan [VG99] est capable de corriger respectivement jusqu'à $\tau = 19$, 19 ou 36 erreurs, au lieu de $t = 15$, 16 ou 29 pour HQC 128, 192 ou 256. Notez qu'un emplacement d'erreur est déjà pris par C_1. En effet, C_1 n'est pas manipulé avec une opération `gf_mul`, donc notre modèle d'attaquant ne permet pas d'effectuer une attaque par modèle sur cette variable. La probabilité de réussite de l'attaque devient :

$$\mathbb{P}\left(\texttt{succès}\right) = \sum_{i=0}^{\tau-1} \mathbb{P}\left(X > \left\lfloor \frac{n}{2} \right\rfloor\right)^{n-i-1} \cdot \mathbb{P}\left(X \leq \left\lfloor \frac{n}{2} \right\rfloor\right)^{i}, X \rightsquigarrow \mathcal{B}(n-k,p) \tag{7.2}$$

7.2.6 Attaque Pratique

Niveaux de sécurité Cette attaque par profilage peut également être menée pour les niveaux de sécurité de HQC. En effet, la fonction `gf_mul` est exactement la même, indépendamment du niveau de sécurité sélectionné, ce qui nous permet de réutiliser les modèles (voir sous-section 7.2.2). Les paramètres de tableau 3.1 montrent que n, le nombre d'octets du mot de code à récupérer, augmente avec le niveau de sécurité. Mais, en même temps, $n-k$, le nombre d'essais indépendants, augmente également, fournissant plus d'informations indépendantes sur chaque octet du mot de code.

Résultats Nous observons une précision de $p = 0.9389$ sur le premier opérande avec 300000 traces d'entraînement et une seule trace d'attaque (voir tableau 7.1). En considérant cette probabilité dans les équations 7.1 et 7.2, nous obtenons des taux de réussite supérieurs à 0.9999 avec ou sans la stratégie de re-décodage pour tous les niveaux de sécurité HQC.

Discussion À partir de équation 7.2, nous calculons la valeur minimale de p telle que le succès de l'attaque reste au-delà de 0.9. Il s'ensuit qu'une précision minimale du modèle de $p_{min} = 0.7262$ est suffisante pour réussir l'attaque pour HQC128. HQC192 et HQC256 nécessitent une précision minimale du modèle de respectivement $p_{min} = 0.7250$ et $p_{min} = 0.6834$. Étant donné que la précision minimale requise est inférieure pour chaque niveau de sécurité que ce que nous avons obtenu en pratique, nous pourrions envisager d'attaquer des cibles avec un niveau de bruit plus élevé.

Cette première attaque est possible uniquement grâce à un choix "malheureux" dans l'ordre des opérandes pour la multiplication dans l'implémentation de référence de HQC [AMAB+23]. On peut raisonnablement supposer, à partir des résultats présentés ici, qu'un développeur informé fera le choix de permuter les premier et deuxième opérandes. Cela permet de manipuler les données sensibles avec l'opérande qui fuit le moins. Étant donné que cette opération de multiplication est commutative, la permutation des opérandes n'implique pas de surcoût computationnel. L'implémentation ainsi permutté ne laisse plus la possibilité d'attaquer HQC avec une simple attaque par profilage. Il faut alors élaborer des stratégies plus complexes que je présente dans les prochaines sections.

7.3 SASCA sur le Décodeur Reed-Solomon

Après la permutation des opérandes, nous ne sommes pas en mesure d'effectuer l'attaque de section 7.2 contre les données sensibles, qui sont maintenant "cachées" derrière le deuxième opérande. De plus, le premier opérande, dont la valeur peut être modélisée avec une grande précision, manipule désormais les éléments de la matrice de parité, qui est déjà publiquement connue. Cependant, les résultats de tableau 7.1 montrent que les poids de Hamming de la deuxième opérande et de la sortie de `gf_mul` peuvent être modélisé avec une bonne précision. Cette information est regroupée dans un graphe de facteurs.

7.3.1 Graphe du Décodeur Reed-Solomon

Nous construisons un graphe de facteurs pour représenter le calcul du syndrome RS illustré dans Algorithme 20 (voir figure 7.4). Chacune des $n-1$ fenêtres correspond à une itération de la deuxième boucle *for* (ligne 3). Dans chaque fenêtre, $m = n - k$ multiplications dans le corps de Galois (ligne 4) sont effectuées, entre un octet du mot de code et un élément de la matrice de parité $\mathbf{H}$, résultant en une valeur de syndrome intermédiaire (ligne 2). Le calcul de chaque octet de syndrome implique l'opération XOR de chaque syndrome intermédiaire à la position correspondante dans chaque fenêtre. Enfin, nous représentons l'étape d'initialisation (ligne 1) par une opération XOR avec C_1 sur chaque octet de syndrome.

Le graphe de facteurs présenté dans figure 7.4 modélise les relations entre chaque valeur intermédiaire utilisée lors du calcul. Dans une utilisation normale d'un décodeur, le syndrome de sortie donne des informations sur l'erreur aléatoire ajoutée au mot de code. Mais ici, nous considérons le syndrome RS comme nul (voir section 3.5), ce qui permet de supprimer la partie inférieure du graphe. LA construction se modélise donc par $n - 1$ fenêtres, chacune représentant un graphe arborescent indépendant et bénéficiant de la preuve de convergence donnée par la théorie de la BP pour une telle topologie de graphe (voir section 4.5). Nous rappelons que la stratégie de re-décodage est disponible pour l'attaquant, lui permettant de récupérer C_1, le premier octet du mot de code qui n'est manipulé dans aucune des fenêtres.

Construction du sous-graphe `gf_mul` La principale sous-opération effectuée lors du calcul du syndrome RS est `gf_mul`, la multiplication dans le corps de Galois $\mathbb{F}_{2^8}$. Cette opération peut être réalisée à l'aide d'une multiplication rapide basée sur la transformée de Fourier rapide (FFT) (voir Algorithme 25) [BGTZ08], qui est le choix des auteurs de HQC depuis avril 2023 dans l'implémentation de référence [AMAB$^+$23]. Cependant, ce calcul peut être effectué différemment, en utilisant la représentation logarith-

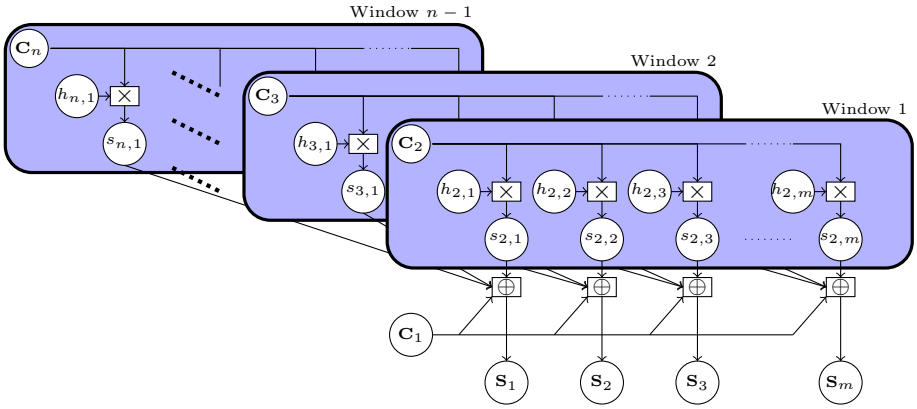


FIGURE 7.4 – Graphe de facteurs du décodeur Reed-Solomon (calcul du syndrome) (avec $m = n - k$).

mique de chaque élément, telle qu'elle était décrite dans l'implémentation de référence de juin 2021 (voir Algorithme 21) : $v := a \times b = \alpha^{\texttt{log}(a)} \times \alpha^{\texttt{log}(b)} = \alpha^{(\texttt{log}(a)+\texttt{log}(b))\%n}$, où α est un élément primitif du corps de Galois. Après cette transformation, si les conversions `log` et `exp` (stockées avec des tables pré-calculées en pratique) sont connues, la multiplication peut être calculée par une simple addition et une réduction modulaire. Cette représentation graphique (voir figure 7.5) nous permet d'optimiser les calculs de mise à jour des nœuds du graphe. En effet, des tables pré-calculées sont utilisées pour accélérer le calcul des marginales de probabilités des facteurs logarithmiques, exponentielles et de réduction modulaire. L'opération d'addition est mise en œuvre avec une convolution, qui peut bénéficier d'une FFT en fonction de la taille du domaine des variables.

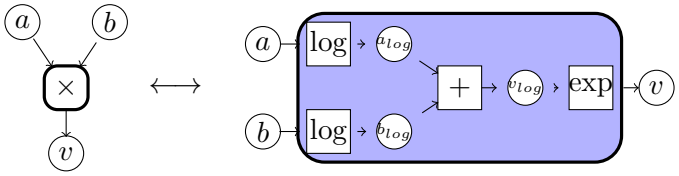


FIGURE 7.5 – Sous-graphe de multiplication dans le corps de Galois. Les facteurs sont indiqués par un carré et les variables par un cercle.

7.3.2 Simulation des Fuites de Poids de Hamming

Dans un premier temps, nous effectuons des simulations sur le graphe du décodeur. Nous avons initialisé les probabilités marginales du deuxième opérande avec les valeurs retournées par notre attaque en profilage provenant

des matrices de prédiction (voir sous-section 7.2.3). Cet opérande donne des informations sur les éléments de la matrice de contrôle de parité. Nous avons également initialisé les probabilités marginales des sorties de toutes les opérations `gf_mul` à partir d'un modèle de fuite de poids de Hamming avec un bruit gaussien. Ainsi, pour une mesure physique x résultant de la manipulation d'une sortie `gf_mul` v, nous avons :

$$x = \alpha \cdot \texttt{HW}(v) + \mathcal{N}(\beta, \sigma^2) \tag{7.3}$$

Avec ce modèle de fuite, nous pouvons simuler la sortie d'un classifieur sur une valeur suivant un modèle de fuite en poids de Hamming parfait (voir sous-section 4.1.1), avec les équations suivantes :

$$\mathbb{P}\left(\texttt{guess} = v \mid \texttt{label} = l\right) = \mathbb{P}(X = v), X \rightsquigarrow \mathcal{N}(l, \sigma^2) \tag{7.4}$$

Résultats de simulation La figure 7.6 montre le taux de réussite de l'attaque en augmentant la valeur de l'écart type du bruit σ étape par étape, et en effectuant l'attaque 400 fois pour chacune d'elles. Jusqu'à un écart type de 2, nous avons un taux de réussite de 100% pour les niveaux de sécurité 128 et 192. Pour le plus haut niveau de sécurité de HQC, nous pouvons atteindre un niveau de bruit de presque $\sigma = 3$ sans perte de précision.

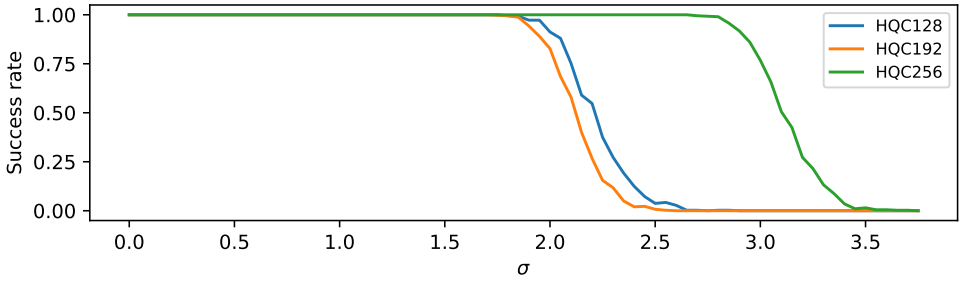


FIGURE 7.6 – Taux de réussite simulé de SASCA sur le décodeur, avec la stratégie de re-décodage, en fonction du niveau de sécurité sélectionné de HQC.

Discussion Nous observons que le taux de réussite pour HQC-192 est inférieur à celui pour HQC-128. En effet, le nombre d'octets à récupérer est plus important (56 au lieu de 46), le nombre d'essais indépendants reste presque le même (32 au lieu de 30) et la capacité de correction d'erreur est la même (*c.-à-d.*, 19). Cela rend les attaques plus difficiles à mener compte tenu de équation 7.2.

Nous nous attendions à ce que l'attaque sur HQC-256 ait un taux de réussite supérieur à celui de HQC-128. En effet, le nombre d'octets du mot de code à récupérer est beaucoup plus important pour ce niveau de sécurité

($n = 90$ au lieu de 46 ou 56), mais le nombre d'essais indépendants est également plus important ($n - k = 58$ au lieu de 30 ou 32), et la capacité de correction d'erreur augmente (36 au lieu de 19). L'attaque doit trouver le double d'octets du mot de code, mais dispose du double de fuites indépendantes sur chacun d'entre eux, et est finalement aidée par une forte capacité de correction.

Une stratégie pour réduire le taux de succès de l'attaque pourrait être de modifier les paramètres de codes de HQC afin de diminuer la probabilité de succès donnée dans équation 7.2. Cela peut être réalisé en diminuant la valeur de la capacité de correction d'erreur. Cette option semble être en contradiction avec le choix attendu pour un code $\mathcal{C}$ pour HQC afin de réduire les tailles de texte chiffré et de clef du schéma. Une autre option peut consister à augmenter n, le nombre d'octets du mot de code à récupérer, sans augmenter $n-k$, le nombre de fuites indépendantes sur chacun d'entre eux. Cela peut être fait en sélectionnant un code avec des taux plus élevés $R = \frac{k}{n}$. Attention cependant car les les suggestions présentées ici nécessitent de repenser les paramètres de HQC, ce qui pourrait être impossible ou avoir un impact significatif sur la sécurité ou les performances du schéma.

7.4 Attaque de *Codeword Masking*

Une stratégie de masquage *Codeword Masking* (voir sous-section 6.5.2)à déjà été proposée pour protéger le décodeur de HQC contre des attaques physiques (voir chapitre 6). Cette stratégie de masquage permet de créer un masque pour le décodeur en utilisant un encodeur. Au lieu de décoder $c + e$ en m, on commence par tirer aléatoirement un masque de message m'. Ce masque de message est encodé en c', le masque de mot de code. Ensuite, l'algorithme de décodage est appliqué sur $c + c' + e$, masquant les données sensibles c, renvoyant $m + m'$ en raison de la linéarité du code impliqué. Le vrai résultat m est récupéré en soustrayant le masque de message m' au résultat précédent. Étant donné que l'encodeur est une opération rapide par rapport au décodeur, *Codeword Masking* permet de réduire le cout de la contremesure de masquage.

Étant donné que la contremesure n'est pas la répétition de la même opération, mais bien l'appel à deux opération distinctes, la sécurité du masquage de mot de code nécessite une étude approfondie de l'algorithme d'encodage. Par conséquent, une attaque ciblant une implémentation masquée de HQC peut être réalisée en deux étapes : *(i)* attaquer le décodeur pour récupérer $c + c'$, la clef partagée masquée, et *(ii)* attaquer l'encodeur pour récupérer le masque c'. Dans ce scénario, le taux de réussite est le produit des deux taux de réussite de ces deux attaques. Le première attaque est déjà abordée dans la section précédente (voir section 7.3). Dans cette section, nous décrivons une approche SASCA contre l'encodeur RS.

7.4.1 Graphe de l'Encodeur Reed-Solomon

La figure 7.7 donne une représentation graphique de Algorithme 11. Afin de représenter toutes les valeurs intermédiaires de l'algorithme, la boucle *for* (ligne 2) est dépliée. Chaque ligne du graphe correspond à une itération de cette boucle. Les valeurs intermédiaires dites *gate values*, notées Γ (ligne 3) sont représentées du côté gauche du graphe. À partir de la deuxième ligne du graphe, elles dépendent de l'élément le plus à droite du tableau, dont l'addition est indiquée par une flèche numérotée sur le graphe. Chaque élément du rectangle bleu représente la sortie de multiplication dans le corps de Galois (`gf_mul`) de la *gate value* correspondante sur la même ligne et de l'élément correspondant du polynôme générateur dans la même colonne (ligne 5). Enfin, ces éléments sont ajoutés en diagonale (XOR) pour produire les octets de redondance (ligne 6-7) en bas du graphe. Ces octets sont ensuite concaténés avec le message initial pour former le mot de code de sortie (ligne 9).

Tout comme le décodeur, l'opération principale effectuée par l'encodeur est `gf_mul`, la multiplication dans le corps de Galois. De plus, cet algorithme d'encodage nécessite la connaissance de g, un polynôme générateur, publiquement connu comme un paramètre de HQC. Cette connaissance préalable peut être implémentée dans le graphe de facteurs. Enfin, l'attaque vise également à récupérer les octets du mot de code RS, permettant d'appliquer la stratégie de re-décodage (voir sous-section 7.2.5).

Comme dans la sous-section 7.2.3, nous avons réutilisé les résultats de profilage pour effectuer des attaques pratiques. Les simulations suivent la théorie du modèle de fuite de poids de Hamming de sous-section 4.1.1, dont les résultats sont dans figure 7.8.

Résultats Les résultats de simulation affichés dans figure 7.8 montrent que l'attaque sur l'encodeur est plus sensible au bruit que celle du décodeur (voir figure 7.6). Nous pensons qu'en pratique, le décodeur masqué n'est pas sécurisé car nous sommes toujours capables de récupérer le masque avec une probabilité de 0.7625, 0.6575 et 0.8075 pour HQC-128, HQC-192 et HQC-256 respectivement.

Discussion La sensibilité de cette attaque au bruit peut potentiellement s'expliquer par les relations éparses entre les valeurs intermédiaires dans le graphe de l'encodeur, ainsi que par les cycles à l'intérieur de ce dernier. Les travaux futurs pourraient se concentrer sur des techniques d'optimisation telles que l'amortissement ou la planification des messages (voir section 4.5) Néanmoins, on observe toujours des taux de réussite plus élevés contre HQC-256, le plus haut niveau de sécurité, que ce soit en simulations ou dans les attaques réelles.

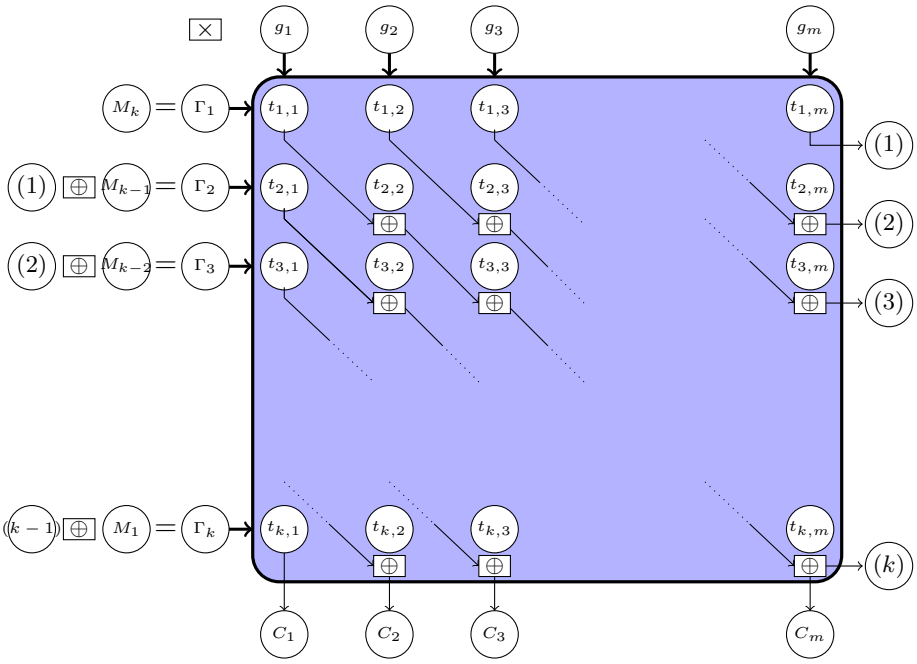


FIGURE 7.7 – Encodeur RS vu comme un graphe (avec $m = n - k$).

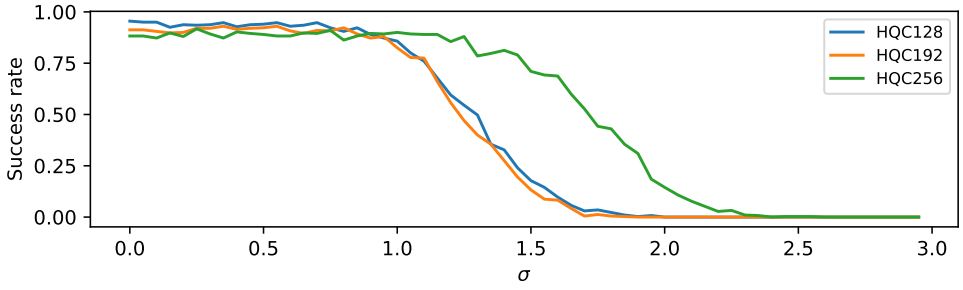


FIGURE 7.8 – Taux de réussite de SASCA contre l'encodeur Reed-Solomon, avec la stratégie de re-décodage, en fonction du niveau de sécurité sélectionné pour HQC.

Stratégie de masquage d'ordre élevé En tirant N masques aléatoires et en les ajoutant ensemble, on peut générer un masquage d'ordre élevé (voir sous-section 6.5.2). Avec une telle construction et une probabilité de succès telle que observé dans la section précédente, il faudrait utiliser un masque d'ordre au moins 17, 11 ou 22, pour sécurisé respectivementHQC-128, HQC-192 et HQC-256. Ce nombre de masque permettrait de garantir que la probabilité de succès de l'attaque générale contre HQC soit inférieur à 0.01. Cette approche ne semble pas être efficace en raison du nombre élevé de masque nécessaire.

Stratégie de masquage alternative Dans ce travail, nous avons uniquement considéré *Codeword Masking*, qui est une forme très spécifique de masquage à un niveau élevé. Il serait intéressant de considérer l'effet du masquage à un niveau inférieur, par exemple en masquant directement la multiplication dans le corps de Galois elle-même. Cette approche porte le nom de gadget et semble une famille de contremesure très prometteuse contre des modèles d'adversaires très robustes, notamment définis par le *t-probing*. De manière similaire à ce qui a été fait pour Dilithium [ABC+22], cette approche pourrait être une manière efficace de protéger HQC contre notre attaque.

7.5 Attaques des Contremesures de Mélange

L'approche SASCA a déjà ciblé la NTT de Kyber [BDK+18] dans divers travaux [PPM17, PP19, HHP+21]. Deux contremesures de mélange, le mélange "grossier" (*coarse*) et le mélange fin (*fine*) [RPBC20], ont été identifiées pour protéger la NTT contre SASCA. Le mélange fin vise à mélanger l'ordre des entrées et des sorties de la NTT, en sélectionnant aléatoirement l'une des 4 combinaisons à chaque appel. Cette stratégie empêche un attaquant d'étiqueter les fuites observées. Le mélange grossier quant à lui consiste à mélanger les éléments de la boucle interne, indépendamment dans chaque couche. Ces stratégies de mélange peuvent être adaptées pour une utilisation dans HQC. En effet, la couche de la NTT se comporte comme les fenêtres du décodeur RS. Le mélange grossier peut être utilisé pour mélanger l'ordre des éléments dans une fenêtre (voir figure 7.4). Le mélange fin peut être utilisé pour mélanger les entrées de `gf_mul`, puisque la sortie est unique, le nombre de combinaisons est simplement de 2.

Nous pouvons également déduire de nouvelles méthodes de mélange pour HQC. Il est possible de calculer chaque fenêtre dans un ordre aléatoire. Cette stratégie est inutile pour la NTT car toutes les couches effectuent la même opération. Cependant, pour le décodeur RS, les fenêtres sont parfaitement indépendantes et peuvent donc être effectuées dans un ordre aléatoire. Nous appelons cette contremesure le mélange de fenêtres (*Window Shuffling*).

Enfin, toutes les opérations `gf_mul` étant indépendantes au cours du calcul, elles peuvent être effectuées dans un ordre entièrement aléatoire, suivant les idées de [ATT+18, CFG+10].

Dans cette section, nous décrivons et analysons la sécurité de ces contre-mesures de mélange.

7.5.1 Mélange Fin

Sous une stratégie de mélange fin, les données sensibles (c'est-à-dire, l'octet du mot de code) sont manipulées sous le premier opérande une fois sur deux en moyenne. Nous réutilisons une stratégie de vote majoritaire de sous-section 7.2.4 pour exploiter le niveau de fuite élevé observé sur le premier opérande. Nous considérons que la sortie du classificateur est une valeur aléatoire lorsque les données sensibles sont cachées derrière le second opérande. Cette hypothèse est un scénario plus défavorable que ce que nous observons en pratique (voir tableau 7.1). En effet, la fuite sur les valeurs de la matrice de parité pourrait aider pour une analyse plus fine. Cependant, si la probabilité que la bonne hypothèse soit classée en première position par le classificateur est suffisamment élevée, le vote majoritaire réussira.

Discussion L'utilisation de la stratégie de mélange fin va à l'encontre du souhait de cacher les données sensibles sous l'opérande qui fuit le moins, comme discuté dans section 7.2. Cette stratégie de mélange n'est donc pas efficace dans notre scénario d'attaque.

7.5.2 Mélange Grossier

La stratégie de mélange grossier vise à mélanger l'ordre des opérations effectuées dans chaque fenêtre. Le mélange sélectionné peut être modifié pour chaque fenêtre, assurant un meilleur niveau de sécurité. L'ordre des opérations à l'intérieur des fenêtres n'affecte pas la construction du graphe (voir sous-section 7.3.1, ni la convergence de ce dernier vers le mot de code cible. Cette assertion est vraie car les $n-1$ fenêtres sont des sous-graphes indépendants (voir section 7.3). Nous rappelons que la valeur de C_1 est récupérée avec la stratégie de re-décodage finale (voir sous-section 7.2.5). Par conséquent, ce cas correspond à notre configuration précédente de BP, donnant les mêmes résultats que l'attaque précédente du décodeur (voir section 7.3).

7.5.3 Mélange de Fenêtres

Le mélange de fenêtres permet d'intervertir l'ordre des calculs des octets du mot de code. Dans un tel cas, même si nous sommes capables de converger avec une attaque BP, les octets du mot de code récupérés sont

mélangés et l'attaque ne réussit pas à moins que la permutation ne soit inversée. Nous commençons par appliquer la même stratégie d'attaque (voir section 7.3), indépendamment de la permutation des fenêtres. Les fenêtres étant des graphes indépendants, cette étape produit le même résultat que précédemment : on retrouve la valeur de chaque octet du mot du code, mais dans un ordre permuté. Par conséquent, la difficulté d'attaquer cette contremesure de mélange réside dans l'inversion de la permutation aléatoire.

Cependant, notre attaque nous permet également de retrouver des marginales de probabilités concernant les élément de la matrice de parité $\mathbf{H}$ connue publiquement. Ces éléments sont également touché par la permutation appliquée sur les fenêtres. Les marginales de probabilités retrouvées sur ces éléments vont nous permettre d'inverser la permutation.

Inversion de la permutation des octets du mot de code La matrice de contrôle de parité $\mathbf{H} = (h_{i,j})_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n-k}}$ peut être transformée en une distribution de probabilités de Dirac sous la matrice T de taille $k \times n \times 256$:

$$T[i,j,l] = \begin{cases} 1 & \text{si } h_{i,j} = l \\ 0 & \text{sinon} \end{cases} \tag{7.5}$$

Nous savons que les lignes de la matrice de contrôle de parité ont été mélangées par le mélange de fenêtres, mais dans chaque ligne, les éléments ont conservé leur arrangement d'origine. Après la première phase de BP, nous obtenons une estimation de T qui est mélangée aléatoirement. Nous notons cette estimation $\tilde{T}$, qui contient les marginales de probabilités de chaque variable représentant $\mathbf{H}$. Plus formellement, si L représente une mesure de canal secondaire :

$$\tilde{T}[i,j,l] = \mathbb{P}\left(h_{i,j} = l \mid L\right) \tag{7.6}$$

L'idée est de réaffecter les lignes de $\tilde{T}$ afin de minimiser une distance avec T. Pour ce faire, nous calculons la matrice D telle que :

$$D[i,i'] = \sum_{j=1}^{256} \mathtt{d}(\tilde{T}[i,j],\ T[i',j]) \tag{7.7}$$

où `d` est une fonction de distance arbitraire. Inverser le mélange de fenêtres revient à sélectionner k éléments de la matrice D. Exactement un élément par ligne et un élément par colonne de manière à ce que la somme de ces éléments soit minimale. La localisation de ces éléments sélectionnés donne l'association entre $\tilde{T}$ et les lignes de T. Ce problème est une instance du problème d'affectation (*Assignement problem*), pour lequel une technique de résolution optimale est connue.

Problème d'affectation Le problème d'affectation est un problème d'optimisation classique dans le domaine de la recherche opérationnelle et de la programmation linéaire. Il s'agit de trouver l'affectation optimale d'un ensemble de tâches à un ensemble d'agents de manière à minimiser le coût total ou le temps nécessaire pour accomplir les tâches, ou inversement, à maximiser le profit total. Chaque tâche doit être affectée à exactement un agent, et chaque agent ne peut être affecté qu'à une seule tâche.

Algorithme Hongrois L'algorithme Hongrois (ou *Hungarian Algorithm* en anglais) est une méthode efficace pour résoudre le problème d'affectation, surtout lorsque le problème implique un nombre égal de tâches et d'agents. Il a été développé par Harold Kuhn [Kuh55] dans les années 1950 et a ensuite été perfectionné par James Munkres [Mun57]. Nous l'avons appliqué en considérant $\tilde{T}$ résultant d'une fuite simulée pour étudier le comportement de l'algorithme en présence de bruit. Plusieurs mesures de distance ont été évaluées pour équation 7.7 : la distance L_1 [1] a donné les meilleurs résultats.

Après la méthode hongroise, et la ré-affectation des lignes, on peut admettre que les valeurs provenant de la matrice de parité sont connus avec certitude. Il est donc possible de ré-initialiser le graphe en inversant la permutation sur les fenêtres en insérant des connaissances parfaites sur les nœuds représentant les éléments de la matrice de parité. Cette seconde phase de BP, qui n'est pas forcément nécessaire si les résultat de la première convergence était satisfaisants, permet d'améliorer la précision de l'attaque.

7.5.4 Mélange Complet

Un mélange plus fort est introduit en combinant les idées du mélange de fenêtres et du mélange grossier. Ces deux méthodes de mélange peuvent être appliquées indépendamment comme dans [GLG22b], mais cela peut conduire à des attaques d'inversion de permutation. Par conséquent, elles sont plutôt utilisées conjointement en tirant de manière complètement aléatoire l'ordre des opérations `gf_mul` pendant le calcul du syndrome de RS. Cette stratégie suit une idée de [ATT+18] et vise à augmenter la complexité combinatoire pour l'attaquant. L'Algorithme 26 présente le calcul du syndrome protégé avec cette méthode de mélange. Cette stratégie que nous appelons "mélange complet" a un surcout qui est le cout de mélange d'une liste de taille $n \times (n-k)$.

Complexité de l'inversion du mélange complet Supposons que nous soyons capables de récupérer, avec une attaque BP ou autre, la valeur exacte du deuxième opérande, provenant de la matrice de parité. Étant donné cette

1. La distance L_1 (également appelée distance de Manhattan ou de taxicab) entre deux vecteurs $\mathbf{x} = (x_1, \cdots, x_n)$ et $\mathbf{y} = (y_1, \cdots, y_n)$ de même longueur est donnée par $\mathtt{d}_{L_1}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$.

Algorithm 26 Calcul du syndrome des codes RS avec un mélange complet.

Require: paramètres : k, n la dimension et la longueur du code
Require: matrice de parité $H \in \mathbb{F}_q^{(n-k,n)}$
Require: mot de code $C \in \mathbb{F}_q^n$
Ensure: $s := H^T \times c$ le syndrome de c
1: Initialiser s à C_1^{n-k}
2: $\mathcal{L}_1 := \{1, \cdots, n-k\}$
3: $\mathcal{L}_2 := \{2, \cdots, n\}$
4: $\mathcal{L} = \texttt{shuffle}(\mathcal{L}_1 \times \mathcal{L}_2)$
5: **for** $(i, j) \in \mathcal{L}$ **do**
6: $\quad s[i] = s[i] \oplus \texttt{gf\_mul}\,(c[j] \times H[i, j-1])$

information, nous voulons inverser le mélange de ces éléments. Cependant, la taille de la matrice est beaucoup plus grande que la taille du corps de Galois, entrainant une grande redondance. Par conséquent, il est impossible d'inverser la permutation sans tester toutes les possibilités pour les éléments redondants. Étant donné que l'on connait la matrice de parité, on peut calculer ce nombre de permutations pour tous les niveaux de sécurité de HQC. Notez que ce nombre augmente avec la taille de la matrice, donc avec le niveau de sécurité, car le corps de Galois reste le même. Ce nombre de permutations est respectivement de 2^{504}, 2^{614} et 2^{1030} pour les trois niveaux de sécurité de HQC. Ce nombre étant plus grand que le niveau de sécurité, nous concluons que l'inversion du mélange n'est pas réalisable avec la stratégie présentée dans section 7.5. Cela nous conduit à penser que le mélange complet est une contre-mesure efficace contre notre attaque.

Discussion Une stratégie pour contourner la contre-mesure du mélange complet pourrait consister à attaquer la génération aléatoire de permutations. Cette stratégie sort du cadre de cette attaque et de mon domaine de compétence. Toutefois, si cette contremesure est choisie pour protéger une implémentation cryptographique, il faudra faire attention à l'implémenter de telle sorte à être elle-même résistante aux attaques connues afin d'éviter que l'inversion de permutation ne soit trop facile à réaliser.

7.6 Attaque de la Décapsulation

La transformée HHK utilisée pour HQC, variation de la transformée FO, implique une étape de re-chiffrement lors de la décapsulation. Ainsi, une clef partagée décodée est également ré-encodée pendant la re-chiffrement. Cette étape supplémentaire permet d'exploiter les fuites à la fois d'un décodeur et d'un encodeur pendant le même processus de décapsulation.

Combinaison des graphes de décodeur et d'encodeur Nous sommes capables de construire un double graphe, créant une connexion entre les graphes d'encodeur et de décodeur. En effet, ces deux graphes partagent les mêmes nœuds de variables d'octets de mot de code (C_i), qui peuvent donc être fusionnés. Nous suivons la stratégie de simulation de sous-section 7.3.2, les résultat sont présentées dans la figure 7.9. Nous montrons que nous sommes capables d'atteindre des niveaux de bruit plus élevés que toutes les attaques précédentes de ce chapitre, et ce pour tous les niveaux de sécurité de HQC.

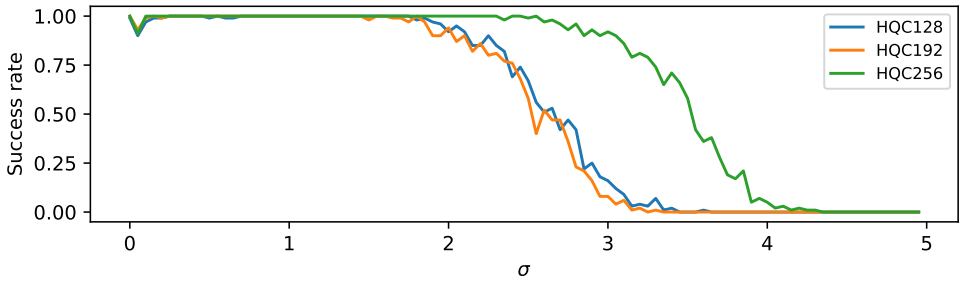


FIGURE 7.9 – Taux de réussite de SASCA sur la décapsulation (décodeur + encodeur combinés), avec une stratégie de re-décodage, en fonction du niveau de sécurité sélectionné pour HQC.

Contremesure Cette attaque combinée qui exploite la redondance de fuites impliqué par l'étape de re-chiffrement, constitue une menace pour la sécurité de HQC. Ainsi, il est nécessaire de trouver une contremesure à la fois pour l'encodeur et le décodeur. L'algorithme actuel de l'encodeur RS (voir Algorithme 11) est implémenté avec une division polynomiale. Protéger cet encodeur avec une stratégie de mélange est une tâche difficile, car la propagation des retenues implique que plusieurs opérations `gf_mul` dépendent du résultat des précédentes. En considérant l'implémentation actuelle de l'encodeur, la stratégie de mélange complet ne peut pas être appliquée directement. Notre idée pour protéger l'encodeur est de changer l'algorithme d'encodage pour un encodage classique de type multiplication de matrice-vecteur (voir chapitre 2) La stratégie de mélange complet peut alors être appliquée, offrant une complexité combinatoire suffisante pour empêcher notre attaque. Le changement de l'algorithme de l'encodeur permet de protéger à la fois l'encodeur et le décodeur avec la même contre-mesure de mélange.

7.7 Conclusion et Travaux Futurs

En fonction des choix d'implémentation de HQC, nos attaques peuvent être soit une attaque classique par profilage ou se baser sur une attaque de

propagation des croyances (BP).

Pour toutes nos attaques pratiques, nous avons utilisé la configuration présentée dans la sous-section 7.2.1. Ces attaques sont effectuées en quelques minutes sur une cible STM32F407 exécutant l'implémentation de référence de HQC [AMAB+23]. Pour chaque niveau de sécurité, chaque attaque a été répétée 400 fois. Nous atteignons une précision parfaite pour chaque attaque, à l'exception de l'attaque de l'encodeur qui présente des taux de réussite supérieurs à 65%. Nous insitons sur le fait que ces attaques sont une menace pour HQC et des contre-mesures efficaces doivent être appliquées. Ce travail tire parti de la structure interne et des propriétés de la cryptographie basée sur les codes pour monter des attaques pratiques de récupération de clef partagée.

— Nous démontrons que des attaques pratiques sont faisable contre le décodeur de Reed-Solomon RS de HQC. Plus précisément, nous exploitons les fuites physiques lors de la multiplication dans le corps de Galois, une opération clé de la logique RS, et modélisons les dépendances entre les variables intermédiaires dans un graphe de facteurs. Nous avons simulé cette attaque avec un modèle de fuites de poids de Hamming et montré que le taux de réussite reste élevé (supérieur à 0.9) jusqu'à $\sigma = 2$ et même $\sigma = 3$ pour le niveau de sécurité le plus élevé de HQC (voir figure 7.6). En pratique, cette attaque a un taux de réussite de 100%.

— Nous effectuons la même analyse contre une version de HQC protégée par masquage de mot de code. Plus précisément, la robustesse de l'encodeur RS contre SASCA est étudiée. Il en ressort que l'attaque de l'encodeur est plus sensible au bruit, ce qui peut potentiellement s'expliquer par les relations eparses entre les valeurs intermédiaires, ainsi que par les cycles dans le graphe de l'encodeur. Les résultats de simulation sont représentés dans figure 7.8 avec de bonnes précisions jusqu'à $\sigma = 1$. En pratique, notre attaque atteint des taux de réussite de 76.25%, 65.75% ou 80.75% selon le niveau de sécurité sélectionné. Ces taux de réussite suffisent à menacer la sécurité du schéma. *Codeword Masking* n'est pas une contre-mesure efficace pour protéger HQC contre SASCA sur un STM32F407.

— Nous analysons la sécurité de plusieurs contre-mesures de mélange du décodeur RS contre nos attaques. Nous démontrons la protection insuffisante apportée par les contre-mesures de mélange adaptées de la littérature liée à Kyber. Nous atteignons une précision parfaite dans un scénario d'attaque réelle. Nous présentons la stratégie de mélange complet qui apporte une complexité combinatoire satisfaisante aux attaques proposées dans cet article. La stratégie de mélange complet du décodeur RS est une contre-mesure intéressante qui pourrait éventuellement contrecarrer d'autres attaques.

— Enfin, en exploitant la transformation Fujisaki-Okamoto FO, un attaquant peut combiner les fuites de l'encodeur et du décodeur en fusionnant les deux graphes de facteurs, pour une récupération réussie de la clé partagée sur des dispositifs avec des niveaux de bruit plus élevés (voir figure 7.9). Encore une fois, dans un scénario pratique, notre attaque a un taux de réussite de 100%. L'attaque combinée exploitant la redondance des fuites créées par la ré-encapsulation est une menace potentielle pour tout schéma de type FO. Nous montrons qu'un changement dans la stratégie d'encodage des codes RS de HQC permet de protéger à la fois l'encodeur et le décodeur avec le mélange complet.

L'analyse du code de RS interne de HQC à travers le regard d'un attaquant exploitant les fuites physiques ouvre plusieurs perspectives pour des travaux futurs. Tout d'abord, comme toutes nos attaques exploitent la multiplication dans le corps de Galois, nous croyons que protéger cette opération, avec un masque de bas niveau, est une voie prometteuse vers des contremesures efficaces. Une option pourrait être de mettre en œuvre un gadget [BBE$^+$18] pour `gf_mul`, assurant la sécurité des opérations dans la manipulations des codes de RS sous un modèle d'attaquant donné. Deuxièmement, l'algorithme complet de mélange doit être soigneusement sélectionné, en particulier le générateur aléatoire, pour éviter les attaques de récupération de permutation. Enfin, la résilience d'autres schémas PQC construits avec la transformation FO doit être évaluée contre des approches SASCA analogues à l'attaque de la décapsulation présentée dans cet article. Les attaques combinant la redondance des fuites créées par la ré-encapsulation pourraient constituer une menace pour les schémas FO.

CHAPITRE 8

CONCLUSION ET PERSPECTIVES

Contents

8.1 Conclusions

Les travaux présentés dans ce manuscrit traitent de la question de la sécurité de la cryptographie post-quantique face aux attaques par canaux auxiliaires. Ce sujet de recherche a été motivé par différents points :

— Le développement de l'ordinateur quantique représente une menace pour la sécurité de la cryptographie utilisée actuellement, dite cryptographie classique. Cette menace crée un besoin d'une nouvelle cryptographie : la cryptographie post-quantique.

— Le constat que les attaques physiques ont également un impact sur la sécurité des algorithmes cryptographiques. Connaître et savoir se protéger face à ces attaques est une étape essentielle afin de déployer des solutions adaptées et sécurisées.

Les travaux présentés dans ce manuscrit ont donc pour objectif d'étudier la cryptographie basée sur les codes correcteurs d'erreurs sous le prisme des attaques physiques. Plus précisément, les travaux se sont concentrés sur un schéma en particulier, Hamming Quasi-Cyclic (HQC). Durant la thèse, différentes vulnérabilités au niveau de l'implémentation de référence de HQC ont été identifiées, notamment au niveau du décodeur. En effet, l'étude de l'état de l'art des attaques par canaux auxiliaires sur la cryptographie basée sur les codes montre que ce sont essentiellement les opérations de décodage qui sont visées, indépendamment du schéma.

Dans un premier temps, l'exploitation de la faible probabilité d'échecs du décodeur, ainsi que sa structure concaténée, a permis de monter une attaque contre HQC afin de retrouver la clé partagée. Pour cette attaque, une méthode basée sur l'analyse de corrélation (ou *glscpa*) et le bien connu coefficient de Pearson a été mise en place pour mener l'attaque. Cette attaque a permis de montrer une vulnérabilité dans l'implémentation de HQC. En effet, en exploitant les fuites observées pendant la mesure des ondes électromagnétiques lors de l'exécution du décodeur de RS, la complexité de recherche de la clé partagée de HQC a été réduite de 2^{128} opérations algébriques dans le corps de Galois à 2^{96}. Malgré la diminution drastique de l'espace de recherche, le coût de l'attaque est bien supérieur au nombre d'opérations acceptable dans le cadre d'une attaque. Ces travaux ont permis d'attester la présence d'une vulnérabilité dans l'implémentation de HQC et ont été publiés à WCC 2022.

L'attaque suivante vise la première partie du décodeur concaténé, dont la cible était la clé secrète de HQC. Des outils de *Machine Learning* ont été utilisés afin d'apprendre le comportement physique de la cible dans différents scénarios de décodage. Être capable de classifier ces scénarios de décodage permet de construire une attaque par canaux auxiliaires avec un taux de réussite parfait, permettant de retrouver la clé secrète. Cette attaque nécessite les mesures physiques de 20000 exécutions du schéma, contre 50000

pour l'attaque de l'état de l'art la plus proche. Ces travaux ont été publiés à International Conference on Post-Quantum Cryptography (PQcrypto) 2022.

Finalement, vers la fin de la thèse, les travaux se sont tournés vers les algorithmes de propagation de croyance, qui offrent une nouvelle manière d'attaquer les schémas cryptographiques. En appliquant cette classe d'attaque sur la vulnérabilité identifiée dans le premier papier, une attaque a pu être montée qui permet de retrouver la clé partagée de HQC en une seule mesure physique. Ces travaux ont fait l'objet d'une publication à CHES 2024.

Le but de ces travaux n'est pas seulement d'attaquer le schéma et d'en exploiter les vulnérabilités, mais aussi de montrer que ces attaques peuvent être évitées en mettant en place des contre-mesures efficaces. Pour chacune des attaques citées ci-dessus, au moins une contre-mesure permettant de déjouer l'attaque et de rendre le schéma plus robuste a été proposée. Afin de protéger l'attaque contre la clé secrète, la solution proposée est l'utilisation d'une permutation aléatoire des opérations sensibles, autrement dit un mélange. Pour protéger le décodeur interne contre les attaques visant à retrouver la clé, la linéarité de la fonction sensible peut être judicieusement utilisée afin de masquer les valeurs sensibles lors de leur manipulation.

8.2 Perspectives

Bien que les travaux présentés dans cette thèse couvrent un large panel d'attaques par canaux auxiliaires sur HQC, il reste encore plusieurs points méritant d'être approfondis.

Dans un premier temps, je vais évoquer les questions relatives à HQC. Durant ces travaux, un certain nombre de contre-mesures ont été proposées et évaluées dans le but de sécuriser HQC face à des attaques par canaux. Deux constats apparaissent :

— La plupart des contre-mesures proposées sont des contremesures "locales", dont le but est de protéger la vulnérabilité identifiée par une attaque spécifique. La question se pose de savoir si ces contre-mesures seront intéressantes dans un besoin de sécuriser globalement le schéma.

— Nous avons montré, dans notre troisième attaque, qu'une contremesure que nous pensions efficace au début de la thèse ne permettait pas, au final, de garantir une sécurité suffisante contre un attaquant avec des stratégies d'attaques différentes.

Ces constats tendent à montrer qu'une sécurité efficace contre les attaques par canaux auxiliaires passe par la mise en place de contre-mesures prouvées et assurées sous un certain modèle d'attaquant. Inspirés des travaux menés par Azouaoui et al. [ABC+22] sur Dilithium et par Demange et Rossi [DR24] sur BIKE, le masquage bas niveau, reposant sur le modèle

du t-probing, semble être la solution la plus prometteuse pour une sécurité pérenne face aux attaques par canaux auxiliaires.

Ces constats ouvrent aussi à la perspective de l'étude systématique des vulnérabilités de HQC. Cette thèse présente un ensemble de vulnérabilités qui peuvent être exploitées afin de monter des attaques par canaux auxiliaires. La question que l'on est en droit de se poser est *Combien en reste-t-il ?*. Afin de répondre à cette question, une étude approfondie de l'algorithmie de HQC, indépendamment d'une implémentation particulière, devra être menée pour identifier et protéger toutes les opérations à risque du schéma.

Écartons-nous maintenant un peu de HQC. Une des améliorations de l'attaque présentée à CHES 2024 exploite l'opération de re-chiffrement qui est nécessaire, pour des questions de sécurité, dans la structure de la transformée FO. Ce re-chiffrement ajoute de la redondance dans la manipulation de données sensibles, redondance qu'il est possible d'exploiter, dans le cadre de HQC, afin d'améliorer la force de l'attaque. La question est donc de savoir si *la transformée FO n'ajoute pas systématiquement une vulnérabilité au schéma lorsque l'on considère les attaques par canaux auxiliaires ? D'autres schémas sont-ils susceptibles d'être attaqués de la même manière ?*

Les techniques de propagation de croyance n'ont, à l'heure actuelle, été utilisées que contre les schémas post-quantiques Kyber et HQC. L'attaque présentée à CHES montre que cette approche est particulièrement efficace contre la cryptologie basée sur les codes correcteurs en raison de la présence forte de redondance dans la structure des données sensibles. *Est-il possible que cette approche soit adaptable à d'autres schémas basés sur les codes correcteurs d'erreurs tels que BIKE et ClassicMcEliece ? Est-il possible que cette approche soit adaptable à tout schéma cryptographique ?*

CHAPITRE 9

BIBLIOGRAPHIE

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779) :505–510, 2019.

[ABB+17] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. BIKE : Bit Flipping Key Encapsulation. 2017.

[ABB+23a] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, et al. PERK. 2023.

[ABB+23b] Gora Adj, Stefano Barbero, Emanuele Bellini, Andre Esser, Luis Rivera-Zamarripa, Carlo Sanna, Javier Verbel, and Floyd Zweydinger. MiRitH : Efficient post-quantum signatures from MinRank in the head. *Cryptology ePrint Archive*, 2023.

[ABB+23c] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, Jean-Pierre Tillich, et al. RYDE specifications. 2023.

[ABC+22] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Markus Schönauer, Tobias Schneider, François-Xavier Standaert, and Christine van Vredendaal. Protecting dilithium against leakage : Revisited sensitivity analysis and improved implementations. *Cryptology ePrint Archive*, 2022.

[ABCD+23] Nicolas Aragon, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. MIRA : a digital signature scheme based on the MinRank problem and the MPC-in-the-head paradigm. *arXiv preprint arXiv :2307.08575*, 2023.

[AEVR23] Guilhèm Assael, Philippe Elbaz-Vincent, and Guillaume Reymond. Improving single-trace attacks on the number-theoretic transform for Cortex-M4. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 111–121. IEEE, 2023.

[AGZ20] Nicolas Aragon, Philippe Gaborit, and Gilles Zémor. HQC-RMRS, an instantiation of the HQC encryption framework with a more efficient auxiliary error-correcting code. *arXiv preprint arXiv :2005.10741*, 2020.

[AMAB+] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC reference implementation. `https://pqc-hqc.org`.

[AMAB+17] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC). 2017.

[AMAB+20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Maxime Bros, Couvreur Alain, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Rank quasi-cyclic (RQC). 2020.

[AMAB+23] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC reference implementation, April, 2023. `https://pqc-hqc.org/implementation.html`.

[AMGH+23] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 564–596. Springer, 2023.

[ATT+18] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE international symposium on hardware oriented security and trust (HOST)*, pages 81–88. IEEE, 2018.

[Bar11] Morgan Barbier. *Décodage en liste et application à la sécurité de l'information*. PhD thesis, Ecole Polytechnique X, 2011.

[BBB+17] Magali Bardet, Élise Barelli, Olivier Blazy, Rodolfo Canto Torres, Alain Couvreur, Phillipe Gaborit, Ayoub Otmani, Nicolas Sendrier, and Jean-Pierre Tillich. Big quake. *NIST submissions*, 2017.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In *Advances in Cryptology–EUROCRYPT 2018 : 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 354–384. Springer, 2018.

[BCL+] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece : conservative code-based cryptography.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004 : 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber : a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[Ber24] Daniel J. Bernstein. Daniel J. Bernstein blog page, 2024. `https://blog.cr.yp.to/20220129-plagiarism.html`.

[BFR23] Ryad Benadjila, Thibauld Feneuil, and Matthieu Rivain. MQ on my mind : Post-quantum signatures from the non-structured multivariate quadratic problem. *Cryptology ePrint Archive*, 2023.

[BGTZ08] Richard P Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann. Faster multiplication in GF(2)[x]. In *Algorithmic Number Theory : 8th International Symposium, ANTS-VIII Banff, Canada, May 17-22, 2008 Proceedings 8*, pages 153–166. Springer, 2008.

[BHK+19] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20 : How 1+ 1= 0 improves information set decoding. In *Advances in Cryptology–EUROCRYPT 2012 : 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 520–536. Springer, 2012.

[BMG+24] Chloé Baïsse, Antoine Moran, Guillaume Goy, Julien Maillard, Nicolas Aragon, Philippe Gaborit, Maxime Lecomte, and Antoine Loiseau. Secret and Shared Keys Re-

covery on Hamming Quasi-Cyclic with SASCA. *Cryptology ePrint Archive*, 2024.

[BMVT78] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3) :384–386, 1978.

[CARG23] Agathe Cheriere, Nicolas Aragon, Tania Richmond, and Benoît Gérard. BIKE key-recovery : Combining power consumption analysis and information-set decoding. In *International Conference on Applied Cryptography and Network Security*, pages 725–748. Springer, 2023.

[CCD+20] Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Dragoi, Alexandre Menu, and Lilian Bossuet. Message-recovery laser fault injection attack on the classic McEliece cryptosystem. Cryptology ePrint Archive, Report 2020/900, 2020. `https://ia.cr/2020/900`.

[CDAMHT22] Kévin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0 : reducing decoding to LPN. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 477–507. Springer, 2022.

[CEVMS15] Cong Chen, Thomas Eisenbarth, Ingo Von Maurich, and Rainer Steinwandt. Differential power analysis of a McEliece cryptosystem. In *International Conference on Applied Cryptography and Network Security*, pages 538–556. Springer, 2015.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *International Conference on Information and Communications Security*, pages 46–61. Springer, 2010.

[CFG+11] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square always exponentiation. In *Progress in Cryptology–INDOCRYPT 2011 : 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings 12*, pages 40–57. Springer, 2011.

[CFMR+17] Antoine Casanova, Jean-Charles Faugere, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. *GeMSS : a great multivariate short signature*. PhD thesis, UPMC-Paris 6 Sorbonne Universités ; INRIA Paris Research Centre, MAMBA Team ..., 2017.

[Chr06] Chris Christensen. Cryptography of the Vigenère cipher. *Proceedings of Computer Sciences Corporation*, pages 1–18, 2006.

[CLG09] Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of CRYPTOLOGY*, 22(1) :93–113, 2009.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH : an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018 : 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 395–427. Springer, 2018.

[CRR03] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic hardware and embedded systems-CHES 2002 : 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.

[CW87] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

[DAT17] Thomas Debris-Alazard and Jean-Pierre Tillich. Statistical decoding. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1798–1802. IEEE, 2017.

[DCE16] A Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, faster, and more robust t-test based leakage detection. In *Constructive Side-Channel Analysis and Secure Design : 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers 7*, pages 163–183. Springer, 2016.

[DFKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign : compact post-quantum signatures from quaternions and isogenies. In *Advances in Cryptology–ASIACRYPT 2020 : 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I 26*, pages 64–93. Springer, 2020.

[DGK19] Nir Drucker, Shay Gueron, and Dusan Kostic. On constant-time qc-mdpc decoding with negligible failure rate. *Cryptology ePrint Archive*, 2019.

[DGK20] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In *International Conference on Post-Quantum Cryptography*, pages 35–50. Springer, 2020.

[DH76] WHITFIELD DIFFIE and MARTIN E HELLMAN. New Directions in Cryptography. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 22(6), 1976.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium : A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[DKSRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber : Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *Progress in Cryptology–AFRICACRYPT 2018 : 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10*, pages 282–305. Springer, 2018.

[DR24] Loïc Demange and Mélissa Rossi. A provably masked implementation of BIKE key encapsulation mechanism. *Cryptology ePrint Archive*, 2024.

[DRBL18] Vlad Drăgoi, Tania Richmond, Dominic Bucerzan, and Axel Legay. Survey on cryptanalysis of code-based cryptography : From theoretical to physical attacks. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*, pages 215–223. IEEE, 2018.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International conference on applied cryptography and network security*, pages 164–175. Springer, 2005.

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52. Moscow, 1991.

[EB22] Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In *IACR International Conference on Public-Key Cryptography*, pages 112–141. Springer, 2022.

[F+18] Richard P Feynman et al. Simulating physics with computers. *Int. j. Theor. phys*, 21(6/7), 2018.

[FEDS13] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. On the complexity of the generalized Min-

Rank problem. *Journal of Symbolic Computation*, 55 :30–58, 2013.

[FJR22] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head : shorter signatures from zero-knowledge proofs. In *Annual International Cryptology Conference*, pages 541–572. Springer, 2022.

[FKI06] Marc PC Fossorier, Kazukuni Kobara, and Hideki Imai. Modeling bit flipping decoding based on nonorthogonal check sums with application to iterative decoding attack of McEliece cryptosystem. *IEEE Transactions on Information Theory*, 53(1) :402–411, 2006.

[FMPR10] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In *International Workshop on Selected Areas in Cryptography*, pages 262–280. Springer, 2010.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.

[For65] G David Forney. Concatenated codes. 1965.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself : Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[Gab85] Ernest Mukhamedovich Gabidulin. Theory of codes with maximum rank distance. *Problemy peredachi informatsii*, 21(1) :3–16, 1985.

[Gab05] Philippe Gaborit. Shorter keys for code based cryptography. 2005.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this : Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 223–263, 2022.

[Gil52] Edgar N Gilbert. A comparison of signalling alphabets. *The Bell system technical journal*, 31(3) :504–522, 1952.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In *Annual International Cryptology Conference*, pages 359–386. Springer, 2020.

[GLG22a] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In *International Conference on Post-Quantum Cryptography*, pages 353–371. Springer, 2022.

[GLG22b] Guillaume Goy, Antoine Loiseau, and Phlippe Gaborit. Estimating the strength of horizontal correlation attacks in the hamming weight leakage model : A side-channel analysis on HQC KEM. In *WCC 2022 : The Twelfth International Workshop on Coding and Cryptography*, page WCC_2022_paper_48, 2022.

[GMGL23] Guillaume Goy, Julien Maillard, Philippe Gaborit, and Antoine Loiseau. Single trace HQC shared key recovery with SASCA. *Cryptology ePrint Archive*, 2023. `https://ia.cr/2023/1590`.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[GS98] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 28–37. IEEE, 1998.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In *Advances in Cryptology–EUROCRYPT 2018 : 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 385–412. Springer, 2018.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2) :147–160, 1950.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, 2021.

[HLS21] Clemens Hlauschek, Norman Lahr, and Robin Leander Schröder. On the timing leakage of the deterministic re-encryption in HQC KEM. Cryptology ePrint Archive, Report 2021/1485, 2021. `https://ia.cr/2021/1485`.

[HSC+23] Senyang Huang, Rui Qi Sim, Chitchanok Chuengsatiansup, Qian Guo, and Thomas Johansson. Cache-timing attack against HQC. *Cryptology ePrint Archive*, 2023.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 60–88, 2023.

[Jab01] A Al Jabri. A statistical decoding algorithm for general linear block codes. In *Cryptography and Coding : 8th IMA International Conference Cirencester, UK, December 17–19, 2001 Proceedings 8*, pages 1–8. Springer, 2001.

[JAC+17] David Jao, Reza Azarderakhsh, Matt Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalili, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. SIKE : Supersingular isogeny key encapsulation. 2017.

[JH04] Jørn Justesen and Tom Høholdt. *A course in error-correcting codes*, volume 1. European Mathematical Society, 2004.

[Kah96] David Kahn. *The Codebreakers : The comprehensive history of secret communication from ancient times to the internet.* Simon and Schuster, 1996.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2) :498–519, 2001.

[KJJ+98] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. Introduction to differential power analysis and related attacks. 1998.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99 : 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[Koc96] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96 : 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer, 1996.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of

convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[KPP20] Matthias J Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *Cryptology ePrint Archive*, 2020.

[Kuh55] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2) :83–97, 1955.

[LNPS19] Norman Lahr, Ruben Niederhagen, Richard Petri, and Simona Samardjiska. Side channel information set decoding using iterative chunking. Cryptology ePrint Archive, Report 2019/1459, 2019. `https://ia.cr/2019/1459`.

[MAB+17] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Gilles Zémor, and IC Bourges. Ouroboros-r. *NIST Submission*, 2017.

[Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244 :114–116, 1978.

[MDS99] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Investigations of power analysis attacks on smartcards. *Smartcard*, 99 :151–161, 1999.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 107–124. Springer, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 203–228. Springer, 2015.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting PUF error correction by codeword masking. *Cryptology ePrint Archive*, 2013.

[Mul54] David E Muller. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the IRE professional group on electronic computers*, (3) :6–12, 1954.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1) :32–38, 1957.

[NAB+19] M Naehrig, E Alkim, J Bos, L Ducas, K Easterbrook, B LaMacchia, P Longa, I Mironov, V Nikolaenko, C Peikert, et al. FrodoKEM : Learning with errors key encapsulation–algorithm specifications and supporting documentation. *NIST Technical Report*, 2019.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2) :157–166, 1986.

[NSP+23] Mohammad Reza Nosouhi, Syed W Shah, Lei Pan, Yevhen Zolotavkin, Ashish Nanda, Praveen Gauravaram, and Robin Doss. Weak-key analysis for BIKE post-quantum key encapsulation mechanism. *IEEE Transactions on Information Forensics and Security*, 18 :2160–2174, 2023.

[Ove06] Raphael Overbeck. Statistical decoding revisited. In *Information Security and Privacy : 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006. Proceedings 11*, pages 283–294. Springer, 2006.

[Per] Ludovic Perret. Biscuit : Shorter MPC-based signature from PoSSo.

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. *Post-Quantum Cryptography Project of NIST*, 2020.

[PK] Wilson Poulter and Justin Kulp. The ADFGVX cipher.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology–LATINCRYPT 2019 : 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017 : 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 513–533. Springer, 2017.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5) :5–9, 1962.

[PRD+16] Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. Differential power analysis attack on the secure bit permutation in the McEliece cryptosystem. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 132–137. IEEE, 2016.

[PT19] Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In *International Conference on Selected Areas in Cryptography*, pages 551–573. Springer, 2019.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[Ree54] Irving S Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, 4(4) :38–49, 1954.

[RHHM17] Mélissa Rossi, Mike Hamburg, Michael Hutter, and Mark E Marson. A side-channel assisted cryptanalytic attack against QcBits. In *Cryptographic Hardware and Embedded Systems–CHES 2017 : 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 3–23. Springer, 2017.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In *International Workshop on Information Security Applications*, pages 440–456. Springer, 2004.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable SCA countermeasures against single trace attacks for the NTT : A performance evaluation study over kyber and dilithium on the ARM Cortex-M4. In *Security, Privacy, and Applied Cryptography Engineering : 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*, pages 123–146. Springer, 2020.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR transactions on cryptographic hardware and embedded systems*, pages 307–335, 2020.

[RS60] Irving S Reed and 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, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, 1978.

[S+69] Volker Strassen et al. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4) :354–356, 1969.

[SGG24] Robin Leander Schröder, Stefan Gast, and Qian Guo. Divide and Surrender : Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks. *Cryptology ePrint Archive*, 2024.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3) :379–423, 1948.

[Sho94] Peter W Shor. Algorithms for quantum computation : discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SHR+22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the Reed-Muller Reed-Solomon version of the HQC cryptosystem. In *International Conference on Post-Quantum Cryptography*, pages 327–352. Springer, 2022.

[SKC+19] Bo-Yeon Sim, Jihoon Kwon, Kyu Young Choi, Jihoon Cho, Aesun Park, and Dong-Guk Han. Novel side-channel attacks on quasi-cyclic code-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 180–212, 2019.

[SRSWZ20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the CCA2-secure HQC KEM. In *19th Smart Card Research and Advanced Application Conference (CARDIS2020)*, 2020.

[Sta10] François-Xavier Standaert. Introduction to side-channel attacks. *Secure integrated circuits and systems*, pages 27–42, 2010.

[Sta19] François-Xavier Standaert. How (not) to use welch's *t*-test in side-channel security evaluations. In *Smart Card Research and Advanced Applications : 17th International Conference, CARDIS 2018, Montpellier, France, November 12–14, 2018, Revised Selected Papers 17*, pages 65–79. Springer, 2019.

[Ste89] Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications : 3rd International Colloquium Toulon, France, November 2–4, 1988 Proceedings 3*, pages 106–113. Springer, 1989.

[STM+08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, pages 216–229, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of complexity*, 13(1) :180–193, 1997.

[SV20] Nicolas Sendrier and Valentin Vasseur. On the existence of weak keys for QC-MDPC decoding. 2020.

[TH08] Stefan Tillich and Christoph Herbst. Attacking state-of-the-art software countermeasures—a case study for AES. In *Cryptographic Hardware and Embedded Systems–CHES 2008 : 10th International Workshop, Washington, DC, USA, August 10-13, 2008. Proceedings 10*, pages 228–243. Springer, 2008.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption : A generic power/EM analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 296–322, 2022.

[Var57] Rom Rubenovich Varshamov. Estimate of the number of signals in error correcting codes. *Docklady Akad. Nauk, SSSR*, 117 :739–741, 1957.

[Vas22] Valentin Vasseur. QC-MDPC codes DFR and the IND-CCA security of BIKE. 2022.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *Advances in Cryptology–ASIACRYPT 2014 : 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 282–296. Springer, 2014.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks : A comprehensive study with cautionary note. In *Advances in Cryptology–ASIACRYPT 2012 : 18th International Conference on the Theory and Application*

of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18, pages 740–757. Springer, 2012.

[VG99] Madhu Sudan Venkatesan Guruswami. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6) :1757–1767, 1999.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[Wal01] Colin D Walter. Sliding windows succumbs to big mac attack. In *Cryptographic Hardware and Embedded Systems—CHES 2001 : Third International Workshop Paris, France, May 14–16, 2001 Proceedings 3*, pages 286–299. Springer, 2001.

[WTBB+20] Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Etienne Marcatel. A practicable timing attack against HQC and its countermeasure. *Advances in Mathematics of Communications*, 2020.

[WWW23] Tianrui Wang, Anyu Wang, and Xiaoyun Wang. Exploring decryption failures of bike : New class of weak keys and key recovery attacks. In *Annual International Cryptology Conference*, pages 70–100. Springer, 2023.

ACRONYMES

AES Advanced Encryption Standard. 10, 81

AoI Area of Interest. 101, 103, 127

BCH Bose, Ray-Chaudhuri and Hocquenghem. 27, 36, 54, 85, 86

BGF Black-Gray-Flip. 48

BIKE Bit Flipping Key Encapsulation. 13, 46, 47, 48, 84, 85, 87, 89, 149, 150, XXIII, XXV

BP Belief Propagation. 78, 79, 80, 81, 124, 133, 140, 141, 142

CBC Cryptographie basée sur les codes. 38

CCA Chosen Ciphertext Attack. 82

CCE Codes Correcteurs d'Erreurs. 20, 21, 25, 28, 52, XXI

CHES Conference on Cryptographic Hardware and Embedded Systems. 15, 124, 149, 150

CPA Correlation Power Analysis. 61, 67, 76

CSIDH Commutative Supersingular Isogeny Diffie–Hellman. 14

CVP Closest Vector Problem. 13

D Problème du Décodage. 40, 44, 45

DES Data Encryption Standard. 10

DFR Decryption Failure Rate. 37, 51, 52, 54, 55, 86, 110

DPA Differential Power Analysis. 61, 75, 88

FFT Fast Fourier Transform. 125, 133, 134

FHT Fast Hadamard Transform. 57, 59, 95, 98, 101, 102, 104, 105, 106, XXI, XXII, XXV

FO Fujisaki-Okamoto. 9, 49, 54, 55, 84, 87, 89, 143, 146, 150

GBA Generalized Birthday Algorithm. 41

GV Gilbert-Varshamov. 26, 27

HHK Hofheinz, Hövelmanns and Kiltz. 49, 143

HQC Hamming Quasi-Cyclic. 13, 15, 16, 49, 50, 51, 52, 53, 54, 55, 56, 57, 70, 84, 85, 86, 87, 89, 93, 94, 95, 96, 99, 100, 101, 105, 107, 109, 110, 111, 112, 113, 118, 119, 120, 122, 124, 125, 126, 127, 131, 132, 133, 135, 136, 137, 138, 139, 143, 144, 145, 146, 148, 149, 150, XXI, XXII, XXIII, XXV, XXVI, XXVIII

HQC-PKE PKE version of HQC. 49, 51

HQC-KEM KEM version of HQC. 37, 49, 51, 96

IND-CPA INDistinguability under Chosen Plaintext Attacks. 52

IND-CCA INDistinguability under Chosen Ciphertext Attacks. 52

ISD Information Set Decoding. 37, 41, 42, 87, 88, 97

KEM Key Encapsulation Mechanism. 9, 46, 49, 81, 88, 110

LDA Linear Discriminant Analysis. 61, 77, 78, 94, 103, 129, 130

LDPC Low-Density Parity-Check. 48

LRA Linear Regrassion Analysis. 61, 68, 70, 71, 77, 128

MDS Maximal Distance Separable. 25, 30

MPC Multi-Party Computation. 14

NIST National Institute of Standards and Technology. 9, 12, 13, 14, 47, 48, 55, 81, 84, 89

NP Non-polynomial. 38

NTT Number Theoric Transform. 80, 81, 82, 83, 124, 139

PKE Public Key Encryption. 37, 44, 45, 46, 49

PKP Permuted Kernel Problem. 14

PoI Point of Interest. 77, 86, 101, 127

PQC Post-Quantum Crryptography. 1, 12, 45, 55, 81, 84, 89, 124, 146

PQcrypto International Conference on Post-Quantum Cryptography. 15, 149

PUF Physical Unclonable Function. 120

QC-MDPC Quasi-Cyclic Moderate Density Parity Check. 47, 87, 88

RM Reed-Muller. 28, 54, 55, 93, 94, 96, 98, 99, 100, 104, 105, 110

RMRS Codes de Reed-Muller et Reed Solomon concaténés. 54, 55, 86, 94, 105, 124, XXI

RQC Rank Quasi-Cyclic. 13

RS Reed-Solomon. 25, 27, 28, 29, 30, 36, 54, 56, 110, 111, 112, 115, 116, 124, 125, 130, 133, 136, 137, 139, 142, 143, 144, 145, 146, 148, XXV, XXVI

RSA Rivest, Shamir et Adleman. 9, 11, 12, 15, 66

RSS Residual Sum of Square. 71

SASCA Soft Analytical Side-Channel Attacks. 78, 81, 82, 124, 136, 139, 145, 146

SCA Side-Channel Attacks. 14, 65

SD Problème du Décodage de Syndrome. 37, 39, 40, 88

SHA Secured Hash Algorithm. 81

SIKE Supersingular Isogeny Key Encapsulation. 14

SNR Signal-to-Noise Ratio. 72, 113, 114, 115, 116, 117, 118, XXII

SPA Simple Power Analysis. 73, 75

SQIsign Short Quaternion and Isogeny Signature. 14

SVP Shortest Vector Problem. 13

WCC International Workshop on Coding and Cryptography. 15, 110, 148

ZK Zero Knowledge. 9

TABLE DES FIGURES

LISTE DES TABLEAUX

LIST OF ALGORITHMS

ANNEXE A

ANNEXES

A.1 Galois Field Reduction

After the carryless multiplication performed in the HQC reference implementation [AMAB+23], the result must be reduced to fit in the Galois field. This operation is performed with a further reduction called Galois field reduction [AMAB+23] and described in the following algorithm (see Algorithme 27).

Algorithm 27 Galois Field Reduction from [AMAB+23]

Require: parameters : m an integer, g a generator polynomial of $\mathbb{F}_{2^m}$, Δ the distance between the primitive polynomial first two set bits
Require: x a polynomial of degre $deg(x)$
Ensure: s be the representant of x in $\mathbb{F}_{2^m}$

```
steps = ⌈(deg(x)−(m−1))/Δ⌉                ▷ Determine the number of steps
for i from 1 to steps do
    mod = x >> m
    x = x ⊕ (1 << m) − 1 ⊕ mod
    z1 = 0
    r = g ⊕ 1
    for j from HW(g) − 1 down to 0 with step 1 do
        z2 = number_of_zeros(r)
        d = z2 − z1
        mod = mod << d
        x = x ⊕ mod
        r = r ⊕ (1 << z2)
        z1 = z2
return s := x
```

Regarding the assembly code, this algorithm is inlined inside the `gf_mul` operation, so there is no explicit call to `gf_reduce`.

A.2 Assembly Codes

In this section, we gather assembly codes of functions we target during our side-channel analyses. There are two function :

— the `gf_mul` function which calld both sub-routines carryless multiplication and Galois field redution

— the carryless multiplication.

Here, we don't decribe the reduction in a specific Table, since the function is called inline by the `gf_mul` operation (see tableau A.1).

0	`gf_mul`

1	push	{lr}	Set Return Adress (link register)
2	mov	ip, r0	Set (intra procedure) ip := r_0, the first argument
3	sub	sp, #12	allocate 12 bytes of space
4	add	r0, sp, #4	Set r_0 to point at the allocated memory
5	movs	r3, #0	Initialized $r_3 := 0$
6	uxtb	r2, r1	Zero-extend (Padding) the lsb of r_1 into r_2
7	uxtb.w	r1, ip	Zero-extend (Padding) the lsb of ip into r_1
8	strh.w	r3, [sp, #4]	Store r_3 as a half-word at location sp +4
9	bl	¡gf_carryless_mul¿	▷ call to carryless multiplication ,see tableau A.2
10	ldrb.w	r3, [sp, #4]	Restore r_3 from location sp +4
11	ldrb.w	r2, [sp, #5]	Load sp +5 into r_2
12	orr.w	r2, r3, r2, lsl #8	Combine r_3 and r_2 into r_2 (lsb from r_2)
			▷ – **Until 24 :** `gf_reduce` **of** r_2 –
13	uxtb	r3, r2	Zero-extend the lsb of r_2 into r_3 (Clear msb of r_3)
14	lsrs	r1, r2, #8	Right-shift r_2 by 8 into r_1. (Clear lsb of r_2)
15	eor.w	r3, r3, r2, lsr #8	$r_3 := (r_2 << 8) \oplus r_3$
16	eor.w	r3, r3, r1, lsl #2	$r_3 := (r_1 << 2) \oplus r_3$
17	eor.w	r3, r3, r1, lsl #3	$r_3 := (r_1 << 3) \oplus r_3$
18	eor.w	r3, r3, r1, lsl #4	$r_3 := (r_1 << 4) \oplus r_3$
19	and.w	r2, r3, #255 ; 0xff	set r_2 with the lsb of r_3
20	lsrs	r0, r3, #8	$r_0 := (r_3 >> 8)$
21	eor.w	r3, r2, r3, lsr #8	$r_3 := (r_3 << 8) \oplus r_2$
22	eor.w	r3, r3, r0, lsl #2	$r_3 := (r_0 << 2) \oplus r_3$
23	eor.w	r3, r3, r0, lsl #3	$r_3 := (r_0 << 3) \oplus r_3$
24	eor.w	r0, r3, r0, lsl #4	$r_0 := (r_0 << 4) \oplus r_3$
25	add	sp, #12	Restore the Stack, delocate the space
26	ldr.w	pc, [sp], #4	Load the return Adress

TABLE A.1 – Assembly code for Galois Field multiplication from [AMAB+23] compiled with -03 optimization

0	`gf_mul_carryless`	
1	stmdb	sp !, {r4, r5, r6, r7, r8, r9, sl, fp, lr}
2	and.w	ip, r2, #127 ; 0x7f
3	sbfx	r5, r2, #7, #1
4	ubfx	r2, r1, #2, #2
5	ubfx	r3, r1, #4, #2
6	subs	r4, r2, #2
7	rsb	r8, r2, #2
8	and.w	lr, r1, #3
9	orr.w	r8, r8, r4
10	rsb	r9, r3, #2
11	subs	r4, r3, #2

```
12  sub.w   r7, lr, #2
13  orr.w   r9, r9, r4
14  rsb     r4, lr, #2
15  add.w   fp, r2, #4294967295 ; 0xffffffff
16  orrs    r4, r7
17  rsb     r7, r2, #1
18  orr.w   r7, r7, fp
19  mov.w   r8, r8, asr #31
20  mvn.w   r8, r8
21  asrs    r7, r7, #31
22  bic.w   r7, ip, r7
23  and.w   r8, r8, ip, lsl #1
24  add.w   fp, r3, #4294967295 ; 0xffffffff
25  eor.w   r8, r7, r8
26  rsb     r7, r3, #1
27  orr.w   r7, r7, fp
28  mov.w   r9, r9, asr #31
29  mvn.w   r9, r9
30  asrs    r7, r7, #31
31  and.w   r9, r9, ip, lsl #1
32  bic.w   r7, ip, r7
33  add.w   sl, lr, #4294967295 ; 0xffffffff
34  eor.w   r7, r7, r9
35  asrs    r4, r4, #31
36  rsb     r9, lr, #1
37  sub     sp, #12
38  orr.w   r9, r9, sl
39  mvns    r4, r4
40  sub.w   sl, lr, #3
41  rsb     lr, lr, #3
42  and.w   r4, r4, ip, lsl #1
43  orr.w   lr, lr, sl
44  mov.w   r9, r9, asr #31
45  eor.w   r6, ip, ip, lsl #1
46  str     r0, [sp, #4]
47  bic.w   r9, ip, r9
48  mov     r0, r4
49  mov.w   lr, lr, asr #31
50  eor.w   r9, r0, r9
51  bic.w   lr, r6, lr
52  eor.w   lr, r9, lr
53  sub.w   sl, r2, #3
54  rsb     r9, r2, #3
55  orr.w   r9, r9, sl
```

```
56  mov.w       r9, r9, asr #31
57  bic.w       r9, r6, r9
58  eor.w       r8, r8, r9
59  eor.w       r2, lr, r8, lsl #2
60  rsb         r9, r3, #3
61  sub.w       lr, r3, #3
62  orr.w       r9, r9, lr
63  mov.w       r9, r9, asr #31
64  asrs        r4, r1, #6
65  bic.w       r9, r6, r9
66  eor.w       r9, r9, r7
67  rsb         r3, r4, #2
68  subs        r7, r4, #2
69  add.w       lr, r4, #4294967295 ; 0xffffffff
70  orrs        r3, r7
71  rsb         r7, r4, #1
72  orr.w       r7, r7, lr
73  asrs        r3, r3, #31
74  asrs        r7, r7, #31
75  mvns        r3, r3
76  and.w       r3, r3, ip, lsl #1
77  bic.w       ip, ip, r7
78  eor.w       ip, ip, r3
79  subs        r3, r4, #3
80  rsb         r4, r4, #3
81  orrs        r4, r3
82  asrs        r4, r4, #31
83  bic.w       r4, r6, r4
84  eor.w       ip, r4, ip
85  eor.w       r2, r2, r9, lsl #4
86  mov.w       r9, r9, asr #4
87  and.w       r3, r5, r1, lsl #7
88  eor.w       r9, r9, r8, asr #6
89  eor.w       r2, r2, ip, lsl #6
90  eors        r2, r3
91  eor.w       ip, r9, ip, asr #2
92  ldr         r3, [sp, #4]
93  and.w       r1, r5, r1, lsr #1
94  eor.w       r1, ip, r1
95  strb        r2, [r3, #0]
96  strb        r1, [r3, #1]
97  add         sp, #12
98  ldmia.w     sp !, {r4, r5, r6, r7, r8, r9, sl, fp, pc}
```

TABLE A.2 – Assembly code for Carryless Galois Field multiplication from [AMAB+23] compiled with -03 optimization