Etude et analyse de cryptosystèmes basés sur les codes correcteurs implantables en pratique

Boly Seck

N° d'ordre NNT : 2023STET025

THÈSE de DOCTORAT
DE L'UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Membre de l'Université de LYON

Opérée dans le cadre d'une cotutelle internationale avec :

L'ÉCOLE SUPÉRIEURE POLYTECHNIQUE de DAKAR

École Doctorale N° 488
Sciences Ingénierie Santé

École Doctorale EDMI
Mathématique et Informatique

Spécialité / discipline de doctorat : Informatique

Soutenue publiquement le 20/09/2023, par :

Boly SECK

Étude et analyse de cryptosystèmes basés sur les codes correcteurs d'erreurs implantables en pratique

---

Devant le jury composé de :

Viktor FISCHER Professeur à l'UJM de Saint-Étienne	**Président**
Cristina ONETE Maître de Conférences HDR à l'Université de Limoges	**Rapporteure**
Nadia EL MRABET Professeure des Universités à l'École des Mines de Saint-Étienne	**Rapporteure**
Augustin Pathé SARR Professeur à l'UGB de Saint-Louis	**Examinateur**
Pierre-Louis CAYREL Maître de Conférences HDR à l'UJM de Saint-Étienne	**Directeur de thèse**
Idy DIOP Professeur à l'ESP de Dakar	**Co-directeur de thèse**
Morgan BARBIER Maître de Conférences à l'ENSICAEN	**Encadrant de thèse**

Remerciements

Je tiens tout d'abord à exprimer ma reconnaissance envers mes directeurs de thèse, Pierre-Louis CAYREL, Maître de Conférences HDR à l'UJM de Saint-Étienne et Idy DIOP, Professeur à l'ESP de Dakar, ainsi que mon encadrant Morgan BARBIER, Maître de Conférences à l'ENSICAEN, qui m'ont accompagné durant ces années de recherches et partagé leurs brillantes expériences. Je les remercie également pour leur disponibilité malgré les aléas de la thèse en cotutelle internationale.

J'exprime particulièrement ma gratitude envers Pierre-Louis CAYREL, qui est plus qu'un directeur de thèse pour moi. Il m'a accueilli et m'a apporté son aide à tous les niveaux durant mes séjours à Saint-Etienne qui est aujourd'hui ma ville d'adoption (je suis stéphanois !).

J'adresse également mes remerciements aux Mesdames Cristina ONETE, Maître de Conférences HDR à l'Université de Limoges et Nadia EL MRABET, Professeure des Universités à l'École des Mines de Saint-Étienne, qui m'ont fait l'honneur d'être rapporteures malgré les délais très courts. Je tiens à remercier Monsieur Viktor FISCHER, Professeur à l'UJM de Saint-Étienne, qui m'a accordé le privilège d'être le président de mon jury. Je remercie aussi Monsieur Augustin Pathé SARR, Professeur à l'UGB de Saint-Louis, l'examinateur de cette thèse.

Je remercie les structures qui ont financé mes séjours en France telles que CEAMITIC, le programme de bourse d'excellence Eiffel ainsi que la bourse d'aide à la recherche doctorale financée par le SCAC de l'ambassade de France à Dakar.

Les travaux de recherches de cette thèse ont été menés au sein des Laboratoires Hubert Curien et LIMBI dont je remercie les membres et particulièrement l'équipe SESAM. J'ai beaucoup apprécié les moments passés ensemble, les discussions pendant la pause-café, les footings sans oublier la piscine. Je me souviens du repas international organisé chez Damien pour découvrir les différentes spécialités culinaires de chacun où j'ai cuisiné pour la première fois du «Yassa» pour l'équipe.

Je tiens à remercier Lilian Bossuet, Professeur à l'UJM de Saint-Étienne et responsable de l'équipe SESAM, ainsi que Jean-Jacques ROUSSEAU, Professeur à l'UJM de Saint-Étienne, pour leur disponibilité et leur aide dans mes démarches administratives.

Je tiens à exprimer ma gratitude envers Julie DEBIESSE, Directrice administrative du Laboratoire Hubert Curien et Fadoua LAFDIL, Responsable administrative des études doctorales, pour leur gentillesse et leur disponibilité pour traiter mes dossiers souvent compliqués.

Je remercie Cheikh Thiécoumba GUEYE, Professeur à l'UCAD de Dakar qui a fait tout son possible pour que je puisse continuer mes travaux de recherches durant mon séjour au Sénégal. Je le remercie également pour m'avoir intégré dans l'équipe du Laboratoire LACGAA dont je remercie particulièrement Ousmane NDIAYE, Gilbert Ndollane DIONE et Jean Belo KLAMTI.

Je remercie aussi mon ami El Mehdi BENHANI et sa femme, pour le séjour à Grenoble. J'ai beaucoup apprécié notre excursion en canoë-kayak au lac d'Aiguebelette.

Je remercie également mes amis Bilal SECK («Bil the boss»), Bassirou GUEYE, Abdoulaye NDIAYE, Doudoud DIONE, Mouhamed FALL, Ibrahima SECK, Oulimata NIANG et Moussa SARR pour leur soutien durant ces années de recherches.

Une mention spéciale à toute ma famille qui m'a soutenu durant tout mon cursus et mis dans un cadre favorable pour cet accomplissement, notamment mes frères Moustapha, Assane et Ibrahima. Je remercie particulièrement mes deux parents, Abdoulaye SECK et Mame awa NDIAYE qui m'ont toujours apporté leur soutien dans les moments difficiles, et appris la persévérance et l'humilité. Je remercie aussi mes sœurs Dieynaba, Ouleymatou, Adja khady et Ndeye Oumou.

Je remercie ma tante Sokhna Marième DIOP pour ses prières et à mes oncles Babacar GAYE et Abdoulaye SENE pour leur soutien.

Je remercie également ma femme Mbeugué FALL, ma force, mon épaule, mon amour tout simplement, je t'aime.

Je vais terminer en remerciant chaleureusement tous les guet-ndariens de Saint-Louis, la plus belle ville au monde, avec notre pont mythique plus long que celui d'Avignon !

«Je dédie ce travail à Khadim Rassoul»

Résumé

La cryptographie basée sur les codes correcteurs d'erreurs est l'une des approches permettant de construire des cryptosystèmes capable de résister à l'informatique quantique : on parle de cryptographie post-quantique. Contrairement aux problèmes difficiles du logarithme discret et de la factorisation de grands entiers (théorie des nombres) qui sont solubles avec l'algorithme de Peter Shor (exécuté sur un ordinateur quantique), la sécurité de la cryptographie à base de codes correcteurs d'erreurs repose sur le problème difficile du décodage par syndrome. Les standards actuels pour l'échange de clefs (Diffie-Hellman) et la signature numérique (RSA), sont basés sur ces problèmes difficiles de la théorie des nombres. Ainsi, depuis 2016, le NIST a lancé un processus de normalisation de la cryptographie post-quantique pour définir de nouvelles normes pour l'échange de clefs et la signature numérique. Dans cette thèse, nous présentons d'abord une implantation logicielle efficace de la version binaire et sécurisée de DAGS. C'est un mécanisme d'encapsulation de clefs basé sur les codes de Srivastava généralisés définis avec des matrices quasi-dyadiques. DAGS a été soumis lors du premier tour de ce processus avec une construction qui permettait d'atteindre de très petites tailles de données, comparée aux autres soumissions basées sur les codes. Cependant, à cause de problèmes de sécurité liés au choix des paramètres, ce candidat a été éliminé au second tour. Depuis, des efforts ont été menés par les auteurs qui ont proposé des jeux de paramètres plus judicieux hors de portée des attaques structurelles connues. Notre implantation donne de meilleures performances par rapport aux implantations précédentes grâce notamment à l'utilisation de techniques qui accélérèrent les opérations matricielles dans le cas quasi-dyadique.

Ensuite, nous montrons que la fonction de chargement des coefficients du polynôme de Goppa utilisée dans l'implantation référence de *Classic McEliece* laisse échapper de l'information secrète pendant la décapsulation. Avec une attaque par canal caché (attaque template), nous avons réussi à trouver de l'information secrète sur les coefficients, notamment leur poids Hamming. Cette information nous a permis de trouver directement le polynôme de Goppa dans le cas des clefs faibles de Sendrier et Loidreau, et d'améliorer la complexité de la recherche exhaustive de ce polynôme sur $\mathbb{F}_{2^m}$. Nous proposons également le problème du produit matrice-vecteur, un nouveau formalisme en cryptographie à base de codes, dont la résolution, à l'aide d'une analyse de consommation, permet de faire une attaque générique pendant le déchiffrement de Niederreiter.

Enfin, nous montrons que le protocole d'identification de Véron en métrique rang publié à CANS 2018 n'est pas zero-knowledge et laisse échapper de l'information sur le secret. Ce qui nous a permis de faire une attaque algébrique et de retrouver le secret. La faille dans ce schéma est liée à un masquage défectueux du secret par une simple permutation aléatoire sur $\mathbb{F}_{q^m}$. À l'aide d'une permutation adaptée, nous avons réparé ce protocole avant de proposer un schéma de signature avec des tailles de données compétitives par rapport aux candidats du NIST basés sur les codes pour la nouvelle campagne de signature.

Mots clefs : Cryptographie post-quantique, codes correcteurs d'erreurs, attaques par canaux cachés, identification, signature et métrique rang.

Abstract

Code-based Cryptography is one of the approaches to building post-quantum cryptosystems. Unlike the difficult problems of discrete logarithm and factorization (number theory), which are soluble with Peter Shor's algorithm, the security of code-based cryptography relies on the syndrome decoding problem. Asymmetric cryptography standards, mainly used today for key exchange (Diffie-Hellman) and digital signature (RSA), are based on these difficult number-theoretic problems. For this reason, since 2016, NIST has launched a post-quantum cryptography standardization process to define new standards for key exchange and digital signature.

In this thesis, we first present an efficient software implementation of the binary and secure version of DAGS. It is a key encapsulation mechanism scheme based on generalized Srivastava codes defined with quasi-dyadic matrices, submitted in the first process round. The DAGS construction achieves very small data sizes, compared to other code-based submissions. However, due to security issues related to parameter selection, DAGS was eliminated in the second round. Since then, efforts have been made by the submission's authors to provide judicious parameter sets, so that DAGS is now beyond the reach of known structural attacks. Our software implementation of binary DAGS performs better than previous implementations, thanks in particular to the use of techniques that speed up matrix operations in the quasi-dyadic case.

Next, we show that the Goppa polynomial coefficient loading function used in the reference implementation of *Classic McEliece* leaks secret information during decapsulation. By performing a side-channel attack (template attack), we managed to find secret information about these coefficients, in particular their Hamming weight. This information enabled us to find this polynomial directly in the case of Sendrier and Loidreau's weak keys, and to improve the complexity of the exhaustive search for this polynomial on $\mathbb{F}_{2^m}$. We also propose a new formalism in code-based cryptography. This is the matrix-vector product problem, whose solution of which, using a profiled attack, enables a generic attack on the private key during decryption.

Finally, we show that Véron's rank-metric identification published at CANS 2018 is not zero-knowledge and leaks information about the secret. This allowed us to make an algebraic attack and recover the secret easily. The flaw in this scheme is linked to faulty masking of the secret by a simple random permutation on $\mathbb{F}_{q^m}$. Using an adapted permutation, we obtained a zero-knowledge identification scheme in rank metric before proposing a signature scheme with data sizes competitive with NIST's code-based candidates for the new normalization process for signature.

Keywords : Post-quantum cryptography, error-correcting codes, side-channel attacks, identification, signature, and rank metric.

Notations

Notation dans les différents chapitres de cette thèse.

Nombre

d_H	distance de Hamming entre deux vecteurs
d	distance minimale d'un code linéaire
n	longueur d'un code linéaire
k	dimension d'un code linéaire
r	codimension d'un code linéaire
t	capacité de correction d'un code linéaire ou degré du polynôme de Goppa
α_i	élément i du support $\mathcal{L}$ de Goppa
$\boldsymbol{x}_i$	coordonnée i du vecteur $\boldsymbol{x}$
x	variable d'un polynôme
$\boldsymbol{h}_{i,j}$	composante i, j de la matrice $\boldsymbol{H}$
p	entier premier
q	puissance de p et taille d'un corps fini
$\mathtt{gen}$	générateur de groupe multiplicatif cyclique
τ	nombre d'échantillons dans une trace
N_{POI}	nombre de points d'intérêts
σ	permutation de longueur n
Σ	permutation de longueur n dans notre permutation spéciale
ℓ	nombre de bits de sécurité
N_Σ	graine pour générer Σ
N_Γ	graine pour générer Γ
N_{irr}	nombre de polynômes irréductible de Goppa

ϑ	nombre de rondes dans un schéma d'identification
$\mathtt{rang}$	poids d'un mot en métrique rang
w_H	poids de Hamming d'un mot

Vecteur

$\boldsymbol{x}$	message à transmettre ou texte clair
$\boldsymbol{y}$	mot reçu contenant du bruit
$\boldsymbol{c}$	mot de code ou texte chiffré
$\boldsymbol{e}$	vecteur d'erreur
$\boldsymbol{e}^T$	transposée de $\boldsymbol{e}$
$\boldsymbol{s}$	syndrome d'un mot
μ	vecteur de moyenne

Matrice

$\boldsymbol{G}$	matrice génératrice d'un code linéaire
$\boldsymbol{G}^T$	transposée de $\boldsymbol{G}$
$\boldsymbol{H}$	matrice de contrôle d'un code linéaire
$\boldsymbol{I}_{n\times n}$	matrice identité de taille $n \times n$
$\boldsymbol{O}_{m\times n}$	matrice nulle de taille $m \times n$
$\boldsymbol{P}$	matrice de permutation
$\boldsymbol{Z}$	matrice carrée inversible dans McEliece
$\boldsymbol{Q}$	matrice carrée inversible dans Niederreiter
$\boldsymbol{S}^*$	matrice carrée inversible dans le problème du produit matrice-vecteur
$\boldsymbol{R}^*$	matrice carrée secrète dans le problème du produit matrice-vecteur

Γ	matrice carrée dans notre permutation spéciale
$\boldsymbol{D}$	matrice dyadique
$\boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$	matrice de Cauchy
$\boldsymbol{L}$	matrice triangulaire inférieure
$\boldsymbol{U}$	matrice triangulaire supérieure
$\boldsymbol{C}$	matrice de covariance

Polynôme

$g(x)$	polynôme de Goppa
$s(x)$	polynôme syndrome
$\sigma(x)$	polynôme localisateur d'erreurs
$\theta(x)$	polynôme évaluateur d'erreurs
$p(x)$	densité de probabilité de la gaussienne multivariée
$u(x)$	somme des monômes de puissances paires de $\sigma(x)$
$v(x)$	somme des monômes de puissances impaires de $\sigma(x)$

Ensemble

$\mathcal{C}$	code linéaire
$\mathbb{F}$	corps fini
$\mathbb{N}$	entiers naturels
$\mathcal{L}$	support de Goppa
$Supp$	support d'un mot
pk	clef publique
sk	clef privée ou secrète
K	clef de session dans un KEM

Opérateur

$\oplus$	opérateur logique OU EXCLUSIF
^	opérateur OU EXCLUSIF (bit à bit) en C/C++
&	opérateur logique ET
$\xleftarrow{\$}$	élément généré de manière uniformément aléatoire
$>>i$	décalage logique de i bits vers la droite

Table des matières

Table des figures

Liste des tableaux

Introduction

Contexte

Les protocoles de sécurité modernes dans la plupart de nos systèmes reposent principalement sur trois fonctions de base de la cryptographie asymétrique : le chiffrement, la signature numérique et l'échange de clefs. En pratique, ces fonctions de base se retrouvent dans les suites cryptographiques de la norme TLS (Transport Layer Security), protocole de sécurisation des échanges par réseau informatique, notamment utilisée dans le HTTP (HyperText Transfer Protocol). Dans un tel protocole ou HTTPS (HyperText Transfer Protocol Secure), une communication sécurisée entre un client et un serveur (authentification unidirectionnelle) ou entre un serveur et un autre serveur pour une application Web (authentification mutuelle) est établie à la fin d'une session TLS. Les messages transmis sont appelés records, qui sont généralement encapsulés dans des segments TCP (Transmission Control Protocol), protocole chargé d'assurer les fonctionnalités de transport réseau. Ces records font intervenir le standard actuel pour l'échange de clefs, le protocole de Diffie-Helleman (ou DH), pour la négociation d'un secret éphémère entre différentes entités. La confidentialité de cet échange est persistante, car les clefs de session sont jetées à la fin. Pour certifier cet échange, on utilise le standard pour la signature numérique, RSA (Rivest-Shamir-Adleman). Cependant, ces standards actuels pour l'échange de clefs et la signature numérique basés, respectivement, sur le problème du logarithme discret [JMV01] et le problème de factorisation de grands entiers [RSA78] sont attaquables, du moins théoriquement dans un modèle d'ordinateur quantique. En effet, Peter Shor a montré dans [Sho94] qu'il est possible de trouver des solutions en temps polynomial à ces problèmes difficiles de la théorie des nombres à l'aide d'un ordinateur quantique avec suffisamment de capacité. De plus, la recherche quantique est érigée aujourd'hui en priorité nationale dans certains pays, notamment la France, ainsi que pour certains géants technologiques comme Google, IBM et Microsoft qui développent des efforts considérables pour voir émerger cette technologie d'ici à quelques années. La menace est donc réelle pour la plupart des protocoles de sécurité actuels. D'où le processus de normalisation de la cryptographie post-quantique lancé par le NIST (National Institute of Standards and Technology) depuis décembre 2016. Un cryptosystème post-quantique est un protocole à clef publique basé sur un problème mathématique prouvé difficile, différent des problèmes de la théorie des nombres : on parle de résistant quantique. Ce processus ou appel à compétition est organisé pour engager les acteurs universitaires, industriels et gouvernementaux à collaborer afin de déterminer les futures normes de résistants quantiques pour le chiffrement à clef publique, l'échange ou mécanisme d'encapsulation de clefs (Key Encapsulation Mechanism ou KEM) et la signature numérique d'ici à 2025. Parmi les principaux problèmes non solubles en temps polynomial avec l'algorithme quantique de Peter Shor, se trouvent les problèmes de recherche de mots de poids faibles dans les réseaux euclidiens, de décodage de codes correcteurs d'erreurs aléatoires, de résolution de systèmes d'équations multivariées, d'isogénies et avec les fonctions de hachage.

En juillet 2022, le NIST a livré les premiers résultats du processus de normalisation de la cryptographie post-quantique. Les algorithmes sélectionnés sont CRYSTALS-Kyber, mécanisme d'encapsulation de clefs basé sur les réseaux euclidiens, CRYSTALS-Dilithium, FALCON, signatures numériques basées également sur les réseaux euclidiens et SPHINCS+, signature numérique basée sur les fonctions de hachage.
Ces futures normes pour l'échange de clefs et la signature numérique seront intégrées à la fin du processus dans la chaine de valeur actuelle et seront les options par défaut pour la cryptographie post-quantique.

Objectif

Les codes correcteurs d'erreurs sont utilisés à la base pour transmettre de manière fiable des informations sur un canal de communication bruité comme dans le domaine des télécommunications et pour le stockage de données sur divers supports tels les disques durs. En pratique, les codes correcteurs d'erreurs utilisés sont des codes pour lesquels on connait un algorithme de décodage efficace ou en temps polynomial : on parle de codes structurés. Cependant, pour des codes aléatoires, le coût du décodage est exponentiel. En 1978, R. McEliece [McE78] a montré que le décodage des codes aléatoires est un problème difficile avant de les utiliser pour faire du chiffrement. C'est le chiffrement de McEliece que nous verrons tout au long de cette thèse. Ce schéma a en fait été proposé en même temps que RSA, mais était peu utilisé à cause notamment de la taille des clefs très élevée. Depuis, plusieurs variantes de ce schéma utilisant d'autres codes ont été proposées pour réduire la taille des données et trouver un compromis entre performance et sécurité (un équilibre souvent difficile à trouver).

L'objectif de cette thèse était de participer à l'analyse des candidats basés sur les codes correcteurs. Nous nous focaliserons en particulier sur la possibilité d'implanter ou d'étudier un ou plusieurs algorithmes sur des cartes de type STM32. Réussir l'implantation d'un algorithme résistant aux attaques quantiques est un challenge important pour le développement de l'internet des objets dans l'ère quantique. Nous étudierons les arithmétiques nécessaires au développement des algorithmes post-quantiques et à leur traduction en langage de programmation C/C++. Nous analyserons la résistance des implantations contre les attaques dites par canaux cachés ou auxiliaires. Ce sont des attaques puissantes utilisant, non pas les failles mathématiques d'un algorithme, mais les fuites d'information lors de son exécution. Elles ont également la particularité de déduire des informations sur le secret en exploitant les propriétés physiques d'un circuit électroniques en fonctionnement. Il peut s'agir par exemple de mesurer le temps de calcul de certaines opérations sensibles dans l'algorithme (attaque temporelle) ou bien d'enregistrer sa consommation électrique ou son rayonnement électromagnétique (SPA, DPA, attaque template, etc).

Contribution

Chiffrement Le processus de normalisation de la cryptographie post-quantique du NIST est composé de différentes étapes (ou tours) durant lesquelles un certain nombre de candidats est choisi en fonction des exigences dynamiques du NIST pour passer à l'étape suivante. En décembre 2016, l'appel du NIST a mobilisé pour le premier tour un total de 69 soumissions, dont DAGS [Gus+17], qui ont satisfait aux critères minimaux et aux exigences en termes de niveau de sécurité du NIST. En janvier 2019, seuls 26 candidats sur les 69 ont passé le second tour du processus. Malheureusement, DAGS n'a pas été choisi pour la suite du processus en raison de certaines attaques soulevées par Barelli-Couvreur dans [BC18] ainsi que Bardet *et al.* dans [Bar+19] a priori applicables sur ses paramètres d'origine. Pour rappel, DAGS est un KEM basé sur les codes de Srivastava généralisés sous forme quasi-dyadiques. Cette construction permet d'atteindre de très petites tailles de clef publique/privée et de texte chiffré par rapport à toutes les autres soumissions basées sur les codes telles que *Classic McEliece*, BIKE, BIG QUAKE. L'analyse détaillée de ces techniques d'attaque ont permis aux auteurs de la soumission de proposer de nouveaux paramètres sécurisés et une construction plus efficace que la version originale dans [Ban+19].

Dans le Chapitre 3, nous proposons une implantation logicielle efficace des paramètres binaires et sécurisés de DAGS. En utilisant les techniques présentées dans [Ban+20] qui visent spécifiquement à améliorer la multiplication des matrices quasi-dyadiques avec l'utilisation de la multiplication de Karatsuba et la décomposition $\boldsymbol{LUP}$ de ces matrices, nous avons réussi à accélérer les opérations dans les différents algorithmes. Notre implantation logicielle de DAGS binaire donne de meilleure performance par rapport aux implantations précédentes de DAGS. Cette contribution à fait l'objet d'une publication à CBCrypto 2022 [Sec+23d].

Attaques par canaux cachés Durant le processus de normalisation de la cryptographie post-quantique, en plus de la performance et la définition des niveaux de sécurité des candidats, un aspect très important soulevé par le NIST est l'impact de la mise en œuvre logicielle ou matérielle sur leur sécurité. En effet, comme nous l'avons évoqué ci-dessus, les attaques par canaux cachés sont des attaques physiques très puissantes capables d'exploiter les propriétés physiques d'un circuit électronique pour obtenir des informations sensibles ou secrètes dans une implantation d'algorithme cryptographique. Pour la plupart des schémas post-quantiques, notamment basés sur les codes correcteurs d'erreurs, on ne sait pas en pratique quelles sont les attaques physiques réalisables et comment s'en protéger.

Dans le Chapitre 4, nous réalisons une attaque template sur la décapsulation du KEM finaliste du NIST, *Classic McEliece*[Alb+20] utilisant les codes de Goppa binaires. Notre attaque cible le chargement des coefficients du polynôme de Goppa dans l'implantation de référence réalisée sur un microcontrôleur STM32 intégrant un cœur ARM Cortex-M4 [CC21]. Cette attaque nous permet, tout d'abord, de retrouver les poids de Hamming

des coefficients du polynôme de Goppa et grâce à ces informations, nous avons réussi à retrouver directement ce polynôme, dans le cas des clefs faibles dans la méthode de Loidreau et Sendrier [LS01]. Dans le cas des «clefs moins faibles», nous montrons aussi que l'on peut retrouver ce polynôme par recherche exhaustive avec une faible complexité. Ensuite, nous proposons une meilleure réduction de la complexité de la recherche exhaustive du polynôme de Goppa sur $\mathbb{F}_{2^m}$. Enfin, nous montrons que notre attaque est plus réaliste en pratique par rapport à d'autres attaques par canaux cachés sur *Classic McEliece*. Ce travail a été accepté et présenté à AFRICACRYPT 2023 [Sec+23c].

Dans le Chapitre 5, nous présentons une attaque plus générique sur le déchiffrement des schémas basés sur les codes correcteurs d'erreurs et sur le schéma de Niederreiter en particulier grâce notamment à des informations obtenues par analyse de canaux cachés. Tout d'abord, nous introduisons un nouveau formalisme en cryptographie basée sur les codes, le problème du produit matrice-vecteur, dont la résolution permet de trouver la clef privée. Ce problème consiste à trouver une matrice privée $\boldsymbol{R}$ connaissant $\boldsymbol{s}^*$ (ses entrées) dans $\mathbb{N}$ tel que $\boldsymbol{R}\boldsymbol{s}^T = \boldsymbol{s}^*$. Ensuite, nous appliquons une attaque par canal caché basée sur l'apprentissage automatique durant la première étape du déchiffrement de Niederreiter pour obtenir $\boldsymbol{s}^*$ dans $\mathbb{N}$. Enfin, nous montrons que si nous parvenons à construire correctement une matrice $\boldsymbol{S}^* = (\boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_{n-k}^*)$ dans $\mathbb{N}$, on trouve directement la matrice privée $\boldsymbol{R}$ sans résoudre le problème difficile de décodage par syndrome sur lequel repose la sécurité de la cryptographie à base de codes correcteurs d'erreurs. Cette contribution a été publiée à ICISC 2022 [Sec+23b].

Identification Comme nous l'avons évoqué ci-dessus, les 3 premières futures normes de signatures post-quantiques sont maintenant connues, CRYSTALS-Dilithium, FALCON et SPHINCS+ qui sont basés respectivement sur les réseaux euclidiens et les fonctions de hachage. Pour diversifier les approches afin d'offrir une gamme plus large de normes de signatures contrairement aux précédentes campagnes comme pour l'AES, le NIST a décidé de faire un processus supplémentaire pour la signature numérique à partir de juin 2023. L'objectif de cette campagne additionnelle est de proposer des schémas avec des tailles de signature courtes et des temps de vérification rapides en plus d'une bonne preuve sécurité. La signature basée sur les codes est un candidat prometteur pour trouver un compris entre la performance et la sécurité.

Dans le Chapitre 6, nous montrons d'abord que le protocole d'identification de Véron en métrique rang proposé dans [Bel+18] laisse échapper de l'information secrète. Ensuite, nous effectuons une attaque sur le support de l'erreur de ce schéma en utilisant l'algorithme de Gaborit, Ruatta, et Schreck [GRS15] pour trouver le secret. Enfin, nous proposons un protocole d'identification à divulgation nulle de connaissance de Véron en métrique rang à l'aide d'une permutation spéciale avant de proposer un schéma de signature efficace avec des tailles de données correctes et une vérification rapide grâce notamment à la métrique rang. Cette contribution est publiée à I4CS 2022 [Sec+23a].

Première partie

Préliminaires

Chapitre 1

Cryptologie

La cryptologie, c'est le moteur de l'internet. Aujourd'hui, plus personne ne regarde sous le capot de sa voiture.

– Jacques Stern, 2006

La cryptologie, étymologiquement la science du secret, regroupe la cryptographie et la cryptanalyse. La cryptographie est l'étude et la conception de procédés (ou algorithmes) de chiffrement des informations. Un algorithme de chiffrement est une méthode mathématique permettant de rendre un message (ou texte clair) incompréhensible à un tiers qui ne possède pas la clef. Pour protéger les données, on utilise généralement la cryptographie. Il ne s'agit pas de dissimuler le message comme en stéganographie, mais de le chiffrer, à l'aide d'une clef : on parle de chiffrement. Ainsi, la donnée transformée est appelée le chiffré (ou texte chiffré) associé au message. Pour retrouver le message original, il faut réaliser l'opération inverse à l'aide d'une clef : on parle de déchiffrement. Historiquement, des techniques cryptographiques existaient déjà dans l'Antiquité, et la plus avancée était celle de Jules César. Il utilisait la technique par décalage de trois lettres de l'alphabet vers la droite. Par exemple, A est remplacé par D, B par E, ainsi de suite jusqu'à W par Z, etc.
La cryptanalyse est la science qui consiste à analyser des chiffrés pour trouver les messages associés et/ou la clef. Elle regroupe des techniques pour décrypter des chiffrés (trouver le message sans connaître la clef) et trouver la clef en évaluant les faiblesses d'un système de chiffrement. Ces deux concepts feront l'objet de définitions plus formelles dans le présent chapitre.

1.1 Cryptographie

Le chiffrement qui est la partie technique de la cryptographie se compose de deux grandes familles : le chiffrement symétrique et le chiffrement asymétrique. Le chiffre-

ment symétrique utilise la même clef secrète pour chiffrer des messages et pour déchiffrer des textes chiffrés, c'est le cas dans l'algorithme de chiffrement appelé AES (Advanced Encryption Standard). Alors que le chiffrement asymétrique utilise une paire de clefs : une clef publique pour chiffrer des messages ou vérifier des signatures et une clef privée (ou secrète) pour déchiffrer des textes chiffrés ou faire des signatures, c'est le cas dans l'algorithme de chiffrement et de signature appelé RSA (Rivest- Shamir-Adleman). Pour mieux décrire ces différents systèmes de chiffrement, nous allons utiliser trois personnages, Alice et Bob qui cherchent à communiquer de manière sécurisée et Eve, une attaquante (passive ou active) qui écoute ou modifie cet échange pour trouver des informations secrètes (message et clef secrète).

1.1.1 Chiffrement symétrique

Le chiffrement symétrique ou à clef secrète qui utilise la même clef pour le chiffrement et le déchiffrement est constitué de trois algorithmes :

Génération des clefs (KeyGen**)** C'est un algorithme probabiliste qui prend en entrée un paramètre de sécurité λ et produit une clef secrète sk.

$$\mathsf{KeyGen}(\lambda) = \mathsf{sk}.$$

Remarque 1.1 *Le paramètre de sécurité λ est obtenu à l'aide d'un algorithme probabiliste d'initialisation, souvent appelé* ***Setup****.*

Chiffrement (Encrypt**)** C'est un algorithme déterministe qui prend en entrée le message $\mathtt{mess}$ et la clef secrète sk et retourne le texte chiffré $\mathtt{chif}$ associé.

$$\mathsf{Encrypt}(\mathtt{mess}, \mathsf{sk}) = \mathtt{chif}.$$

Déchiffrement (Decrypt**)** C'est un algorithme probabiliste qui prend en entrée le texte chiffré $\mathtt{chif}$ et la clef secrète sk et retourne le message de départ $\mathtt{mess}$.

$$\mathsf{Decrypt}(\mathtt{chif}, \mathsf{sk}) = \mathtt{mess}.$$

Le principal avantage du chiffrement à clef secrète est la rapidité des opérations de chiffrement comme dans AES. Cependant, l'inconvénient majeur dans ce type de chiffrement est le partage de la clef secrète sk via un canal qu'il faut sécuriser.

1.1.2 Échange de clefs

Définition 1.1 (Fonction à sens unique) *Soit $f(x) = y$ une fonction. f est dite à sens unique si :*

— *étant donné x, il est facile de calculer $f(x)$ (calculable en un temps humainement raisonnable);*

— *étant donné y, il est difficile de retrouver x.*

En d'autres termes, une fonction à sens unique est une fonction telle que retrouver x à partir de $f(x)$ est un problème mathématique difficile, comme dans l'exponentiation modulaire par exemple.

Définition 1.2 (Exponentiation modulaire) *Soit p un entier premier et $\mathtt{gen}$ un générateur du groupe multiplicatif cyclique $(\mathbb{Z}/p\mathbb{Z})^*$. L'exponentiation modulaire est la fonction :*

$$f : (\mathbb{Z}/p\mathbb{Z})^* \longrightarrow (\mathbb{Z}/p\mathbb{Z})^*$$
$$x \longmapsto \mathtt{gen}^x \mod p$$

L'exponentielle modulaire est facile à calculer dans un sens. Et dans l'autre sens ?

Définition 1.3 (Problème du logarithme discret) *Soit $y \in (\mathbb{Z}/p\mathbb{Z})^*$. Le problème du logarithme discret consiste à trouver x tel que $\mathtt{gen}^x \mod p = y$.*

En 1976, Whitfiel Diffie et Martin Hellman [DH76] ont proposé un protocole d'échange de données à distance sans être d'accord au préalable sur un secret commun à l'aide de fonctions à sens unique : c'est le protocole d'échange de clefs de Diffie-Helleman. La complexité de la meilleure attaque pour trouver une solution au problème du logarithme discret dans un groupe cyclique est sous-exponentielle [Jou13]. La sécurité du protocole de Diffie-Hellman, décrit dans la figure 1.1, repose sur ce problème.

Alice		**Bob**
$\mathsf{pk} = (p, \mathtt{gen})$		$\mathsf{pk} = (p, \mathtt{gen})$
$x_a \xleftarrow{\$} \mathbb{N}$		$x_b \xleftarrow{\$} \mathbb{N}$
$y_a \leftarrow \mathtt{gen}^{x_a} \mod p$		$y_b \leftarrow \mathtt{gen}^{x_b} \mod p$
	$\xrightarrow{y_a}$	
		$\mathsf{sk}_b = y_a^{x_b} \mod p$
	$\xleftarrow{y_b}$	
$\mathsf{sk}_a = y_b^{x_a} \mod p$		

FIGURE 1.1 – Protocole de Diffie-Hellman.

Nous pouvons vérifier que :

$$y_b^{x_a} = (\mathtt{gen}^{x_b})^{x_a} = \mathtt{gen}^{x_b x_a} = (\mathtt{gen}^{x_a})^{x_b} = y_a^{x_b}.$$

Le secret commun partagé par Alice et Bob est $K = \mathtt{gen}^{x_a x_b}$, qui peut être utilisé comme clef secrète dans un algorithme de chiffrement symétrique tel que AES.

Toutefois, la méthode de Diffie-Hellman ne permet pas de faire du chiffrement. Mais Diffie et Hellman pensent que c'est possible avec une fonction à sens unique avec trappe sans donner d'exemple.

Définition 1.4 (Fonction à sens unique avec trappe) *La fonction $f(x) = y$ est dite à sens unique avec trappe si :*

- *étant donné x, il est facile de calculer $f(x)$;*
- *étant donné y, il est difficile de retrouver x, sauf si on connaît une trappe.*

En 1977, Ronald Rivest, Adi Shamir et Leonard Adleman [RSA77] ont proposé la fonction puissance avec trappe pour faire du chiffrement et de la signature : c'est le début du chiffrement asymétrique ou à clef publique.

1.1.3 Chiffrement asymétrique

Le chiffrement à clef publique (Public Key Encryption ou PKE) utilise une clef publique pour chiffrer et une clef privée (équivalent de la clef secrète dans le chiffrement symétrique) pour déchiffrer ; il est constitué de trois algorithmes :

Génération des clefs (KeyGen**)** C'est un algorithme probabiliste qui prend en entrée un paramètre de sécurité λ et produit une clef publique pk et une clef privée sk.

$$\mathsf{KeyGen}(\lambda) = (\mathsf{pk}, \mathsf{sk}).$$

Chiffrement (Encrypt**)** C'est un algorithme probabiliste qui prend en entrée le message `mess` et la clef publique pk et retourne le texte chiffré `chif` associé.

$$\mathsf{Encrypt}(\texttt{mess}, \mathsf{pk}) = \texttt{chif}.$$

Déchiffrement (Decrypt**)** C'est un algorithme déterministe qui prend en entrée le texte chiffré `chif` et la clef privée sk et retourne le message de départ `mess`.

$$\mathsf{Decrypt}(\texttt{chif}, \mathsf{sk}) = \texttt{mess}.$$

L'avantage de la cryptographie asymétrique repose sur la souplesse de la gestion des clefs à travers une infrastructure de gestion à clefs publiques (Public Key Infrastructure ou PKI). Aujourd'hui, les protocoles asymétriques sont essentiellement utilisés pour l'échange de clefs (Diffie-Hellman) et la signature numérique (RSA).

Pour tirer profit des avantages à la fois des systèmes symétrique et asymétrique, on utilise un protocole hybride comme TLS (Transport Layer Security) pour la sécurisation des réseaux informatiques. Par exemple, Dans une session TLS, nous avons le protocole de Diffie-Hellman, pour la négociation de clefs éphémères, la signature numérique RSA, pour certifier cet échange et l'AES pour le chiffrement des données.

Un mécanisme d'encapsulation de clefs (Key Encapsulation Mechanism ou KEM), Figure 1.2), est de base un schéma de chiffrement à clef publique qui sert la même fonction que l'échange de clefs, à la différence qu'ici la clef est entièrement générée par l'une des parties (Alice ou Bob), ce qui permet entre autres de commencer à transmettre les données encapsulées. Dans un schéma de KEM Bob générer une paire de clefs publique/-privée avant d'envoyer la clef publique à Alice : c'est la génération des clefs (KeyGen). Ensuite, Alice chiffre une clef de session K (choisie aléatoirement) en K' à l'aide de la clef publique de Bob : c'est l'encapsulation (Encaps). Enfin, cette clef chiffrée K' est transmise à Bob qui la déchiffre avec sa clef privée pour obtenir K : c'est la décapsulation (Decaps).

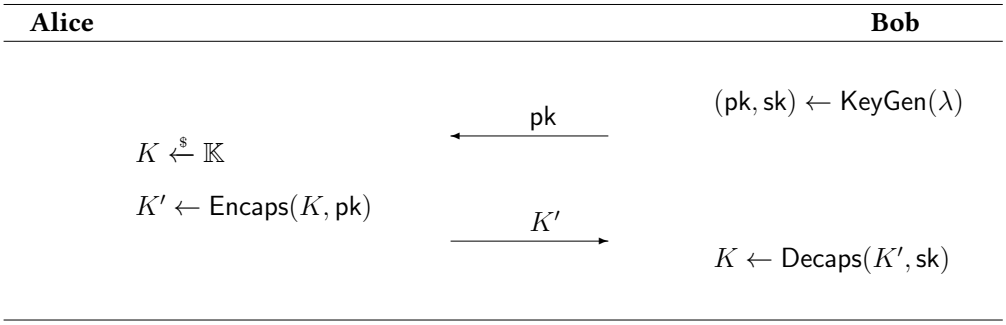

FIGURE 1.2 – Mécanisme d'encapsulation de clefs ou KEM.

Dans cette thèse, nous nous intéresserons particulièrement à la cryptographie asymétrique et nous verrons en détail quelques KEM basés sur les codes correcteurs d'erreurs dans les Chapitres 3 et 4.

1.1.4 Signature

L'identification (procédure permettant de connaître l'identité d'une entité) est un principe cryptographique qui assure l'authentification (procédure permettant de vérifier l'identité d'une entité) d'Alice auprès de Bob et/ou inversement. Dans un tel protocole, Alice utilise un algorithme pour prouver son identité à Bob qui utilise à son tour un autre algorithme pour la vérification. Alice est appelée le prouveur et Bob, le vérifieur.

La signature numérique est un principe cryptographique qui, en plus de l'authentification, assure l'intégrité (procédure permettant de vérifier si les données échangées entre Alice et Bob ont été modifiées ou non). C'est un protocole asymétrique qui utilise une fonction de hachage (permettant de passer du message $\mathtt{mess}$ de taille variable à un bloc de taille fixe et plus petite $\mathtt{hash}(\mathtt{mess})$, appelé empreinte) et est composé de trois algorithmes :

Génération des clefs (KeyGen**)** C'est un algorithme probabiliste qui prend en entrée un paramètre de sécurité λ et produit une clef publique pk et une clef privée sk.

$$\mathsf{KeyGen}(\lambda) = (\mathsf{pk}, \mathsf{sk}).$$

Signature (Sign**)** C'est un algorithme probabiliste qui prend en entrée le message $\mathtt{mess}$ ou son empreinte $\mathtt{hash}(\mathtt{mess})$ et la clef privée sk et retourne la signature $\mathtt{s}(\mathtt{mess})$.

$$\mathsf{Sign}(\mathtt{hash}(\mathtt{mess}), \mathsf{sk}) = \mathtt{s}(\mathtt{mess}).$$

Le document signé est composé du message $\mathtt{mess}$ et de la signature $\mathtt{s}(\mathtt{mess})$.

Vérification (verif**)** C'est un algorithme déterministe qui prend en entrée la signature $\mathtt{s}(\mathtt{mess})$ et la clef publique pk et compare les empreintes $\mathtt{hash}'(\mathtt{mess})$ et $\mathtt{hash}(\mathtt{mess})$.

$$\mathsf{verif}(\mathtt{s}(\mathtt{mess}), \mathsf{pk}) = \mathtt{hash}'(\mathtt{mess}),$$

$$\mathtt{hash}'(\mathtt{mess}) \stackrel{?}{=} \mathtt{hash}(\mathtt{mess}).$$

La signature est grossièrement l'inverse d'un PKE. Nous verrons en détail dans la Section 2.5 (page 36) l'identification et la signature.

1.2 Cryptanalyse

L'étude de la sécurité des procédés du chiffrement repose sur le principe de Kerckhoffs [Ker83] : «la sécurité d'un système cryptographique ne doit pas reposer sur l'ignorance de la méthode de chiffrement employée». Elle repose essentiellement sur la clef secrète (ou privée). Cependant, Eve peut disposer de différentes informations sur le message et/ou sur le texte chiffré qu'elle peut exploiter pour retrouver cette clef. Nous avons donc différents niveaux de sécurité en fonction des informations qu'Eve peut détenir sur le protocole cryptographique.

1.2.1 Différents niveaux de sécurité

On évalue le niveau de sécurité d'un protocole cryptographique en fonction des quatre attaques ci-dessous classées par ordre décroissant de difficulté.

1. **Attaque à texte chiffré connu** (Known-Ciphertext Attack) : Eve ne connaît que les textes chiffrés.
2. **Attaque à texte clair connu** (Known-Plaintext Attack) : Eve connaît des couples de textes clairs/chiffrés.
3. **Attaque à texte clair choisi** (Chosen-Plaintext Attack ou CPA) : Eve choisit des textes clairs, les chiffre et obtient les textes chiffrés correspondants. Ce type d'attaque nécessite d'avoir accès à une machine chiffrante.

4. **Attaque à texte chiffré choisi** (Chosen-Ciphertext Attack ou CCA1) : Eve choisit des textes chiffrés et peut obtenir les textes clairs correspondants. Pour ce type d'attaque aussi, Eve a besoin d'avoir accès à une machine déchiffrante (temporairement). Lorsque Eve fait ses choix de textes chiffrés de manière adaptative, c'est-à-dire en fonction du résultat des déchiffrements précédents : on parle d'**attaque adaptative à texte chiffré choisi** (Adaptative Chosen-Ciphertext Attack ou CCA2). Ce modèle d'attaque est plus fort que l'attaque CCA1. Il est très important dans les preuves de sécurité contre les attaques à texte chiffré choisi. La résistance d'un protocole cryptographique contre l'attaque CCA2 implique qu'aucune attaque pratique à texte chiffré choisi ne peut être réalisée.

De manière générale, il existe deux concepts d'attaques en fonction du comportement de Eve qui peut être passif ou actif. Une attaque est dite passive lorsque Eve ne fait qu'observer ou écouter la communication entre Alice et Bob. En revanche, lorsqu'elle peut modifier le texte chiffré ou créer une faute sur une implantation d'un système cryptographique (ou cryptosystème) : on parle d'attaque active. Les attaques par canaux cachés ou auxiliaires sont des attaques passives et les attaques par injection de fautes sont dites actives.

1.2.2 Attaques par canaux cachés

Les attaques par canaux cachés ou auxiliaires (Side-Channel Attack ou SCA) sont des attaques en pratique qui exploitent les propriétés physiques d'un circuit électronique afin d'obtenir des informations sensibles ou secrètes sur une implantation d'algorithme cryptographique. Les canaux cachés sont des canaux physiques qui existent naturellement dans un système par lesquels de l'information s'échappe. Ce sont, par exemple, la consommation de puissance (courant ou tension) d'un circuit, les émissions électromagnétiques, etc. Pour réaliser une SCA (Figure 1.3), on surveille généralement ces canaux qui laissent échapper de l'information secrète sur le circuit attaqué (ou cible) à l'aide d'un oscilloscope (ou d'un ChipWhisperer [1]). Ces derniers sont des appareils de captures qui échantillonnent des signaux analogiques continus et les transforment en séquences numériques discrètes. Ces séquences sont appelées des «traces de consommation».

Les protocoles cryptographiques, bien que théoriquement très robustes, leurs implantations peuvent être très sensibles à ces fuites d'informations qui sont exploitées dans les SCAs. Connues depuis la fin des années 1990 avec les travaux de Kocher [Koc96 ; KJJ99], les SCAs sont devenues aujourd'hui incontournables pour évaluer la sécurité des implantations embarquées notamment. Les principales SCAs sont énumérées ci-après.

Attaque temporelle L'attaque temporelle (Timing Attack ou TA) proposée pour la première fois en 1996 par Kocher [Koc96] sur RSA, consiste à mesurer le temps d'exécu-

1. Plate-forme open-source d'analyse de sécurité des systèmes embarqués pour l'acquisition et l'analyse de traces de consommation.
https://store.newae.com/

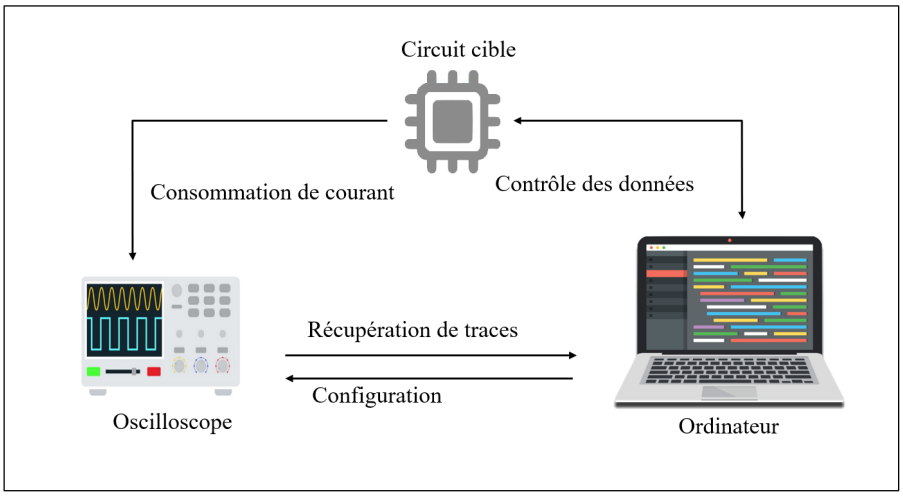

FIGURE 1.3 – Banc d'attaque par canaux cachés.

tion de certaines étapes dans le déchiffrement afin de retrouver des informations secrètes. Kocher exploite une faille dans l'implantation du calcul de l'exponentiation modulaire dans RSA en mesurant le temps de calcul de variables sensibles ($\mathsf{gen}^{\mathsf{sk}} \mod p$) qui dépend de la taille en bits de la clef privée sk. Pour ce type d'attaque, Eve doit connaître les détails de l'implantation matérielle ou logicielle du déchiffrement dans RSA. Pour se prémunir de ce type d'attaque (on parle de contre-mesures), il faut éviter les branchements conditionnels (test **Si**) dans une implantation.

Analyse simple de consommation En 1999, Kocher *et al.* [KJJ99] ont proposé une nouvelle technique d'attaque qui consiste à observer à l'aide d'un oscilloscope la consommation de courant d'un microprocesseur pour déterminer si un bit dans la clef privée est un «0» ou un «1» : c'est l'attaque par analyse de la consommation de puissance. L'analyse simple de consommation (Simple Power Analysis ou SPA) consiste à mesurer une trace de consommation d'un microprocesseur exécutant un algorithme cryptographique pour en déduire les opérations effectuées. En effet, chaque instruction exécutée par un microprocesseur met en œuvre un certain nombre de transistors et la mesure du courant consommé par ces derniers reflète l'activité du microprocesseur. Dans une SPA, une fine analyse de la trace de consommation permet généralement de distinguer la différence entre un «0» et un «1», et parfois, de retrouver la clef privée sk comme dans RSA. Pour ce type d'attaque, aussi, Eve doit connaître l'implantation. Cependant, si Eve n'arrive pas à distinguer de différences sur la trace de consommation liée par exemple à une contre-mesure qui entraîne de faibles variations de courant sur l'implantation, elle peut faire une analyse de consommation d'ordre supérieur.

Analyse différentielle de consommation Contrairement à la SPA, dans une analyse différentielle de consommation (Differential Power Analysis ou DPA), Eve doit disposer

de plusieurs traces de consommation pour une même hypothèse de clef privée. Elle effectue ensuite une analyse statistique des traces en calculant la différence des moyennes pour vérifier si son hypothèse de clef est correcte (différence de moyennes non nulle) ou non (différence de moyennes nulle).

À côté des attaques par analyse simple et différentielle de consommation qui s'effectue en une seule étape (attaques non profilées), il existe aussi des attaques par analyse de consommation qui requièrent une phase d'apprentissage (attaques profilées).

Attaques profilées Les attaques profilées sont composées de deux phases : une phase d'apprentissage (ou profilage) et une phase d'attaque. Dans la phase d'apprentissage, Eve cherche à caractériser la consommation de puissance du circuit qu'elle souhaite attaquer (ou cible) à l'aide d'un autre circuit identique (ou clone). Dans la phase d'attaque, elle utilise cette caractérisation pour déterminer les informations secrètes sur la cible. Parmi les attaques profilées, nous avons les attaques stochastiques, les attaques utilisant des techniques d'intelligence artificielle (comme le deep learning) et l'attaque template (Template Attack ou TA), que nous décrirons en détail dans le chapitre 4.

1.2.3 Attaques par injection de fautes

Les attaques par injection de fautes (Fault Injection Attack ou FIA) proposées en 1997 par Boneh *et al.* [BDL97], sont des attaques actives qui nécessitent souvent plus de ressources (attaques invasives) que les SCAs (attaques non-invasives). Elles consistent à créer volontairement des erreurs dans le cryptosystème pour provoquer un comportement inhabituel des opérations cryptographiques afin d'en extraire des informations secrètes. L'injection de fautes peut être une modification de la tension électrique du circuit (glitch), une perturbation lumineuse avec un laser, ou par impulsion électromagnétique.

Attaque par glitch Dans une attaque par glitch, Eve procède par une perturbation de la tension d'alimentation ou une modification de la fréquence de l'horloge du circuit afin de provoquer une erreur pouvant révéler des informations secrètes sur l'implantation du cryptosystème. Ce type d'attaque est non-invasive et entraîne la non-exécution ou une exécution erronée d'instructions dans un algorithme d'authentification par exemple.

Attaque par injection laser L'attaque par injection laser est une attaque invasive qui nécessite une préparation du circuit. Eve effectue d'abord, le de-packaging (retrait du boîtier) du circuit intégré pour enlever les différentes protections. Ensuite, elle utilise une sonde électromagnétique reliée à un laser finement calibré générant une perturbation qui modifie des bits d'information pendant l'exécution du cryptosystème. Enfin, elle utilise différentes techniques comme l'analyse statistique ou l'apprentissage automatique pour révéler des informations secrètes. Ce type d'attaque est très puissant, coûteux, généralement irréversible et compliqué à réaliser.

Nous allons nous intéresser par la suite à la cryptographie à clef publique basée sur les codes correcteurs qui est l'une des principales approches de la cryptographie post-quantique (Post-Quantum Cryptography ou PQC).

Chapitre 2

Codes correcteurs d'erreurs en cryptographie

Le problème fondamental de la communication est de reproduire en un point, soit exactement, soit approximativement, un message sélectionné en un autre point.

– Claude E. Shannon, 1948

En 1948, Claude Shannon a formalisé la théorie de l'information [Sha48] comme une branche mathématique à la suite des premiers codes correcteurs inventés par Richard W. Hamming : c'est le début de la théorie des codes [Ham50]. Elle consiste à ajouter de la redondance (augmenter le nombre de symboles) au message avant de l'envoyer sur un canal bruité. Si cette redondance est structurée, il est alors possible de corriger d'éventuelles erreurs introduites par le canal. On peut alors, malgré le bruit, retrouver l'intégralité du message de départ. Les codes correcteurs rendent possible la correction d'erreurs lorsque la communication se fait dans un canal bruité. Ils sont utilisés dans beaucoup d'applications, notamment en cryptographie. En 1978, Robert McEliece [McE78] a proposé le premier cryptosystème basé sur les codes correcteurs d'erreurs. Dans ce chapitre, nous allons faire quelques rappels sur la théorie des codes, notamment sur les codes linéaires et leur utilisation en cryptographie.

2.1 Codes linéaires

Soit $\mathcal{C}$ un code de dimension k et de longueur n et $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ un message à transmettre dans un canal bruité. Le message $\boldsymbol{x}$ est d'abord transformé en un mot de code $\boldsymbol{c}$ de longueur n : c'est l'encodage. Le mot de code $\boldsymbol{c}$ est ensuite transmis dans le canal et on obtient en sortie le mot bruité $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ ($\boldsymbol{e}$ représente les erreurs de transmission).

Enfin, le destinataire doit enlever ces erreurs de transmission dans $\boldsymbol{y}$ pour retrouver le message de départ $\boldsymbol{x}$: c'est le décodage.

Définition 2.1 (Code linéaire) *Soient q une puissance d'un nombre premier p et $\mathbb{F}_q$ le corps fini à q éléments. Un code $\mathcal{C}$ linéaire de paramètres $[n,k]_q$ sur $\mathbb{F}_q$, est un sous-espace vectoriel de $\mathbb{F}_q^n$ de dimension k et les éléments de $\mathcal{C}$ sont appelés mots de code.*

Un code correcteur d'erreurs peut être représenté de deux manières équivalentes, soit avec une matrice génératrice ou avec une matrice de contrôle.

Définition 2.2 (Matrice génératrice) *Une matrice génératrice du code $\mathcal{C}$ est une matrice $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ de k lignes et n colonnes telles que les k lignes forment une base de l'espace vectoriel $\mathcal{C}$ et on a*

$$\mathcal{C} = \{\boldsymbol{x}\boldsymbol{G} | \boldsymbol{x} \in \mathbb{F}_q^k\}.$$

Remarque 2.1 *La matrice génératrice d'un code n'est pas unique, il y en a autant que l'espace vectoriel $\mathcal{C}$ admet de bases. Elle permet de réaliser les opérations d'encodage et de permutation de lignes.*

Définition 2.3 (Fonction d'encodage) *Soit $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ une matrice génératrice de $\mathcal{C}$. La fonction d'encodage f associée à $\boldsymbol{G}$ est l'application*

$$\begin{aligned} f : \mathbb{F}_q^k &\longrightarrow \mathbb{F}_q^n \\ \boldsymbol{x} &\longmapsto \boldsymbol{x}\boldsymbol{G}, \end{aligned}$$

telle que son image $Im(f) = \mathcal{C}$ est le code linéaire associé.

Définition 2.4 (Codes équivalents) *Soient deux codes linéaires $\mathcal{C}$ et $\mathcal{C}'$ avec leurs matrices génératrices $\boldsymbol{G}$ et $\boldsymbol{G}'$ respectivement. Les deux codes $\mathcal{C}$ et $\mathcal{C}'$ sont dits équivalents par permutation si*

$$\boldsymbol{G} = \boldsymbol{Z}\boldsymbol{G}'\boldsymbol{P},$$

avec $\boldsymbol{Z} \in \mathbb{F}_q^{k\times k}$ une matrice inversible et $\boldsymbol{P} \in \mathbb{F}_q^{n\times n}$ une matrice de permutation.

Définition 2.5 (Support d'un mot) *Soit un vecteur $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n) \in \mathbb{F}_q^n$. On appelle support de $\boldsymbol{a}$, l'ensemble des indices de ses coordonnées non nulles*

$$Supp(\boldsymbol{a}) = \{i; \boldsymbol{a}_i \neq 0\}.$$

Définition 2.6 (Poids de Hamming) *Soit un vecteur $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n) \in \mathbb{F}_q^n$. Le poids de Hamming de $\boldsymbol{a}$ noté $w_H(\boldsymbol{a})$, est le cardinal de l'ensemble de ses coordonnées non nulles*

$$w_H(\boldsymbol{a}) = |\{i; \boldsymbol{a}_i \neq 0\}| = |Supp(\boldsymbol{a})|.$$

Définition 2.7 (Distance de Hamming) *Soient deux vecteurs* $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$ *et* $\boldsymbol{b} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{F}_q^n$. *La distance de Hamming entre* $\boldsymbol{a}$ *et* $\boldsymbol{b}$ *noté* $d_H(\boldsymbol{a}, \boldsymbol{b})$ *est le nombre de coordonnées distinctes entre les deux*

$$d_H(\boldsymbol{a}, \boldsymbol{b}) = |\{i; \boldsymbol{a}_i \neq \boldsymbol{b}_i\}|.$$

Autrement dit, c'est le nombre de positions i où $\boldsymbol{a}$ et $\boldsymbol{b}$ sont différents.
Dans le cas binaire ($q = 2$), on obtient

$$d_H(\boldsymbol{a}, \boldsymbol{b}) = w_H(\boldsymbol{a} \oplus \boldsymbol{b}),$$

où $\oplus$ est l'opérateur OU exclusif.

Définition 2.8 (Distance minimale) *Soit* $\mathcal{C}$ *un code,* $\forall \boldsymbol{c}, \boldsymbol{c}' \in \mathcal{C}$ *et* $\boldsymbol{c} \neq \boldsymbol{c}'$, *la distance minimale de* $\mathcal{C}$, *notée* d, *est la plus petite distance de Hamming* $d_H(\boldsymbol{c}, \boldsymbol{c}')$ *entre* $\boldsymbol{c}$ *et* $\boldsymbol{c}'$.

Remarque 2.2 *Si le code* $\mathcal{C}$ *est longueur* n, *de dimension* k *et de distance minimale,* d *on dit qu'il est de paramètres* $[n, k, d]_q$. *Un code* $\mathcal{C}$ *avec une distance minimale* d *peut détecter jusqu'à* $d - 1$ *erreurs. On parle de capacité de détection de* $\mathcal{C}$.

Définition 2.9 (Capacité de détection) *Soit* $\mathcal{C}$ *un code linéaire de paramètres* $[n, k, d]_q$. *La capacité de détection de* $\mathcal{C}$ *est la quantité* $d - 1$.

Définition 2.10 (Capacité de correction) *Soient* $\mathcal{C}$ *un code linéaire de paramètres* $[n, k, d]_q$ *et* t *un entier. La capacité de correction de* $\mathcal{C}$ *est définie par*

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

Définition 2.11 (Rayon de recouvrement) *Soient* $\boldsymbol{c} \in \mathcal{C}$ *et* $\boldsymbol{a} \in \mathbb{F}_q^n$. *Le rayon de recouvrement* ρ *du code* $\mathcal{C}$ *est la plus grande distance entre* $\mathcal{C}$ *et* $\mathbb{F}_q^n$ *définie par*

$$\rho = \max_{\boldsymbol{a} \in \mathbb{F}_q^n} \{d_H(\mathcal{C}, \boldsymbol{a})\} = \max\{\min_{\boldsymbol{c} \in \mathcal{C}} \{d_H(\boldsymbol{c}, \boldsymbol{a})\}\}.$$

Remarque 2.3 *Dans le cas où le rayon de recouvrement* ρ *de* $\mathcal{C}$ *est égal à sa capacité de correction* t, *on dit que le code est parfait. En d'autres termes, le code* $\mathcal{C}$ *est capable de corriger toutes les erreurs détectées. La correction des erreurs est réalisée par l'algorithme de décodage du code s'il existe (nous verrons dans la Section 2.2, page 20, le problème du décodage).*

Définition 2.12 (Matrice de contrôle) *Une matrice de contrôle (ou de parité) du code* $\mathcal{C}$ *est une matrice* $\boldsymbol{H} \in \mathbb{F}_q^{(n-k)\times n}$ *de* $n - k$ *lignes et* n *colonnes telles que*

$$\mathcal{C} = \{\boldsymbol{y} \in \mathbb{F}_q^n | \boldsymbol{H}\boldsymbol{y}^T = \boldsymbol{O}_{1,n-k}\},$$

où T *représente le symbole de la transposée.*

Remarque 2.4 *Une matrice de contrôle $\boldsymbol{H}$ de $\mathcal{C}$ est une matrice génératrice de son code dual (ou code orthogonal), noté $\mathcal{C}^{\perp}$. La matrice de contrôle permet de calculer la fonction syndrome.*

Définition 2.13 (Fonction syndrome) *Soient $\boldsymbol{y} \in \mathbb{F}_q^n$ et $\boldsymbol{H}$ une matrice de contrôle de $\mathcal{C}$. La fonction syndrome $s_{\boldsymbol{H}}$ associée à $\boldsymbol{H}$ est l'application*

$$\begin{aligned} s_{\boldsymbol{H}} : \mathbb{F}_q^k &\longrightarrow \mathbb{F}_q^{n-k} \\ \boldsymbol{y} &\longmapsto \boldsymbol{H}\boldsymbol{y}^T, \end{aligned}$$

où $s_{\boldsymbol{H}}(\boldsymbol{y})$ est appelé le syndrome de $\boldsymbol{y}$.

Remarque 2.5 *Si $\boldsymbol{y} \in \mathcal{C}$, alors $s_{\boldsymbol{H}}(\boldsymbol{y}) = 0$ et réciproquement.*

Il existe aussi une représentation particulière du code $\mathcal{C}$ avec $\boldsymbol{G}$ ou $\boldsymbol{H}$ dont les k premières lignes dans le mot de code $\boldsymbol{c}$ représentent le message $\boldsymbol{x}$ suivi de la redondance de taille $n-k$. Cette forme de représentation permet de réduire le coût de communication et d'accélérer le décodage. On parle de forme systématique.

Définition 2.14 (Forme systématique) *Une matrice génératrice $\boldsymbol{G}$ du code $\mathcal{C}$ est sous forme systématique si*

$$\boldsymbol{G} = [\boldsymbol{I}_k | \boldsymbol{A}],$$

où $\boldsymbol{I}_k \in \mathbb{F}_q^{k \times k}$ est la matrice identité et $\boldsymbol{A} \in \mathbb{F}_q^{k \times (n-k)}$.

Proposition 2.1 *Soient $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_q$ et $\boldsymbol{G}$ une matrice de $\mathcal{C}$ sous-forme systématique. Le code $\mathcal{C}$ admet alors une matrice de contrôle*

$$\boldsymbol{H} = [-\boldsymbol{A}^T | \boldsymbol{I}_{n-k}].$$

2.2 Problème du décodage

Contrairement à l'opération d'encodage, qui est un produit entre une matrice et un vecteur, le décodage est une opération plus complexe et implique différentes approches.

Problème 2.1 (Décodage borné) *Soient $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_q$, $\boldsymbol{y} \in \mathbb{F}_q^n$ et $i \leq t$ un entier. Le problème du décodage borné par i consiste à trouver un mot de code $\boldsymbol{c} \in \mathcal{C}$ tel que $d_H(\boldsymbol{c}, \boldsymbol{y}) \leq i$.*

Remarque 2.6 *L'algorithme de décodage borné ou à poids borné consiste d'abord à supposer qu'il n'y ait pas d'erreur sur le mot reçu $\boldsymbol{y}$ et donc on vérifie si $\boldsymbol{y}$ est un mot de code. Dans le cas où il y a des erreurs, on fait des hypothèses sur le poids (poids de Hamming) de l'erreur, $w_H(\boldsymbol{e}) = i$ avec $1 \leq i \leq t$, du mot reçu $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ et on vérifie si $\boldsymbol{y}$ est dans la liste ou non des mots de code avec une erreur de poids i.*

Problème 2.2 (Décodage complet) *Soient $\mathcal{C}$ un code linéaire de paramètres $[n, k, d]_q$ et $\boldsymbol{y} \in \mathbb{F}_q^n$. Le problème du décodage complet consiste à trouver un mot de code $\boldsymbol{c} \in \mathcal{C}$ tel que $d_H(\boldsymbol{c}, \boldsymbol{y}) = d_H(\boldsymbol{y}, \mathcal{C})$.*

Remarque 2.7 *L'algorithme de décodage complet consiste à trouver le mot de code le plus proche du mot de l'espace ambiant $\boldsymbol{y} \in \mathbb{F}_q^n$.*

Problème 2.3 (Décodage par syndrome) *Soient $\boldsymbol{H} \in \mathbb{F}_q^{(n-k)\times n}$ une matrice de contrôle, $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$ un vecteur et t un entier. Le problème du décodage par syndrome (Syndrome Decoding Problem ou SDP) consiste à trouver un vecteur d'erreur $\boldsymbol{e} \in \mathbb{F}_q^n$ de poids $w_H(\boldsymbol{e}) \leq t$ tel que $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}$.*

Données.

$\boldsymbol{H}$:	*matrice de taille $(n-k) \times n$ dans $\mathbb{F}_q$*
$\boldsymbol{s}$:	*vecteur de $\mathbb{F}_q^{n-k}$*
t :	*entier*

Problème.

Existe-t-il un vecteur $\boldsymbol{e} \in \mathbb{F}_q^n$ de poids $w_H(\boldsymbol{e}) \leq t$ tel que : $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}$?

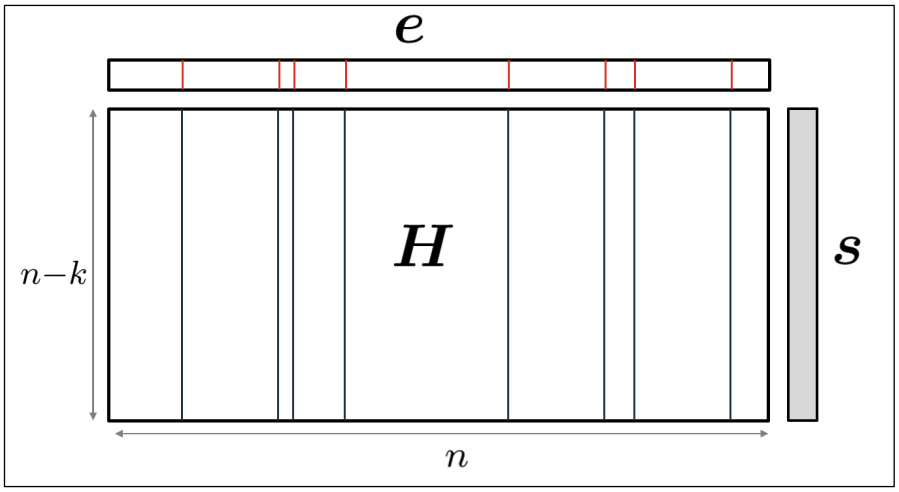

FIGURE 2.1 – Problème de décodage par syndrome.

Remarque 2.8 *Dans le cas binaire, Le problème du décodage par syndrome consiste à trouver les t colonnes de $\boldsymbol{H}$ qui ont été XORées ($\oplus$) sur $\mathbb{F}_2$ pour trouver le vecteur $\boldsymbol{s}$ (Figure*

2.1). Ce problème a été montré $\mathcal{NP}$-complet dans [BMVT78], c'est-à-dire qu'il ne peut être résolu que par une classe restreinte d'algorithmes de recherche exhaustive. C'est autour de ce problème que repose la cryptographie basée sur les codes. La meilleure stratégie pour résoudre le SDP sur $\mathbb{F}_2$ est le décodage par ensemble d'information.

Problème 2.4 (Décodage par ensemble d'information) *Soient $\boldsymbol{c} \in \mathcal{C}$, $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ une matrice génératrice du code $\mathcal{C}$, $\boldsymbol{e} \in \mathbb{F}_q^n$ un vecteur d'erreur de poids $w_H(\boldsymbol{e}) \leq t$ et $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_q^n$. Le problème du décodage par ensemble d'information (Information Set Decoding ou ISD) consiste à trouver les k symboles d'information de $\boldsymbol{x} \in \mathbb{F}_q^k$ dans $\boldsymbol{c}$ tels que $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{G} = \boldsymbol{c}$.*

Données.

$\boldsymbol{G}$:	*matrice de taille $k \times n$ dans $\mathbb{F}_q$*
$\boldsymbol{e}$:	*vecteur d'erreur de $\mathbb{F}_q^n$*
$\boldsymbol{y}$:	*vecteur d'erreur de $\mathbb{F}_q^n$*
t :	*entier*

Problème.

Peut-on trouver $\boldsymbol{x} \in \mathbb{F}_q^k$ tel que : $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{G} = \boldsymbol{c}$?

Remarque 2.9 *L'algorithme de décodage par ensemble d'information consiste d'abord à choisir aléatoirement k des n coordonnées dans $\boldsymbol{c}$ dans l'espoir qu'elles soient toutes sans erreurs. Ensuite, on essaie de récupérer le message $\boldsymbol{x}$ en résolvant un système linéaire $k \times k$ sur $\mathbb{F}_q$ (ou $\mathbb{F}_2$, dans le cas binaire).*

2.3 Exemples de codes

Dans cette section, nous allons présenter quelques codes linéaires dont les propriétés seront utilisées dans cette thèse pour construire des cryptosystèmes basés sur les codes.

2.3.1 Code quasi-dyadique

Les codes quasi-dyadiques (Quasi-Dyadic ou QD) que nous présentons ici, sont des codes avec une construction particulière dont nous allons exploiter les propriétés dans notre implantation logicielle du cryptosystème appelé DAGS [Gus+17] au Chapitre 3.

Définition 2.15 (Matrice dyadique) *Soit $\boldsymbol{d} = (\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1}) \in \mathbb{F}_q^n$ un vecteur (avec q, une puissance de 2). Une matrice $\boldsymbol{D} \in \mathbb{F}_q^{n\times n}$ est dite dyadique si*

$$\boldsymbol{D}_{i,j} = \boldsymbol{d}_{i\oplus j}.$$

On dit que $\boldsymbol{d}$ est la signature de $\boldsymbol{D}$ (ou la première ligne de $\boldsymbol{D}$).

Proposition 2.2 *Une matrice dyadique $\boldsymbol{D}$ est entièrement déterminée par sa première ligne (ou colonne).*

Preuve 2.1 *Soit $\boldsymbol{D}$, une matrice dyadique avec $\boldsymbol{d} = (\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1})$.*
Pour tout $j = 0 \ldots n-1$, on a :

$$\boldsymbol{D}_{0,j} = \boldsymbol{d}_{0 \oplus j} = \boldsymbol{d}_j.$$

Proposition 2.3 *Soit $\boldsymbol{D} \in \mathbb{F}_q^{n \times n}$ une matrice dyadique, il existe $s \in \mathbb{N}$ tel que $n = 2^s$.*

Preuve 2.2 *Supposons que $n = 2^s m$, avec $m > 1$ et impair. Alors, il existe $p \geq 1$ tel que $m = 2p + 1$ et on a :*

$$n = 2^s(2p+1) = 2^s + 2^{s+1}p.$$

Donc il existe i tel que 0 est l'élément à l'indice i dans la représentation binaire de $n-1$ et on a :

$$2^{i-1} \geq n - 1.$$

Alors pour l'élément $\boldsymbol{D}_{n-1,2^{i-1}}$, on a :

$$n - 1 \oplus 2^{i-1} = n - 1, \text{ ce qui est contradictoire.}$$

Proposition 2.4 *Soit $n = 2^s$, le produit de 2 matrices dyadiques $2^s \times 2^s$ est une matrice dyadique $2^s \times 2^s$.*

Proposition 2.5 *L'inverse d'une matrice dyadique $2^s \times 2^s$ est aussi une matrice $2^s \times 2^s$ dyadique.*

Définition 2.16 (Matrice quasi-dyadique) *Une matrice QD (Figure 2.2) est une matrice par blocs dont les composantes (blocs) sont des matrices dyadiques.*

Définition 2.17 (Matrice de permutation dyadique) *Une matrice de permutation dyadique est une matrice dyadique dont la signature admet un seul élément égal à 1, les autres éléments étant nuls.*

Définition 2.18 (Code quasi-dyadique) *Soient $n_0 \in \mathbb{N}^*$ et $\ell = 2^s$, avec $s \geq 2$. Un code QD est un code linéaire de longueur $n = \ell n_0$ qui admet une matrice de contrôle QD. Un code quasi-dyadique est déterminé par son indice n_0 (nombre de blocs dyadique) et son ordre ℓ (taille de chaque bloc).*

Définition 2.19 (Matrice de Cauchy) *Soient $\boldsymbol{u} = (\boldsymbol{u}_1 \ldots, \boldsymbol{u}_n) \in \mathbb{F}_q^n$ et $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in \mathbb{F}_q^m$ tels que $\boldsymbol{u}_i \neq \boldsymbol{v}_j$ pour tout (i,j). Une matrice de Cauchy $\boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_q^{n \times m}$ associée à $\boldsymbol{u}$ et $\boldsymbol{v}$ est une matrice définie par $\boldsymbol{C}_{ij} = \frac{1}{\boldsymbol{u}_i - \boldsymbol{v}_j}$ telle que*

$$\boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v}) = \begin{pmatrix} \frac{1}{\boldsymbol{u}_1 - \boldsymbol{v}_1} & \cdots & \frac{1}{\boldsymbol{u}_1 - \boldsymbol{v}_m} \\ \vdots & \vdots & \vdots \\ \frac{1}{\boldsymbol{u}_n - \boldsymbol{v}_1} & \cdots & \frac{1}{\boldsymbol{u}_n - \boldsymbol{v}_m} \end{pmatrix}.$$

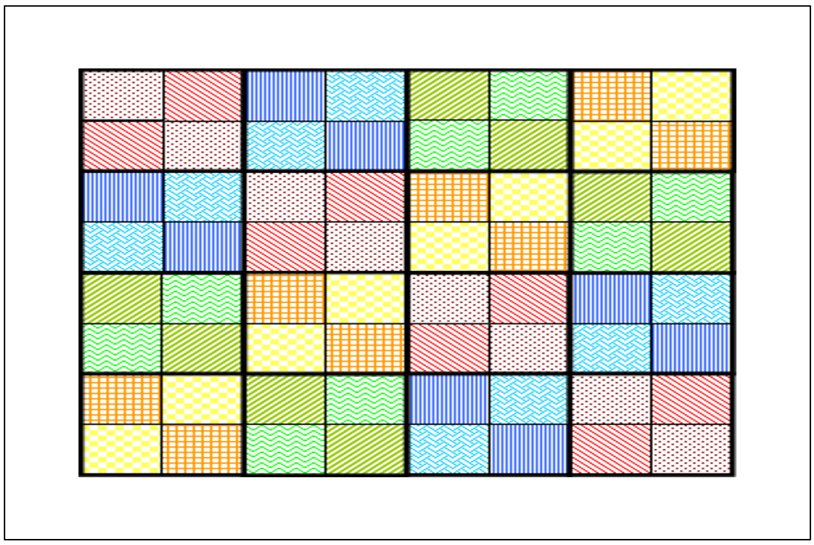

FIGURE 2.2 – Matrice dyadique.

2.3.2 Codes de Srivastava Généralisés

Les codes de Srivastava ont été proposés en 1967, mais publiés 5 ans plus tard par Hermann Helgert [Hel72].

Définition 2.20 (Code de Srivastava Généralisé) *Soient $m, n, s, t \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_n$ et $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_s$ éléments distincts de $\mathbb{F}_{q^m}$, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ éléments non nuls de $\mathbb{F}_{q^m}$. Un code de Srivastava Généralisé (Generalized Srivastava ou GS) d'ordre st, de longueur $n \leq q^m - st$, de dimension $k \geq n - mst$ et de distance minimale $d \geq st + 1$ (distance construite) est défini par une matrice de contrôle $\boldsymbol{H}$ de la forme*

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_1 \\ \vdots \\ \boldsymbol{H}_s \end{pmatrix},$$

où chaque bloc $\boldsymbol{H}_i$ est donné par

$$\boldsymbol{H}_i = \begin{pmatrix} \frac{\boldsymbol{z}_1}{\alpha_1 - \boldsymbol{w}_i} & \cdots & \frac{\boldsymbol{z}_n}{\alpha_n - \boldsymbol{w}_i} \\ \vdots & \vdots & \vdots \\ \frac{\boldsymbol{z}_1}{(\alpha_1 - \boldsymbol{w}_i)^t} & \cdots & \frac{\boldsymbol{z}_n}{(\alpha_n - \boldsymbol{w}_i)^t} \end{pmatrix}.$$

Proposition 2.6 *Si on permute les lignes de $\boldsymbol{H}$ pour constituer des $s \times n$ blocs de la forme*

$$\hat{\boldsymbol{H}}_i = \begin{pmatrix} \frac{\boldsymbol{z}_1}{(\alpha_1 - \boldsymbol{w}_1)^i} & \cdots & \frac{\boldsymbol{z}_n}{(\alpha_n - \boldsymbol{w}_1)^i} \\ \vdots & \vdots & \vdots \\ \frac{\boldsymbol{z}_1}{(\alpha_1 - \boldsymbol{w}_s)^i} & \cdots & \frac{\boldsymbol{z}_n}{(\alpha_n - \boldsymbol{w}_s)^i} \end{pmatrix},$$

nous obtenons une matrice de contrôle équivalente pour un code GS, donnée par

$$\hat{\boldsymbol{H}} = \begin{pmatrix} \hat{\boldsymbol{H}}_1 \\ \vdots \\ \hat{\boldsymbol{H}}_t \end{pmatrix}.$$

où chaque bloc $\hat{\boldsymbol{H}}_i$ (avec $i = 1, \ldots, t$) est une matrice de Cauchy.

2.3.3 Codes de Reed-Solomon

Les codes de Reed-Solomon sont des codes d'évaluation et ont été proposés par Irving Reed et Gustave Solomon en 1960 [RS60].

Définition 2.21 (Code de Reed-Solomon) *Soient k un entier et $p \in \mathbb{F}_q[x]$ un polynôme de degré strictement inférieur à k. Un code de Reed-Solomon (RS) de longueur n, de dimension k et de support $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$, avec $\forall i \neq j, \alpha_i \neq \alpha_j$, est défini par*

$$RS_{n,k}(\alpha) = \{(p(\alpha_1), \ldots, p(\alpha_n)) \in \mathbb{F}_q^n : deg(p) < k\}.$$

Remarque 2.10 *La définition des codes RS impose que les éléments α_i du support α soient tous distincts, ce qui entraîne $n \leq q$.*

Théorème 2.1 *Un code RS de longueur n et de dimension k a pour distance minimale $d = n - k + 1$.*

Proposition 2.7 *Une matrice génératrice $\boldsymbol{G}$ du code RS de support $\alpha = (\alpha_1, \ldots, \alpha_n)$ est définie par*

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_{n-1} & \alpha_n \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_{n-1}^{k-1} & \alpha_n^{k-1} \end{pmatrix}.$$

Définition 2.22 (Code de Reed-Solomon généralisé) *Soient $\mathcal{C}$ un code de Reed-Solomon $RS_{n,k}(\alpha)$ et $\beta = (\beta_1, \ldots, \beta_n)$ un vecteur dans $\mathbb{F}_{q^m}$. Un code de Reed-Solomon généralisé (Generalized Reed-Solomon ou GRS) de multiplicateur β est défini par*

$$GRS_k(\alpha, \beta) = \{(\beta_1 \cdot p(\alpha_1), \ldots, \beta_n \cdot p(\alpha_n)) \in \mathbb{F}_q^n : p \in \mathbb{F}_q[x], deg(p) < k\}.$$

Remarque 2.11 *Un code RS est un code GRS avec $\beta_i = 1, \forall i \in \{1, \ldots, n\}$ et son rayon de recouvrement est $\rho = n - k$.*

Proposition 2.8 *Une matrice génératrice $\boldsymbol{G}$ du code GRS de support $\alpha = (\alpha_1, \ldots, \alpha_n)$ et de multiplicateur $\beta = (\beta_1, \ldots, \beta_n)$ est définie par*

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_{n-1} & \alpha_n \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_{n-1}^{k-1} & \alpha_n^{k-1} \end{pmatrix} \begin{pmatrix} \beta_1 & & & & 0 \\ & \beta_2 & & & \\ & & \ddots & & \\ & & & \beta_{n-1} & \\ 0 & & & & \beta_n \end{pmatrix}.$$

Définition 2.23 (Code alternant) *Soient r un entier positif, $GRS_{n-r}(\alpha, \beta)$, un code GRS de paramètres $[n, n-r, r+1]$ construit sur $\mathbb{F}_{q^m}$ et $\gamma = (\gamma_1, \ldots, \gamma_n)$, un vecteur à éléments non nuls (non nécessairement distincts) dans $\mathbb{F}_{q^m}$. Un code alternant $A_k(\alpha, \gamma)$ est défini comme l'ensemble des mots de $GRS_{n-r}(\alpha, \beta)$ tel que*

$$A_k(\alpha, \gamma) = GRS_{n-r}(\alpha, \beta) \cap \mathbb{F}_q^n.$$

Théorème 2.2 *Une matrice de contrôle $\boldsymbol{H}$ du code alternant $A_k(\alpha, \gamma)$ est une matrice sur $\mathbb{F}_q$ dont les $h_{i,j} \in \mathbb{F}_{q^m}$ sont des vecteurs colonnes à m éléments sur $\mathbb{F}_{q^m}$ telle que*

$$\boldsymbol{H} = \underbrace{\begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_{n-1} & \alpha_n \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \ldots & \alpha_{n-1}^{r-1} & \alpha_n^{r-1} \end{pmatrix}}_{\boldsymbol{V}} \begin{pmatrix} \gamma_1 & & & & 0 \\ & \gamma_2 & & & \\ & & \ddots & & \\ & & & \gamma_{n-1} & \\ 0 & & & & \gamma_n \end{pmatrix}$$

$$= \begin{pmatrix} \gamma_1 & \gamma_2 & \ldots & \gamma_{n-1} & \gamma_n \\ \alpha_1\gamma_1 & \alpha_2\gamma_2 & \ldots & \alpha_{n-1}\gamma_{n-1} & \alpha_n\gamma_n \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{r-1}\gamma_1 & \alpha_2^{r-1}\gamma_2 & \ldots & \alpha_{n-1}^{r-1}\gamma_{n-1} & \alpha_n^{r-1}\gamma_n \end{pmatrix},$$

où $\boldsymbol{V}$ est la matrice de Vandermonde.

2.3.4 Codes de Goppa classiques

Les codes de Goppa sont des codes linéaires proposés en 1970 par V. D. Goppa [Gop70]. Il existe deux classes de codes de Goppa : les codes de Goppa géométriques définis sur des courbes algébriques et les codes de Goppa classiques définis sur les corps finis qui nous intéressent particulièrement dans cette thèse.

Définition 2.24 (Code de Goppa) *Soient $m > 0$, $n \leq q^m$, $\mathcal{L} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{q^m}^n$ le support et $g(x) \in \mathbb{F}_{q^m}[x]$ un polynôme de degré t tel que $g(\alpha_i) \neq 0$ pour tous les α_i. Un code de Goppa de longueur n est défini par*

$$\Gamma(\mathcal{L}, g) = \left\{ \boldsymbol{c} \in \mathbb{F}_q^n : \sum_{i=1}^{n} \frac{\boldsymbol{c}_i}{x - \alpha_i} \equiv 0 \mod g(x) \right\},$$

où g est appelé polynôme de Goppa.

Proposition 2.9 *Soit le code de Goppa $\Gamma(\mathcal{L}, g)$ de longueur n, alors sa dimension $k \geq n - mt$ et sa distance minimale $d \geq t + 1$.*

Proposition 2.10 *Soit un polynôme de Goppa $g(x) \in \mathbb{F}_{q^m}[x]$ sans racine multiple sur $\mathbb{F}_{q^m}$. Alors*

$$\Gamma(\mathcal{L}, g^{q-1}) = \Gamma(\mathcal{L}, g^q).$$

Corollaire 2.1 *Soit le code de Goppa $\Gamma(\mathcal{L}, g^{q-1})$ avec $g(x)$ ne possédant pas de racine sur $\mathbb{F}_q^m$. Alors sa distance minimale $d \geq qt + 1$.*

Un résultat très intéressant de ce corollaire est que pour le cas binaire, nous obtenons la distance minimale $d \geq 2t + 1$. Par conséquent, l'algorithme de décodage d'un code de Goppa binaire permet de corriger jusqu'à t erreurs.

Remarque 2.12 *Le code de Goppa $\Gamma(\mathcal{L}, g)$ est irréductible si le polynôme de Goppa $g(x)$ est irréductible sur $\mathbb{F}_q$.*

Remarque 2.13 *Le nombre de polynômes de Goppa irréductibles $g(x)$ de degré t à coefficients dans $\mathbb{F}_{q^m}$ est d'environ $\frac{q^{mt}}{t}$.*

Proposition 2.11 *Un code de Goppa $\Gamma(\mathcal{L}, g)$ est un code alternant de dimension $n - r$ tel que*

$$\Gamma(\mathcal{L}, g) = GRS_{n-r}(\mathcal{L}, \beta_i) \cap \mathbb{F}_q^n,$$

où $\beta_i = \frac{g(\alpha_i)}{\prod_{j \neq i}^{n}(\alpha_i - \alpha_j)}$.

Théorème 2.3 *Une matrice de contrôle $\boldsymbol{H}$ du code de Goppa $\Gamma(\mathcal{L}, g)$ est une matrice dans $\mathbb{F}_{q^m}$ telle que*

$$\boldsymbol{H} = \underbrace{\begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_{n-1} & \alpha_n \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_{n-1}^{t-1} & \alpha_n^{t-1} \end{pmatrix}}_{\boldsymbol{V}} \begin{pmatrix} \frac{1}{g(\alpha_1)} & & & & 0 \\ & \frac{1}{g(\alpha_2)} & & & \\ & & \ddots & & \\ & & & \frac{1}{G(\alpha_{n-1})} & \\ 0 & & & & \frac{1}{G(\alpha_n)} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \dots & \frac{1}{g(\alpha_{n-1})} & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_2}{g(\alpha_2)} & \dots & \frac{\alpha_{n-1}}{g(\alpha_{n-1})} & \frac{\alpha_n}{g(\alpha_n)} \\ \vdots & \vdots & & \vdots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} & \dots & \frac{\alpha_{n-1}^{t-1}}{g(\alpha_{n-1})} & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{pmatrix}.$$

Dans la suite de cette thèse, nous ne considérons que le cas des codes de Goppa binaires irréductibles ($q = 2$).

Encodage Pour encoder un message $\boldsymbol{x}$ de longueur k sur $\mathbb{F}_2$ en un mot de code $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$, on a

$$\boldsymbol{c} = \boldsymbol{x}\boldsymbol{G}.$$

Décodage Soit le mot erroné reçu

$$\boldsymbol{y} = \boldsymbol{c} \oplus \boldsymbol{e} = \underbrace{(\boldsymbol{c}_1, \dots, \boldsymbol{c}_n)}_{\text{mot de code}} + \underbrace{(\boldsymbol{e}_1, \dots, \boldsymbol{e}_n)}_{\text{vecteur d'erreur}},$$

avec $\boldsymbol{e}$ un vecteur d'erreur de poids $w_H(\boldsymbol{e}) \leq t$.
Pour corriger l'erreur dans $\boldsymbol{y}$, il faut

1. localiser les $w_H(\boldsymbol{e})$ positions dans $\boldsymbol{e}$;
2. trouver les valeurs de $\boldsymbol{e}$ correspondantes.

Pour cela, nous allons définir les polynômes suivants.

Définition 2.25 (Polynôme localisateur d'erreurs) *Le polynôme localisateur $\sigma(x)$ de l'erreur $\boldsymbol{e}$ dans $\boldsymbol{y}$ est défini par :*

$$\sigma(x) = \prod_{\substack{i=1 \\ \boldsymbol{e}_i \neq 0}}^{n} (x - \alpha_i), \text{ avec } deg(\sigma) \leq t.$$

Les racines du polynôme $\sigma(x)$ permettent de retrouver les positions $w_H(\boldsymbol{e})$ positions dans $\boldsymbol{e}$.

Définition 2.26 (Polynôme évaluateur d'erreurs) *Le polynôme évaluateur $\theta(x)$ de l'erreur $\boldsymbol{e}$ dans $\boldsymbol{y}$ est défini par :*

$$\theta(x) = \sum_{\substack{i=1 \\ \boldsymbol{e}_i \neq 0}}^{n} \prod_{\substack{j=1 \\ \boldsymbol{e}_j \neq 0 \\ j \neq i}}^{n} (x - \alpha_j), \text{ avec } deg(\theta) \leq (t-1).$$

Le polynôme $\theta(x)$ permet de trouver les valeurs de $\boldsymbol{e}$.

Définition 2.27 (Polynôme syndrome) *Le polynôme syndrome $s_{\boldsymbol{y}}(x)$ du mot reçu $\boldsymbol{y}$ est donné par :*

$$s_{\boldsymbol{y}}(x) \equiv \sum_{i=1}^{n} \frac{\boldsymbol{y}_i}{x - \alpha_i} \mod g(x).$$

En remplaçant $\boldsymbol{y} = \boldsymbol{c} \oplus \boldsymbol{e}$ dans $s_{\boldsymbol{y}}(x)$, on obtient

$$\begin{aligned} s_{\boldsymbol{y}}(x) &\equiv \sum_{i=1}^{n} \frac{\boldsymbol{c}_i}{x - \alpha_i} \mod g(x) \oplus \sum_{i=1}^{n} \frac{\boldsymbol{e}_i}{x - \alpha_i} \mod g(x) \\ &\equiv \sum_{\substack{i=1 \\ \boldsymbol{e}_i = 1}}^{n} \frac{1}{x - \alpha_i} \mod g(x) \end{aligned}$$

avec $s_{\boldsymbol{c}}(x) = 0$ car $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$.

D'après les Définitions 2.25 et 2.26, on a

$$\frac{\theta(x)}{\sigma(x)} = \sum_{\substack{i=1 \\ \boldsymbol{e}_i = 1}}^{n} \frac{1}{x - \alpha_i},$$

et donc

$$s_{\boldsymbol{y}}(x) \equiv \frac{\theta(x)}{\sigma(x)} \mod g(x).$$

Ainsi pour corriger $\boldsymbol{y}$ en $\boldsymbol{c}$ connaissant $s(x)$ il faut résoudre

$$s(x)\sigma(x) \equiv \theta(x) \mod g(x) : \text{ c'est l'équation-clef.} \tag{2.1}$$

Pour alléger les notations pour la suite $s_{\boldsymbol{y}}(x) = s(x)$.
Il existe différentes méthodes permettant de résoudre cette équation-clef notamment avec l'algorithme d'Euclide étendu ou celui de Patterson. Mais avant nous allons rappeler les grandes lignes du principe de décodage d'un code de Goppa dans l'Algorithme 1.

Algorithme 1 Décodage d'un code de Goppa

Entrée: un mot reçu $\boldsymbol{y} = \boldsymbol{c} \oplus \boldsymbol{e}$ avec $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$ et $\boldsymbol{e}$ un vecteur d'erreur de poids $w_H(\boldsymbol{e}) \leq t$, le support $\mathcal{L} = (\alpha_1, \ldots, \alpha_n)$ et le polynôme de Goppa $g(x)$.
Sortie: $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$.

1: Calculer le syndrome de $\boldsymbol{y}$: $s(x)$.
2: Résoudre l'équation-clef : $s(x)\sigma(x) \equiv \theta(x) \mod g(x)$.
// afin de trouver le polynôme localisateur d'erreurs $\sigma(x)$
3: Construire le vecteur $\boldsymbol{e}$ tel que si $\sigma(\alpha_i) = 0$, $\boldsymbol{e}_i = 1$ et $\boldsymbol{e}_i = 0$ sinon.
// on cherche les racines de σ sur le support $\mathcal{L}$, on parle aussi d'évaluation de σ sur $\mathcal{L}$
4: **Retourner** $\boldsymbol{c} = \boldsymbol{y} - \boldsymbol{e}$.

Algorithme d'Euclide étendu (Extended Euclidean Algorithm ou EEA) Grâce à l'algorithme d'Euclide étendu, on peut résoudre l'équation-clef 2.1, c'est-à-dire trouver le polynôme localisateur $\sigma(x)$ et le polynôme évaluateur $\theta(x)$ de l'erreur tels que :

$$s(x)\sigma(x) \equiv \theta(x) \mod g(x)$$

Plus précisément, il existe un polynôme $u(x) \in \mathbb{F}_{2^m}[x]$ tel que

$$s(x)\sigma(x) + u(x)g(x) = \theta(x) \tag{2.2}$$

Pour retrouver $\sigma(x)$ et $\theta(x)$ avec l'EEA, il suffit de mettre en entrée le polynôme syndrome $s(x)$ que l'ont peut calculer à partir de $\boldsymbol{y}$ et le polynôme de Goppa $g(x)$. En appliquant le principe de l'EEA dont le résultat est unique à multiplication par un inversible près, dans notre cas par un scalaire $\alpha \in \mathbb{F}_2$, on a donc en sortie un couple candidat (σ, θ) qui satisfait l'égalité dans l'Équation 2.2. On obtient alors l'Algorithme 2.

Algorithme de Patterson L'algorithme le plus efficace pour le décodage des codes de Goppa binaires $\Gamma(\mathcal{L}, g)$ dans $\mathbb{F}_{2^m}$ est l'algorithme Patterson en 1975 [Pat75]. La méthode de Patterson, décrit dans l'Algorithme 3, permet de décoder les codes de Goppa binaires jusqu'à t, sa capacité de correction, en utilisant les propriétés de la caractéristique ($q = 2$) pour réduire de moitié la taille des polynômes dans l'équation-clef. Le polynôme localisateur $\sigma(x)$ est divisé en puissances paire et impaire de x telles que

$$\sigma(x) = u^2(x) + xv^2(x).$$

Les codes de Goppa binaires ont été utilisés pour la première fois en 1978 par Robert McEliece [McE78] pour faire du chiffrement : c'est le début de la cryptographie basée sur les codes.

Algorithme 2 Décodage par Euclide étendu

Entrée: un mot reçu $\boldsymbol{y} = \boldsymbol{c} \oplus \boldsymbol{e}$ avec $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$ et $\boldsymbol{e}$ un vecteur d'erreur de poids $w_H(\boldsymbol{e}) \leq t$, le support $\mathcal{L} = (\alpha_1, \ldots, \alpha_n)$ et le polynôme de Goppa $g(x)$.
Sortie: $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$.

$r_{-1} \leftarrow s(x), r_0 \leftarrow g(x)$;
$u_{-1} \leftarrow 1, v_{-1} \leftarrow 0$;
$u_0 \leftarrow 0, v_0 \leftarrow 1$;
$i \leftarrow 1$;
tant que $deg(r_i) \neq 0$ **faire**
    $q \leftarrow r_{i-1}/r_i$;
    $r_{i+1} \leftarrow r_{i-1} - qr_i$;
    $u_{i+1} \leftarrow u_{i-1} - qu_i$;
    $v_{i+1} \leftarrow v_{i-1} - qv_i$;
    $i++$;
fin tant que
$\sigma(x) \leftarrow u_i(x)$;
$\theta(x) \leftarrow r_i(x)$;
Retourner$(\sigma(x), \theta(x))$.

2.4 Chiffrement

Le cryptosystème de McEliece est un schéma de chiffrement asymétrique utilisant les codes de Goppa binaires et repose sur le problème difficile du décodage par syndrome binaire. Au début, ce schéma n'a pas rencontré un véritable soutien dans la communauté cryptographique, car la clef publique est particulièrement grande et que le texte chiffré est deux fois plus long que le message. Pourtant, le schéma de McEliece possède des propriétés intéressantes : la sécurité croît beaucoup plus avec la taille des clefs comparé à RSA et le chiffrement est plus rapide. Un autre avantage est de reposer sur un problème très différent des algorithmes asymétriques usuels. Cela signifie, par exemple, qu'une percée théorique dans le domaine de la factorisation, qui briserait éventuellement RSA, n'affecterait pas ce schéma.

Algorithme 3 Décodage par Patterson

Entrée: $s(x)$, le polynôme syndrome de $\boldsymbol{y} = \boldsymbol{c} \oplus \boldsymbol{e}$, le support $\mathcal{L} = (\alpha_1, \ldots, \alpha_n)$ et le polynôme de Goppa $g(x)$.
Sortie: $\boldsymbol{c} \in \Gamma(\mathcal{L}, g)$.

si $s(x) = 0$ **alors**
  $\sigma(x) = 1$
sinon
  Calculer $t(x) = s^{-1}(x) \mod g(x)$
  // en utilisant l'Algorithme 2 : premier appel à l'EEA.
  si $t(x) = x$ **alors**
    $\sigma(x) = x$
  sinon
    Calculer $s(x) = \sqrt{x} \mod g(x)$
    // en posant $s(x) = x^{2^{tm-1}} \mod g(x)$.
    Calculer $h(x) = \sqrt{t(x) + x} \mod g(x)$
    Calculer $u(x)$ et $v(x)$ tels que $u(x) = v(x)h(x) \mod g(x)$
    // en utilisant l'Algorithme 2 : deuxième appel à l'EEA.
    Construire $\sigma(x) = u^2(x) + xv^2(x)$
  fin si
fin si
Construire le vecteur $\boldsymbol{e}$ tel que si $\sigma(\alpha_i) = 0$, $\boldsymbol{e}_i = 1$ et $\boldsymbol{e}_i = 0$ sinon.
Retourner $\boldsymbol{c} = \boldsymbol{y} - \boldsymbol{e}$.

De nombreuses variantes du schéma de McEliece utilisant d'autres codes ont été proposés pour réduire la taille de cette clef publique tout en espérant garder le même niveau de sécurité. On verra dans le chapitre 3 un moyen de réduire la taille de la clef publique. Le processus de standardisation lancé par le NIST (National Institute of Standards and Technology) depuis décembre 2016 [Nis], que nous verrons dans la Section 2.6 (page 39), a donné un élan considérable à la recherche sur la cryptographie basée sur les codes correcteurs d'erreurs.

2.4.1 Cryptosystème de McEliece

L'idée de McEliece [McE78] est de masquer la structure (qui permet de décoder) du mot de code correspondant au message en lui ajoutant autant d'erreurs que possible tout en gardant la possibilité de les corriger. Les algorithmes de génération des clefs, de chiffrement et de déchiffrement sont décrits ci-dessous.

Génération des clefs La génération des clefs consiste à choisir une matrice génératrice $\boldsymbol{G}$ (c'est la clef privée sk) d'un code linéaire $\mathcal{C}$ que l'on sait décoder. Cette matrice génératrice est ensuite transformée en $\boldsymbol{G}'$ qui semble aléatoire (c'est la clef publique pk). L'algorithme de génération des clefs dans le schéma de McEliece (Algorithme 4) prend

en entrée les paramètres du code $\mathcal{C}$ n, k et t et retourne la paire de clefs publique $(\boldsymbol{G}', t)$ et privée $(\boldsymbol{Z}, \boldsymbol{G}, \boldsymbol{P})$.

Algorithme 4 Génération des clefs McEliece

Entrée: n, k et t des entiers, avec $n \leq 2^m$.
Sortie: $\mathsf{pk} = (\boldsymbol{G}', t)$ et $\mathsf{sk} = (\boldsymbol{Z}, \boldsymbol{G}, \boldsymbol{P})$.

1: Choisir une matrice génératrice $\boldsymbol{G} \in \mathbb{F}_2^{k \times n}$ d'un code linéaire $\mathcal{C}$ que l'on sait décoder.
2: Choisir une matrice de permutation $\boldsymbol{P} \in \mathbb{F}_2^{n \times n}$.
3: Choisir une matrice inversible $\boldsymbol{Z} \in \mathbb{F}_2^{k \times k}$.
4: Calculer $\boldsymbol{G}' = \boldsymbol{Z}\boldsymbol{G}\boldsymbol{P}$.
5: $\mathsf{pk} \leftarrow (\boldsymbol{G}', t)$
6: $\mathsf{sk} \leftarrow (\boldsymbol{Z}, \boldsymbol{G}, \boldsymbol{P})$
7: **Retourner** (pk, sk).

Chiffrement Le chiffrement dans le schéma de McEliece est grossièrement une opération d'encodage qui consiste à masquer le mot de code $\boldsymbol{c}$ correspondant au message $\boldsymbol{x}$ en lui ajoutant des erreurs $\boldsymbol{e}$ que l'on sait corriger. L'algorithme de chiffrement (Algorithme 5) prend en entrée un message $\boldsymbol{x} \in \mathbb{F}_2^k$ et la clef publique $(\boldsymbol{G}', t)$ et retourne le texte chiffré $\boldsymbol{y} \in \mathbb{F}_2^n$.

Algorithme 5 Chiffrement McEliece

Entrée: $\mathsf{pk} = (\boldsymbol{G}', t)$ la clef publique et $\boldsymbol{x} \in \mathbb{F}_2^k$ un message à chiffrer.
Sortie: $\boldsymbol{y} \in \mathbb{F}_2^n$ le texte chiffré associé à $\boldsymbol{x}$.

1: Choisir $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_2^n$, une erreur aléatoire de poids t.
2: Calculer $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{G}' \oplus \boldsymbol{e}$.
3: **Retourner** $\boldsymbol{y}$.

Déchiffrement Le déchiffrement dans le schéma de McEliece est une opération de décodage qui consiste à retrouver le message original $\boldsymbol{x}$ à partir du texte chiffré $\boldsymbol{y}$. L'algorithme de déchiffrement (Algorithme 6) prend en entrée la clef privée $(\boldsymbol{Z}, \boldsymbol{G}, \boldsymbol{P})$ ainsi que le texte chiffré $\boldsymbol{y} \in \mathbb{F}_2^n$ et retourne le message $\boldsymbol{x} \in \mathbb{F}_2^k$.

Sécurité Eve possède deux vecteurs d'attaques sur le schéma de McEliece. Elle peut essayer de reconstruire la matrice privée $\boldsymbol{G}'$ à partir de la matrice publique $\boldsymbol{G}$ et utiliser ensuite l'algorithme de décodage pour décoder. Cependant, le cryptosystème de McEliece utilise les codes de Goppa binaires qui sont faciles à décoder, mais une fois leur structure masquée (en engendrant un code de Goppa équivalent par multiplication de $\boldsymbol{Z}$ puis de $\boldsymbol{P}$), même si l'on arrive à les distinguer d'un code aléatoire, on ne peut pas retrouver cette structure (retrouver les matrices $\boldsymbol{Z}$ et $\boldsymbol{P}$) du code de Goppa utilisé [Fau+13].

Algorithme 6 Déchiffrement McEliece

Entrée: $\mathsf{sk} = (\boldsymbol{Z}, \boldsymbol{G}, \boldsymbol{P})$ la clef privée et $\boldsymbol{y} \in \mathbb{F}_2^n$ le texte chiffré.
Sortie: $\boldsymbol{x} \in \mathbb{F}_2^k$ le texte clair.

1: Calculer $\boldsymbol{y}\boldsymbol{P}^{-1} = (\boldsymbol{x}\boldsymbol{Z})\boldsymbol{G} \times \boldsymbol{P}\boldsymbol{P}^{-1} \oplus \boldsymbol{e}\boldsymbol{P}^{-1}$.
// $\boldsymbol{e}\boldsymbol{P}^{-1}$ est de poids t et $(\boldsymbol{x}\boldsymbol{Z})\boldsymbol{G} \in \mathcal{C}$.
2: Décoder $\boldsymbol{y}\boldsymbol{P}^{-1}$ pour retrouver $\boldsymbol{x}\boldsymbol{Z}$.
// en utilisant l'Algorithme 3 par exemple.
3: Calculer $(\boldsymbol{x}\boldsymbol{Z})\boldsymbol{Z}^{-1} = \boldsymbol{x}$.
4: **Retourner** $\boldsymbol{x}$.

Elle peut aussi décider de décoder directement avec la matrice publique $\boldsymbol{G}$. Cependant, décoder t erreurs d'un code aléatoire est un problème $\mathcal{NP}$-complet : **c'est le problème du décodage d'un code aléatoire**.
On peut potentiellement utiliser ce système avec de nombreuses familles de codes. Néanmoins, si l'on utilise une famille trop structurée comme les codes GRS ou de Reed-Muller, il est possible de faire des attaques structurelles qui permettent de retrouver la structure du code masqué.

2.4.2 Variante de Niederreiter

Cette variante du cryptosystème de McEliece a été proposée par H. Niederreiter en 1986 [Nie86]. Elle est exactement équivalente au schéma de McEliece du point de vue de la sécurité et est un peu plus efficace en temps de calcul. Elle fonctionne comme le chiffrement de McEliece, mais en utilisant la matrice de contrôle $\boldsymbol{H}$ du code $\mathcal{C}$ à la place de la matrice génératrice $\boldsymbol{G}$ et l'erreur $\boldsymbol{e}$ pour contenir le message. Les algorithmes de génération des clefs, de chiffrement et de déchiffrement sont décrits ci-dessous. On pose $r = n - k$.

Génération des clefs La génération des clefs consiste à choisir une matrice de contrôle $\boldsymbol{H}$ d'un code linéaire $\mathcal{C}$ que l'on sait décoder (c'est la clef privée sk). Cette matrice de contrôle est ensuite transformée en $\boldsymbol{H}'$ qui semble aléatoire (c'est la clef publique pk). L'algorithme de génération des clefs dans le schéma de Niederreiter (Algorithme 7) prend en entrée les paramètres du code $\mathcal{C}$ n, k et t et retourne la paire de clefs publique $(\boldsymbol{H}', t)$ et privée $(\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{P})$.

Chiffrement Le chiffrement dans le schéma de Niederreiter est une opération de calcul de syndrome du message $\boldsymbol{x}$ converti en un vecteur d'erreur $\boldsymbol{e}$. L'algorithme de chiffrement (Algorithme 8) prend en entrée un message $\boldsymbol{x} \in \mathbb{F}_2^\ell$ (avec $\ell = \lfloor \log_2 \binom{n}{t} \rfloor$) et la clef publique $(\boldsymbol{H}', t)$ et retourne le texte chiffré $\boldsymbol{s} \in \mathbb{F}_2^r$.

Algorithme 7 Génération des clefs Niederreiter

Entrée: n, k et t des entiers, avec $n \leq 2^m$.
Sortie: $\mathsf{pk} = (\boldsymbol{H}', t)$ et $\mathsf{sk} = (\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{P})$.

1: Choisir une matrice génératrice $\boldsymbol{H} \in \mathbb{F}_2^{r \times n}$ d'un code linéaire $\mathcal{C}$ que l'on sait décoder.
2: Choisir une matrice de permutation $\boldsymbol{P} \in \mathbb{F}_2^{n \times n}$.
3: Choisir une matrice inversible $\boldsymbol{Q} \in \mathbb{F}_2^{r \times r}$.
4: Calculer $\boldsymbol{H}' = \boldsymbol{QHP}$.
5: $\mathsf{pk} \leftarrow (\boldsymbol{H}', t)$
6: $\mathsf{sk} \leftarrow (\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{P})$
7: **Retourner** (pk, sk).

Algorithme 8 Chiffrement Niederreiter

Entrée: $\mathsf{pk} = (\boldsymbol{H}', t)$ la clef publique et $\boldsymbol{x} \in \mathbb{F}_2^\ell$ un message à chiffrer.
Sortie: $\boldsymbol{s} \in \mathbb{F}_2^r$ le texte chiffré associé à $\boldsymbol{x}$.

1: Convertir le message $\boldsymbol{x} \in \mathbb{F}_2^\ell$ en $\boldsymbol{e} \in \mathbb{F}_2^n$, une erreur de poids t.
// en utilisant un algorithme d'encodage en mots de poids constants.
2: Calculer $\boldsymbol{s} = \boldsymbol{H}'\boldsymbol{e}^T$.
3: **Retourner** $\boldsymbol{s}$.

Déchiffrement Le déchiffrement dans le schéma de Niederreiter est une opération de décodage qui consiste à retrouver le message original $\boldsymbol{x}$ à partir du texte chiffré $\boldsymbol{s}$. L'algorithme de déchiffrement (Algorithme 9) prend en entrée la clef privée $(\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{P})$ ainsi que le texte chiffré $\boldsymbol{s} \in \mathbb{F}_2^r$ et retourne le message $\boldsymbol{x} \in \mathbb{F}_2^\ell$.

Algorithme 9 Déchiffrement Niederreiter

Entrée: $\mathsf{sk} = (\boldsymbol{Q}, \boldsymbol{H}, \boldsymbol{P})$ la clef privée et $\boldsymbol{s} \in \mathbb{F}_2^r$ le texte chiffré.
Sortie: $\boldsymbol{x} \in \mathbb{F}_2^\ell$ le texte clair.

1: Calculer $\boldsymbol{Q}^{-1}\boldsymbol{s} = \boldsymbol{Q}^{-1}\boldsymbol{QH}(\boldsymbol{Pe})$.
2: Décoder $\boldsymbol{Q}^{-1}\boldsymbol{s}$ pour retrouver $\boldsymbol{Pe}$.
// en utilisant un algorithme de décodage jusqu'à t erreurs
3: Calculer $\boldsymbol{P}^{-1}(\boldsymbol{Pe}) = \boldsymbol{e}$.
4: reconvertir $\boldsymbol{e}$ en $\boldsymbol{x}$.
5: **Retourner** $\boldsymbol{x}$.

La table 2.3 compare la taille des clefs publiques et taux de transmission des cryptosystèmes de McEliece et Niederreiter pour un même code de Goppa de longueur n et de dimension k. Cette table montre aussi que le schéma de Niederreiter est un plus efficace que celui de McEliece en temps de calcul.

TABLE 2.3 – Comparaison des schémas (bits).

	McEliece	McEliece	Niederreiter	Niederreiter
(n, k)	(2 048, 1 696)	(4 096, 3 604)	(2 048, 1 696)	(4 096, 3 604)
Sécurité	$2^{86.8}$	$2^{128.5}$	$2^{86.8}$	$2^{128.5}$
Clef publique	3 392 000	14 416 000	704 000	1 968 000
Message	1 696	3 604	233	327
texte Chiffré	2 048	4 096	352	492
Taux de trans.	0.83	0.88	0.66	0.66

2.5 Identification et Signature

Un schéma d'identification permet à une entité (Alice) détenant une clef privée de prouver son identité à toute autre entité (Bob) détenant la clef publique correspondante. L'authentification est la procédure qui consiste, pour un système informatique, de vérifier l'identité d'une personne ou d'un ordinateur afin d'autoriser l'accès de cette entité à des ressources (systèmes, réseaux, applications, etc). L'identification permet donc de connaître l'identité d'une entité alors que l'authentification permet de vérifier cette identité.

Une preuve à **divulgation nulle de connaissance ou zero-knowledge** est un concept utilisé en cryptologie dans le cadre de l'identification. Elle désigne un protocole interactif sécurisé dans lequel une entité, nommée **prouveur**, prouve mathématiquement à une autre entité, le **vérifieur**, qu'une proposition est vraie sans toutefois révéler d'autre information que la véracité de la proposition. Un schéma d'identification zero-knowledge doit vérifier les trois propriétés suivantes.

1. **Consistance/complétude (ou completeness) :** si le prouveur et le vérifieur suivent le protocole, alors le vérifieur doit toujours accepter la preuve.
2. **Solidité (ou soundness) :** si la proposition est fausse, aucun prouveur malicieux ne peut convaincre un vérifieur honnête que la proposition est vraie et ceci avec une forte probabilité.
3. **Aucun apport d'information (ou zero-knowledge) :** le vérifieur n'apprend de la part du prouveur rien de plus que la véracité de la proposition, il n'obtient aucune information qu'il ne connaissait déjà sans l'apport du prouveur.

En pratique, les schémas d'identification à divulgation nulle de connaissance se présentent souvent sous forme d'un protocole de type défis (**challenge/réponse**). Le vérifieur et le prouveur s'échangent des informations et le vérifieur doit contrôler si la réponse finale du prouveur est vraie ou fausse. Le premier schéma d'identification à divulgation nulle de connaissance (à plusieurs tours ou rondes ϑ) basé sur le problème du décodage par syndrome est le schéma de Stern [Ste88].

Schéma d'identification de Stern Le schéma de Stern utilise une matrice de contrôle $\boldsymbol{H} \in \mathbb{F}_2^{r\times n}$ commune à tous les utilisateurs. La clef privée d'un utilisateur est un vecteur $\boldsymbol{e} \in \mathbb{F}_2^n$ de poids t. L'algorithme de génération des clefs est décrit dans l'Algorithme 10. Dans ce protocole à trois passes, le détenteur de la clef privée ou secrète peut prou-

Algorithme 10 Génération des clefs Stern

Entrée: $n, k, t \in \mathbb{N}^*$.
Sortie: $\mathsf{pk} = (\boldsymbol{s}, \boldsymbol{H}, t)$ et $\mathsf{sk} = (\boldsymbol{e})$.

1: Générer $\boldsymbol{H} \xleftarrow{\$} \mathbb{F}_2^{r\times n}$, une matrice aléatoire.
2: Choisir $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_2^n$, une erreur aléatoire de poids t.
3: Calcule $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{e}^T$.
4: **Retourner** $(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{s}, \boldsymbol{H}, t), \boldsymbol{e})$.

ver sa connaissance de $\boldsymbol{e}$ en utilisant deux facteurs de mélange : une permutation σ et un vecteur aléatoire $\boldsymbol{y}$. Cependant, un prouveur malhonnête ne connaissant pas $\boldsymbol{e}$ peut tromper le vérifieur dans le protocole avec une probabilité de $2/3$: c'est **la probabilité de triche** (ou **cheating**). Ainsi, le protocole doit être exécuté plusieurs fois dont chacun est défini dans la Figure 2.3 pour détecter les prouveurs qui trichent. Afin de minimiser la quantité de données transmises, le prouvreur calcule l'empreinte (avec la fonction de hachage $\mathtt{hash}$) de ses propositions : c'est la notion d'**engagement (ou commitment)**. On note $\mathcal{S}_n$ l'ensemble des permutations sur n éléments et $|$ le symbole de concaténation.

Une variation du schéma de Stern a été proposée en 1996 par P. Véron [Vér97].

Schéma d'identification de Véron Le schéma de Véron utilise une matrice génératrice $\boldsymbol{G} \in \mathbb{F}_2^{k\times n}$ commune à tous les utilisateurs. La clef privée d'un utilisateur est composée du message $\boldsymbol{x} \in \mathbb{F}_2^k$ et un vecteur $\boldsymbol{e} \in \mathbb{F}_2^n$ de poids t. L'algorithme de génération des clefs est décrit dans l'Algorithme 11. Dans ce protocole à trois passes, le détenteur

Algorithme 11 Génération des clefs Véron.

Entrée: $n, k, t \in \mathbb{N}^*$.
Sortie: $\mathsf{pk} = (\boldsymbol{y}, \boldsymbol{G}, t)$ et $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{e})$.

1: Générer $\boldsymbol{G} \xleftarrow{\$} \mathbb{F}_2^{k\times n}$, une matrice aléatoire.
2: Choisir $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_2^n$, une erreur aléatoire de poids t.
3: Calculer $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{G} \oplus \boldsymbol{e}$.
4: **Retourner** $(\mathsf{sk}, \mathsf{pk}) = ((\boldsymbol{x}, \boldsymbol{e}), (\boldsymbol{y}, \boldsymbol{G}, t))$.

de la clef privée ou secrète peut prouver qu'il connaît $\boldsymbol{e}$ en utilisant deux facteurs de mélange : une permutation σ et un vecteur aléatoire $\boldsymbol{u}$. Cependant, un prouveur malhonnête peut tromper le vérifieur dans le protocole avec une probabilité de cheating de $2/3$. Comme dans Stern, le protocole de Véron (Figure 2.4) doit être répété plusieurs fois pour atteindre le niveau de sécurité souhaité.

Prouveur		**Vérifieur**
$(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{s}, \boldsymbol{H}, t), \boldsymbol{e})$		$\mathsf{pk} = (\boldsymbol{s}, \boldsymbol{H}, t)$
$\boldsymbol{y} \xleftarrow{\$} \mathbb{F}_2^n, \sigma \xleftarrow{\$} \mathcal{S}_n$		
$c_1 \leftarrow \mathtt{hash}(\sigma \vert \boldsymbol{H}\boldsymbol{y}^T)$		
$c_2 \leftarrow \mathtt{hash}(\sigma(\boldsymbol{y}))$		
$c_3 \leftarrow \mathtt{hash}(\sigma(\boldsymbol{y} \oplus \boldsymbol{e}))$	$\xrightarrow{c_1, c_2, c_3}$	
	$\xleftarrow{b}$	$b \xleftarrow{\$} \{0, 1, 2\}$
Si $b = 0$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow \boldsymbol{y}$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1 \vert \boldsymbol{H}\mathbf{rp}_2^T)$ et $c_2 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2))$
Si $b = 1$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow \sigma(\boldsymbol{y} \oplus \boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1 \vert \boldsymbol{H}\mathbf{rp}_2^T \oplus \boldsymbol{s})$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2))$
Si $b = 2$		
$\mathbf{rp}_1 \leftarrow \sigma(\boldsymbol{y})$ $\mathbf{rp}_2 \leftarrow \sigma(\boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_2 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1 \oplus \mathbf{rp}_2)$ ainsi que $w_H(\sigma(\boldsymbol{e})) \stackrel{?}{=} t$

FIGURE 2.3 – Protocole de Stern.

La table 2.4 donne les performances des deux schémas d'identification à 3 passes en termes de complexité du prouveur qui donne le nombre d'opérations binaires effectuées par celui-ci et de complexité de communication (ou coût) qui mesure le nombre de bits moyen échangés par les deux entités. La probabilité de cheating est bornée à 10^{-6} pour un niveau de sécurité de 2^{70} opérations binaires. On considère des empreintes de 160 bits et des graines de 128 bits pour les nombres aléatoires.

Remarque 2.14 *Une graine (ou seed) est une chaîne binaire (de quelques bits) utilisée pour l'initialisation d'un générateur de nombres pseudo-aléatoires et le choix de cette graine, devant parfois satisfaire un certain nombre de contraintes, est fondamental en cryptographie.*

Signature à partir d'identification On peut obtenir un schéma de signature à partir de schéma d'identification en utilisant la transformation (ou heuristique) de Fiat-Shamir [FS86]. L'heuristique de Fiat-Shamir est une technique permettant de transformer de manière générique une preuve à divulgation nulle de connaissance en preuve non interactive à divulgation nulle de connaissance et permet de construire un schéma de signature numérique. L'idée est de voir le signataire comme un prouveur alors que le vérifieur sera

Prouveur		**Vérifieur**
$(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{y}, \boldsymbol{G}, t), (\boldsymbol{x}, \boldsymbol{e}))$		$\mathsf{pk} = (\boldsymbol{y}, \boldsymbol{G}, t)$
$\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_2^k, \sigma \xleftarrow{\$} \mathcal{S}_n$		
$c_1 \leftarrow \mathtt{hash}(\sigma)$		
$c_2 \leftarrow \mathtt{hash}(\sigma((\boldsymbol{u} \oplus \boldsymbol{x})\boldsymbol{G}))$		
$c_3 \leftarrow \mathtt{hash}(\sigma(\boldsymbol{u}\boldsymbol{G} \oplus \boldsymbol{y}))$	$\xrightarrow{c_1, c_2, c_3}$	
	$\xleftarrow{b}$	$b \xleftarrow{\$} \{0, 1, 2\}$
Si $b = 0$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow (\boldsymbol{u} \oplus \boldsymbol{x})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_2 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2\boldsymbol{G}))$
Si $b = 1$		
$\mathbf{rp}_1 \leftarrow \sigma((\boldsymbol{u} \oplus \boldsymbol{x})\boldsymbol{G})$ $\mathbf{rp}_2 \leftarrow \sigma(\boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_2 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1 \oplus \mathbf{rp}_2)$ ainsi que $w_H(\sigma(\boldsymbol{e})) \stackrel{?}{=} t$
Si $b = 2$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow \boldsymbol{u}$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2\boldsymbol{G} \oplus \boldsymbol{y}))$

FIGURE 2.4 – Protocole de Véron.

remplacé par un oracle aléatoire. Considérons un schéma à 3 passes pour lequel le prouveur calcule des engagements regroupés sous la valeur cmt, des challenges ch générés par un vérifieur et les réponses aux différents challenges contenues dans **rp**.

Le signataire du message $\boldsymbol{x}$ calcule $ch = \mathtt{hash}(cmt, \boldsymbol{x})$. Le secret $\boldsymbol{e}$ dans le schéma d'identification de Stern constituera la clef de signature et permettra au signataire de générer les **rp**. Ainsi la retranscription $(cmt, ch, \mathbf{rp})$, simulant des interactions entre un prouveur et un vérifieur, constituera la signature du message $\boldsymbol{x}$. Une des instanciations les plus célèbres de la transformation de Fiat-Shamir concerne le schéma d'identification de Stern en signature. Les avantages du paradigme de Fiat-Shamir repose sur la sécurité prouvée d'un tel protocole dans le modèle de l'oracle aléatoire. La taille des clefs de signatures est souvent très grande avec cette méthode à cause de nombreuses simulations du protocole interactif.

2.6 Compétition du NIST

La plupart des protocoles de sécurité reposent principalement sur trois fonctions de base de la cryptographie : le chiffrement, la signature numérique et l'échange de clefs. Le protocole de communication sécurisé sur internet, HTTPS (Hypertext Transfer Protocol

TABLE 2.4 – Comparaison des schémas d'identification de Stern et de Véron.

	Stern	Véron
Nombre de rondes ϑ	35	35
Données publiques (bits)	65 792	66 048
Complexité prouveur (op.binaires)	$2^{22.14}$	$2^{22.13}$
Coût communication (bits)	43 750	37 777

Secure) utilise du TLS pour assurer la confidentialité et l'intégrité des données échangées entre client et serveur. Pour l'échange de clefs, TLS utilise le principe de Diffie-Hellman (basé sur le problème difficile du logarithme discret). La confidentialité de ce protocole est persistante, car les clefs de session sont jetées à la fin et pour certifier cet échange, on le signe essentiellement avec du RSA (basé sur le problème difficile de factorisation de grands entiers). Cependant, ces cryptosystèmes basés sur des problèmes difficiles de la théorie des nombres sont attaquables, du moins théoriquement dans un modèle d'ordinateur quantique. En effet, grâce aux travaux de Shor [Sho94], de tels problèmes seront vulnérables à des attaques en temps polynomial (humainement raisonnable) lorsque des ordinateurs quantiques dotés d'une puissance de calcul suffisante seront disponibles. Ce qui rendra les solutions cryptographiques actuelles obsolètes. Bien que les ressources nécessaires pour exécuter efficacement l'algorithme de Shor sur des paramètres cryptographiques réels (ou d'autres algorithmes quantiques pertinents comme celui de Grover [Gro96] sur les systèmes symétriques) ne sont pas encore disponibles. Cependant, les communications chiffrées d'aujourd'hui pourraient être facilement stockées par des attaquants pour les décryptées plus tard avec un ordinateur quantique, compromettant ainsi les systèmes qui visent une sécurité à long terme. Il est donc essentiel que le temps nécessaire pour développer de telles ressources ne soit pas inférieur à la somme du temps nécessaire pour développer et déployer de nouvelles normes cryptographiques, et de la durée de vie souhaitée d'un secret. C'est dans cette optique que le NIST a lancé, depuis décembre 2016, un appel à propositions de schémas de chiffrement, d'échange de clefs et de signature numérique, afin de sélectionner une gamme de primitives post-quantiques qui deviendront la nouvelle norme de la cryptographie à clef publique.
L'objectif principal de ce processus est de remplacer les trois normes les plus vulnérables aux attaques quantiques, à savoir les normes FIPS 186-4 [1], pour la signature numérique,

1. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

NIST SP 800-56A [2] et NIST SP 800-56B [3], pour l'échange de clefs ou KEM. Le processus devrait durer environ 5 ou 6 ans, et 5 autres années supplémentaires pour la phase de déploiement des nouvelles normes. Il existe actuellement cinq grandes familles d'approches post-quantiques :

1. la recherche de mots de petits poids sur les réseaux euclidiens (ou lattices).
2. le problème de décodage des codes aléatoires (ou codes).
3. la résolution des systèmes de polynômes à plusieurs variables (ou multivariés).
4. les graphes d'isogénies.
5. les fonctions de hachage pour la signature numérique (ou hash).

Les deux premières approches sont les plus étudiées et représentent près des $3/4$ du nombre total des 69 soumissions qui ont satisfait aux critères minimaux et aux exigences en termes de niveau de sécurité du NIST (table 2.5) pour le premier tour. En janvier 2019, 26 candidats sur les 69 ont été annoncés pour passer au second tour du processus. Parmi ces 26 candidats sélectionnés, 17 sont des PKE/KEM et les 9 sont des schémas de signature numérique.

Table 2.5 – Niveau de sécurité requis par le NIST.

Niveau sécu.	Description
I	aussi difficile que casser AES-128 (par recherche exhaustive)
II	aussi difficile que casser SHA-256 (par recherche de collision)
III	aussi difficile que casser AES-192
IV	aussi difficile que casser SHA-384
V	aussi difficile que casser AES-256

En juillet 2020, pour le troisième tour, Le NIST a annoncé les 7 finalistes (dont 4 PKE/-KEM et 3 schémas de signature numérique) et 8 candidats alternatifs et en juillet 2022, les 4 premiers algorithmes gagnants du processus sont publiés. Ces futures normes seront des options par défaut pour l'utilisation des algorithmes post-quantiques. La table 2.6 présente les futurs standards post-quantiques pour la signature numérique (CRYSTALS-Dilithium, FALCON, et SPHINCS+) et l'échange de clefs (CRYSTALS-Kyber).

Pour diversifier les futurs standards pour l'échange de clefs post-quantique, le NIST a prolongé le processus pour les trois KEM basés sur les codes *Classic McEliece*, HQC et

2. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf
3. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf

Table 2.6 – Nouveaux standards de la cryptographie post-quantique.

Schéma	Fonction	Famille
CRYSTALS-Dilithium	Signature numérique	Lattice
FALCON	Signature numérique	Lattice
SPHINCS+	Signature numérique	Hash
CRYSTALS-Kyber	PKE/KEM	Lattice

BIKE pour un quatrième tour. De même pour les normes de signature post-quantique, le NIST a aussi lancé une campagne supplémentaire. Les propositions doivent être des schémas de signature autre que ceux basés sur les lattices avec des tailles de signatures courtes et une vérification rapide avant juin 2023. Les schémas basés sur les codes sont des candidats prometteurs pour ce nouveau processus.

Deuxième partie

Étude de la soumission au NIST : DAGS

Chapitre 3

Implantation efficace de DAGS

Ce chapitre correspond à notre publication pour Code-Based Cryptography (CBC), 2022 [Sec+23d] :

Software implementation of a code-based KEM from binary QD-GS Codes
avec Cheikh Thiécoumba Gueye, Gilbert Ndollane Dione, Jean Belo Klamti, Pierre-Louis Cayrel, Idy Diop et Ousmane N'diaye. CBC 2022, p.77-89, LNCS.

Dans cette partie, nous proposons une implantation logicielle des paramètres binaires sécurisés de DAGS que nous appellerons DAGS binaire. DAGS (ᛞ) n'est pas qu'un simple acronyme, mais plutôt la 23ème lettre du vieux Futhark (la plus ancienne forme d'alphabet runique, utilisée par les peuples germaniques) dont la forme rappelle la propriété dyadique d'une matrice qui est au cœur de la construction du schéma. En effet, DAGS est un mécanisme d'encapsulation de clefs (Key Encapsulation Mechanism ou KEM) basé sur les codes de Srivastava généralisés (Generalized Srivastava ou GS) quasi-dyadiques (Quasi-Dyadic ou QD) et était candidat pour le processus de normalisation de la cryptographie post-quantique (Post-Quantum Cryptography ou PQC) du NIST (National Institute of Standards and Technology) [Nis]. Cette construction avec des matrices QD permet d'atteindre de très petites tailles de clefs publique et privée ainsi que de texte chiffré par rapport à toutes les autres soumissions basées sur les codes telles que *Classic McEliece*, BIKE, BIG QUAKE. Malheureusement, DAGS original (soumis au NIST) [Gus+17] n'a pas été sélectionné pour le deuxième tour du processus de normalisation en raison de certains problèmes de sécurité soulevés par Barelli-Couvreur ainsi que Bardet *et al.* L'analyse détaillée de ces techniques d'attaque ont permis aux auteurs de la soumission de proposer de nouveaux paramètres sécurisés et plus performant que la version originale. Dans ce chapitre, nous allons, d'une part, rappeler la construction de DAGS original et sa variante que nous appellerons «SimpleDAGS» [Ban+19]. D'autre part, nous proposons une implantation efficace de DAGS binaire [Sec+23d] en utilisant les techniques présentées dans [Ban+20] qui visent spécifiquement à améliorer la multiplication des matrices QD. Celles-ci impliquent une

version dédiée de la multiplication de Karatsuba et l'application de la décomposition $\boldsymbol{LUP}$ des matrices QD pour accélérer les opérations dans les différents algorithmes. Notre implantation logicielle de DAGS binaire donne de meilleurs temps d'exécution par rapport aux implantations précédentes de DAGS (DAGS original et «SimpleDAGS») et place le KEM DAGS comme une alternative crédible pour les KEM finalistes du NIST (*Classic McEliece*, HQC et BIKE) pour le quatrième tour.

3.1 Construction

La construction du KEM DAGS (Figure 3.1) est basée sur l'utilisation des codes de Srivastava généralisés (Définition 2.20, page 24) définis par des matrices QD (Définition 2.18, page 23) : on parle de codes GS-QD. Comme tout KEM, DAGS est composé de trois algorithmes (génération des clefs, encapsulation et décapsulation) qui suivent le paradigme du «McEliece Randomisé» [Noj+08], basé sur le schéma de McEliece original. Le fait que cette version de McEliece soit prouvée sûre par IND-CPA a permis la conversion de DAGS en KEM et d'atteindre la sécurité IND-CCA. Les paramètres de DAGS sont la longueur du code n, la dimension k (la codimension $r = n - k$), les valeurs s et t qui définissent un code GS, la cardinalité du corps de base q, le degré de l'extension du corps m en plus des sous-paramètres k' et k'' tels que $k = k' + k''$ (où k' est arbitraire et fixé comme «petit»).

Alice		**Bob**
		$(\boldsymbol{G}, (\boldsymbol{v}, \boldsymbol{y})) \leftarrow \mathsf{KeyGen}(n, m, q, s, t)$
	$\xleftarrow{\boldsymbol{G}}$	
$\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_q^{k'}$		
$((\boldsymbol{c}, \pi), K) \leftarrow \mathsf{Encaps}(\boldsymbol{x}, \boldsymbol{G})$		
	$\xrightarrow{(\boldsymbol{c}, \pi)}$	
		K ou $\perp \leftarrow \mathsf{Decaps}((\boldsymbol{c}, \pi), (\boldsymbol{v}, \boldsymbol{y}))$

FIGURE 3.1 – Principe général du KEM DAGS.

Pour une meilleure compréhension de notre implantation logicielle dans la Section 3.4 (page 59), nous allons d'abord définir les algorithmes de génération des clefs, d'encapsulation et de décapsulation de DAGS original.

Génération des clefs L'algorithme de génération des clefs (Algorithme 12) prend en entrée les paramètres du code GS n, m, q, s, t et retourne la clef publique (pk), la matrice génératrice $\boldsymbol{G}$, et la clef privée (sk), le couple $(\boldsymbol{v}, \boldsymbol{y})$. Il est essentiellement composé de deux grandes étapes.

— **Étape 1 : Construction de la matrice $\boldsymbol{H}_{base}$**
Nous rappelons que tout code GS avec $t = 1$ est un code de Goppa. De plus, les codes de Goppa admettent une matrice de contrôle sous forme de Cauchy à condition que le polynôme générateur $g(x)$ soit unitaire et sans zéros multiples [MS77]. Misoczki et Barreto [MB09] ont aussi montré que l'intersection de l'ensemble des matrices de Cauchy avec l'ensemble des matrices dyadiques n'est pas vide à condition que le code soit défini sur un corps de caractéristique 2, et que la signature dyadique $\boldsymbol{d} = (\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1})$ satisfait l'équation «fondamentale» suivante :

$$\frac{1}{\boldsymbol{d}_{i\oplus j}} = \frac{1}{\boldsymbol{d}_i} + \frac{1}{\boldsymbol{d}_j} + \frac{1}{\boldsymbol{d}_0}. \tag{3.1}$$

Cela signifie que toute matrice de Goppa sous forme de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$ peut se mettre sous forme dyadique.
Le processus de génération des clefs dans DAGS exploite cette équation pour construire une matrice de Cauchy $\hat{\boldsymbol{H}}$ (Définition 2.19, page 23). Avant, il faut :

1. Générer une signature dyadique $\boldsymbol{d}$ dans $\mathbb{F}_{q^m}$
   On choisit d'abord aléatoirement des éléments non nuls distincts $\boldsymbol{d}_0$ et $\boldsymbol{d}_j$ dans $\mathbb{F}_{q^m}$ avec j une puissance de 2. Ensuite, on construit les éléments restants en utilisant l'Équation 3.1 pour des choix appropriés de i et j avant de sélectionner des blocs de dimension entre s et n (avec $n = n_0 s$).
2. Obtenir le support de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$
   En partant de cette signature $\boldsymbol{d}$, on construit le support de Cauchy en choisissant au hasard un décalage $\mathtt{dec}$ tel que $\boldsymbol{u}_i = \frac{1}{\boldsymbol{d}_i} + \mathtt{dec}$ et $\boldsymbol{v}_i = \frac{1}{\boldsymbol{d}_j} + \frac{1}{\boldsymbol{d}_0} + \mathtt{dec}$ avec $i = 0, \ldots, s-1$ et $j = 0, \ldots, n-1$.
   On forme ainsi un premier bloc de taille $s \times n$. Ce bloc correspond au premier bloc $\hat{\boldsymbol{H}}_1$ de $\hat{\boldsymbol{H}}$ (Proposition 2.6, page 24) avec $\boldsymbol{u}_i = \boldsymbol{w}_i$ et $\boldsymbol{v}_j = \alpha_j$.
3. Construire la matrice $\hat{\boldsymbol{H}}$
   La matrice $\hat{\boldsymbol{H}}$ est ensuite alimentée successivement par les blocs suivants $(\hat{\boldsymbol{H}}_2, \ldots, \hat{\boldsymbol{H}}_t)$ en élevant élément par élément de $\hat{\boldsymbol{H}}_1$ à la puissance $i = 1, \ldots, t$. Enfin, $\hat{\boldsymbol{H}}$ est obtenue en superposant tous les t blocs $\hat{\boldsymbol{H}}_i$ et on a

$$\hat{\boldsymbol{H}} = \begin{pmatrix} \hat{\boldsymbol{H}}_1 \\ \vdots \\ \hat{\boldsymbol{H}}_t \end{pmatrix},$$

avec

$$\hat{\boldsymbol{H}}_i = \begin{pmatrix} \frac{z_1}{(\boldsymbol{u}_1 - \boldsymbol{v}_1)^i} & \cdots & \frac{z_n}{(\boldsymbol{u}_n - \boldsymbol{v}_1)^i} \\ \vdots & \vdots & \vdots \\ \frac{z_1}{(\boldsymbol{u}_1 - \boldsymbol{v}_s)^i} & \cdots & \frac{z_n}{(\boldsymbol{u}_n - \boldsymbol{v}_s)^i} \end{pmatrix}.$$

Cette matrice $\hat{\boldsymbol{H}}$ va servir ensuite pour la construction de la matrice $\boldsymbol{H}_{base}$.

4. Construire la matrice $\boldsymbol{H}_{base}$
   On choisit d'abord aléatoirement un vecteur $\boldsymbol{z}$ dans $\mathbb{F}_{q^m}$ tel que $\boldsymbol{z}_{is+j} = \boldsymbol{z}_{is}$ pour $i = 0, \ldots, n_0 - 1, j = 0, \ldots, s-1$. Ensuite, on cherche un vecteur $\boldsymbol{y} = (\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{n-1}) \in \mathbb{F}_{q^m}$ tel que $\boldsymbol{y}_j = \frac{\boldsymbol{z}_j}{\prod_{i=0}^{s-1}(\boldsymbol{u}_i - \boldsymbol{v}_j)^t}$ pour $j = 0, \ldots, n-1$.
   Ce vecteur $\boldsymbol{y}$ en plus de $\boldsymbol{v}$ vont servir pour la reconstruction de la matrice alternante pour le décodage. Enfin, on calcule $\boldsymbol{H} = \hat{\boldsymbol{H}} \cdot Diag(\boldsymbol{z}_i)$ avant de la projeter sur $\mathbb{F}_q$ pour obtenir une matrice $\boldsymbol{H}_{base}$ de taille $mst \times n$ qui peut être mise sous forme systématique.

— **Étape 2 : Systématisation de la matrice $\boldsymbol{H}_{base}$**
En appliquant un pivot de Gauss, on obtient $\boldsymbol{H}_{base} = (\boldsymbol{I}_r | \boldsymbol{M})$ avec une grande probabilité. $\boldsymbol{M}$ est une matrice de taille $mst \times k$, composée de s blocs par ligne. Cette opération de systématisation préserve la structure dyadique, car la réduction de lignes se fait bloc par bloc et par conséquent les matrices $\boldsymbol{M}$ et $\boldsymbol{G} = (\boldsymbol{I}_k | \boldsymbol{M}^T)$ sont aussi QD (Proposition 2.4, page 23). Finalement, la matrice $\boldsymbol{G}$ constitue la clef publique (instanciation à la McEliece) et le couple $(\boldsymbol{v}, \boldsymbol{y})$ est la clef privée.

Algorithme 12 Génération des clefs

Entrée: n, m, q, s, t.
Sortie: $\mathsf{pk} = \boldsymbol{G}$ et $\mathsf{sk} = (\boldsymbol{v}, \boldsymbol{y})$.

1: Générer une signature dyadique $\boldsymbol{d}$.
2: Construire le support de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$.
3: Former la matrice de Cauchy $\hat{\boldsymbol{H}}_1 = \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$.
4: Construire $\hat{\boldsymbol{H}}$.
5: Générer $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_{q^m}^n$.
6: Calculer $\boldsymbol{y}_j = \frac{\boldsymbol{z}_j}{\prod_{i=0}^{s-1}(\boldsymbol{u}_i - \boldsymbol{v}_j)^t}$.
7: Former $\boldsymbol{H} = \hat{\boldsymbol{H}} \cdot Diag(\boldsymbol{z}_i)$.
8: Projeter $\boldsymbol{H}$ sur $\mathbb{F}_q$ pour obtenir la matrice $\boldsymbol{H}_{base}$.
9: Écrire $\boldsymbol{H}_{base}$ sous forme systématique $(\boldsymbol{I}_r | \boldsymbol{M})$.
10: $\mathsf{pk} \leftarrow \boldsymbol{G} = (\boldsymbol{I}_k | \boldsymbol{M}^T)$
11: $\mathsf{sk} \leftarrow (\boldsymbol{v}, \boldsymbol{y})$
12: **Retourner** $(\mathsf{pk}, \mathsf{sk})$.

Encapsulation L'algorithme d'encapsulation (Algorithme 13) prend en entrée le message $\boldsymbol{x} \in \mathbb{F}_q^{k'}$ et la clef publique $\boldsymbol{G}$ et retourne le texte chiffré $(\boldsymbol{c}, \pi)$ ainsi que la clef de session K. Il est principalement composé de calcul de hachages, d'opération d'encodage et de génération aléatoire de vecteurs.

Remarque 3.1 *Les algorithmes d'encapsulation et de décapsulation utilisent des fonctions de hachage :*

1. $\mathtt{hash}_1 \colon \mathbb{F}_q^{k'} \longrightarrow \mathbb{F}_q^k$, *pour générer des données aléatoires.*
2. $\mathtt{hash}_2 \colon \mathbb{F}_q^{k'} \longrightarrow \mathbb{F}_q^{k'}$, *pour fournir une confirmation en clair comme dans [HHK17].*
3. $\mathtt{hash}_3 \colon \{0,1\}^* \longrightarrow \{0,1\}^\ell$, *pour extraire la clef de session (ou encapsulée)* K, ℓ *étant la longueur de bits souhaitée (généralement 256).*

Algorithme 13 Encapsulation

Entrée: pk $= \boldsymbol{G}$ la clef publique et $\boldsymbol{x}$ un message à chiffrer.
Sortie: $(\boldsymbol{c}, \pi)$ le texte chiffré et la clef de session K.

1: Choisir $\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_q^{k'}$.
2: Calculer $\mathtt{hash}_1(\boldsymbol{x})$ et $\mathtt{hash}_2(\boldsymbol{x}) = \pi$.
3: Décomposer $\mathtt{hash}_1(\boldsymbol{x})$ en $(\rho|\mathtt{graine})$ (de longueurs respectives k'' et k').
4: Calculer $\mu = (\rho|\boldsymbol{x})$.
5: Générer $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_q^n$, une erreur de poids w à partir de $\mathtt{graine}$.
6: Calculer $\boldsymbol{c} = \mu\boldsymbol{G} + \boldsymbol{e}$.
7: Calculer $K = \mathtt{hash}_3(\boldsymbol{x})$.
8: **Retourner** $((\boldsymbol{c}, \pi), K)$.

La chaîne $\mathtt{graine}$ de longueur $k' \ll n$, est utilisée pour l'initialisation d'un générateur pseudo-aléatoire de longueur n avec un certain nombre de contraintes sur le poids (Remarque 2.14, page 38).

Décapsulation L'algorithme de décapsulation (Algorithme 14) prend en entrée la clef privée $(\boldsymbol{v}, \boldsymbol{y})$ ainsi que le texte chiffré $(\boldsymbol{c}, \pi)$ et retourne la clef de session (ou décapsulée) K. Les deux premières étapes de la décapsulation représentent un décodage de code alternant (Définition 2.23, page 26) comme présenté dans [MS77]. Le décodage est constitué de trois étapes distinctes.

1. Calculer le polynôme syndrome $s(\boldsymbol{z})$ du mot reçu $\boldsymbol{c} = \mu\boldsymbol{G} + \boldsymbol{e}$ avec la matrice alternante $\boldsymbol{H}'$ construite à partir de $(\boldsymbol{v}, \boldsymbol{y})$.
2. Utiliser ce syndrome pour calculer le polynôme localisateur d'erreurs $\sigma(\boldsymbol{z})$ et le polynôme évaluateur d'erreurs $\theta(\boldsymbol{z})$, en résolvant l'Équation-clef 2.1 (page 29).
3. Chercher les racines de $\sigma(\boldsymbol{z})$ et $\theta(\boldsymbol{z})$ pour révéler, respectivement, les emplacements et les valeurs (si le code n'est pas binaire) des erreurs $\boldsymbol{e}_i$.

Après le décodage ($\boldsymbol{e}$ retrouvé), les étapes restantes consistent à dériver la bonne clef de session K pour le KEM.

Algorithme 14 Décapsulation

Entrée: $\boldsymbol{H}' \leftarrow (\boldsymbol{v}, \boldsymbol{y})$ la clef privée et $(\boldsymbol{c}, \pi)$ le texte chiffré.
Sortie: K la clef de session.

1: Décoder $\boldsymbol{c}$ avec $\boldsymbol{H}'$ pour obtenir $\mu'\boldsymbol{G}$ et $\boldsymbol{e}'$.
2: Renvoyer $\perp$, si le décodage échoue ou si $w_H(\boldsymbol{e}') \neq w$.
3: Retrouver μ' et le décomposer en $(\rho'|\boldsymbol{x}')$.
4: Calculer $\texttt{hash}_1(\boldsymbol{x}')$ et $\texttt{hash}_2(\boldsymbol{x}') = \pi'$.
5: Décomposer $\texttt{hash}_1(\boldsymbol{x}')$ en $(\rho''|\texttt{graine}')$
6: Générer $\boldsymbol{e}'' \xleftarrow{\$} \mathbb{F}_q^n$, une erreur de poids w à partir de $\texttt{graine}'$.
7: **si** $\boldsymbol{e}'' \neq \boldsymbol{e}' \vee \rho'' \neq \rho' \vee \pi \neq \pi'$ **alors**
8: Renvoyer $\perp$.
9: **sinon**
10: Calculer $K = \texttt{hash}_3(\boldsymbol{x}')$.
11: **fin si**
12: **Retourner** K.

Il est naturel de penser que l'introduction d'une structure algébrique supplémentaire, comme la «quasi-dyadicité», dans un schéma basé sur les codes correcteurs d'erreurs, peut donner plus de pouvoirs à Eve (attaquante) pour mener potentiellement des attaques structurelles.

3.2 Attaques algébriques et choix de paramètres

Attaque FOPT Récupérer une matrice privée à partir d'une matrice publique peut-être en général une tâche très difficile (comme nous l'avons évoqué dans la Section 2.4, page 31). Mais, la présence d'une structure supplémentaire dans les propriétés du code peut en effet réduire cette difficulté. L'attaque de Faugère, Otmani, Perret et Tillich (ou FOPT) [Fau+10a] et ses variantes [Fau+16 ; Fau+10b] exploitent cette structure algébrique pour résoudre un système d'équations multivariées afin de reconstruire une matrice alternante. Cette attaque visait à l'origine une variante de McEliece qui utilisait des codes de Goppa QD [MB09]. La plupart des paramètres proposés dans [MB09] ont été cassés, à l'exception du cas binaire. En réalité, cela n'était pas lié au corps de base q, mais dépendait plutôt du fait qu'avec un q plus petit, les auteurs utilisaient un degré d'extension m beaucoup plus élevé avec une constante $q^m = 2^{16}$. Le degré d'extension m joue un rôle-clef dans l'attaque, car il définit la dimension de l'espace solution, qui est égale à $m - 1$. Dans [Fau+10b] les auteurs donnent une limite de complexité théorique pour laquelle tout schéma avec $m - 1 \leq 20$ peut-être vulnérable à l'attaque FOPT.

Étant donné que les codes GS sont des codes alternants, l'attaque peut également être appliquée au schéma de DAGS. Cependant, dans le cas des codes GS la dimension de l'espace solution est définie par $mt - 1$ [Per12] contrairement aux codes de Goppa QD. Cela offrait plus de flexibilité lors de la conception du DAGS original et a permis aux

auteurs de choisir, par exemple, un m faible. Dans [Fau+16], les auteurs présentent une nouvelle technique appelée «folding» et montrent qu'il est possible de réduire la complexité de l'attaque FOPT pour un code beaucoup plus petit (ou code raccourci), grâce aux propriétés fortes du groupe d'automorphismes des codes alternants. Cette variante de FOPT s'avère très efficace contre les codes de Goppa. Par contre, l'efficacité contre, les codes GS n'est pas claire, car les auteurs ne fournissent que de résultats expérimentaux sur cette famille de code. Pour ces raisons, la sélection des paramètres de DAGS original est faite en ne tenant compte que de la dimension de l'espace solution du système algébrique telle que $mt - 1 > 20$.

Paramètres originaux D'une part, la condition pour être hors de portée de l'espace solution de l'attaque FOPT est $mt \geq 21$. Ceci garantit au moins 128 bits de sécurité selon la limite présentée dans [Fau+10b]. D'autre part, pour que la méthode du décodage par ensemble d'information (Problème 2.4, page 22) soit «calculatoirement» difficile, le nombre d'erreurs w à décoder doit être suffisamment important et correspond à $st/2$ conformément à la distance minimale des codes GS. En outre, pour optimiser les performances du KEM, le degré d'extension m doit être le plus petit possible. Cela nécessite toutefois que q soit suffisamment grand sachant que la longueur n du code GS est plafonnée à $q^m - s$. En pratique, cela signifie que q^m est au moins égal à 2^{12} avec un choix optimal de $q = 2^6$ et $m = 2$. Enfin, s doit être une puissance de 2 et les valeurs impaires de t offrent de meilleures performances. Les Tables 3.1 et 3.2 présentent les paramètres de DAGS (DAGS_1, DAGS_3 et DAGS_5) tels que proposés dans [Gus+17] et correspondent aux trois niveaux de sécurité (I, III et V) pour le processus de normalisation du NIST (Table 2.5).

Table 3.1 – Paramètres DAGS original.

Niveau sécu.	q	m	n	k	k'	s	t	w	Complexité FOPT
I	2^5	2	832	416	43	2^4	13	104	≥ 128
III	2^6	2	1 216	512	43	2^5	11	176	≥ 128
V	2^6	2	2 112	704	43	2^6	11	352	≥ 128

Cependant, en 2018, Barelli et Couvreur ont présenté dans [BC18] deux approches d'attaques structurelles visant précisément les paramètres de DAGS original. La seconde approche sera améliorée un an plus tard par Bardet *et al.* dans [Bar+19].

Attaque de Barelli-Couvreur Cette attaque utilise des sous-codes des codes de Reed-Solomon sur des sous-corps. Leur construction est spécifique au cas $m = 2$ (paramètres de DAGS original). Les auteurs présentent deux stratégies pour identifier le sous-code secret.

TABLE 3.2 – Taille des données dans DAGS original (octets).

Ensemble de paramètres	Clef publique	Clef privée	Chiffré
DAGS_1	6 760	432 640	552
DAGS_3	8 448	1 284 096	944
DAGS_5	11 616	2 230 272	1 616

La première est essentiellement une recherche exhaustive sur tous les codes avec la codimension appropriée. Cette codimension est donnée par $c = \frac{2q}{s}$ ce qui est non trivial dans le cas d'un code GS-QD. Alors qu'une telle force brute serait en principe trop coûteuse, les auteurs présentent quelques raffinements qui la rend réalisable, notamment avec l'utilisation de codes raccourcis comme dans FOPT.
La deuxième, au contraire, consiste à résoudre un système bilinéaire obtenu en utilisant la matrice de contrôle publique et en considérant comme inconnues les éléments d'une matrice génératrice pour le code secret. Ce système est résolu à l'aide des techniques des bases de Gröbner (qui ne font pas partie de notre cadre de recherche), et bénéficie d'une réduction du nombre de variables similaire à celle effectuée dans FOPT.
Pour la première stratégie, les auteurs fournissent une analyse de complexité inférieure au niveau de sécurité souhaité pour tous les paramètres de DAGS proposés au moment de la soumission (Table 3.3). Bien que l'attaque se comporte très bien contre les paramètres de DAGS original, elle n'est pas globalement très critique. En fait, dans [Bar+19], les auteurs montrent qu'il suffit de modifier la taille de q pour se prémunir de cette attaque. C'est le cas pour DAGS_1 et DAGS_3, où il suffit de changer les valeurs de q de 2^5 à 2^6 et de 2^6 à 2^8 respectivement pour pousser la complexité de l'attaque au-delà du niveau de sécurité revendiqué. Concernant DAGS_5, en plus de la taille de q, l'ordre dyadique s ainsi que les autres paramètres du code (m, n, k, w) ont été modifiés. Une mise à jour des paramètres de DAGS original a été proposée dans [Bar+19] (Tables 3.4 et 3.5).

TABLE 3.3 – Complexité de l'attaque de Barelli-Couvreur.

Niveau sécu.	q	m	n	k	k'	s	t	w	Complexité dans [BC18]
I	2^5	2	832	416	43	2^4	13	104	2^{70}
III	2^6	2	1 216	512	43	2^5	11	176	2^{80}
V	2^6	2	2 112	704	43	2^6	11	352	2^{55}

TABLE 3.4 – Paramètres DAGS original mis à jour.

Niveau sécu.	q	m	n	k	s	t	w	Complexité dans [BC18]
I	2^6	2	832	416	2^4	13	104	$\approx 2^{128}$
III	2^8	2	1 216	512	2^5	11	176	$\approx 2^{288}$
V	2^8	2	1 600	896	2^5	11	176	$\approx 2^{289}$

TABLE 3.5 – Taille des données dans DAGS orginal mis à jour (octets).

Ensemble de paramètres	Clef publique	Clef privée	Chiffré
DAGS_1	8 112	2 496	656
DAGS_3	11 264	4 864	1 248
DAGS_5	19 712	6 400	1 632

Ces ajustements montrent que la première stratégie de l'attaque de Barelli-Couvreur devrait être considérée comme une contrainte supplémentaire sur le choix des paramètres plutôt qu'une attaque sur DAGS. La complexité de l'attaque dépend fortement du rapport q/s, ainsi pour se prémunir, les paramètres doivent être choisis de telle sorte que q soit suffisamment grand et que s ne le soit pas trop.
Cependant, ces paramètres mis à jour restent particulièrement vulnérables à la deuxième stratégie de Barelli et Couvreur, améliorée en pratique par Bardet *et al.* dans [Bar+19].

Attaque de Bardet *et al.* Les auteurs montrent que l'approche proposée dans [BC18] n'est pas optimale, notamment sur le ratio entre le nombre d'équations et le nombre de variables du système polynomial. En effet, Bardet *et al.* montrent que le succès de l'attaque dépend fortement de la dimension du code invariant qui conduit à un déséquilibre entre le nombre d'équations et le nombre de variables. Par exemple, pour DAGS_1 mis à jour dans [Bar+19], ce rapport est d'environ $2,5$, ce qui est assez élevé pour rendre l'attaque possible sur cet ensemble de paramètres (Table 3.6).

«N.A.» dans la dernière colonne de la Table 3.6 signifie que l'attaque de Bardet *et al.* n'est pas applicable sur DAGS_3 et DAGS_5. En effet, le système dans DAGS_5 admet trop de variables comparé à DAGS_1, ce qui conduit à un rapport de $1,1$ insuffisant pour rendre l'attaque possible. La situation de DAGS_3 est plus extrême, puisque les paramètres dans cet ensemble ne permettent pas de définir un code invariant. Ainsi, les auteurs ont proposé d'utiliser le code dual à la place pour obtenir le code invariant.

TABLE 3.6 – Complexité de l'attaque de Bardet *et al.*

Niveau sécu.	q	m	n	k	s	t	w	Complexité dans [Bar+19]
I	2^6	2	832	416	2^4	13	104	$\approx 2^{83}$
III	2^8	2	1 216	512	2^5	11	176	N.A.
V	2^8	2	1 600	896	2^5	11	176	N.A.

Mais les paramètres de DAGS_3 produisent un rapport de $0,7$ qui est encore plus faible que dans DAGS_5.

Au final, l'attaque de Barelli-Couvreur [BC18], considérée comme une contrainte supplémentaire sur la sélection de paramètres, et l'attaque de Bardet *et al.* [Bar+19] ont permis aux auteurs de la soumission de DAGS [Gus+17] de proposer une nouvelle approche du sujet dans [Ban+19] : le «SimpleDAGS».

3.3 Version révisée ou «SimpleDAGS»

Le KEM «SimpleDAGS» [Ban+19] est une version plus compacte de DAGS utilisant une instanciation de Niederreiter (Figure 3.2). Nous rapportons la nouvelle description ci-dessous suivant les mêmes conventions que celles utilisées dans la spécification de DAGS.

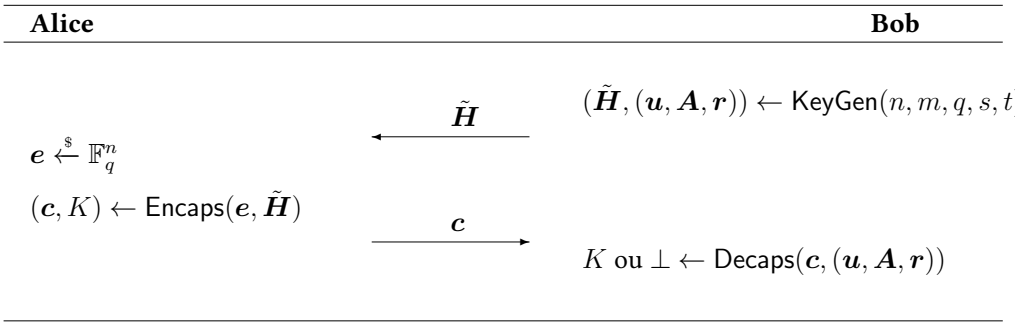

FIGURE 3.2 – Principe général du KEM «SimpleDAGS».

Contrairement au DAGS original, nous n'avons plus besoin des sous-paramètres k' et k''. Nous présentons dans cette section les nouveaux algorithmes de génération des clefs, d'encapsulation et de décapsulation de «SimpleDAGS».

3.3.1 Nouveaux algorithmes

Génération des clefs L'algorithme de génération des clefs (Algorithme 15) est légèrement différent par rapport à DAGS original (Algorithme 12, page 49) notamment sur la systématisation de $\boldsymbol{H}_{base}$ (**Étape 2**). La clef publique est constituée d'une matrice de contrôle $\tilde{\boldsymbol{H}}$ plutôt qu'une matrice génératrice $\boldsymbol{G}$ et la clef privée contient le vecteur $\boldsymbol{u}$, une chaîne aléatoire $\boldsymbol{r}$ et une sous-matrice $\boldsymbol{A}$.

- **Étape 1 : Construction de la matrice $\boldsymbol{H}_{base}$**
  Cette étape dans «SimpleDAGS» est identique à celle décrite dans DAGS original (Section 3.1, page 47).
- **Étape 2 : Systématisation de la matrice $\boldsymbol{H}_{base}$**
  On écrit d'abord $\boldsymbol{H}_{base}$ sous la forme $(\boldsymbol{B}|\boldsymbol{A})$, avec $\boldsymbol{A}$ une sous-matrice QD et inversible composée de taille $r \times r$. Ensuite, on calcule le produit $\boldsymbol{A}^{-1}\boldsymbol{H}_{base} = (\boldsymbol{I}_r|\boldsymbol{M})$ avec une grande probabilité ($\boldsymbol{M}$ est une matrice QD de taille $mst \times k$). Pour augmenter la sécurité du KEM et ne pas avantager l'attaquant (Eve) en cas d'échec du décodage, un vecteur privé $\boldsymbol{r}$ de n bits est généré aléatoirement sur $\mathbb{F}_q$. Enfin, la matrice $\tilde{\boldsymbol{H}} = (\boldsymbol{I}_r|\boldsymbol{M})$ est la clef publique (instanciation à la Niederreiter) et le triplet $(\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$ constitue la clef privée.

Algorithme 15 Génération des clefs

Entrée: n, m, q, s, t.
Sortie: $\mathsf{pk} = \tilde{\boldsymbol{H}}$ et $\mathsf{sk} = (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$.

1: Générer une signature dyadique $\boldsymbol{d}$.
2: Construire le support de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$.
3: Former la matrice de Cauchy $\hat{\boldsymbol{H}}_1 = \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$.
4: Construire $\hat{\boldsymbol{H}}$.
5: Générer $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_{q^m}^n$.
6: Former $\boldsymbol{H} = \hat{\boldsymbol{H}} \cdot Diag(\boldsymbol{z}_i)$.
7: Projeter $\boldsymbol{H}$ sur $\mathbb{F}_q$ pour obtenir la matrice $\boldsymbol{H}_{base}$.
8: Écrire $\boldsymbol{H}_{base} = (\boldsymbol{B}|\boldsymbol{A})$.
9: Obtenir une forme systématique $(\boldsymbol{I}_r|\boldsymbol{M}) = \boldsymbol{A}^{-1}\boldsymbol{H}_{base}$ que l'on note $\tilde{\boldsymbol{H}}$.
   // Si $\boldsymbol{A}$ n'est pas inversible, revenir à l'étape 1.
10: Générer $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_q^n$.
11: $\mathsf{pk} \leftarrow \tilde{\boldsymbol{H}}$
12: $\mathsf{sk} \leftarrow (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$
13: **Retourner** (pk, sk).

Encapsulation L'algorithme d'encapsulation (Algorithme 16) prend en entrée la clef publique $\tilde{\boldsymbol{H}}$ et retourne le texte chiffré $\boldsymbol{c}$ ainsi que la clef de session K.

Remarque 3.2 *Dans cette version révisée de DAGS, les algorithmes d'encapsulation et de décapsulation utilisent une seule fonction de hachage* $\boldsymbol{h} = \mathtt{hash}_3$: $\{0,1\}^* \longrightarrow \{0,1\}^\ell$, *pour fournir une confirmation en clair et d'extraire la clef de session* K.

Algorithme 16 Encapsulation

Entrée: $\mathsf{pk} = \tilde{\boldsymbol{H}}$ la clef publique.
Sortie: $\boldsymbol{c}$ le texte chiffré et la clef encapsulée K.

1: Générer $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_q^n$, une erreur de poids w.
2: Calculer $\boldsymbol{c}_0 = \tilde{\boldsymbol{H}}\boldsymbol{e}$ et $\boldsymbol{c}_1 = \mathrm{h}(2|\boldsymbol{e})$.
3: Définir $\boldsymbol{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$.
4: Calculer $K = \mathrm{h}(1|\boldsymbol{e}|\boldsymbol{c})$.
5: **Retourner** $(\boldsymbol{c}, K)$.

Décapsulation L'algorithme de décapsulation (Algorithme 17) prend en entrée la clef privée $(\boldsymbol{v}, \boldsymbol{y})$ ainsi que le texte chiffré $\boldsymbol{c}$ et retourne la clef de session K.
L'utilisation de Niederreiter dans «SimpleDAGS» implique une stratégie de décodage différente comme indiqué dans l'Algorithme 17. L'entrée de l'algorithme de décapsulation est un syndrome, plutôt qu'un mot de code bruité comme dans l'Algorithme 14. Dans ce cas, il faut d'abord transformer le syndrome reçu $\boldsymbol{c}_0$ en $\boldsymbol{c}'_0$ à partir de $\boldsymbol{H}'$ selon deux étapes.

- **Étape 1 : transformation de $\boldsymbol{H}_{base}$ en $\boldsymbol{H}$**
  1. Calculer $\boldsymbol{A}\boldsymbol{c}_0 = \boldsymbol{A}\tilde{\boldsymbol{H}}\boldsymbol{e} = \boldsymbol{A}\boldsymbol{A}^{-1}\boldsymbol{H}_{base}\boldsymbol{e} = \boldsymbol{H}_{base}\boldsymbol{e}$.
  2. Inverser la projection de $\boldsymbol{H}_{base}$ (de $\mathbb{F}_q$ vers $\mathbb{F}_{q^m}$) pour obtenir $\boldsymbol{H}$.
  3. Calculer $\boldsymbol{H}\boldsymbol{e}$.
- **Étape 2 : calcul de $\boldsymbol{c}'_0$ à partir de $\boldsymbol{H}'$**
  1. Calculer
  $$g(x)_{(\ell-1)t+1} = \frac{\prod_{j=1}^{s}(x_i - \boldsymbol{u}_j)^t}{(x - \boldsymbol{u}_\ell)^i},$$
  où $\ell = 1, \ldots, s$ et $i = 1, \ldots, t$.
  2. Chercher les coefficients $g_1(x), \ldots, g_r(x)$.
  3. Former la matrice inversible $\boldsymbol{C}$ de taille $r \times r$.
  4. Construire $\boldsymbol{H}' = \boldsymbol{C}^{-1}\boldsymbol{H}$.
  5. Calculer $\boldsymbol{c}'_0 = \boldsymbol{H}'\boldsymbol{e}$.

Après avoir obtenu $\boldsymbol{c}'_0$, on le décode pour obtenir $\boldsymbol{e}'$ selon les trois étapes du décodage d'un code alternant (Section 3.1, page 50). Cette erreur $\boldsymbol{e}'$ va servir par la suite en plus de $\boldsymbol{c}_1$ à dériver la bonne clef de session K du KEM «SimpleDAGS».

Algorithme 17 Décapsulation

Entrée: $\boldsymbol{H}' \leftarrow (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$ la clef privée et $\boldsymbol{c}$ le texte chiffré.
Sortie: K la clef de session.

1: Construire le syndrome $\boldsymbol{c}'_0$ à partir de $\boldsymbol{H}'$.
2: Décoder $\boldsymbol{c}'_0$ pour obtenir $\boldsymbol{e}'$.
3: Définir $b = 0$ et $\eta = \boldsymbol{r}$, si le décodage échoue ou si $w_H(\boldsymbol{e}') \neq w$.
4: **si** $\tilde{\boldsymbol{H}}\boldsymbol{e}' = \boldsymbol{c}_0$ et $\mathrm{h}(2|\boldsymbol{e}') = \boldsymbol{c}_1$ **alors**
5: Mettre $b = 1$ et $\eta = \boldsymbol{e}'$.
6: **sinon**
7: Renvoyer $b = 0$ et $\eta = \boldsymbol{r}$.
8: **fin si**
9: Calculer $K = \mathrm{h}(b, \eta, \boldsymbol{c})$.
10: **Retourner** K.

Le passage du DAGS original à la variante «SimpleDAGS» n'est pas sans conséquence. D'une part, le changement dans la conversion en KEM est plus simple avec une réduction plus fine dans le modèle de l'oracle aléatoire. De plus, l'introduction de l'étape de confirmation du texte clair durant l'encapsulation et la décapsulation apporte une couche supplémentaire de sécurité au coût d'une seule valeur de hachage h. Cette approche est similaire à celle effectuée dans *Classic McEliece* [Alb+20].

D'autre part, l'utilisation du schéma de Niederreiter a un impact négatif sur la performance globale du KEM «SimpleDAGS». En effet, le calcul de l'inverse $\boldsymbol{A}^{-1}$ de la matrice secrète $\boldsymbol{A}$ et le produit avec $\boldsymbol{H}_{base}$ pendant l'étape 2 de la systématisation ont augmenté le nombre d'opérations dans la génération des clefs. Aussi, les opérations avec $\boldsymbol{A}$ et l'inverse $\boldsymbol{C}^{-1}$ de la matrice $\boldsymbol{C}$ (construite à partir de $\boldsymbol{u}$) lors de la construction de $\boldsymbol{c}'_0$ à partir $\boldsymbol{H}'$ (première étape avant le décodage) ont entraîné un coût supplémentaire dans la décapsulation.

Finalement, la variante «SimpleDAGS» est clairement un compromis qui sacrifie la performance en faveur d'une description plus compacte et d'une sécurité plus stricte, notamment avec les nouveaux paramètres proposés dans [Ban+19].

Table 3.7 – Paramètres de «SimpleDAGS».

Niveau sécu.	q	m	n	k	s	t	w	Complexité dans [BC18]	Complexité dans [Bar+19]
I	2^6	2	832	416	2^4	13	104	$\approx 2^{542}$	N.A.
III	2^8	2	1 216	512	2^5	11	176	$\approx 2^{288}$	N.A.
V	2^8	2	1 600	896	2^5	11	176	$\approx 2^{289}$	N.A.

3.3.2 Nouveaux paramètres

Les travaux effectués dans [BC18] et [Bar+19] ont permis aux auteurs de DAGS de proposer des paramètres hors de portée des attaques structurelles connues. En tenant compte des résultats rapportés dans la Table 3.6, les auteurs de DAGS ont mis à jour DAGS_1 [Ban+19] en choisissant des paramètres qui ne permettent pas de définir un code invariant ni son dual. Ceci met complètement en échec l'attaque de Bardet *et al.* alors que la première approche de Barelli-Couvreur produit une complexité largement supérieure au niveau de sécurité revendiqué. En définitive, nous pouvons affirmer que l'ensemble des paramètres de «SimpleDAGS» (noté $\mathsf{DAGS}_{\mathsf{simp}}\_1$, $\mathsf{DAGS}_{\mathsf{simp}}\_3$ et $\mathsf{DAGS}_{\mathsf{simp}}\_5$) est maintenant sûr contre toutes les attaques structurelles (Table 3.7) même si la taille des clefs publiques a légèrement augmenté par rapport au DAGS original (Table 3.8).

TABLE 3.8 – Taille des données dans «SimpleDAGS» (octets).

Ensemble de paramètres	Clef publique	Clef privée	Chiffré
$\mathsf{DAGS}_{\mathsf{simp}}\_1$	7 744	2 816	736
$\mathsf{DAGS}_{\mathsf{simp}}\_3$	11 264	4 864	1 248
$\mathsf{DAGS}_{\mathsf{simp}}\_5$	19 712	6 400	1 632

En plus de ces corrections apportées dans «SimpleDAGS», les auteurs ont aussi proposé un nouvel ensemble de paramètres binaires, que nous appellerons DAGS binaire. Dans notre article de mise œuvre logicielle efficace du DAGS binaire [Sec+23d], nous avons tout d'abord passé en revue les aspects de sécurité de cette nouvelle version de DAGS.

3.4 DAGS binaire

3.4.1 Construction

Dans la construction du KEM DAGS binaire (Figure 3.3), le principe d'encapsulation et de décapsulation reste le même que dans «SimpleDAGS» (Algorithmes 16 et 17) sauf que le corps de base ici est $\mathbb{F}_2$ ($q = 2$) au lieu de l'extension de corps $\mathbb{F}_{q^m}$ comme le montre l'Algorithme 18 pour la génération des clefs.

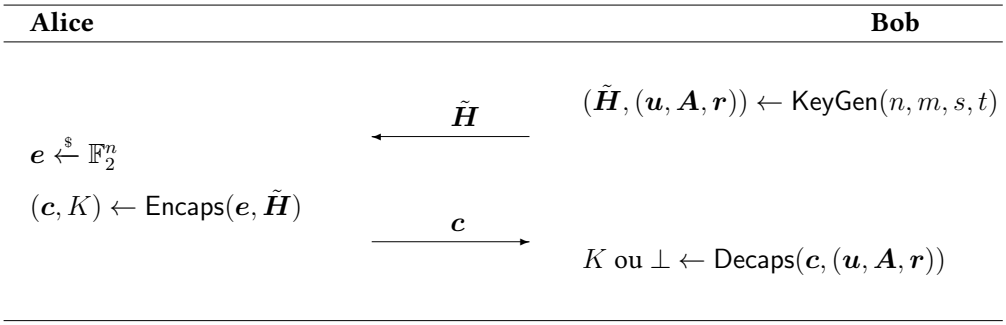

FIGURE 3.3 – Principe général du KEM DAGS binaire.

Algorithme 18 Génération des clefs

Entrée: n, m, s, t.
Sortie: $\mathsf{pk} = \tilde{\boldsymbol{H}}$ et $\mathsf{sk} = (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$.

1: Générer une signature dyadique $\boldsymbol{d}$.
2: Construire le support de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$.
3: Former la matrice de Cauchy $\hat{\boldsymbol{H}}_1 = \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$.
4: Construire $\hat{\boldsymbol{H}}$.
5: Générer $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_{2^m}^n$.
6: Former $\boldsymbol{H} = \hat{\boldsymbol{H}} \cdot Diag(\boldsymbol{z}_i)$.
7: Projeter $\boldsymbol{H}$ sur $\mathbb{F}_2$ pour obtenir la matrice $\boldsymbol{H}_{base}$.
8: Écrire $\boldsymbol{H}_{base} = (\boldsymbol{B}|\boldsymbol{A})$.
9: Obtenir une forme systématique $(\boldsymbol{I}_r|\boldsymbol{M}) = \boldsymbol{A}^{-1}\boldsymbol{H}_{base}$ que l'on note $\tilde{\boldsymbol{H}}$.
// Si $\boldsymbol{A}$ n'est pas inversible, revenir à l'étape 1.
10: Générer $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_2^n$.
11: $\mathsf{pk} \leftarrow \tilde{\boldsymbol{H}}$
12: $\mathsf{sk} \leftarrow (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$
13: **Retourner** (pk, sk).

3.4.2 Choix de paramètres et sécurité

DAGS binaire est un nouvel ensemble de paramètres ($\mathsf{DAGS}_{\mathsf{bin}}\_1$, $\mathsf{DAGS}_{\mathsf{bin}}\_3$ et $\mathsf{DAGS}_{\mathsf{bin}}\_5$) proposé dans [Ban+19]. Comme nous l'avons évoqué dans les sections précédentes, la sélection de paramètres pour les schémas basés sur les codes GS-QD n'est pas simple et doit satisfaire un certain nombre de contraintes. Tout d'abord, il faut que la dimension du code $k = n - mst$ soit approximativement égale à $n/2$, car le fait d'avoir un taux proche de 1/2 est un choix optimal. Ensuite, pour éviter l'attaque de Barelli-Couvreur, il faut que la valeur de q soit suffisamment grande et que l'ordre dyadique de s ne le soit pas. D'un autre côté, il nous faut un s suffisamment grand pour obtenir une réduction significative de la taille de la clef publique. En outre, q et s doivent être

des puissances de 2, ce qui accroît la difficulté de trouver un équilibre entre ces deux paramètres. Parallèlement, les valeurs du degré d'extension m et du nombre de blocs t doivent être, conjointement, suffisamment grandes pour atteindre le seuil de $mt \geq 21$, nécessaire pour éviter l'attaque FOPT. Nous avons également observé qu'un meilleur résultat est obtenu quand m et t sont de tailles opposées (une grande et une petite). Or, comme s et t déterminent le nombre d'erreurs corrigibles $(st/2)$, la valeur de t du coup ne peut pas être trop petite, tandis qu'un petit m est utile pour éviter les corps d'extension très grands. Il faut aussi noter que q^m doit toujours être au moins aussi grand que la longueur du code n. Le fait de choisir de grands m et de petits t permet de réduire au minimum la taille du corps de base q. Dans la proposition originale de Misoczki et Barreto [MB09] utilisant les codes Goppa QD, les paramètres binaires étaient le seul choix sûr. En effet, pour des corps de base plus grands, m doit être petit afin d'éviter de travailler sur des corps d'extension trop grands. Mais, cela signifie aussi que l'attaque FOPT risque d'être très efficace (car pas de paramètre t pour les codes de Goppa). Donc, afin de garantir la sécurité dans le cas des codes GS-QD, il faut choisir m aussi grand que possible et par conséquent $q = 2$. Or, dans ce cas aussi, si t est petit, s doit être grand (pour la correction des erreurs), ce qui fait augmenter n et k en conséquence. En définitive, les paramètres de DAGS binaire proposés sont hors de portée des attaques de Barelli-Couvreur et de Bardet *et al.* comme le montre la Table 3.9.

TABLE 3.9 – Paramètres DAGS binaire.

Niveau sécu.	q	m	n	k	s	t	w	Complexité dans [BC18]	Complexité dans [Bar+19]
I	2	13	6 400	3 072	2^7	2	126	N.A.	N.A.
III	2	14	11 520	4 352	2^8	2	256	N.A.	N.A.
V	2	14	14 080	6 912	2^8	2	256	N.A.	N.A.

Comme évoqué ci-dessus, la sélection des paramètres binaires implique un choix de code plus long, ce qui aboutit à des clefs légèrement plus grandes (Table 3.10) par rapport au DAGS original (environ 1,3 fois).

Malgré ces ajustements de paramètres par rapport à la version originale, DAGS binaire a toujours une taille de clef publique plus petite que celle des finalistes du NIST tels que *Classic McEliece* et BIKE (Table 3.11) pour le même niveau de sécurité III.

Par ailleurs, l'utilisation du corps de base binaire apporte des avantages en termes d'arithmétique, et permet une meilleure mise en œuvre. Cette variante de DAGS doit être considérée comme un autre compromis auquel la taille de la clef est sacrifiée au profit d'une sécurité accrue en plus d'une implantation logicielle plus efficace que les précédentes comme présenté dans [Sec+23d].

Table 3.10 – Taille des données dans DAGS binaire (octets).

Ensemble de paramètres	Clef publique	Clef privée	Chiffré
$\mathsf{DAGS}_{\mathsf{bin}}\_1$	9 984	20 800	832
$\mathsf{DAGS}_{\mathsf{bin}}\_3$	15 232	40 320	1 472
$\mathsf{DAGS}_{\mathsf{bin}}\_5$	24 192	49 280	1 792

Table 3.11 – Comparaison de clefs publiques (octets).

Schéma	DAGS original	DAGS binaire	BIKE	*Classic McEliece*
Clef publique	8 448	15 232	24 659	524 160

3.4.3 Implantation

Dans cette section, nous présentons les temps d'exécution obtenus dans notre implantation logicielle de DAGS binaire [Sec+23d]. Nos efforts ont porté sur plusieurs aspects du code, dans le but de fournir des algorithmes plus rapides, mais également plus clairs et plus accessibles. Dans cette implantation logicielle, nous avons exploité la structure particulière des matrices QD afin de rendre les opérations matricielles plus efficaces en ne considérant que les signatures des matrices comme indiqué dans [Ban+20]. De plus, nous avons utilisé l'algorithme de multiplication de Karatsuba, ainsi que la décomposition $\boldsymbol{LUP}$ des matrices QD pour améliorer de manière significative le calcul des inverses. Ces techniques de mise en œuvre nous ont permis d'accélérer la génération des clefs et la décapsulation de DAGS binaire par rapport aux implantations de DAGS original [Gus+17] (Table 3.12) et de «SimpleDAGS» [Ban+19] (Table 3.13).

Table 3.12 – Temps pour l'implantation de référence (en cycles).

Algorithme	Ensemble de paramètres		
	DAGS_1	DAGS_3	DAGS_5
Génération des clefs	2 540 311 986	4 320 206 006	7 371 897 084
Encapsulation	12 108 373	26 048 972	96 929 832
Décapsulation	215 710 551	463 849 016	1 150 831 538

TABLE 3.13 – Temps pour l'implantation de «SimpleDAGS» (en cycles).

Algorithme	Ensemble de paramètres		
	$\text{DAGS}_{\text{simp\_1}}$	$\text{DAGS}_{\text{simp\_3}}$	$\text{DAGS}_{\text{simp\_5}}$
Génération des clefs	408 342 881	1 560 879 328	2 061 117 168
Encapsulation	5 061 697	14 405 500	35 655 468
Décapsulation	192 083 862	392 435 142	388 316 593

Multiplication de Karatsuba Comme mentionné plus haut, certaines opérations de matrices QD, en particulier la multiplication, peuvent être effectuées plus efficacement en ne considérant que les signatures. Nous allons adapter l'algorithme de Karatsuba pour multiplier deux entiers au produit de deux matrices QD. Soient $\boldsymbol{M}$ et $\boldsymbol{B}$ deux matrices QD sur $\mathbb{F}_2^{n\times n}$ ($n = 2^s$) que nous voulons multiplier avec $\boldsymbol{m} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{n-1})$ et $\boldsymbol{b} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$, les signatures respectives. La matrice $\boldsymbol{D} = \boldsymbol{M}\boldsymbol{B}$ est également QD de signature $\boldsymbol{d} = (\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1})$.

Dans un algorithme de multiplication standard, un élément de $\boldsymbol{D}$ en position (i, j) est obtenu par multiplication entre la i-ème ligne de $\boldsymbol{M}$ et la j-ème colonne de $\boldsymbol{B}$. Comme les matrices QD sont symétriques, cela correspond au produit entre la i-ème ligne de $\boldsymbol{M}$ et la j-ème ligne de $\boldsymbol{B}$. La signature $\boldsymbol{d}$ (c'est-à-dire la première ligne de $\boldsymbol{D}$) est obtenue par des produits impliquant uniquement $\boldsymbol{m}$ (c'est-à-dire la première ligne de $\boldsymbol{M}$). Ainsi, nous pouvons simplement construire les lignes de $\boldsymbol{B}$, par permutations des éléments de $\boldsymbol{b}$, avant de calculer les produits. La complexité de cet algorithme est estimée à $\mathcal{O}(s \cdot (2^{2s} - 2^s) + 2^{2s} \cdot C_{mult} + (2^{2s} - 2^s) \cdot C_{som})$ dans [Ban+20], avec C_{mult} et C_{som} respectivement les coûts d'une multiplication et d'une somme sur $\mathbb{F}_2$.

Dans l'algorithme de Karatsuba pour multiplier deux matrices QD, nous allons considérer que la moitié des signatures. Soient $\boldsymbol{m}_{(1)}$ et $\boldsymbol{m}_{(2)}$ respectivement, la première et la seconde moitié de $\boldsymbol{m}$ tel que :

$$\boldsymbol{m}_{(1)} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{\frac{n}{2}-1})$$

$$\boldsymbol{m}_{(2)} = (\boldsymbol{m}_{\frac{n}{2}}, \ldots, \boldsymbol{m}_{n-1}).$$

De même, $\boldsymbol{b}_{(1)}$ et $\boldsymbol{b}_{(2)}$ ainsi que $\boldsymbol{d}_{(1)}$ et $\boldsymbol{d}_{(2)}$ correspondent respectivement aux moitiés de $\boldsymbol{b}$ et $\boldsymbol{d}$.

Le produit de deux matrices QD $\boldsymbol{D} = \boldsymbol{M}\boldsymbol{B}$ avec la méthode de Karatsuba donne

$$\begin{aligned} \boldsymbol{d}_{(1)} &= \boldsymbol{m}_{(1)}\boldsymbol{b}_{(1)} + \boldsymbol{m}_{(2)}\boldsymbol{b}_{(2)} \\ \boldsymbol{d}_{(2)} &= (\boldsymbol{m}_{(1)} + \boldsymbol{m}_{(2)})(\boldsymbol{b}_{(1)} + \boldsymbol{b}_{(2)}) + \boldsymbol{d}_{(1)}. \end{aligned} \tag{3.2}$$

L'application itérative de l'Équation 3.2 permet de calculer le produit de matrices QD

avec une complexité de $\mathcal{O}(3^s \cdot C_{mult} + 4 \cdot (3^s - 2^s) \cdot C_{som})$ [Ban+20], inférieure à la méthode standard.

Inversion de matrices QD L'inverse d'une matrice QD est une matrice QD (Proposition 2.5) entièrement définie par sa signature. La décomposition $\boldsymbol{LUP}$ est une méthode qui factorise la matrice QD $\boldsymbol{D}$ telle que

$$\boldsymbol{D} = \boldsymbol{LUP},$$

où $\boldsymbol{L}$ et $\boldsymbol{U}$ sont respectivement des matrices triangulaires inférieure et supérieure, et $\boldsymbol{P}$ une matrice de permutation.

L'inverse de $\boldsymbol{D}$ peut donc être exprimé comme suit

$$\boldsymbol{D}^{-1} = \boldsymbol{P}^{-1}\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}. \tag{3.3}$$

L'avantage de cette méthode est que les inverses de $\boldsymbol{L}$, $\boldsymbol{U}$ et $\boldsymbol{P}$ qui sont aussi sous forme QD, peuvent être facilement calculés, en raison de leurs structures particulières. En effet, l'inverse d'une matrice triangulaire s'obtient par substitution tandis que l'inverse de $\boldsymbol{P}$ est sa transposée. De plus, la décomposition se fait en blocs afin d'exploiter l'algèbre simple et efficace des matrices QD en utilisant directement les signatures.

L'algorithme de la décomposition $\boldsymbol{LUP}$ prend en entrée la matrice QD $\boldsymbol{D}$ et retourne sa factorisation en $\boldsymbol{L}$, $\boldsymbol{U}$ et $\boldsymbol{P}$ comme indiqué dans Algorithme 19.

Soit $\boldsymbol{d}_{i,j}$, la signature en position (i, j) dans $\boldsymbol{D}$ en sortie de l'Algorithme 19, les matrices $\boldsymbol{L}$ et $\boldsymbol{U}$ sont données par

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ \boldsymbol{d}_{1,0} & 1 & 0 & \dots & 0 \\ \boldsymbol{d}_{2,0} & \boldsymbol{d}_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{d}_{n-1,0} & \boldsymbol{d}_{n-1,1} & \boldsymbol{d}_{n-1,2} & \dots & 1 \end{pmatrix}, \boldsymbol{U} = \begin{pmatrix} \boldsymbol{d}_{0,0} & \boldsymbol{d}_{0,1} & \boldsymbol{d}_{0,2} & \dots & \boldsymbol{d}_{0,n-1} \\ 0 & \boldsymbol{d}_{1,1} & \boldsymbol{d}_{1,2} & \dots & \boldsymbol{d}_{1,n-1} \\ 0 & 0 & \boldsymbol{d}_{2,2} & \dots & \boldsymbol{d}_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \boldsymbol{d}_{n-1,n-1} \end{pmatrix}.$$

La matrice $\boldsymbol{P}$ est représentée par un vecteur de longueur n contenant une permutation d'entiers $(0, 1, \dots, n-1)$. Les lignes de $\boldsymbol{D}$ sont permutées en fonction des éléments de $\boldsymbol{P}$. Une fois la factorisation de $\boldsymbol{D}$ obtenue, il ne reste plus qu'à effectuer le calcul de $\boldsymbol{D}^{-1}$ à partir de 3.3. Comme les inverses $\boldsymbol{L}^{-1}$ et $\boldsymbol{U}^{-1}$ conservent leurs structures triangulaires d'origine et $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$. par conséquent, le calcul de $\boldsymbol{D}^{-1}$ s'effectue de manière plus efficace.

Algorithme 19 Décomposition $\boldsymbol{LUP}$ de la matrice $\boldsymbol{D}$

Entrée: $n = 2^s$ et $\boldsymbol{D} \in \mathbb{F}_2^{n \times n}$.
Sortie: $\boldsymbol{D} \in \mathbb{F}_2^{n \times n}$ et $\boldsymbol{P} \in \mathbb{N}^n$.

$\boldsymbol{P} \leftarrow (0, 1, \ldots, n-1)$.
pour $j \leftarrow 0$ jusqu'à $n-1$ **faire**
  pour $i \leftarrow j+1$ jusqu'à n **faire**
    $\boldsymbol{d}_{i,j} \leftarrow \boldsymbol{d}_{i,j}\boldsymbol{d}_{j,j}^{-1}$.
  fin pour
  pour $i \leftarrow j+1$ jusqu'à $n-1$ **faire**
    pour $\ell \leftarrow j+1$ jusqu'à $n-1$ **faire**
      $\boldsymbol{d}_{i,\ell} \leftarrow \boldsymbol{d}_{i,\ell} + \boldsymbol{d}_{i,j}\boldsymbol{d}_{j,\ell}$.
    fin pour
  fin pour
fin pour
Retourner $\boldsymbol{D}, \boldsymbol{P}$.

Application Nous allons maintenant appliquer ces méthodes qui accélèrent les opérations matricielles QD dans notre implantation logicielle de DAGS binaire.

Dans la génération des clefs Nous avons réalisé que la systématisation de la matrice $\boldsymbol{H}_{base}$ dans l'étape 9 (Algorithme 18, page 60) représente la quasi-totalité du temps d'exécution. Pour réduire ce coût, nous proposons quelques optimisations dans notre implantation logicielle. D'abord, au lieu de projeter toute la matrice $\boldsymbol{H}$ sur $\mathbb{F}_2$ pour obtenir $\boldsymbol{H}_{base}$ dans l'étape 7, nous avons juste besoin de projeter la signature $\boldsymbol{h}_{i,j}$ d'un bloc en position (i, j) de $\boldsymbol{H}$ pour obtenir la signature $\boldsymbol{h}_{base_{(i,j)}}$ de $\boldsymbol{H}_{base}$. Ainsi, la première ligne de $\boldsymbol{H}_{base}$ sera essentiellement composée des signatures des premiers blocs, la deuxième ligne par les signatures des deuxièmes blocs, et ainsi de suite (comme pour la construction de $\hat{\boldsymbol{H}}$, page 48). Finalement, on obtient une matrice $\boldsymbol{H}_{base}$ de taille $mt \times n$, composée de n_0 lignes par blocs avec $n_0 = n/s$. Cette méthode nous a permis de diminuer le temps nécessaire avant de mettre $\boldsymbol{H}_{base}$ sous forme systématique.

Ensuite, pour calculer l'inverse $\boldsymbol{A}^{-1}$ de la matrice secrète $\boldsymbol{A}$, qui est aussi QD (Proposition 2.5, page 23), et le produit $\boldsymbol{A}^{-1}\boldsymbol{H}_{base}$ dans l'étape 9 (Algorithme 18, page 60), nous combinons les techniques de la décomposition $\boldsymbol{LUP}$ et de la multiplication de Karatsuba (Algorithme 20, couleur bleue) qui donnent de meilleurs résultats par rapport aux méthodes standards.

Algorithme 20 Génération des clefs dans notre implantation [Sec+23d]

Entrée: n, m, s, t.
Sortie: $\mathsf{pk} = \tilde{\boldsymbol{H}}$ et $\mathsf{sk} = (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$.

1: Générer une signature dyadique $\boldsymbol{d}$.
2: Construire le support de Cauchy $(\boldsymbol{u}, \boldsymbol{v})$.
3: Former la matrice de Cauchy $\hat{\boldsymbol{H}}_1 = \boldsymbol{C}(\boldsymbol{u}, \boldsymbol{v})$.
4: Construire $\hat{\boldsymbol{H}}$.
5: Générer $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_{2^m}$.
6: Former $\boldsymbol{H} = \hat{\boldsymbol{H}} \cdot Diag(\boldsymbol{z}_i)$.
7: Projeter la signature $\boldsymbol{h}$ de $\boldsymbol{H}$ sur $\mathbb{F}_2$ pour obtenir $\boldsymbol{h}_{base}$.
8: Construire $\boldsymbol{H}_{base}$ à partir de $\boldsymbol{h}_{base}$.
9: Écrire $\boldsymbol{H}_{base} = (\boldsymbol{B}|\boldsymbol{A})$.
10: Factoriser $\boldsymbol{A}$ en $\boldsymbol{LUP}$.
11: Calculer $\boldsymbol{A}^{-1} = \boldsymbol{P}^{-1}\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}$.
12: Obtenir une forme systématique $(\boldsymbol{I}_r|\boldsymbol{M}) = Karatsuba(\boldsymbol{A}^{-1}\boldsymbol{H}_{base})$ que l'on note $\tilde{\boldsymbol{H}}$.
// Si $\boldsymbol{A}$ n'est pas inversible, revenir à l'étape 1.
13: Générer $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_2^n$.
14: $\mathsf{pk} \leftarrow \tilde{\boldsymbol{H}}$
15: $\mathsf{sk} \leftarrow (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$
16: **Retourner** (pk, sk).

Enfin, l'utilisation de ces techniques dans notre implantation nous ont permis de diviser par 10 le temps d'exécution de la génération des clefs dans DAGS original et par 5 dans «SimpleDAGS» (comme proposé dans [Ban+19]) pour le niveau de sécurité I (Table 3.14).
Dans ce qui suit, nous appelons $\mathsf{DAGS}_{\mathsf{improved}}$, les paramètres binaires utilisés dans notre implantation.

TABLE 3.14 – Comparaison des implantations de DAGS pour le niveau I (en cycles).

Algorithme	Ensemble de paramètres		
	$\mathsf{DAGS}_{\mathsf{simp}}\_1$	$\mathsf{DAGS}_{\mathsf{bin}}\_1$	$\mathsf{DAGS}_{\mathsf{improved}}\_1$
Génération des clefs	408 342 881	679 076 980	77 768 093
Encapsulation	5 061 697	6 564 782	4 641 252
Décapsulation	192 083 862	298 987 096	15 091 456

Dans l'encapsulation Il n'y a pas eu besoin d'optimisation, car c'est l'algorithme le plus rapide du KEM.

Dans la décapsulation La première étape (Algorithme 17, page 58) consiste à reconstruire la matrice secrète $\boldsymbol{H}'$ à partir de la clef privée avant de chercher le syndrome correspondant $\boldsymbol{c}'_0$ (Sous-section 3.3.1, page 56). Nous avons observé durant l'implantation que cette étape consomme près de la moitié du temps d'exécution total dans la décapsulation. Ainsi, pour réduire ce coût comme dans la génération des clefs, nous avons d'abord utilisé la multiplication de Karatsuba pour calculer $\boldsymbol{A}\tilde{\boldsymbol{H}} = \boldsymbol{H}_{base}$.

Ensuite, à partir de $\boldsymbol{H}_{base}$ nous inversons le processus de projection pour obtenir $\boldsymbol{H}$ à coefficients dans $\mathbb{F}_{2^m}$ avec la même méthode que dans la génération des clefs avant de calculer la matrice $\boldsymbol{C}$ avec le support secret $\boldsymbol{u}$.

Enfin, nous calculons $\boldsymbol{H}' = \boldsymbol{C}^{-1}\boldsymbol{H}$ en utilisant la même technique que pour la systématisation. La décomposition $\boldsymbol{LUP}$ de la matrice $\boldsymbol{C}$, en travaillant directement avec les signatures, nous a permis de calculer son inverse $\boldsymbol{C}^{-1}$ plus efficacement et la méthode de Karatsuba pour effectuer le produit $\boldsymbol{C}^{-1}\boldsymbol{H}$. Ces optimisations dans notre implantation (Algorithme 21, en bleue) nous ont permis de diviser le temps d'exécution de la décapsulation encore plus rapide que dans les implantations précédentes de DAGS (Tables 3.14 et 3.15).

Algorithme 21 Décapsulation dans notre implantation [Sec+23d]

Entrée: $\boldsymbol{H}' \leftarrow (\boldsymbol{u}, \boldsymbol{A}, \boldsymbol{r})$ la clef privée et $\boldsymbol{c}$ le texte chiffré.
Sortie: K la clef de session.

1: Calculer $\boldsymbol{H}_{base} = Karatsuba(\boldsymbol{A}\tilde{\boldsymbol{H}})$.
2: Projeter la signature $\boldsymbol{h}_{base}$ de $\boldsymbol{H}_{base}$ sur $\mathbb{F}_{2^m}$ pour obtenir $\boldsymbol{h}$ de $\boldsymbol{H}$.
3: Construire la matrice $\boldsymbol{H}$ à partir de $\boldsymbol{h}$.
4: Former la matrice $\boldsymbol{C}$ à partir de $\boldsymbol{u}$.
5: Factoriser $\boldsymbol{C}$ en $\boldsymbol{LUP}$.
6: Calculer $\boldsymbol{C}^{-1} = \boldsymbol{P}^{-1}\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}$.
7: Calculer $\boldsymbol{H}' = Karatsuba(\boldsymbol{C}^{-1}\boldsymbol{H})$.
8: Construire le syndrome $\boldsymbol{c}'_0$ à partir de $\boldsymbol{H}'$.
9: Décoder $\boldsymbol{c}'_0$ pour obtenir $\boldsymbol{e}'$.
10: Définir $b = 0$ et $\eta = \boldsymbol{r}$, si le décodage échoue ou si $w_H(\boldsymbol{e}') \neq w$.
11: **si** $\tilde{\boldsymbol{H}}\boldsymbol{e}' = \boldsymbol{c}_0$ et $\mathrm{h}(2|\boldsymbol{e}') = \boldsymbol{c}_1$ **alors**
12: Mettre $b = 1$ et $\eta = \boldsymbol{e}'$.
13: **sinon**
14: Renvoyer $b = 0$ et $\eta = \boldsymbol{r}$.
15: **fin si**
16: Calculer $K = \mathrm{h}(b, \eta, \boldsymbol{c})$.
17: **Retourner** K.

TABLE 3.15 – Comparaison des implantations de DAGS pour le niveau III (en cycles).

Algorithme	Ensemble de paramètres		
	$\text{DAGS}_{\text{simp}}\_3$	$\text{DAGS}_{\text{bin}}\_3$	$\text{DAGS}_{\text{improved}}\_3$
Génération des clefs	1 560 879 328	1 597 980 876	847 980 876
Encapsulation	14 405 500	15 200 232	8 020 732
Décapsulation	392 435 142	454 765 478	34 656 844

3.4.4 Performance

Nous comparons les performances de notre implantation de DAGS binaire avec les KEM finalistes du NIST et nous rapportons les temps obtenus dans les Tables 3.16 et 3.17. En plus d'être sécurisé contre les attaques structurelles connues, la variante binaire de DAGS a des performances correctes par rapport aux meilleurs KEM actuels basés sur les codes correcteurs d'erreurs. Les temps ont été acquis en exécutant notre code C sur un compilateur CLANG version 8.0.0 avec les flags de compilation -O3 -g3 -Wall -march=native -mtune=native -fomit-frame-pointer -ffast-math. Le processeur est un Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz.

TABLE 3.16 – Comparaison avec les finalistes du NIST pour le niveau I (en cycles).

Algorithme	Ensemble de paramètres			
	HQC	BIKE	*Classic McEliece*	$\text{DAGS}_{\text{improved}}\_1$
Génération des clefs	98 570	650 638	49 758 742	77 768 093
Encapsulation	356 980	247 976	56 672	4 641 252
Décapsulation	467 891	2 575 687	253 864	15 091 456

3.5 Conclusion

DAGS était l'une des deux propositions pour la normalisation du NIST basées sur des codes algébriques structurés (BIG QUAKE est basé sur des codes quasi-cycliques) et la seule à utiliser les codes GS-QD. L'approche de DAGS original était de choisir des paramètres «agressifs» ($m = 2$), dans le but d'atteindre des tailles de données (clefs publique/privée et chiffré) intéressantes. Malheureusement, cela a conduit à l'attaque de Barelli-Couvreur et à la décision du NIST de ne pas sélectionner DAGS pour le deuxième tour du processus. Néanmoins, de nouveaux paramètres, résistants à toutes les attaques

TABLE 3.17 – Comparaison avec les finalistes du NIST pour le niveau III (en cycles).

Algorithme	Ensemble de paramètres			
	HQC	BIKE	*Classic McEliece*	$\mathsf{DAGS}_{\mathsf{improved\_3}}$
Génération des clefs	267 983	3 674 894	364 756 564	847 980 876
Encapsulation	567 836	564 896	245 794	8 020 732
Décapsulation	947 920	4 298 673	387 678	34 656 844

structurelles existantes, ont été proposés par les auteurs de la soumission grâce notamment à l'analyse de l'attaque de Bardet *et al.* Les auteurs de DAGS ont également adopté une toute nouvelle approche du sujet. D'une part, la variante «SimpleDAGS» qui est essentiellement une conversion du protocole original dans le cadre de Niederreiter, est un schéma plus compact et permet une analyse de sécurité plus simple comme dans *Classic McEliece*. D'autre part, un nouvel ensemble de paramètres binaires est proposé, dont nous avons étudié la sécurité avant de faire son implantation logicielle. Cette variante binaire de DAGS offre des avantages considérables, notamment une sécurité contre les attaques structurelles connues. Notre mise en œuvre logicielle de DAGS binaire utilisant les techniques de multiplication de Karatsuba et de décomposition $\boldsymbol{LUP}$ dans le cas dyadique, a permis de réduire le temps d'exécution des algorithmes de génération des clefs et de décapsulation. Cela nous a également permis d'avoir des performances plus rapides par rapport aux implantations précédentes, mais on reste moins performant par rapport aux finalistes du NIST basés sur les codes. Dans nos travaux futurs, nous prévoyons d'accélérer encore le temps d'exécution en optimisant le choix des paramètres pour le DAGS binaire, par exemple pour le niveau de sécurité I, le choix de $n = 6400$ est trop élevé. Nous envisageons également de réaliser une implantation matérielle avec une analyse des attaques physiques exploitant la structure particulière de cette construction comme dans *Classic McEliece* qui justement sera la cible de notre analyse par canaux cachés au prochain chapitre.

Troisième partie

Attaques de cryptosystèmes basés sur les codes

Chapitre 4

Attaque template sur *Classic McEliece*

Ce chapitre correspond à notre papier accepté pour AFRICACRYPT, 2023 [Sec+23c] :

A Side-Channel Attack against *Classic McEliece* when loading the Goppa Polynomial
avec Pierre-Louis Cayrel, Vlad-Florin Dragoi, Idy Diop, Jean Belo Klamti Morgan Barbier, Vincent Grosso et Brice Colombier. AFRICACRYPT 2023, LNCS.

Le processus de normalisation de la cryptographie post-quantique (Post-Quantum Cryptography ou PQC) lancé par le NIST (National Institute of Standards and Technology) [Nis] depuis décembre 2016, a livré ses premiers résultats pour les futures normes de signature numérique et de mécanisme d'encapsulation de clefs (Key Encapsulation Mechanism ou KEM) au mois de juillet 2022. Ce processus a donné un élan considérable à la recherche sur l'étude des algorithmes post-quantiques, notamment leurs vulnérabilités faces aux attaques par canaux cachés (Side-Channel Attack ou SCA). Les SCA sont des attaques en pratique qui exploitent les propriétés physiques d'un circuit électronique pour obtenir des informations sensibles ou secrètes dans une implantation de schéma cryptographique. En plus de la définition de schémas et de choix de paramètres sécurisés, un aspect très important dans ce processus, est l'impact de la mise en œuvre sur la sécurité. Une exigence généralement couverte par les implantations des candidats du NIST, est le temps d'exécution des opérations qui ne varie pas en fonction des données secrètes (clef privée ou texte clair) pour se prémunir d'une attaque temporelle. C'est ce que l'on appelle une mise en œuvre à temps constant. Toutefois, il existe d'autres types de SCA sur la consommation de puissance notamment, qui peuvent permettre à Eve (attaquante) d'accéder à ces informations secrètes dans une implantation d'algorithme post-quantique. Ainsi, pour la plupart des schémas post-quantiques, on ne sait pas en pratique quelles sont les SCA réalisables et comment s'en protéger.

Dans ce chapitre, nous présentons notre attaque template (Template Attack ou TA) [Sec+23c] réalisée sur la décapsulation (ou déchiffrement) du KEM finaliste pour le 4ème

tour du NIST, *Classic McEliece*[Alb+20] utilisant les codes de Goppa binaires. Notre attaque cible le chargement des coefficients du polynôme de Goppa dans l'implantation référence du KEM *Classic McEliece*[Alb+20] sur un microcontrôleur STM32 intégrant un cœur ARM Cortex-M4 [CC21]. Notre SCA, pratiquement réalisable pour tous les ensembles de paramètres proposés dans *Classic McEliece* nous a permis, tout d'abord, de retrouver les poids de Hamming $w_H(g_i)$ des coefficients du polynôme de Goppa g. Grâce à ces informations, nous avons réussi à retrouver directement le polynôme $g(x)$, dans le cas des clefs faibles, avec la méthode de Loidreau et Sendrier [LS01]. Dans le cas des «clefs moins faibles», nous montrons aussi que l'on peut retrouver g en temps polynomial. Ensuite, nous proposons une meilleure réduction de la complexité pour la recherche exhaustive du polynôme g sur $\mathbb{F}_{2^m}$. Enfin, nous montrons que notre attaque est plus réaliste en pratique par rapport à d'autres SCA sur *Classic McEliece*.

4.1 Construction

Classic McEliece [Alb+20] est un KEM (Figure 4.1) basé sur les codes de Goppa binaires (Définition 2.24, page 27). Sa construction est similaire à celle de DAGS binaire vu dans le chapitre précédent (Section 3.4, page 59). Le KEM *Classic McEliece* est conçu pour fournir une sécurité IND-CCA2 et est construit de manière conservatrice à partir du schéma de chiffrement à clef publique (Public Key Encryption ou PKE) de Niederreiter [Nie86]. Les paramètres dans ce schéma sont la longueur n, le corps de base $q = 2$, le degré de l'extension m, la capacité de correction t, la dimension $k = n - mt$ et la codimension $r = n - k = mt$ du code de Goppa.

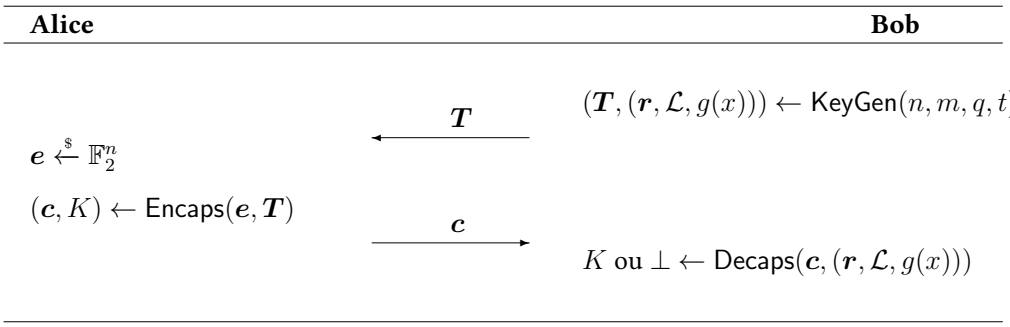

FIGURE 4.1 – Principe général du KEM *Classic McEliece*.

Pour mieux comprendre notre TA dans la Section 4.2 (page 78), nous allons d'abord rappeler les algorithmes dans *Classic McEliece*.

Génération des clefs L'algorithme de génération des clefs (Algorithme 22) prend en entrée les paramètres du code de Goppa binaire ($q = 2$) n, m, q, t et retourne la clef

publique (pk), la matrice $\boldsymbol{T}$ de taille $mt \times (n - mt)$ sur $\mathbb{F}_2$, et la clef privée (sk), le triplet $(\boldsymbol{r}, \mathcal{L}, g(x))$. La principale étape de cet algorithme est la génération de la matrice de Goppa $\hat{\boldsymbol{H}}$.

Génération de $\hat{\boldsymbol{H}}$

1. Construire une matrice $\tilde{\boldsymbol{H}} \in \mathbb{F}_{2^m}$ de Goppa $\Gamma(\mathcal{L}, g)$ de taille $t \times n$ où ses éléments $\tilde{\boldsymbol{h}}_{i,j}$ sont donnés par
$$\tilde{\boldsymbol{h}}_{i,j} = \frac{\alpha_j^{i-1}}{g(\alpha_i)},$$
pour $i = 1, \ldots, t$ et $j = 1, \ldots, n$, comme dans le Théorème 2.3.
2. Utiliser l'isomorphisme de l'espace vectoriel entre $\mathbb{F}_{2^m}$ et $\mathbb{F}_2^m$ pour remplacer chaque élément $\tilde{\boldsymbol{h}}_{i,j}$ de $\tilde{\boldsymbol{H}}$ dans $\mathbb{F}_{2^m}$ par des vecteurs dans $\mathbb{F}_2^m$ (disposés en colonnes) pour obtenir une matrice $\hat{\boldsymbol{H}}$ de taille $mt \times n$.

Algorithme 22 Génération des clefs

Entrée: n, m, q, t.
Sortie: $\mathsf{pk} = \boldsymbol{T}$ et $\mathsf{sk} = (\boldsymbol{r}, \mathcal{L}, g(x))$.

1: Générer un polynôme irréductible et unitaire $g(x) \in \mathbb{F}_{2^m}[x]$ de degré t.
2: Construire le support $\mathcal{L} = (\alpha_1, \ldots, \alpha_n) \subseteq \mathbb{F}_{2^m}$.
3: Générer la matrice $\hat{\boldsymbol{H}}$.
4: Obtenir une forme systématique de $\hat{\boldsymbol{H}} = (\boldsymbol{I}_{mt} | \boldsymbol{T}_{mt \times (n-mt)})$ que l'on note $\boldsymbol{H}$.
// Sinon revenir à l'étape 1.
5: Générer $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}_2^n$.
6: $\mathsf{pk} \leftarrow \boldsymbol{T}$
7: $\mathsf{sk} \leftarrow (\boldsymbol{r}, \mathcal{L}, g(x))$
8: **Retourner** (pk, sk).

Encapsulation L'algorithme d'encapsulation (Algorithme 23) prend en entrée la clef publique $\boldsymbol{T}$ et retourne le texte chiffré $\boldsymbol{c}$ ainsi que la clef de session K.

Remarque 4.1 *La fonction de hachage h utilisée dans les algorithmes d'encapsulation et de décapsulation est un SHAKE256 pour fournir une confirmation en clair et d'extraire la clef de session K.*

Algorithme 23 Encapsulation

Entrée: pk $= \boldsymbol{T}$ la clef publique.
Sortie: $\boldsymbol{c}$ le texte chiffré et la clef encapsulée K.

1: Générer $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_2^n$, une erreur de poids t.
2: Calculer $\boldsymbol{H} = (\boldsymbol{I}_{mt}|\boldsymbol{T}_{mt\times(n-mt)})$.
3: Calculer $\boldsymbol{c}_0 = \boldsymbol{H}\boldsymbol{e}$ de longueur mt.
4: Calculer $\boldsymbol{c}_1 = \mathrm{h}(2|\boldsymbol{e})$.
5: Définir $\boldsymbol{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$.
6: Calculer $K = \mathrm{h}(1|\boldsymbol{e}|\boldsymbol{c})$.
7: **Retourner** $(\boldsymbol{c}, K)$.

Décapsulation L'algorithme de décapsulation (Algorithme 24) prend en entrée la clef privée sk $= (\boldsymbol{r}, \mathcal{L}, g(x))$ ainsi que le texte chiffré $\boldsymbol{c}$ et retourne la clef de session K. L'étape cruciale dans la décapsulation de *Classic McEliece* est le décodage du code de Goppa $\Gamma(\mathcal{L}, g)$ binaire utilisant le même principe que dans Algorithme 1 (page 30). Cet algorithme prend en entrée cette fois-ci le syndrome $\boldsymbol{c}$ (instanciation de Niederreiter), le support $\mathcal{L} = (\alpha_1, \ldots, \alpha_n)$ et le polynôme de Goppa g et retourne l'erreur $\boldsymbol{e}'$. Une fois le décodage terminé (obtention de $\boldsymbol{e}'$ de poids $w_H(\boldsymbol{e}') = t$ tel que $\boldsymbol{H}\boldsymbol{e}' = \boldsymbol{c}_0$), on utilise $\boldsymbol{c}_1$ pour dériver la bonne clef de session K.

Algorithme 24 Décapsulation

Entrée: $(\boldsymbol{r}, \mathcal{L}, g(x))$ la clef privée et $\boldsymbol{c}$ le texte chiffré.
Sortie: K la clef de session.

1: Ajouter $n - mt$ zéros à $\boldsymbol{c}_0$ pour avoir $\boldsymbol{c}_0 \in \mathbb{F}_2^n$.
2: Décoder $\boldsymbol{c}$ pour obtenir $\boldsymbol{e}'$.
   // Avec $\boldsymbol{e}' \leftarrow$ Algorithme 1 $(\boldsymbol{c}, \mathcal{L}, g(x))$.
3: Définir $b = 0$ et $\eta = \boldsymbol{r}$, si le décodage échoue ou si $w_H(\boldsymbol{e}') \neq t$.
4: **si** $\boldsymbol{H}\boldsymbol{e}' = \boldsymbol{c}_0$ et $\mathrm{h}(2|\boldsymbol{e}') = \boldsymbol{c}_1$ **alors**
5: Mettre $b = 1$ et $\eta = \boldsymbol{e}'$.
6: **sinon**
7: Renvoyer $b = 0$ et $\eta = \boldsymbol{r}$.
8: **fin si**
9: Calculer $K = \mathrm{h}(b, \eta, \boldsymbol{c})$.
10: **Retourner** K.

La décapsulation de *Classic McEliece* est inspirée de schéma de «McBits» proposé par Bernstein *et al.* [BCS13], révisé par Chou dans [Cho17]. Le «McBits» est une implantation rapide en temps constant qui utilise des algorithmes tels que Berlekamp-Massey (Annexe A) pour chercher le polynôme localisateur d'erreurs $\sigma(x)$, la FFT (Fast Fourier Transform) pour l'évaluation de $\sigma(x)$ sur les α_i, ainsi que pour le calcul du syndrome secret $\boldsymbol{c}_0 = \boldsymbol{H}\boldsymbol{e}'$, le réseau de Beneš pour la permutation des éléments de $\mathcal{L}$,

et une représentation en «bitsliced» (par morceau) pour accélérer les opérations de bas niveau. Pour rappel, le bilslicing est une technique d'implantation logicielle proposée dans [Bih97] pour accélérer les opérations dans l'algorithme de chiffrement symétrique DES (Data Encryption Standard). L'idée de base est de transposer m données à n bits dans n différents registres à m bits. Cela permet, entre autres, d'effectuer des opérations logiques en parallèle.

Les paramètres de *Classic McEliece* tels que proposés dans [Alb+20], correspondant aux trois niveaux de sécurité I, III et V (Table 2.5), sont présentés dans les Tables 4.1 et 4.2. La variante mceliece8192128 avec un niveau de sécurité V est la cible de notre attaque profilée dans la section suivante.

TABLE 4.1 – Paramètres dans *Classic McEliece*.

Niveau sécu.	Ensemble de paramètres	q	m	n	t
I	mceliece348864	2	12	3488	64
III	mceliece460896	2	13	4608	96
V	mceliecee6688128	2	13	6688	128
V	mceliece6960119	2	13	6960	119
V	mceliece8192128	2	13	8192	128

TABLE 4.2 – Taille des données dans *Classic McEliece* (octets).

Ensemble de paramètres	Clef publique	Clef privée	Chiffré
mceliece348864	261 120	6 492	128
mceliece460896	524 160	13 608	188
mceliecee6688128	1 044 992	13 932	240
mceliece6960119	1 047 319	13 948	226
mceliece8192128	1 357 824	14 120	240

4.2 Détails de notre attaque

Proposée pour la première fois par Chari *et al.* dans [CRR02], l'attaque template est une SCA considérée aujourd'hui parmi les plus puissantes pour exploiter les failles dans une implantation de schéma cryptographique. Elle fait partie de la famille des attaques profilées (Sous-section 1.2.2, page 15), dans laquelle Eve cherche d'abord à caractériser le profil (la consommation de puissance ou de courant) d'un circuit sensible (ou cible) à l'aide d'un autre circuit identique (ou clone) pour trouver des informations secrètes. C'est une attaque composée d'une phase d'apprentissage (ou building) pour la caractérisation de la cible qui nécessite plusieurs traces de consommation et une phase d'attaque (ou matching), qui avec seulement quelques traces (voire une seule), permet de retrouver le secret. Dans cette section, nous allons d'abord rappeler les bases d'une TA, ensuite, nous montrerons que le chargement des coefficients du polynôme irréductible et unitaire de Goppa g (une partie de la clef privée) dans [CC21] laisse échapper de l'information sensible et, enfin, nous présenterons notre attaque pour retrouver les poids de Hamming $w_H(g_i)$ des coefficients de g. Cette information sur les poids de Hamming va nous permettre d'obtenir de nouveaux résultats en cryptographie basée sur les codes correcteurs d'erreurs (Section 4.3).

4.2.1 Attaque template

L'attaque template exploite le fait que la consommation de puissance d'un circuit dépend des données traitées. L'idée générale dans une telle attaque repose sur le fait qu'une trace de consommation peut être modélisée à l'aide d'une loi de probabilité telle que la distribution normale (ou gaussienne) multivariée.

Distribution normale multivariée C'est une généralisation de la loi normale dont les composantes des vecteurs sont des variables aléatoires gaussiennes. Dans un tel modèle de consommation, la corrélation entre les points voisins (ou échantillons) dans une trace est prise en compte afin d'améliorer la caractérisation. Soit $\boldsymbol{t}_r$ une trace de consommation. Une gaussienne multivariée est entièrement définie par sa densité de probabilité donnée par :

$$p(\boldsymbol{t}_r; (\mu, \boldsymbol{C})) = \frac{1}{\sqrt{(2\pi)^\tau \times \det(\boldsymbol{C})}} \exp\left(-\frac{1}{2}(\boldsymbol{t}_r - \mu)^T \boldsymbol{C}^{-1}(\boldsymbol{t}_r - \mu)\right), \tag{4.1}$$

avec τ le nombre d'échantillons dans la trace $\boldsymbol{t}_r$, $\boldsymbol{C}$ une matrice de covariance de taille $\tau \times \tau$ et μ un vecteur de moyenne de τ éléments. C'est le couple $(\mu, \boldsymbol{C})$ qui constitue une template que l'on note φ.

La matrice de covariance $\boldsymbol{C}$ caractérise essentiellement les fluctuations dans une trace telles que le bruit électronique. Cependant, l'un des inconvénients en pratique avec la distribution normale multivariée est que la taille (ou coût) de cette matrice de covariance $\boldsymbol{C}$ croît de manière quadratique avec τ. En effet, pendant la caractérisation,

si nous considérons tous les échantillons (nombre souvent très élevé en pratique) de la trace $\boldsymbol{t}_r$, nous obtenons une distribution à τ dimension. Par conséquent, le calcul de l'inverse $\boldsymbol{C}^{-1}$ de $\boldsymbol{C}$, nécessaire pour obtenir la densité de la distribution (Équation 4.1), peut s'avérer très complexe comme expliqué dans [EPW10]. De plus, dans une trace, tous les échantillons ne contiennent pas forcément de l'information (donnée sensible) relative à l'opération cible dans le circuit à attaquer. C'est pourquoi dans une TA (ou dans une attaque profilée), il est indispensable de trouver une stratégie pour déterminer les échantillons qui contiennent le plus d'information par rapport à l'opération ou l'instruction que l'on souhaite caractériser. On réduit ainsi la dimension de la gaussienne multivariée en ne considérant que ces échantillons : ce sont les points d'intérêts (Point Of Interest ou POI).

Recherche de points d'intérêts Les points d'intérêts ou POI sont des échantillons dans une trace qui varient fortement suivant des hypothèses sur la clef privée (ou sur son poids de Hamming) pendant la caractérisation du circuit cible. Il existe plusieurs méthodes statistiques pour trouver le nombre de POI ($N_{POI} \ll \tau$) dans une trace. On peut citer notamment la somme des différences (Sum of Differences ou SOD) [RO04], la somme des carrés des écarts (Sum Of Squared Differences ou SOSD) [GLRP06], l'analyse en composante principale (Pincipal Component Analysis ou PCA) [Arc+06], l'analyse discriminante linéaire (Linear Discriminant Analysis ou LDA)[SA08], etc. Dans notre attaque, nous allons utiliser la méthode SOD pour construire des templates $\varphi_i = (\mu, \boldsymbol{C})_i$ à faible coût mémoire. En effet, pour une opération (cible) de données data_i, on effectue d'abord des acquisitions de $N_{\boldsymbol{t}_r}$ traces ($N_{\boldsymbol{t}_r}$ est le nombre de traces $\boldsymbol{t}_r$ souhaité) correspondant à l'hypothèse $j_{(j=1,\ldots,N)}$ (N est le nombre d'hypothèses) sur τ échantillons, définis en pratique, avant de calculer la moyenne μ_j. Ensuite, on répète les mêmes opérations pour les N hypothèses et on calcule les différences de moyennes absolues deux à deux. Enfin, on effectue la somme en valeur absolue de ces différences de moyenne avant de reporter le résultat sur un graphe que l'on appelle : la trace SOD. Ainsi, les pics les plus importants dans la trace SOD indiquent les POI dans l'opération cible. La méthode SOD est résumée dans l'Algorithme 4.8.

Algorithme 25 Somme des différences

Entrée: $N, N_{\boldsymbol{t}_r}, \tau$.
Sortie: SOD.

1: **pour** $j \leftarrow 1$ jusqu'à N **faire**
2: **pour** $i \leftarrow 1$ jusqu'à $N_{\boldsymbol{t}_r}$ **faire**
3: Faire les acquisitions de traces $\boldsymbol{t}_{r_i}$ sur τ échantillons.
4: **fin pour**
5: Calculer la moyenne $\mu_j = \frac{1}{N_{\boldsymbol{t}_r}} \sum_{i=1}^{N_{\boldsymbol{t}_r}} \boldsymbol{t}_{r_i}$.
6: **fin pour**
7: Calculer la somme des différences $SOD = \sum_{j,j+1}^{N} |\mu_j - \mu_{j+1}|$.
8: **Retourner** SOD.

Par conséquent, on réduit clairement la dimension de la distribution multivariée avec une matrice $\boldsymbol{C}$ de taille $N_{POI} \times N_{POI}$ et l'Équation 4.1 de la densité de probabilité devient

$$p(\boldsymbol{t}_r; (\mu, \boldsymbol{C})) = \frac{1}{\sqrt{(2\pi)^{N_{POI}} \times \det(\boldsymbol{C})}} \exp\left(-\frac{1}{2}(\boldsymbol{t}_r - \mu)^T \boldsymbol{C}^{-1} (\boldsymbol{t}_r - \mu)\right). \tag{4.2}$$

On sait désormais qu'en pratique, la caractérisation des traces de consommation pendant la phase d'apprentissage ne porte que sur quelques échantillons spécifiques (POI) et cette étape est cruciale pour le succès d'une attaque template durant la phase de matching. On sait maintenant qu'en pratique, la caractérisation des traces de consommation pendant phase d'apprentissage ne concerne que quelques échantillons spécifiques. Cette étape de rechercher des POI est cruciale pour le succès des attaques profilées durant la phase de matching, notamment l'attaque template.

Déroulement d'une attaque template Le déroulement générique d'une attaque template est décrit dans la Figure 4.2. Comme nous l'avons évoqué ci-dessus, une TA est une attaque profilée composée de deux étapes.

Phase d'apprentissage Dans cette étape de caractérisation, on utilise un autre circuit du même type que la cible qu'on appelle clone. Dans ce circuit clone dont nous avons entièrement le contrôle, on y exécute d'abord les mêmes séquences d'instructions que dans la cible avec différentes données data_i et d'hypothèses de clefs sk_j afin d'enregistrer la consommation de puissance qui en résulte. Ensuite, on regroupe les traces correspondant à chaque paire $(\mathsf{data}_i, \mathsf{sk}_j)$ avant de calculer leur moyenne μ et leur matrice de covariance $\boldsymbol{C}$. Enfin, nous obtenons les templates

$$\varphi_{\mathsf{data}_i, \mathsf{sk}_j} = (\mu, \boldsymbol{C})_{\mathsf{data}_i, \mathsf{sk}_j}. \tag{4.3}$$

Phase d'attaque Dans cette phase, nous utilisons la caractérisation (les templates) dans l'étape d'apprentissage avec une trace $\boldsymbol{t}_r$ du circuit cible pour déterminer la clef privée sk. En fait, on évalue d'abord la densité de probabilité de la distribution multivariée de $\boldsymbol{t}_r$ avec chaque template $\varphi_{\mathsf{data}_i,\mathsf{sk}_j}$ et on a

$$p(\boldsymbol{t}_r; (\mu, \boldsymbol{C})_{\mathsf{data}_i,\mathsf{sk}_j}) = \frac{1}{\sqrt{(2\pi)^{N_{POI}} \times \det(\boldsymbol{C})}} \exp\left(-\frac{1}{2}(\boldsymbol{t}_r - \mu)^T \boldsymbol{C}^{-1}(\boldsymbol{t}_r - \mu)\right). \quad (4.4)$$

Ensuite, on établit un classement par ordre croissant de densité de probabilité $p(\boldsymbol{t}_r; (\mu, \boldsymbol{C})_{\mathsf{data}_i,\mathsf{sk}_j})$. Enfin, le template ayant la plus grande densité de probabilité vérifie l'hypothèse de clef privée $\mathsf{sk} = \mathsf{sk}_j$ correspondant à la trace $\boldsymbol{t}_r$ du circuit cible.

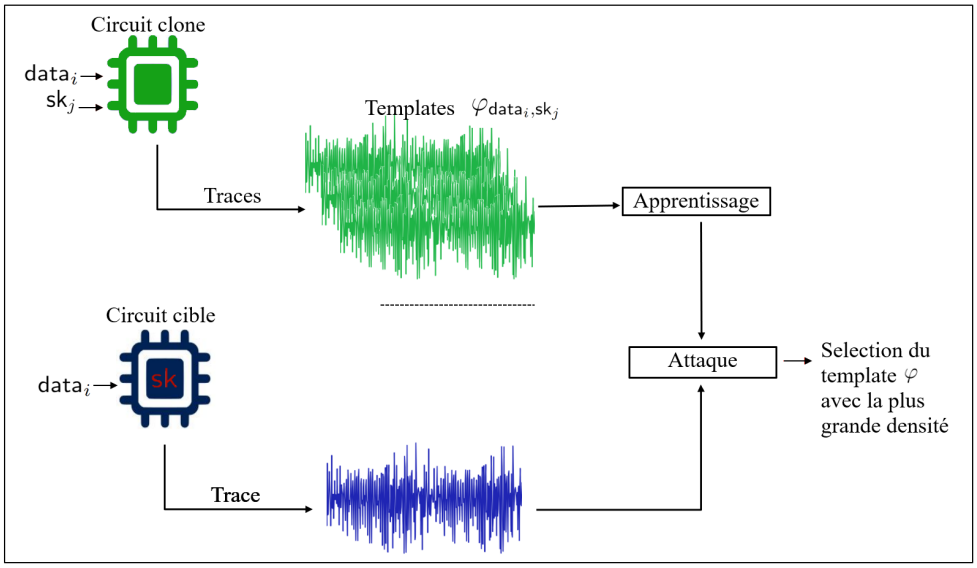

FIGURE 4.2 – Modèle générique d'une attaque template.

Avant de réaliser notre attaque template dans l'implantation de *Classic McEliece* [CC21], nous allons d'abord identifier la source de fuite d'information.

Remarque 4.2 *Il est également possible de construire des templates pour des données secrètes avec différents poids de Hamming. Supposons que nous ayons une séquence d'instructions secrètes dont la sortie est sur un octet. Nous pouvons alors construire 9 templates de 0 à 8 poids de Hamming différents. C'est l'approche que nous allons adopter pour construire nos templates dans la Sous-section 4.2.3.*

4.2.2 Fuite dans l'implantation [CC21]

Le choix de l'implantation dans [CC21] se justifie par le fait que Chen et Chou ont réalisé la mise en œuvre de la proposition référence de *Classic McEliece* sur un microcontrôleur STM32 intégrant un cœur ARM Cortex-M4. Comme nous l'avons souligné

ci-dessus, notre cible est la fonction de chargement du polynôme $g(x)$ de degré t dans la décapsulation de mceliece8192128, avec les paramètres $m = 13, n = 8192$ et $t = 128$ où chaque coefficient de g est représenté sur m bits. Mais, dans [CC21], chaque coefficient de g est codé sur 2 octets au lieu d'un octet et 5 bits pour des raisons de mise en œuvre. Cette implantation est réalisée sur un microcontrôleur 32 bits à cœur ARM Cortex-M4 avec 1 MB de mémoire Flash et 192 kB de SRAM. Notre banc d'attaque template sur la décapsulation de mceliece8192128 est donné dans la Figure 4.3. Nous avons récupéré les

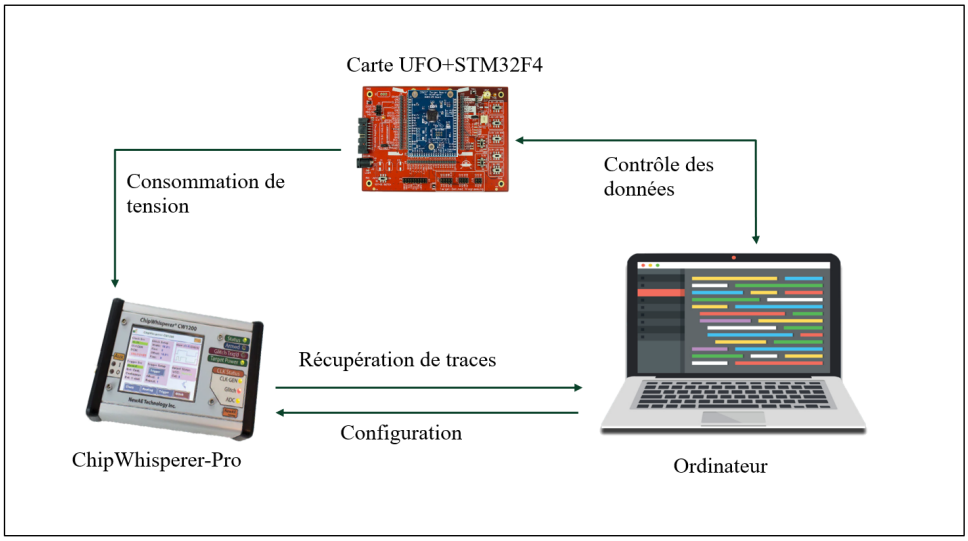

FIGURE 4.3 – Banc d'attaque template.

traces de consommation (ou acquisition) pendant le chargement du polynôme de Goppa à l'aide d'un ChipWhisperer-Pro (ou CW1200) [OC14] avec une carte UFO (ou CW308) contenant la cible (STM32F4). L'acquisition des traces est contrôlée par des scripts exécutés sur un PC. Une fois l'acquisition terminée, les traces sont stockées sur le disque dur du PC et on répète le processus de mesure en fonction du nombre de traces souhaité.

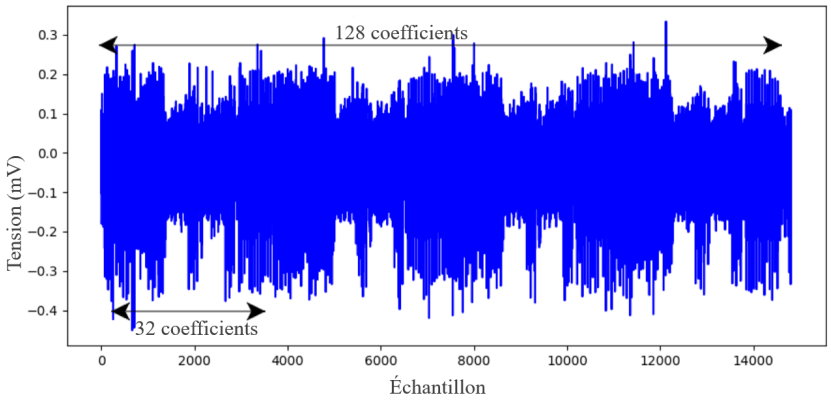

FIGURE 4.4 – Chargement des 128 coefficients de g.

La trace de consommation ($\tau = 16\ 000$) correspondant au chargement des 128 coefficients de g (polynôme unitaire) est présentée dans la Figure 4.4. Nous remarquons la présence de 4 motifs identiques liés à la représentation en «bitsliced» des données dans [CC21]. En effet, chaque motif représente le chargement de 32 coefficients de 16 bits dans 16 différents registres à 32 bits comme indiqué dans la fonction `irr_load_32x()` pour le chargement des 128 coefficients de g (Figure 4.5). Cette

```
static inline void irr_load_32x(uint32_t out[][GFBITS], const
unsigned char * in, int len )
{
 int i, j;
 uint32_t mat[16];
 uint16_t *mat16 = (uint16_t*)&mat[0];
 for(i=0;i<len;i+=32) {
    for(j=0;j<16;j++) mat16[j] = load_gf(in + (i+j)*2);
    for(j=0;j<16;j++) mat16[16+j] = load_gf(in + (i+j+16)*2);
    transpose_16x16( mat16 , mat16 );
    transpose_16x16( (mat16+16), (mat16+16) );
    bs16_to_bs32( out[i>>5] , mat16 , mat16+16 , GFBITS );
  }
}
```

FIGURE 4.5 – Fonction de chargement des coefficients de g.

fonction `irr_load_32x()` est constituée d'une boucle principale imbriquée avec deux boucles consécutives pour charger les 128 coefficients avec un pas de 32 (d'où les quatre motifs de la Figure 4.4). D'abord, les deux boucles consécutives chargent chacune 16 données de 16 bits. Ensuite, chacune de ces 16 données est transposée dans 16 registres de 32 bits. Enfin, une opération d'addition de ces deux groupes de coefficients est effectuée pour obtenir les 32 coefficients de 16 bits dans 16 registres de 32 bits différents. Un zoom effectué sur un motif de la Figure 4.4 montre la consommation de puissance

($\tau = 4\ 000$) correspondant à cette stratégie d'implantation en «bitsliced» (Figure 4.6). Nous allons maintenant effectuer une analyse de consommation de cette fonction de chargement `irr_load_32x()`. On vérifie si de petites variations de puissance peuvent être observées dans la trace en changeant la valeur d'un bit dans la donnée chargée. En d'autres termes, nous analysons comment la consommation de puissance dans la fonction `irr_load_32x()` dépend du bit de poids faible (Least Significant Bit ou LSB) d'une donnée de 16 bits (un coefficient) chargée en mémoire.

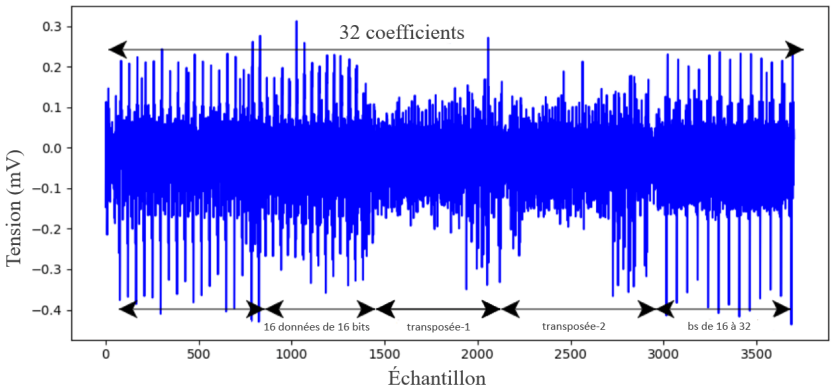

FIGURE 4.6 – Chargement de 32 coefficients de g.

En général, dans un circuit, les bus de communications sont pré-chargés à 0 et la consommation de puissance dépend des poids de Hamming des données manipulées. Ainsi, si la sortie en LSB produite par la fonction `irr_load_32x()` est égale à 1, alors, en théorie, le circuit devrait consommer plus d'énergie par rapport à une valeur de LSB égale à 0. Nous pouvons, dès lors, déduire que notre circuit cible admet un canal caché sur les poids de Hamming des coefficients de g pendant leur chargement. Pour mettre en évidence ces faibles variations, nous allons utiliser une technique d'analyse différentielle de consommation (Differential Power Analysis ou DPA) appelée différence des moyennes (Difference Of Means ou DOM) pour déterminer l'effet du LSB sur la consommation dans la fonction `irr_load_32x()`. La DOM entre chaque sous-ensemble (LSB=1 et LSB=0) permet de vérifier si l'hypothèse est significative ou non. Dans le cas d'une hypothèse significative, la DOM entre ces deux sous-ensembles mettra en évidence les variations de consommation par la présence de pics dans la trace DOM.

Dans notre expérimentation, nous avons tout d'abord effectué l'acquisition 10 000 traces de consommation ($\tau = 1\ 000$) pendant le chargement de 10 000 coefficients aléatoires de 16 bits. Nous obtenons ainsi 10 000 traces de consommation que l'on regroupe en deux sous-ensembles de 5 000 traces pour des LSB=1 et 5 000 traces pour des LSB=0 avant de calculer la moyenne de chaque sous-ensemble. Ensuite, la moyenne de chaque sous-ensemble est calculée point par point. Ainsi, la différence des moyennes peut alors être calculée en faisant simplement une soustraction entre les points du premier sous-ensemble (LSB=1) et ceux du second (LSB=0). Enfin, la trace DOM est présentée dans la

Figure 4.7.

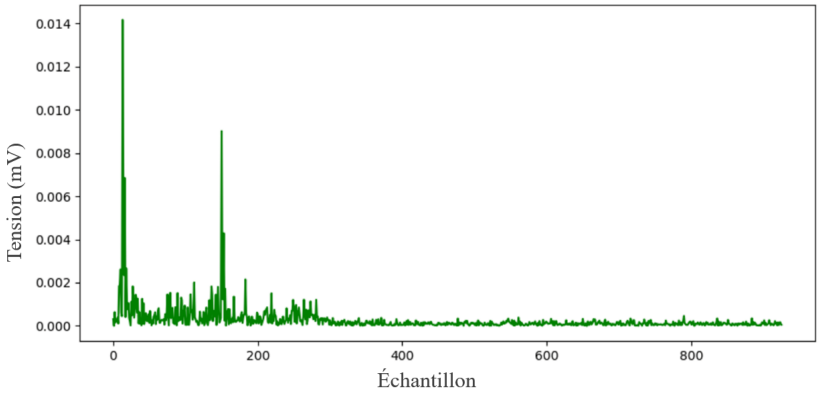

FIGURE 4.7 – DOM entre les traces pour LSB=1 et LSB=0.

Le premier pic significatif dans la trace DOM révèle que la consommation de puissance de notre circuit cible dépend du LSB. Ainsi, nous pouvons affirmer que cette fonction de chargement `irr_load_32x()` fuit de l'information pendant le chargement des coefficients de g. Le second pic, correspond à la représentation en «bitsliced» (transposition) des mêmes données chargées et donc cette technique conduit également à une fuite d'information. Il convient également de noter que cette expérience a été réalisée pour les 15 autres bits d'un coefficient et le résultat était identique. Nous allons exploiter cette fuite sur le chargement du premier coefficient de g (sans la transposition) pour réaliser une TA afin de déterminer son poids de Hamming.

4.2.3 Principe de notre attaque

Rappelons que dans *Classic McEliece* [Alb+20], la clef privée est constituée du polynôme $g(x)$ et du support $\mathcal{L}$. Comme chaque coefficient de g est représenté sur 16 bits dans [CC21], nous allons construire 17 templates correspondant aux poids de Hamming compris entre 0 à 16 (Remarque 4.2). L'un des points essentiels dans une TA est qu'elle nécessite également un grand nombre de traces pendant la phase de caractérisation. De plus, la probabilité d'avoir à la sortie de la fonction `irr_load_32x()` un coefficient avec les poids de Hamming 0 ou 16 est très faible ($1/256^2$) pour calculer une matrice de covariance et un vecteur de moyenne.

Remarque 4.3 *Lors de nos tests, nous avons utilisé le même circuit (STM32F4) en phase d'apprentissage comme clone (implantation sans clef privée) et comme circuit cible (implantation avec la clef privée) pour la phase d'attaque.*

Durant la phase d'apprentissage de notre attaque, nous avons fait l'acquisition de 350 000 traces lors du chargement des coefficients de g afin de construire nos 17 templates. En fait, le premier template est constitué des traces avec des coefficients de poids de

Hamming 0, le second avec des poids de Hamming 1 et ainsi de suite. Après avoir trié les traces en fonction de leur poids de Hamming, on cherche ensuite la «trace moyenne» pour chaque poids et on obtient ainsi 17 «traces moyennes». Enfin, on utilise ces traces pour chercher les points d'intérêts avec la méthode SOD détaillée ci-dessus. Nous avons obtenu un seul point d'intérêt (un seul pic) dans la trace SOD correspondant au 13ème échantillon dans la Figure 4.8.

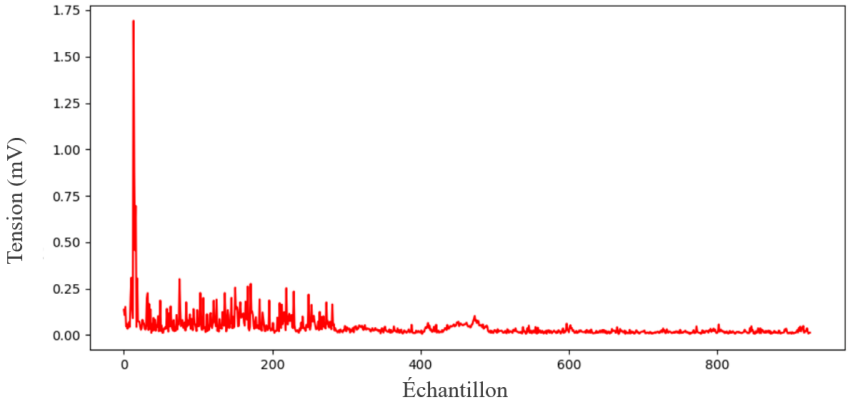

FIGURE 4.8 – SOD pour les 350 000 traces.

Remarque 4.4 *Dans le cas où il y aurait plus de pics dans la trace SOD, il faudra s'assurer que deux points d'intérêts consécutifs soient séparés par une certaine distance.*

Ce point d'intérêt peut être aussi identifié en superposant les 17 «traces moyennes» en poids de Hamming comme dans la Figure 4.9.

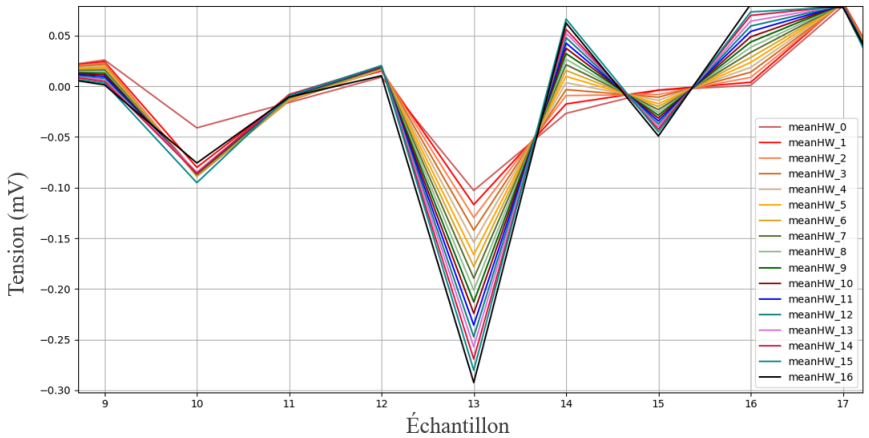

FIGURE 4.9 – Supperposition des «traces moyennes».

Avec ce nombre de points d'intérêts $N_{POI} = 1$, le vecteur de moyenne μ est un scalaire et la matrice de covariance $\boldsymbol{C}$ se réduit à la variance, ce qui va considérablement

accélérer les opérations dans la phase d'attaque. Ainsi, nous pouvons calculer la densité de probabilité de la distribution en utilisant l'Équation 4.2 pour la phase d'attaque.

Pour une décapsulation utilisant des textes chiffrés aléatoires, nous avons enregistré 17 traces de puissance sur le circuit cible pendant le chargement du premier coefficient de g. Ces 17 traces de puissance correspondent au chargement de 17 premiers coefficients de g avec différents poids de Hamming fixés compris entre 0 et 16. Pour chacune de ces traces, nous utilisons les 17 templates pour calculer la densité de probabilité (Équation 4.4), afin de déterminer lequel des templates s'adapte le mieux à la trace cible. On définit ainsi un classement par ordre croissant de densité de probabilité avant de vérifier si le template en poids de Hamming avec la plus grande probabilité correspond au poids de Hamming de la trace cible. Cette phase de matching avec une trace cible a pris environ 4 secondes sur un processeur à 8 cœurs fonctionnant à 3,6 GHz. Ainsi, après 1 000 tests, nous avons réussi à trouver les différents poids de Hamming (de 0 à 16) pour le chargement du premier coefficient de g sur le circuit cible lors de la décapsulation de mceliece8192128 avec un taux de succès de 99,86 %. Pour trouver les poids de Hamming des $t-1$ coefficients restants, il faut construire $t-1$ groupes de $m+1$ templates en poids de Hamming. Nous procéderons de la même manière que pour le premier coefficient, sauf que le nombre de POI ainsi que leurs positions vont dépendre de l'ordre des coefficients de g pendant leur chargement.

La connaissance des poids de Hamming des coefficients de g nous a permis d'établir de nouveaux résultats en cryptographie basée sur les codes correcteurs et d'améliorer notamment la complexité de la recherche exhaustive de ce polynôme sur $\mathbb{F}_{2^m}$.

4.3 Nouveaux résultats

Dans cette section, nous allons montrer comment cette information sur les poids de Hamming des coefficients de g permet d'améliorer la complexité de la recherche exhaustive de ce polynôme sur $\mathbb{F}_{2^m}$. Dans le schéma original de McEliece [McE78] décrit dans le Chapitre 2 (Sous-section 2.4.1, page 31), le nombre de polynômes g de degré t à coefficients dans $\mathbb{F}_{2^m}$ est $2^{mt}/t$ (Remarque 2.13, page 27). Pour rappel, une attaque sur le déchiffrement de ce schéma consiste à récupérer ce polynôme $g(x)$ et le support $\mathcal{L}$. La meilleure démarche connue à ce jour est celle utilisant l'algorithme de division du support (Support Splitting Algorithm ou SSA) présenté dans [LS01] par Loidreau et Sendrier. Le SSA est essentiellement composé de deux étapes décrites ci-dessous.

1. Trouver un polynôme irréductible et unitaire $g'(x)$ de degré t tel que le code $\Gamma(\mathcal{L}, g')$ soit équivalent au code de Goppa public $\Gamma(\mathcal{L}, g)$ (Définition 2.4).
2. Chercher la permutation $\mathcal{L}$ en utilisant l'algorithme SSA.

Le coût de cette attaque énumérative est donné par

$$\mathcal{O}(n^3 \varepsilon N_{irr}), \tag{4.5}$$

où n est la longueur du code de Goppa, ε est une constante et N_{irr} représente le nombre de polynômes irréductibles et unitaires. En d'autres termes N_{irr} est le cardinal de l'espace de recherche de ces polynômes irréductibles et unitaires sur $\mathbb{F}_{2^m}$ défini dans [LS01] par

$$N_{irr} \approx \frac{2^{m(t-3)}}{mt}. \tag{4.6}$$

En supposant que les polynômes irréductibles et unitaires ont une distribution uniforme sur $\mathbb{F}_{2^m}$ et en connaissant leurs poids de Hamming $w_H(g_i) = \delta_i$, on obtient le cardinal de l'espace de recherche suivant

$$\tilde{N}_{irr} = \frac{\prod_{i=0}^{t-1} \binom{m}{\delta_i}}{t}. \tag{4.7}$$

À notre connaissance, il n'existe pas d'algorithme permettant de construire un polynôme irréductible et unitaire à partir de la connaissance des poids de Hamming de ses coefficients sur $\mathbb{F}_{2^m}$. Par conséquent, trouver un tel polynôme en pratique revient à regarder dans l'espace de recherche des polynômes irréductibles et unitaires dont les coefficients sont dans l'ensemble des poids de Hamming connu grâce à l'attaque template. Ce qui aura pour effet de réduire considérablement la taille de l'espace de recherche de ces polynômes.

Dans le cas où ce polynôme a des coefficients binaires correspondant dans la méthode de Loidreau et Sendrier [LS01] au cas des clefs faibles, la connaissance des poids de Hamming des coefficients permet de le retrouver directement sans recherche exhaustive. En effet, tous les coefficients avec un poids de Hamming non nul ont leur valeur égale à 1 et inversement. Ainsi le cardinal de l'espace de recherche se réduit à 1 (Table 4.3). Par contre, dans [LS01] pour trouver ce polynôme binaire irréductible dans leur implantation avec les paramètres $m = 10$, $n = 1024$, et $t = 50$ il faudrait 500 ans. Dans le

TABLE 4.3 – Espace de recherche des clefs faibles.

Type de clef	Espace de recherche
Clefs faibles dans [LS01]	$\frac{2^{m(t-3)}}{mt}$
Clefs faibles avec $\delta_i = 1$ (ou 0)	1
clefs moins faibles avec $\delta_i = 1$	$\frac{m^t}{t}$
clefs moins faibles avec $\delta_i = 2$	$\frac{\binom{m}{2}^t}{t}$

cas où l'on souhaiterait accélérer l'algorithme de génération des clefs dans *Classic McEliece* (Algorithme 22), nous pouvons, par exemple, utiliser des polynômes irréductibles

et unitaires non-binaires de degré t avec des poids de Hamming fixés pour tous les coefficients : c'est le cas des «clefs moins faibles». Grâce à l'information sur les poids de Hamming des coefficients de g, il suffit d'effectuer une recherche exhaustive dans l'ensemble des poids fixés pour retrouver ce polynôme. Dans la table 4.3, nous donnons les expressions de la taille de l'espace de recherche pour deux cas particuliers de ce type de clefs. Nous augmentons ainsi l'ensemble des clefs faibles proposé dans [LS01] grâce au résultat de notre TA.

Dans le cas général de notre implantation cible, nous avons des polynômes irréductibles et unitaires non-binaires dont le degré d'extension m est supérieur à 8 où chaque élément est représenté sur 2 octets. Ainsi pour des valeurs $0 \leq \delta'_i \geq \delta_i$, on recherche des polynômes dont les coefficients ont un poids de Hamming δ'_i sur le premier octet et $\delta_i - \delta'_i$ sur les $m-8$ bits restants. Avec cette méthode, le cardinal $\tilde{N}_{irr}$ (Équation 4.7) de l'espace de recherche devient

$$\tilde{N}_{irr} = \frac{\prod_{i=0}^{t-1} \binom{8}{\delta'_i}\binom{m-8}{\delta_i-\delta'_i}}{t} \leq \frac{\binom{8}{4}^t \binom{m-8}{\lfloor \frac{m-8}{2} \rfloor}^t}{t}. \tag{4.8}$$

Dans le cadre de la démarche de Loidreau et Sendrier (1ère étape), le nombre de polynômes irréductibles et unitaires $g'(x)$ de degré t tel que les codes $\Gamma(\mathcal{L}, g')$ soient équivalents aux codes de Goppa $\Gamma(\mathcal{L}, g)$ est donné par

$$N'_{irr} = 2^m(2^m - 1)m. \tag{4.9}$$

Ainsi, le cardinal $\tilde{N}_{irr}$ de l'espace de recherche de polynômes irréductibles et unitaires connaissant les poids de Hamming des coefficients devient

$$\tilde{N}_{irr} \leq \frac{\binom{8}{4}^t \binom{m-8}{\lfloor \frac{m-8}{2} \rfloor}^t}{2^m(2^m - 1)mt}. \tag{4.10}$$

Ainsi, avec cette cardinalité, nous réduisons considérablement le coût de la recherche exhaustive du polynôme $g(x)$ sur $\mathbb{F}_{2^m}$. Comme le montre la table 4.4, la connaissance des poids de Hamming des coefficients du polynôme $g(x)$ nous a permis par exemple de diviser par 2^{441} la taille de l'espace de recherche dans la variante mceliece8192128. À ce jour, nous proposons une meilleure réduction de la complexité pour la recherche exhaustive du polynôme irréductible et unitaire de Goppa sur $\mathbb{F}_{2^m}$.

TABLE 4.4 – Réduction de la recherche exhaustive de g dans *Classic McEliece*.

Paramètres	$\log_2(N_{irr})$	$\log_2(\tilde{N}_{irr})$				$\log_2(N_{irr}/\tilde{N}_{irr})$
		$\delta_i = 1$	$\delta_i = 2$	$\delta_i = 3$	$\delta_i = m/2$	
mceliece348864	725	158	274	338	534	191
mceliece460896	1 199	251	425	521	871	328
mceliecee6688128	1 615	347	578	706	1 174	441
mceliece6960119	1 498	320	535	654	1 089	409
mceliece8192128	1 615	347	578	706	1 174	441

Notre TA réalisée pendant la décapsulation de *Classic McEliece* nous a permis sous certaines conditions de retrouver une partie de la clef privée. Nous avons aussi amélioré la complexité de la recherche exhaustive du polynôme $g(x)$ pour la première fois avec une SCA. Nous allons maintenant comparer notre attaque avec d'autres SCA sur ce schéma en termes de mise en œuvre pratique, de résultats obtenus et de contre-mesures proposés.

4.4 Comparaison

Nous rappelons que la décapsulation de *Classic McEliece* est grossièrement une opération de décodage par syndrome des codes de Goppa binaires qui est souvent la principale cible des attaques par canaux cachés.

Récemment, Guo *et al.* [GJJ22] ont proposé une SCA pour retrouver les clefs privées dans les implantations de *Classic McEliece* sur FPGA [WSN18] et sur ARM Cortex-M4 [CC21]. Leur stratégie d'attaque est de choisir des textes chiffrés spécifiques correspondant à des textes clairs ou erreurs $\boldsymbol{e}_i$ de poids de Hamming $w_H(\boldsymbol{e}_i) = 1$ pendant la décapsulation. Les auteurs exploitent une fuite d'information dans l'algorithme de la FFT lors du calcul du syndrome secret, avant d'utiliser un algorithme de classification basé sur l'apprentissage automatique pour déterminer le polynôme localisateur $\sigma(x)$. En réalité, l'attaque profilée leur permet de trouver ce polynôme secret parmi les 2^m possibilités afin d'obtenir un élément α_i du support $\mathcal{L}$. Enfin, avec cette information sur le support, ils conçoivent d'autres algorithmes pour récupérer le polynôme irréductible et unitaire $g(x)$ et d'obtenir ainsi la clef privée $(\mathcal{L}, g(x))$.

Le principal inconvénient de l'attaque de Guo *et al.* est la restriction sur les poids de Hamming des textes clairs pour retrouver la clef privée dans *Classic McEliece*. Tout d'abord, avec la forme systématique de $\boldsymbol{H}$, nous pouvons facilement détecter le problème des mauvais textes chiffrés avec des poids de Hamming inférieurs ou égaux à t avant même de commencer la décapsulation. Ensuite, durant le décodage dans *Classic*

McEliece, on sait que le polynôme localisateur d'erreurs $\sigma(x)$ est obtenu grâce à l'algorithme de Berlekamp-Massey. Ensuite, durant le décodage dans *Classic McEliece*, on sait que le polynôme localisateur d'erreurs $\sigma(x)$ est obtenu avec l'algorithme de Berlekamp-Massey. Cet algorithme prend en entrée le syndrome secret et retourne le polynôme $\sigma(x)$ de degré inférieur ou égale à t. Ainsi, cette condition sur le degré de ce polynôme permet aussi de détecter toute erreur intentionnelle, notamment sur le texte chiffré, pour arrêter le décodage et éviter l'attaque de Guo *et al.* Tous ces éléments indiquent que cette SCA pour trouver la clef privée dans *Classic McEliece* n'est pas réaliste.

La table 4.5 compare notre attaque à celle de Guo *et al.* sur la décapsulation de *Classic McEliece*. D'abord, nous n'avons aucune contrainte sur les poids de Hamming des textes chiffrés et avec moins de traces dans la phase de matching, nous avons récupéré le polynôme de Goppa dans le cas des clefs faibles de Loidreau et Sendrier (Table 4.3). Ensuite, pour la mise en œuvre pratique de notre attaque, nous avons simplement suivi les étapes de l'algorithme de décapsulation dans l'implantation de Chen et Chou [CC21], jusqu'au chargement des coefficients du polynôme de Goppa avant de mesurer la consommation de puissance correspondante.

TABLE 4.5 – Attaques profilées sur la décapsulation de *Classic McEliece*

SCA	Poids de Hamming de $\boldsymbol{e}$	Cible
Attaque de [GJJ22]	1	FFT et Berlekamp-Massey
Notre attaque	t	Fonction `irr_load_32x()`

Enfin, notre méthode a la propriété d'être étendue à d'autres étapes du décodage dans *Classic McEliece* ou appliquée dans d'autres schémas à base de code. En effet, le chargement d'un vecteur contenant les α_i (données sensibles) est également effectué juste avant la FFT additive pour l'évaluation du polynôme σ sur les α_i. Nous pouvons suivre les étapes d'exécution de l'algorithme de décapsulation pour reproduire notre attaque et récupérer les poids de Hamming des points d'évaluation α_i. Cela met encore en évidence que cette fonction de chargement dans [CC21] ne doit pas être utilisée pour des données sensibles.

Il convient également de rappeler que tout algorithme de décodage des codes de Goppa binaires utilise cette fonction (ou une fonction similaire) pour charger ces points d'évaluation. Ainsi, l'information sur les poids de Hamming des α_i, combinée avec nos résultats actuels sur les coefficients du polynôme de Goppa, améliorera grandement notre connaissance sur le secret dans *Classic McEliece*. Nous pouvons également utiliser ces informations sur les points d'évaluation avec la méthode de «Decoding with a hint» dans [KM22] pour récupérer directement le polynôme de Goppa.

Le succès des SCA repose sur l'existence de failles (fuites d'information sensibles ou secrètes) dans la mise en œuvre des algorithmes cryptographiques. Donc, pour se

prémunir contre de telles attaques, il faut des techniques qui agissent à la fois au niveau matériel qu'au niveau algorithmique afin de masquer ou d'effectuer un traitement aléatoire des opérations sensibles (chargement des coefficients de g) : on parle de contre-mesures. L'une des contre-mesures couramment utilisées aujourd'hui pour se protéger des SCA est le shuffling [CMJ22]. Cette technique consiste à randomiser les instructions dans la fonction cible afin de rendre la fuite aléatoire à chaque exécution. Ce qui rend une SCA difficilement réalisable. Cependant, dans notre attaque, le shuffling peut potentiellement rendre notre attaque plus difficile à réaliser, mais peut ne pas avoir d'impact sur la fonction cible, car le chargement des coefficients de g dans un ordre aléatoire ne change pas leurs poids de Hamming. D'ailleurs, une attaque physique est prévue contre une implantation intégrant cette contre-mesure sans nos travaux futurs. Notre attaque n'est pas certes générique, mais elle est réalisée contre l'implantation référence optimisée de *Classic McEliece* sur ARM-Cortex M4 [CC21], qui est également la cible d'attaques par canaux cachés dans de nombreux travaux récents [Cay+21 ; Col+22 ; GJJ22]. De plus, notre proposition est un premier pas vers des attaques plus puissantes et potentiellement génériques. En fait, dans le déchiffrement des schémas basés sur les codes, le chargement de données secrètes est effectué à plusieurs endroits. Si on arrive à isoler le moment exact où ces données sont chargées comme dans [CC21], notre TA reste applicable. D'une maniere generale, nous avons montré que la fonction `irr_load_32x()` dans [CC21] n'est pas adaptée pour charger des données sensibles en mémoire.

4.5 Conclusion

Dans ce chapitre, nous avons réalisé une attaque par canal caché (attaque template) contre l'implantation référence du KEM *Classic McEliece* (finaliste de la compétition du NIST) sur ARM Cortex-M4. Le canal caché correspond au chargement des coefficients du polynôme de Goppa en mémoire pendant la décapsulation. La phase de matching de notre attaque profilée est très rapide (environ 4 secondes) et nous a permis d'obtenir les poids de Hamming des coefficients du polynôme de Goppa qui est une partie importante de la clef privée. Ce résultat nous a permis tout d'abord de récupérer le polynôme de Goppa dans le cas des clefs faibles avec la méthode de Loidreau et Sendrier. Dans le cas des «clefs moins faibles», nous parvenons à trouver ce polynôme par recherche exhaustive avec une faible complexité. Nous avons ainsi augmenté l'ensemble des clefs faibles par rapport à la méthode de Loiderau et Sendrier. Ensuite, cette information sur les poids de Hamming des coefficients nous a également permis d'obtenir une meilleure réduction de la complexité pour la recherche exhaustive du polynôme de Goppa sur $\mathbb{F}_{2^m}$. Enfin, nous avons montré que notre attaque est réaliste (décapsulation d'un texte chiffré seulement) par rapport à d'autres attaques par canaux cachés sur *Classic McEliece*. Dans nos travaux futurs, nous espérons rendre notre attaque plus générique en l'appliquant à l'étape du chargement des points d'évaluation pendant la décapsulation afin d'améliorer les récents résultats des attaques physiques sur les schémas post-quantiques.

Chapitre 5

Attaque sur le problème du produit matrice-vecteur

Ce chapitre correspond à notre publication pour Information Security and Cryptology-ICISC, 2022 [Sec+23b] :

Key-Recovery by Side-Channel Information on the Matrix-Vector Product in Code-Based Cryptosystems
avec Pierre-Louis Cayrel, Idy Diop, Vlad-Florin Dragoi, Kalen Couzon, Brice Colombier et Vincent Grosso. ICISC 2022, LNCS.

Une attaque par canal caché (Side Channel Attack ou SCA) désigne une attaque informatique qui, sans remettre en cause la robustesse théorique des schémas cryptographiques, recherche et exploite des failles dans leur implantation, logicielle ou matérielle. La sécurité des schémas basés sur les codes correcteurs d'erreurs repose sur le problème du décodage par syndrome (Syndrome Decoding Problem ou SDP). Dans un contexte informel, le SDP consiste à trouver un vecteur $\boldsymbol{x}$ avec une certaine contrainte sur son poids de Hamming, connaissant un vecteur $\boldsymbol{s}$ tel que $\boldsymbol{H}\boldsymbol{x}^T = \boldsymbol{s}$. La meilleure stratégie pour résoudre le SDP est le décodage par ensemble d'information (Information Set Decoding ou ISD) de Prange [Pra62]. Malgré de nombreuses améliorations [Bec+12; LB88; Leo88; MMT11; MO15; Ste88], la complexité de l'ISD pour retrouver $\boldsymbol{x}$ reste encore exponentielle. Une solution possible pour résoudre le SDP est de le transformer en un problème de programmation linéaire en nombres entiers (Integer Linear Programming ou ILP) [Tan+10]. Récemment, Cayrel *et al.* [Cay+21] ont montré que le SDP devient plus facile à résoudre lorsque le syndrome $\boldsymbol{s}$ est calculé dans $\mathbb{N}$ avec une complexité polynomiale. Les auteurs ont réalisé une attaque par injection de fautes laser (attaque invasive) sur le produit $\boldsymbol{H}\boldsymbol{x}$ durant l'encapsulation (ou chiffrement) dans *Classic McEliece* [Alb+20] pour obtenir $\boldsymbol{s}$ dans $\mathbb{N}$ avant de définir le SDP en ILP comme dans [Tan+10] pour obtenir $\boldsymbol{x}$. L'attaque de Cayrel *et al.* sur le chiffrement de *Classic McEliece* est améliorée par Colombier *et al.* dans [Col+22]. Les auteurs ont proposé une

approche moins invasive en utilisant une SCA de type profilée au lieu de l'injection de fautes pour récupérer les coordonnées de $\boldsymbol{s}$ dans $\mathbb{N}$ avant d'utiliser une fonction «score» plus efficace que l'ILP pour retrouver $\boldsymbol{x}$ avec une complexité polynomiale.

Dans ce chapitre, nous présentons une attaque générique par textes chiffrés choisis sur le déchiffrement des schémas basés sur les codes correcteurs d'erreurs. Nous analysons en particulier l'opération secrète du produit matrice-vecteur dans le déchiffrement de Niederreiter à l'aide d'une SCA. Tout d'abord, nous introduisons le problème du produit matrice-vecteur (Matrix-Vector Product Problem ou MVPP), un nouveau formalisme en cryptographie basée sur les codes dont la résolution permet de trouver la clef privée (ou une partie). Le MVPP consiste à trouver une matrice privée $\boldsymbol{R}$ connaissant $\boldsymbol{s}^*$ dans $\mathbb{N}$ tel que $\boldsymbol{R}\boldsymbol{s}^T = \boldsymbol{s}^*$. Ensuite, nous appliquons l'attaque profilée dans [Col+22] durant la première étape du déchiffrement de Niederreiter pour obtenir $\boldsymbol{s}^*$ dans $\mathbb{N}$. Enfin, nous montrons que si nous parvenons à construire une matrice $\boldsymbol{S}^* = (\boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_{n-k}^*)$ sans erreurs, on obtient directement la matrice $\boldsymbol{R}$ sans résoudre un SDP.

5.1 Définitions

Cette section présente quelques définitions importantes pour la compréhension de ce chapitre.

Définition 5.1 (Problème du décodage par syndrome binaire) *Soient $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$, $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ et $t \in \mathbb{N}$, un entier. Le problème du décodage par syndrome (ou SDP) est de trouver un vecteur $\boldsymbol{x} \in \mathbb{F}_2^n$ de poids $w_H(\boldsymbol{x}) \leq t$ tel que $\boldsymbol{H}\boldsymbol{x}^T = \boldsymbol{s}$.*

Données.

$\boldsymbol{H}$:	*matrice de taille* $(n-k) \times n$ *dans* $\mathbb{F}_2$
$\boldsymbol{s}$:	*vecteur de* $\mathbb{F}_2^{n-k}$
t :	*entier*

Problème.

Existe-t-il un vecteur $\boldsymbol{e} \in \mathbb{F}_2^n$ de poids $w_H(\boldsymbol{e}) \leq t$ tel que : $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}$?

Trouver un tel vecteur revient à chercher les t colonnes de $\boldsymbol{H}$ dont leur somme (opération de XOR) permet d'obtenir $\boldsymbol{s}$ (Figure 2.1, page 21). C'est un problème qui est $\mathcal{NP}$-complet [BMVT78] et les meilleurs algorithmes de décodage par syndrome sont encore exponentiels. Cependant, Cayrel *et al.* [Cay+21] ont montré que ce problème devient plus facile à résoudre si le syndrome $\boldsymbol{H}\boldsymbol{x}^T$ est calculé dans $\mathbb{N}$ au lieu de $\mathbb{F}_2$: c'est le problème du décodage par syndrome dans $\mathbb{N}$ (ou $\mathbb{N}$-SDP).

Définition 5.2 (Problème du décodage par syndrome dans $\mathbb{N}$) *Soient $\boldsymbol{H} \in \mathcal{M}_{n-k,n}(\mathbb{N})$, une matrice de contrôle avec $\boldsymbol{h}_{i,j} \in \{0,1\}$ pour tout i, j, $\boldsymbol{s} \in \mathbb{N}^{n-k}$, et t, un entier. Le problème du décodage par syndrome dans $\mathbb{N}$ (ou $\mathbb{N}$-SDP) consiste à trouver $\boldsymbol{x} \in \mathbb{N}^n$ (avec $\boldsymbol{x}_i \in \{0,1\}$) de poids $w_H(\boldsymbol{x}) \leq t$ tel que $\boldsymbol{H}\boldsymbol{x}^T = \boldsymbol{s}$.*

Données.

$\boldsymbol{H}$:	*matrice de taille $(n-k) \times n$ à coefficients dans $\mathbb{F}_2$*
$\boldsymbol{s}$:	*vecteur de longueur $(n-k)$ à coordonnées dans $\mathbb{F}_2$*
t :	*entier*

Problème.

Existe-t-il un vecteur $\boldsymbol{x} \in \mathbb{F}_2^n$ de poids $w_H(\boldsymbol{x}) \leq t$ tel que : $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s} \in \mathbb{N}$?

Remarque 5.1 *Dans le $\mathbb{N}$-SDP, $\boldsymbol{H}$ et $\boldsymbol{x}$ sont binaires comme dans le SDP, alors que $\boldsymbol{s}$ est entier. Seul le produit matrice-vecteur $\boldsymbol{H}\boldsymbol{x}^T$ change, et donc le résultat par rapport au SDP binaire.*

En raison des similitudes avec le SDP, on peut tenter de résoudre le $\mathbb{N}$-SDP en utilisant les techniques de la théorie des codes.

1. **Résoudre un système linéaire**
   On sait que le système linéaire $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{s}$ comportant $n-k$ équations et n inconnues peut être résolu directement. Cependant, il y a k variables libres, et donc 2^k solutions possibles, car $\boldsymbol{x} \in \{0,1\}$ et pour chaque instance, on calcule le poids de Hamming de $\boldsymbol{x}$, et on s'arrête lorsque $w_H(\boldsymbol{x}) \leq t$. Étant donné que la valeur du paramètre k est généralement très élevée dans les schémas basés sur le code, cette procédure n'est pas réalisable en pratique.
2. **Chercher des combinaisons dans $Supp(\boldsymbol{H})$**
   On choisit des sous-ensembles de $\boldsymbol{s}_i$ éléments de $Supp(\boldsymbol{H})$ pour des valeurs de i croissantes, jusqu'à trouver les bonnes combinaisons. Cette solution peut être encore optimisée en sélectionnant un sous-ensemble à partir d'un ensemble plus petit à chaque itération et en supprimant les positions précédemment sélectionnées de l'ensemble mis à jour (on effectue une sorte de recherche exhaustive). Pour cela, on rejette les positions dans l'intersection du support et de l'union des supports précédents. Cependant, avec cette méthode, la complexité temporelle est dominée par le produit de coefficients binomiaux asymptotiquement exponentiel en t.
3. **Faire un ISD**
   On choisit aléatoirement k des n coordonnées dans $\boldsymbol{H}$ aléatoirement dans l'espoir qu'elles soient toutes sans erreurs. Ensuite, on essaie de récupérer $\boldsymbol{x}$ en résolvant

un système linéaire $k \times k$ sur $\mathbb{F}_2$ (Remarque 2.9). Cette méthode donne également une complexité exponentielle.

Étant donné que toutes les méthodes ci-dessus, ne sont pas réalisables en pratique (paramètres n et k élevés), Cayrel *et al.* [Cay+21] ont proposé de formuler le $\mathbb{N}$-SDP comme un problème d'optimisation avant de chercher la solution optimale $\boldsymbol{x}^*$ à l'aide de l'ILP. Cette approche a permis de trouver le message $\boldsymbol{x}$ avec une complexité polynomiale.

Nous allons maintenant nous inspirer de la définition du problème du décodage par syndrome dans $\mathbb{N}$, qui permet de trouver une solution en temps polynomial, pour aboutir au nouveau formalisme suivant.

Définition 5.3 (Problème du produit matrice-vecteur binaire) *Soient $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$, un vecteur de poids quelconque et $\boldsymbol{s}^* \in \mathbb{N}$. Le problème du produit matrice-vecteur (ou MVPP) consiste à trouver une matrice $\boldsymbol{R} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ tel que $\boldsymbol{R}\boldsymbol{s}^T = \boldsymbol{s}^*$.*

Données.

$\boldsymbol{s}$: *vecteur de $\mathbb{F}_2^{n-k}$*

$\boldsymbol{s}^*$: *vecteur de longueur $(n-k)$ dont le poids de Hamming est dans $\mathbb{N}$*

Problème.

Existe-t-il une matrice $\boldsymbol{R}$ de taille $(n-k) \times (n-k)$ dans $\mathbb{F}_2$ tel que : $\boldsymbol{R}\boldsymbol{s}^T = \boldsymbol{s}^*$?

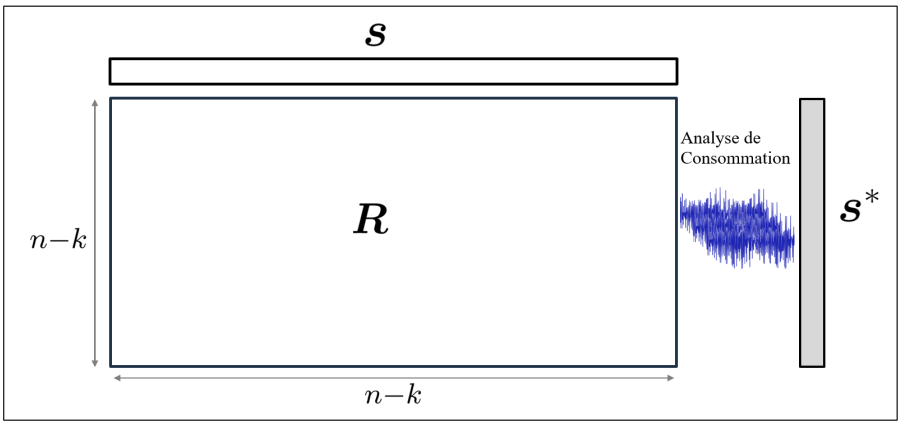

FIGURE 5.1 – Problème du produit matrice-vecteur ou MVPP.

L'intérêt du MVPP (Figure 5.1) repose sur le fait que, si nous parvenons à trouver $\boldsymbol{s}^* \in \mathbb{N}^{n-k}$ à l'aide d'une analyse de consommation de l'opération $\boldsymbol{R}\boldsymbol{s}^T$ comme dans [Col+22], nous pouvons obtenir la matrice privée $\boldsymbol{R}$ en résolvant un système linéaire.

Comme nous l'avons mentionné ci-dessus, le produit matrice-vecteur est une opération secrète utilisée dans le déchiffrement pour la plupart des schémas à base de codes, notamment dans Niederreiter [Nie86]. Pour rappel, dans le déchiffrement du schéma de Niederreiter (Algorithme 9, page 35), la première étape consiste à faire la multiplication de l'inverse de la matrice privée $\boldsymbol{Q}^{-1} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ avec le syndrome reçu (ou texte chiffré) $\boldsymbol{s}$. Appliquer le MVPP sur cette étape revient à chercher une matrice $\boldsymbol{S}^* = (\boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_{n-k}^*)$ tel que $\boldsymbol{R} = \boldsymbol{S}^*\boldsymbol{S}^{-1}$ avec $\boldsymbol{R} = \boldsymbol{Q}^{-1}$ et $\boldsymbol{S} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n-k})$, une matrice inversible obtenue grâce à des textes chiffrés aléatoires.

Algorithme 26 Produit matrice-vecteur packed dans [Alb+20]

Entrée: $\boldsymbol{R}$ et $\boldsymbol{s}$.
Sortie: $\boldsymbol{s}'$.

pour $r \leftarrow 0$ jusqu'à $((n-k)/8 - 1)$ **faire**
    $\boldsymbol{s}'_r = 0$.
fin pour
pour $r \leftarrow 0$ jusqu'à $(n-k-1)$ **faire**
    $b = 0$.
    pour $c \leftarrow 0$ jusqu'à $(n/8 - 1)$ **faire**
        b ^= $\boldsymbol{R}_{[r,c]}$ & $\boldsymbol{s}_c$.
        // Multiplication et addition.
    fin pour
    b ^= b>>4.
    b ^= b>>2.
    b ^= b>>1.
    // XOR suivi de décalages.
    b &= 1.
    // Extraction du LSB.
    $\boldsymbol{s}'_{\lfloor r/8 \rfloor}$ |= b>>$(r \mod 8)$.
    // Mise en paquet.
fin pour
Retourner $\boldsymbol{s}'$.

5.2 Détails de notre attaque

Le problème du produit matrice-vecteur est un nouveau formalisme dont la résolution permet d'obtenir directement la matrice privée $\boldsymbol{R} = \boldsymbol{Q}^{-1}$ dans le déchiffrement de Niederreiter. La première étape de notre attaque consiste à trouver $\boldsymbol{s}^* \in \mathbb{N}^{n-k}$ avec une analyse de consommation. Pour cela, nous utilisons l'implantation du produit matrice-vecteur $\boldsymbol{Rs}$ avec les paramètres $n = 6\,960$, $k = 5\,413$ et $t = 119$ correspondant à un niveau de sécurité V (Table 2.5, page 41), sur un cœur ARM Cortex-M4 comme le recommande le NIST (National Institute of Standards and Technology). La multiplication

$\boldsymbol{Rs}$ est effectuée en utilisant la technique du produit matrice-vecteur «packed» comme dans l'implantation du chiffrement dans *Classic McEliece* [Alb+20]. Dans cette méthode (Algorithme 26) les bits consécutifs des lignes de la matrice $\boldsymbol{R}$ sont regroupés dans un octet pour mieux exploiter la capacité des mots machines. Ce qui va entraîner une modification des dimensions de $\boldsymbol{R}$, du vecteur $\boldsymbol{s}$ et du résultat de leur produit $\boldsymbol{s}'$. Ainsi, la matrice $\boldsymbol{R}$ est maintenant composée de $n-k$ lignes et $n/8$ colonnes, les vecteurs $\boldsymbol{s}$ et $\boldsymbol{s}'$ ont des entrées respectives de $n/8$ et $(n-k)/8$. Pour réaliser le produit matrice-vecteur $\boldsymbol{Rs}$, on utilise d'abord, une variable b (sur un octet) pour stocker le résultat intermédiaire de la multiplication et de l'addition. Ensuite, une série d'opérations de OU-exclusif entre la moitié inférieure et supérieure de b est effectuée avant de faire des décalages de 4, 2 et 1 position à chaque fois. Enfin, le bit de poids faible (Least Significant Bit ou LSB) est extrait avant d'être placé à la bonne position dans l'octet $\boldsymbol{s}'$. Notre attaque est également applicable dans la version «schoolbook» du produit matrice-vecteur (Algorithme 27).

Algorithme 27 Produit matrice-vecteur schoolbook

Entrée: $\boldsymbol{R}$ et $\boldsymbol{s}$.
Sortie: $\boldsymbol{s}'$.

1: **pour** $r \leftarrow 0$ jusqu'à $(n-k-1)$ **faire**
2: $\quad \boldsymbol{s}'_r = 0$.
3: **fin pour**
4: **pour** $r \leftarrow 0$ jusqu'à $(n-k-1)$ **faire**
5: $\quad$ **pour** $c \leftarrow 0$ jusqu'à $(n-1)$ **faire**
6: $\quad\quad \boldsymbol{s}'_r \ \text{^}= \boldsymbol{R}_{[r,c]} \ \& \ \boldsymbol{s}_c$.
$\quad\quad$ // Multiplication et addition.
7: $\quad$ **fin pour**
8: **fin pour**
9: **Retourner** $\boldsymbol{s}'$.

5.2.1 Mesure des traces de consommation

L'implantation de la multiplication $\boldsymbol{Rs}$ est réalisée sur un microcontrôleur 32 bits intégrant un cœur ARM Cortex-M4 avec 1 MB de mémoire Flash pour contenir toute la matrice $\boldsymbol{R}$ et 192 kB de SRAM. Notre banc d'attaque est présenté dans la Figure 5.2. Nous avons récupéré les traces de consommation du produit matrice-vecteur $\boldsymbol{Rs}$ de manière instantanée à l'aide d'un oscilloscope numérique avec une bande passante de 50 MHz à 6 GHz. L'acquisition des traces pendant la multiplication est contrôlée par des scripts exécutés sur un PC avant de les stocker sur son disque dur. La trace de consommation présentée dans la Figure 5.3 correspond au produit des 10 premières lignes de $\boldsymbol{R}$ avec $\boldsymbol{s}$. Pour la suite, nous allons considérer que la première composante du produit $\boldsymbol{Rs}$ afin de déterminer $\boldsymbol{s}^*$ dans $\mathbb{N}$. En réalité, on cherche le poids de Hamming de $\boldsymbol{s}^*$ dans $\mathbb{N}$ avec une démarche similaire à celle détaillée dans [Col+22] sur le chiffrement de *Classic McEliece*.

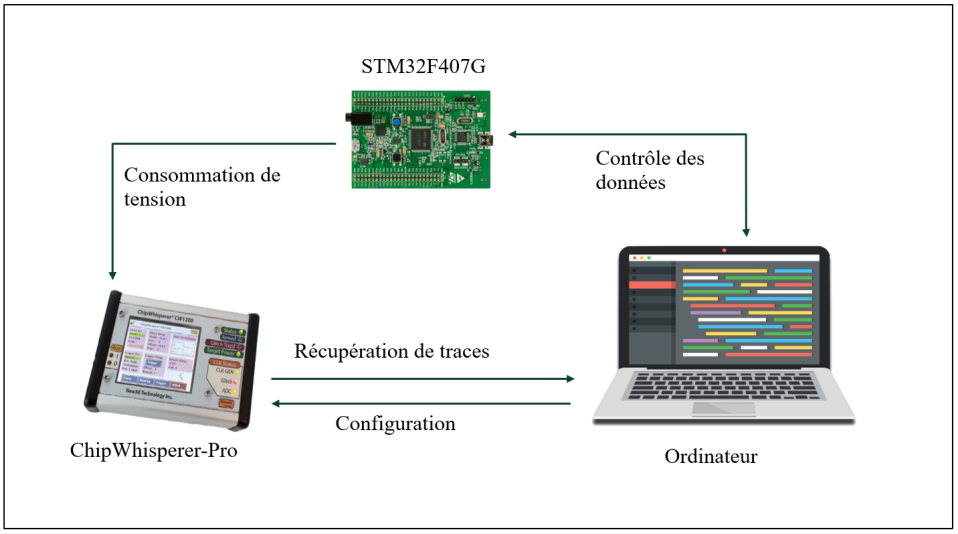

FIGURE 5.2 – Banc d'attaque sur le produit $\boldsymbol{Rs}$

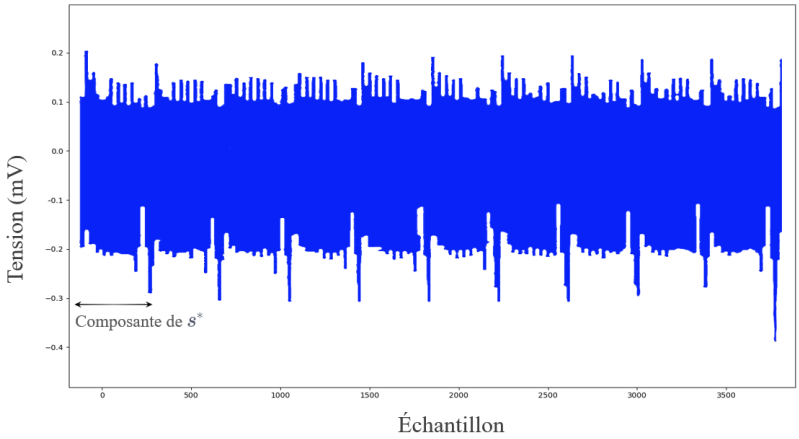

FIGURE 5.3 – Trace de consommation du produit $\boldsymbol{Rs}$.

5.2.2 Récupération de s^* dans $\mathbb{N}$

L'approche dans [Col+22] est une attaque profilée (Sous-section 1.2.2, page 15) basée sur l'apprentissage automatique (ou machine learning) utilisant l'algorithme du random forest [Bre01]. C'est un algorithme de classification (ou classificateur) constitué d'un ensemble d'arbres de décision indépendants dont et le résultat de leurs décisions individuelles est combiné par un vote à la majorité. Comme dans une attaque profilée classique, l'algorithme est entrainé avec des échantillons d'entrées (τ) associés à des étiquettes ($\boldsymbol{t}_r$) durant la phase d'apprentissage, avant d'évaluer sa précision avec des échantillons et étiquettes de tests ($\boldsymbol{t}_{test}$) durant la phase d'attaque. Le random forest est une méthode de

classification plus légère que celles de l'apprentissage profond et donne de bons résultats [HGG20].

Durant la phase d'apprentissage, nous avons enregistré qu'une seule trace $\boldsymbol{t}_r$ avec τ échantillons. Cette trace $\boldsymbol{t}_r$ correspond à la consommation de puissance lors du déchiffrement de Niederreiter (ligne 1 dans Algorithme 9, page 35) avec des entrées $\boldsymbol{R}$ et $\boldsymbol{s}$ aléatoires. Cependant, le nombre d'échantillons τ dans la trace $\boldsymbol{t}_r$ est très élevé, lié à la taille de la matrice $\boldsymbol{R} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ qui est très grande pour les paramètres de notre implantation. Ainsi, la trace $\boldsymbol{t}_r$ est pré-traitée comme dans [Col+22] pour réduire sa dimension avant de la charger dans le classificateur.

Durant la phase d'attaque, une deuxième trace $\boldsymbol{t}_{test}$ est enregistrée avec une matrice $\boldsymbol{R}$ fixée pour évaluer l'entraînement du classificateur. Pour chacune des deux traces, nous avons utilisé les poids de Hamming de la variable intermédiaire b (Algorithme 26) comme étiquettes pour le classificateur.

Après 10 tests indépendants, nous avons réussi à retrouver les poids de Hamming des valeurs intermédiaires b du produit matrice-vecteur $\boldsymbol{Rs}$ avec une précision de 98,65 %. Ainsi, nous pouvons obtenir les entrées de $\boldsymbol{s}^*$ dans $\mathbb{N}$.

5.2.3 Déroulement de l'attaque

Notre attaque par textes chiffrés choisis sur le déchiffrement de Niederreiter et de manière générique sur le déchiffrement des schémas basés sur les codes correcteurs d'erreurs, consiste essentiellement en quatre étapes.

- **Étape 1 : Construire la matrice $\boldsymbol{S}$**
  On choisit $n-k$ textes chiffrés $\boldsymbol{s}_i$ dans $\mathbb{F}_2$ qui sont linéairement indépendants. On dispose ensuite les $\boldsymbol{s}_i$ en colonnes jusqu'à obtenir une matrice $\boldsymbol{S} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n-k})$ de taille $(n-k) \times (n-k)$. Il faut s'assurer que $\boldsymbol{S}$ est inversible avant de passer aux étapes suivantes.
- **Étape 2 : Récupérer les entrées de $\boldsymbol{s}^*$ dans $\mathbb{N}$**
  Pour chacun des $\boldsymbol{s}_i$, on récupère les coordonnées des $\boldsymbol{s}_i^* = \boldsymbol{Rs}_i$ dans $\mathbb{N}$ grâce à l'attaque profilée dans la Sous-section 5.2.2. On construit également une matrice $\boldsymbol{S}^* = (\boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_{n-k}^*)$ de taille $(n-k) \times (n-k)$.
- **Étape 3 : Résoudre le système linéaire**
  Après ces deux étapes, on obtient le système linéaire suivant

$$\boldsymbol{RS} = \boldsymbol{S}^*. \tag{5.1}$$

  Comme la matrice $\boldsymbol{S}$ est inversible, on a

$$\boldsymbol{R} = \boldsymbol{S}^*\boldsymbol{S}^{-1}. \tag{5.2}$$

- **Étape 4 : Retrouver la matrice privée**
  On récupère la matrice privée $\boldsymbol{Q}$ dans le schéma de Niederreiter en cherchant l'inverse de $\boldsymbol{R}$.

L'attaque telle que présentée ici permet de trouver directement la matrice privée $\boldsymbol{R}$. Toutefois, dans l'étape 2, nous pouvons soulever deux questions essentielles.

1. Comment savoir si la matrice $\boldsymbol{R}$ contient des erreurs ?
2. Si c'est le cas, comment peut-on les corriger et retrouver $\boldsymbol{R}$?

Nous discuterons de la deuxième question, plus délicate, dans la Section 5.3. Pour le premier point, supposons que l'on obtienne une matrice $\boldsymbol{S}^{**}$ au lieu de $\boldsymbol{S}^{*}$ dans l'**Étape 2** de l'attaque. Nous avons donc

$$\boldsymbol{R}' = \boldsymbol{S}^{**}\boldsymbol{S}^{-1} \tag{5.3}$$

et

$$\boldsymbol{S}^{**} = \boldsymbol{S}^{*} + \boldsymbol{E} \tag{5.4}$$

où $\boldsymbol{E}$ est une matrice d'erreur.

Comment peut-on distinguer $\boldsymbol{R}$ de $\boldsymbol{R}'$? On sait que les informations disponibles sur la matrice privée $\boldsymbol{R}$ sont peu nombreuses. Néanmoins, nous savons que $\boldsymbol{R}$ est une matrice inversible dans $\mathbb{F}_2$. Donc il faut d'abord regarder directement ses coefficients avant de vérifier son déterminant pour détecter les erreurs dans $\boldsymbol{R}$.

En résumé, nous savons que notre attaque sur le MVPP (définition 5.3) permet de trouver directement la matrice privée $\boldsymbol{R}$ dans le cas où il n'y a pas d'erreurs à l'**Étape 2**. Dans le cas contraire, nous savons aussi comment les détecter dans $\boldsymbol{R}$. En outre, nous proposons d'optimiser notre attaque en minimisant les risques d'erreurs dans $\boldsymbol{S}^{*}$ à l'**Étape 2** et accélérer ainsi les opérations. En effet, à l'**Étape 1**, nous pouvons choisir judicieusement des textes chiffrés $\boldsymbol{s}_i$ de faibles poids de Hamming (par exemple $w_H(\boldsymbol{s}_i) = 1$) afin d'obtenir des mots réguliers. Cela va abord réduire considérablement les risques d'erreurs lors de l'analyse de consommation. Ensuite, nous aurons une matrice $\boldsymbol{S}$ de la forme $\boldsymbol{S} = \boldsymbol{I}_{n-k}$. Ce qui va accélérer la résolution du système linéaire (Équation 5.1) sans avoir à calculer son inverse $\boldsymbol{S}^{-1}$. Enfin, dans un scénario d'attaque où la victime refuserait de déchiffrer $n-k$ textes chiffrés (c'est souvent le cas en pratique), le choix des mots réguliers de faibles poids (implique $\boldsymbol{S} = \boldsymbol{I}_{n-k}$) permet d'éviter ce genre de problème pour la réalisation de notre attaque.

Nous allons maintenant voir comment corriger les erreurs dans la matrice $\boldsymbol{S}^{*}$.

5.3 Correction des erreurs

Dans cette section, nous allons essayer de répondre à la question 2 évoquée ci-dessus et de monter que nous pouvons effectivement corriger les erreurs dans la matrice $\boldsymbol{S}^{*}$ dans certains cas. Nous considérons le cas où la matrice $\boldsymbol{S}^{*}$ contient des erreurs $\boldsymbol{E}$ dans l'**Étape 2** (Équation 5.4). Nous avons les hypothèses suivantes

1. La matrice $\boldsymbol{E}$ a des coefficients binaires.
2. La matrice $\boldsymbol{E}$ a, au plus, un 1 sur chaque ligne.

Ces deux hypothèses ne sont pas restrictives et nous verrons qu'on peut en déduire un cas général. On suppose que $\boldsymbol{E}$ peut être contrôlée dans une certaine mesure, c'est-à-dire $\boldsymbol{S}^{**}$ ne diffère pas «trop» de $\boldsymbol{S}^{*}$.

D'après les hypothèses ci-dessus, il existe deux ensembles finis $\mathcal{I}$ et $\mathcal{J}$ tels que

$$\boldsymbol{E} = \sum_{(i,j)\in\mathcal{I}\times\mathcal{J}} \boldsymbol{E}_{i,j} \tag{5.5}$$

où $\boldsymbol{E}_{i,j}$ est une matrice carrée d'ordre $n-k$ dans laquelle tous les coefficients sont nuls sauf celui de la ligne i et la colonne j qui vaut 1. Soit le lemme suivant

Lemme 5.1 *Soient $1 \leq a, b \leq n$ et $\boldsymbol{M} = (\boldsymbol{m}_{i,j})$, une matrice carrée d'ordre $n-k$. On a*

$$\boldsymbol{E}_{a,b}\boldsymbol{M} = \begin{pmatrix} 0 & \dots & 0 \\ 0 & \dots & 0 \\ \vdots & \dots & \vdots \\ m_{b,1} & \dots & m_{b,n-k} \\ \vdots & \dots & \vdots \\ 0 & \dots & 0 \\ 0 & \dots & 0 \end{pmatrix} \leftarrow \textit{a-ème ligne}$$

En d'autres termes : $[\boldsymbol{E}_{a,b}\boldsymbol{M}]_{i,j} = \begin{cases} 0 & \text{si } i \neq a \\ \boldsymbol{m}_{b,j} & \text{si } i = a \end{cases}$

Pour retrouver $\boldsymbol{R}$ malgré les erreurs dans $\boldsymbol{S}^{*}$, on a

$$\boldsymbol{R}\boldsymbol{S} = \boldsymbol{S}^{*} + \boldsymbol{E} = \boldsymbol{S}^{*} + \sum_{(i,j)\in\mathcal{I}\times\mathcal{I}} \boldsymbol{E}_{i,j},$$

et donc

$$\boldsymbol{R} = \boldsymbol{S}^{*}\boldsymbol{S}^{-1} + \boldsymbol{E}\boldsymbol{S}^{-1} = \boldsymbol{S}^{*}\boldsymbol{S}^{-1} + \sum_{(i,j)\mathcal{I}\times\mathcal{I}} (\boldsymbol{E}_{i,j}\boldsymbol{S}^{-1}).$$

De ce qui précède, on déduit le théorème suivant

Théorème 5.1 *Pour tout $i \in [1, n-k]$, il existe $j \in [1, n-k]$ et $\varepsilon \in \{0, 1\}$ tel que $\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] - \varepsilon\boldsymbol{S}^{-1}[j]$ soit binaire et $\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] - \varepsilon\boldsymbol{S}^{-1}[j] = \boldsymbol{R}[i]$.*

Preuve 5.1 *Supposons $|\mathcal{I}\times\mathcal{J}| = 1$ et on pose $\mathcal{I}\times\mathcal{J} = (a, b)$. on a alors*

$$\boldsymbol{R} = \boldsymbol{S}^{*}\boldsymbol{S}^{-1} + \boldsymbol{E}_{a,b}\boldsymbol{S}^{-1}.$$

D'après le Lemme 5.1, seule la ligne a dans la matrice $\boldsymbol{E}_{a,b}\boldsymbol{S}^{-1}$ est non nulle.

Alors pour tout $i \neq a$, on a d'abord

$$\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] = \boldsymbol{R}[i],$$

et il suffit donc de prendre $\varepsilon = 0$ et j quelconque.

Ensuite, d'après toujours le Lemme 5.1,

$$\boldsymbol{E}_{a,b}\boldsymbol{S}^{-1}[a] = \boldsymbol{S}^{-1}[b],$$

et il suffit cette fois-ci de prendre $\varepsilon = 1$ et $j = b$.

Soient $(a,b) \in \mathcal{I}\times\mathcal{J}$ et $(c,d) \in (\mathcal{I}\times\mathcal{J})\setminus(a,b)$. Si $|\mathcal{I}\times\mathcal{J}| \geq 2$, la deuxième hypothèse implique $c \neq a$.

Enfin, on a $\boldsymbol{E}_{c,d}\boldsymbol{S}^{-1}[a] = 0$, d'après le Lemme 5.1. Donc

$$\boldsymbol{S}^{*}\boldsymbol{S}^{-1}[a] + \sum_{(i,j)\in\mathcal{I}\times\mathcal{J}} (\boldsymbol{E}_{i,j}\boldsymbol{S}^{-1}[a]) = \boldsymbol{S}^{*}\boldsymbol{S}^{-1}[a] + \boldsymbol{E}_{a,b}\boldsymbol{S}^{-1}[a].$$

Nous avons au plus une contribution pour chaque ligne et on a

1. *Si $i \notin \mathcal{I}$, alors $\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] = \boldsymbol{R}[i]$, il suffit de prendre $\varepsilon = 0$ et j quelconque.*
2. *Si $i \in \mathcal{I}$, alors il existe j tel que $(i,j) \in \mathcal{I}\times\mathcal{J}$ et on a $\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] = \boldsymbol{R}[i] + \boldsymbol{S}^{-1}[j]$. Donc, il suffit de prendre le même j précédent et $\varepsilon = 1$.*

Ainsi, on voit que dans certains cas, il est possible de trouver $\boldsymbol{R}$ malgré des erreurs dans $\boldsymbol{S}^{*}$. Les quatre étapes pour corriger les erreurs dans $\boldsymbol{R}'$ sont décrites dans l'Algorithme 28 ci-dessous.

Algorithme 28 Retrouver $\boldsymbol{R}$ à partir de $\boldsymbol{R}'$

1: On suppose que nous pouvons identifier la ou les lignes erronées dans $\boldsymbol{R}'$ ($\boldsymbol{R}'$ n'est pas binaire).
2: Soit $\boldsymbol{\ell}_i$, une ligne erronée de $\boldsymbol{R}'$. On soustrait de $\boldsymbol{\ell}_i$ les lignes de la matrice $\boldsymbol{S}^{-1}$ et on conserve celles qui sont binaires.
3: Ainsi, on obtient une liste de candidats possibles pour $\boldsymbol{R}$, et pour chaque matrice candidate, on calcule son déterminant pour vérifier son inversibilité dans $\mathbb{F}_2$.
4: Dans le cas où cette liste est réduite à un seul élément qui vérifie la condition dans l'étape précédente, on obtient directement $\boldsymbol{R}$.

Un exemple simplifié de notre attaque est fourni en Annexe B.
Nous allons maintenant alléger les hypothèses en supprimant la deuxième. Ainsi, d'après le Théorème 5.1, nous pouvons déduire le cas général de notre approche.

Corollaire 5.1 *Soient $i \in [1, n-k]$ et r_i, le poids de Hamming du vecteur $\boldsymbol{E}[i]$. Il existe deux suites $(j_k)_{1\leq k\leq r_i}$ et $(\varepsilon_k)_{1\leq k\leq r_i}$ à éléments distincts deux à deux, tel que*

$$\boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] - \sum_{k=1}^{r_i} \varepsilon_k \boldsymbol{S}^{-1}[j_k] \text{ soit binaire et } \boldsymbol{S}^{**}\boldsymbol{S}^{-1}[i] - \sum_{k=1}^{r_i} \varepsilon_k \boldsymbol{S}^{-1}[j_k] = \boldsymbol{R}[i].$$

Preuve 5.2 *Nous utilisons les mêmes notations que précédemment. Soient* $|\mathcal{I} \times \mathcal{J}| \geq 2$ *et* $(a, b) \in \mathcal{I} \times \mathcal{J}$.

On cherche à dénombrer les couples (a, t) *avec* $t \in \mathcal{J}$. *Il y en a autant que le nombre de «1» sur la ligne* $\boldsymbol{E}[a]$, *autrement dit, il y a* r_a *couples* (a, t).

Soit $\mathcal{G}_a = \{(a, t) \mid t \in \mathcal{J}\} = \{(a, j_1), (a, j_2), \ldots, (a, j_{r_a})\}$. *d'après le Lemme 5.1, pour tout* $c \neq a$ *and* $d \in \mathcal{J}$, *la* a*-ème ligne de* $\boldsymbol{E}_{c,d}\boldsymbol{S}^{-1}$ *est nulle. Alors, on a*

$$\boldsymbol{R}[a] = \boldsymbol{S}^{**}\boldsymbol{S}^{-1}[a] = \boldsymbol{S}^{*}\boldsymbol{S}^{-1}[a] + \boldsymbol{E}\boldsymbol{S}^{-1}[a] = \boldsymbol{S}^{*}\boldsymbol{S}^{-1}[a] + \sum_{(i,j)\in\mathcal{I}\times\mathcal{J}} (\boldsymbol{E}_{i,j}\boldsymbol{S}^{-1})[a]$$

$$= \boldsymbol{S}^{*}\boldsymbol{S}^{-1}[a] + \sum_{k=1}^{r_a} \boldsymbol{S}^{-1}[j_k].$$

Nous obtenons donc le résultat suivant :

1. *Si* $i \notin \mathcal{I}$, *Alors la suite* (ε_k) *est nulle et on peut prendre des* (j_k) *quelconques et deux à deux distincts.*
2. *Si* $i \in \mathcal{I}$ *et compte tenu de l'ensemble* $\mathcal{G}_i$ *défini ci-dessus, il suffit de prendre la suite* (ε_k) *constante égale à* 1 *et les* (j_k) *donnés par* $\mathcal{G}_i$.

Dans les cas où la matrice $\boldsymbol{E}$ a des coefficients négatifs ou n'est pas binaire, on peut adapter le Théorème 5.1. Il suffit de permettre à la suite ε_k de prendre la valeur -1 et de pouvoir soustraire (ou additionner) la même ligne plusieurs fois.
Contrairement au cas précédent, nous ne pouvons pas donner d'algorithme pour déterminer une liste de candidats possibles pour la matrice $\boldsymbol{R}$. Nous ne connaissons pas, a priori, le nombre de lignes erronées à retirer dans $\boldsymbol{R}'$. Bien que la correction de l'erreur dans la matrice $\boldsymbol{S}^*$ à l'**Étape 2** soit théoriquement possible, elle peut être difficilement réalisable dans certains cas.

5.4 Comparaison avec d'autres attaques

Rappelons que le but de notre attaque est d'obtenir la matrice privée $\boldsymbol{Q}$ dans le schéma de Niederreiter. Plusieurs travaux antérieurs ont montré qu'il était possible d'obtenir des informations secrètes sur le déchiffrement de McEliece original [McE78] (Algorithme 6, page 34). Strenzke a proposé dans [Str10 ; Str13], deux attaques sur chacun des deux appels de l'algorithme d'Euclide étendu (Extended Euclidean Algorithm ou EEA) (lignes 4 et 10 dans Algorithme 3, page 32) dans le déchiffrement de McEliece avec les paramètres $n = 2\,048$ et $t = 50$. Cette vulnérabilité dans l'algorithme de décodage de Patterson [Pat75] permet à Eve (attaquante) de recueillir des informations sur la matrice de permutation privée $\boldsymbol{P} \in \mathbb{F}_2^{n\times n}$ à l'aide d'une SCA de type temporelle.

La première attaque de Strenzke [Str10] cible le deuxième appel de l'EEA dans l'algorithme de Patterson (ligne 10 dans Algorithme 3, page 32) pour déterminer les deux

polynômes formant le polynôme de localisateur d'erreurs $\sigma(x)$. Ce polynôme de degré $deg(\sigma(x)) \leq t$ consiste en deux polynômes $u(x)$ et $v(x)$ dont le degré $deg(v(x))$ a un impact sur le nombre d'itérations dans l'EEA. Cette variation du nombre d'itérations implique une différence dans les temps d'exécution et rend possible une attaque temporelle. Ainsi, Eve peut effectuer une attaque par textes chiffrés choisis en utilisant des vecteurs d'erreurs $\boldsymbol{e}_i$ aléatoires avec des poids de Hamming fixés ($w_H(\boldsymbol{e}_i) = 4$). Dans ce cas, $deg(\sigma(x) = 4)$, $deg(u(x) = 2)$ et $deg(v(x) \leq 1)$. Eve évalue, d'abord, si une itération s'est produite dans l'EEA ou non. Dans le cas d'une itération, le degré du polynôme $v(x)$ est égale à 1 et le vecteur d'erreur correspondant est ajouté comme une nouvelle ligne dans une matrice binaire composée de n colonnes. Ensuite, à chaque fois qu'une nouvelle ligne est ajoutée dans cette matrice, Eve effectue une élimination de Gauss-Jordan avant de déterminer son rang. Enfin, une fois le rang maximum atteint (2 036), l'attaque se termine avec 7 848 229 textes chiffrés avec les paramètres proposés. Comme les positions des erreurs choisies par Eve sont dans un ordre dépendant de la permutation $\boldsymbol{P}$, cette approche permet de récupérer cette matrice privée. Toute fois, cette attaque ne marche que pour des valeurs de poids de Hamming de $\boldsymbol{e}$ égales à 2 ou 4.

La seconde attaque de Strenzke [Str13] est basée sur une fuite d'information dans l'inversion du syndrome de l'erreur dans l'algorithme de Patterson lors du premier de l'EEA (ligne 4 dans Algorithme 3, page 32). Strenzke a montré qu'avec une attaque temporelle dans l'inversion du syndrome, on peut obtenir une information sur l'élément nul dans le support privée $\mathcal{L}$ du code de Goppa (Définition 2.24, page 27). Cette attaque est basée sur l'analyse de l'équation-clef (Équation 2.2) pour déduire les relations entre les degrés des polynômes $\sigma(x)$ et $\theta(x)$ qui dépendent directement du poids de Hamming de l'erreur $\boldsymbol{e}$. Dans ce cas aussi, le nombre d'itérations dans l'EEA dépend du degré de $\theta(x)$ et rend une attaque temporelle possible. Comme dans [Str10], Eve effectue une attaque par textes chiffrés choisis $\boldsymbol{e}_i$ avec des poids de Hamming fixés afin de trouver la matrice $\boldsymbol{P}$. Cependant, l'attaque ne marche que pour des poids de Hamming de 2, 4 ou 6 de $\boldsymbol{e}$. Malgré les améliorations apportées par [Buc+16], le principal inconvénient de ces deux attaques reste la contrainte sur les poids de Hamming de $\boldsymbol{e}$ pour retrouver la matrice privée $\boldsymbol{P}$.

Les attaques de Strenzke ciblent principalement le déchiffrement de McEliece original. Bien que les schémas de Niederreiter et de McEliece original soient légèrement différents, la condition préalable pour ces attaques est de choisir des textes chiffrés aléatoires pour le déchiffrement. Ce qui rend notre attaque également applicable dans le déchiffrement de McEliece original. Nous aurons besoins cette fois-ci de n textes chiffrés aléatoires $\boldsymbol{s}_i$, de construire une matrice $\boldsymbol{S}^* \in \mathbb{N}^{n \times n}$ à l'**Étape 2** avant de résoudre le MVPP (Définition 5.3) et d'obtenir la matrice privée $\boldsymbol{R}^{-1} = \boldsymbol{P} \in \mathbb{F}_2^{n \times n}$. Contrairement aux attaques de Strenzke, nous n'avons pas de contraintes sur le poids de Hamming des textes chiffrés $\boldsymbol{e}_i$ pour trouver la matrice privée $\boldsymbol{P}$. De plus, nous n'avons besoin que de $n = 2\,048$ textes chiffrés au lieu de 7 848 229 dans [Str10] pour trouver la matrice de permutation privée $\boldsymbol{P}$. La Table 5.4 ci-dessous montre aussi que notre attaque cible la première étape du déchiffrement (produit matrice-vecteur) qui nécessite moins d'opé-

rations que les différents appels de l'EEA.

Table 5.4 – Comparaison de SCA dans McEliece Original.

Attaque	Poids de Hamming	Textes chiffrés	Cible dans [McE78]
Attaques de Strenzke	2, 4 ou 6	7 848 229 dans [Str10]	EEA
Notre attaque	pas de contraintes	2 048	Première étape du déchiffrement

5.5 Conclusion

Dans ce chapitre, nous avons présenté une attaque par canal caché sur le problème du produit matrice-vecteur (MVPP) qui est un nouveau formalisme que nous avons proposé pour la cryptographie à base de codes. L'intérêt de ce nouveau formalisme repose sur le fait que, si nous parvenons à trouver le résultat du produit dans $\mathbb{N}$ à l'aide d'une analyse de consommation, nous pouvons obtenir directement la matrice privée dans un schéma basé sur les codes en résolvant un système linéaire. Nous avons aussi montré qu'on pouvait retrouver le secret dans de tel schéma sans résoudre un problème de décodage par syndrome comme dans certains récents travaux. En plus des informations récupérées par l'analyse de consommation, qui est une étape de notre attaque, nous avons proposé une attaque par textes chiffrés choisis sur le déchiffrement de Niederreiter. Nous avons réussi à trouver directement la matrice privée $\boldsymbol{Q}$ durant la première étape du déchiffrement. Dans le cas où il y a des erreurs (à cause du bruit par exemple) dans l'étape d'analyse de consommation de notre attaque, nous avons montré que dans certains cas, il était également possible de retrouver cette matrice privée. De plus, notre attaque n'a pas de contraintes sur les poids de Hamming des textes chiffrés et nécessite moins de données par rapport à d'autres attaques physiques pour retrouver la matrice de permutation privée $\boldsymbol{P}$ dans McEliece original. Dans nos travaux futurs, nous tenterons d'identifier les opérations secrètes du produit matrice-vecteur dans la décapsulation des candidats finalistes du NIST basés sur les codes correcteurs d'erreurs, notamment HQC pour appliquer notre attaque sur le MVPP avant de proposer une contre-mesure.

Chapitre 6

Attaque sur l'identification en métrique rang

Ce chapitre correspond à notre publication pour Cryptography, Codes and Cyber Security-I4CS, 2022 [Sec+23a] :

Cryptanalysis of a code-based identification scheme presented in CANS 2018
avec Pierre-Louis Cayrel, Idy Diop et Morgan Barbier. I4CS 2022, LNCS.

Le NIST (National Institute of Standards and Technology) a récemment publié les quatre premiers algorithmes gagnants du processus de normalisation de la cryptographie post-quantique (Post-Quantum Cryptography ou PQC) lancé depuis décembre 2016. Les algorithmes sélectionnés sont CRYSTALS-Kyber, un mécanisme d'encapsulation de clefs (Key Encapsulation Mechanism ou KEM), CRYSTALS-Dilithium, FALCON et SPHINCS+, des schémas de signature numérique. Les trois premiers schémas sont basés sur les lattices et le dernier sur les fonctions de hachage. Ces futures normes vont être intégrées à la fin du processus de normalisation dans la chaine de valeur actuelle et seront les options par défaut pour la cryptographie post-quantique. Afin de diversifier les normes des schémas de signature post-quantique, le NIST a lancé un appel à soumission de schémas avec des tailles de signatures courtes et un temps de vérification rapide pour juin 2023. L'approche basée sur les codes est un candidat prometteur pour ce processus supplémentaire.

Dans ce chapitre, nous allons d'abord montrer que le protocole d'identification de Véron en métrique rang proposé dans [Bel+18] laisse échapper des informations sur le vecteur d'erreur. Ensuite, nous effectuons une attaque sur le support de l'erreur de ce schéma en utilisant l'algorithme de Gaborit, Ruatta, et Schreck (ou GRS) pour trouver le secret. Enfin, nous proposons un protocole d'identification zero-knowledge de Véron en métrique rang à l'aide d'une permutation spéciale avant de proposer un schéma de

signature efficace avec une vérification rapide.

6.1 Définitions

Cette section présente quelques définitions importantes pour l'identification basée sur la métrique rang. La métrique rang a été introduite dans la théorie des codes en 1985 par E. Gabidulin [Gab85]. Il a montré que la complexité du problème de décodage par syndrome avec cette métrique est exponentielle et est aussi difficile qu'en métrique de Hamming pour les mêmes paramètres. C'est ainsi que les codes en métrique de rang ont été diversement proposés comme alternatives en cryptographie à base de codes. Nous renvoyons le lecteur dans [Loi06] pour plus de détails sur la métrique de rang et ses applications en cryptographie. La métrique rang d'un vecteur est le rang de la matrice obtenue en décomposant sur le corps de base ses composantes défini sur une extension.

Définition 6.1 (Poids rang d'un vecteur) *Soit* $(\beta_1, \ldots, \beta_m) \in \mathbb{F}_{q^m}$, *une base de* $\mathbb{F}_{q^m}/\mathbb{F}_q$. *À tout vecteur* $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{F}_{q^m}^n$, *on associe la matrice*

$$\boldsymbol{M}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{x}_{1,1} & \ldots & \boldsymbol{x}_{1,n} \\ \vdots & & \vdots \\ \boldsymbol{x}_{m,1} & \ldots & \boldsymbol{x}_{m,n} \end{pmatrix} \in \mathbb{F}_q^{m \times n}$$

telle que $\forall j \in \{1 \ldots n\}, \boldsymbol{x}_j = \sum_{i=1}^{m} \boldsymbol{x}_{ij}\beta_i$.
Le poids rang de $\boldsymbol{x}$, *noté* $\mathtt{rang}(\boldsymbol{x})$, *est défini par*

$$\mathtt{rang}(\boldsymbol{x}) = \mathtt{rang}(\boldsymbol{M}(\boldsymbol{x})) = w.$$

Nous allons tout au long de ce chapitre utiliser la notion de support de l'erreur en métrique rang.

Définition 6.2 (Support en métrique rang) *Soit* $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{F}_{q^m}^n$, *un vecteur. On appelle support de* $\boldsymbol{x}$, *le sous-espace vectoriel de* $\mathbb{F}_{q^m}$ *engendré par les* $\boldsymbol{x}_i$ *tel que*

$$Supp(\boldsymbol{x}) = \langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \rangle_{\mathbb{F}_q},$$

et la dimension du support $dim(Supp(\boldsymbol{x})) = w$.

Remarque 6.1 *Le nombre de supports de dimension* w *dans* $\mathbb{F}_{q^m}$ *est donné par le coefficient binomial de Gauss* $\begin{bmatrix} m \\ w \end{bmatrix}_q$ *[LW01] et on a*

$$\begin{bmatrix} m \\ w \end{bmatrix}_q = \prod_{i=0}^{w-1} \frac{q^m - q^i}{q^w - q^i} \approx q^{w(m-w)}.$$

L'idée d'utiliser des codes correcteurs d'erreurs pour construire un schéma d'identification vient de Stern [Ste88]. Le schéma de Stern est un protocole à divulgation nulle de connaissance (ou zero-knowledge) composé de 3 passes avec une probabilité de triche égale à 2/3 (Section 2.5, page 38). Il est basé sur le problème du décodage par syndrome binaire (Définition 5.1, page 94) utilisant une matrice de contrôle publique d'un code linéaire aléatoire. Le schéma de Véron [Vér97], est la version duale de Stern utilisant une matrice génératrice publique (Section 2.5, page 39) afin d'obtenir, entre autres, une amélioration du taux de transmission avec une meme probabilité de triche. Cayrel *et al.* ont proposé un schéma d'identification zero-knowledge appelé CVE [CVEYA10]. Ce protocole est basé sur problème du décodage par syndrome des codes q-aires et est composé de 5 passes, mais avec une probabilité de triche plus faible de 1/2 en augmentant la taille du corps de base q. Comme nous l'avons expliqué dans la Section 2.5 (page 38), tous ces schémas d'identification peuvent être transformés en schémas de signature avec l'heuristique de Fiat-Shamir [FS86]. Cependant, la plupart des schémas de signature à base de codes ne sont pas réalisables en pratique, notamment en raison de leur taille de signature et de taille de clef publique souvent assez grande.

Dans [Che95], Chen a proposé le premier protocole d'identification basé sur le problème de décodage par syndrome en métrique rang (Syndrome Decoding problem in the Rank metric ou RSD).

Définition 6.3 (Problème du décodage par syndrome en métrique rang) *Soient* $\boldsymbol{H} \in \mathbb{F}_{q^m}^{(n-k)\times n}$, $\boldsymbol{s} \in \mathbb{F}_{q^m}^{n-k}$, *un vecteur et* $w \in \mathbb{N}$, *un entier. Le problème du décodage par syndrome en métrique rang (ou RSD) est de trouver un vecteur* $\boldsymbol{e} \in \mathbb{F}_{q^m}^{n}$ *de poids de rang* $\mathtt{rang}(\boldsymbol{x}) \leq w$ *tel que* $\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}$.

Ce problème est moins connu et beaucoup moins étudié que le SDP en métrique de Hamming, mais il est considéré comme $\mathcal{NP}$-complet.

Suite à deux attaques dans [CS96] et [OJ02], le protocole d'identification à divulgation nulle de connaissance de Chen a été complètement cassé par Gaborit *et al.* dans [GSZ11]. Les deux attaques proposées sont rendues possibles en raison de failles dans les preuves à divulgation nulle de connaissance du protocole Chen. La première attaque repose sur un défaut de masquage du secret par une simple multiplication matricielle dans le protocole. En effet, un point crucial dans un schéma d'identification est de masquer le secret durant les étapes de réponse pour assurer la preuve à divulgation nulle de connaissance. La deuxième attaque exploite la non-utilisation d'une fonction de hachage dans l'étape d'engagement du protocole. Cela conduit aussi à une fuite d'informations secrètes dans le protocole. Nous proposons une transformation spéciale qui permet de masquer correctement le secret en métrique rang.

Définition 6.4 (Transformation spéciale) *Soient* $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{F}_{q^m}^{n}$, *un vecteur de poids rang* $\mathtt{rang}(\boldsymbol{x}) = w$ *et de support* $Supp(\boldsymbol{x})$, $\Sigma \in \mathcal{S}_n$, *une matrice de permutation et* $\Gamma \in \mathbb{F}_{q}^{m\times m}$ *(*$\Gamma_{i,j} \neq 0$, $i = 1, \ldots, m$ *et* $j = 1, \ldots, n$*), une matrice de rang* w. *On définit la transformation* $\Pi_{\Gamma,\Sigma}$ *par*

$$\begin{aligned}\Pi_{\Gamma,\Sigma}: \quad & \mathbb{F}_{q^m}^n \longrightarrow \mathbb{F}_{q^m}^n \\ & (\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n) \longmapsto (\Gamma_{1,1}\boldsymbol{x}_{1,\Sigma(1)}+\ldots+\Gamma_{1,m}\boldsymbol{x}_{m,\Sigma(1)}\ldots\Gamma_{m,1}\boldsymbol{x}_{1,\Sigma(n)}+\ldots+\Gamma_{m,m}\boldsymbol{x}_{m,\Sigma(n)}),\end{aligned}$$

et la matrice associée $\Pi_{\Gamma,\Sigma}(\boldsymbol{x})$ *est définie par*

$$\begin{aligned}\Pi_{\Gamma,\Sigma}(\boldsymbol{x}) &= \Sigma\left(\begin{pmatrix}\Gamma_{1,1} & \ldots & \Gamma_{1,m}\\ \vdots & & \vdots \\ \Gamma_{m,1} & \ldots & \Gamma_{m,m}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}_{1,1} & \ldots & \boldsymbol{x}_{1,n}\\ \vdots & & \vdots \\ \boldsymbol{x}_{m,1} & \ldots & \boldsymbol{x}_{m,n}\end{pmatrix}\right) \\ &= \begin{pmatrix}\Gamma_{1,1}\boldsymbol{x}_{1,\Sigma(1)}+\ldots+\Gamma_{1,m}\boldsymbol{x}_{m,\Sigma(1)} & \ldots & \Gamma_{1,1}\boldsymbol{x}_{1,\Sigma(n)}+\ldots+\Gamma_{1,m}\boldsymbol{x}_{m,\Sigma(n)}\\ \vdots & \ddots & \vdots \\ \Gamma_{m,1}\boldsymbol{x}_{1,\Sigma(1)}+\ldots+\Gamma_{m,m}\boldsymbol{x}_{m,\Sigma(1)} & \ldots & \Gamma_{m,1}\boldsymbol{x}_{1,\Sigma(n)}+\ldots+\Gamma_{m,m}\boldsymbol{x}_{m,\Sigma(n)}\end{pmatrix} \in \mathbb{F}_q^{m\times n},\end{aligned}$$

et $\mathtt{rang}(\Pi_{\Gamma,\Sigma}(\boldsymbol{x})) = \mathtt{rang}(\boldsymbol{x}) = dim(Supp(\boldsymbol{x})) = w.$

Cette fonction, en plus de masquer correctement le vecteur $\boldsymbol{x}$ par transformation linéaire (Γ) et permutation de coordonnées (Σ), a la propriété de préserver son rang dans $\mathbb{F}_{q^m}^n$. Cette définition peut être généralisée comme dans [GSZ11]. Ainsi, tout schéma d'identification à divulgation nulle de connaissance en métrique de Hamming peut être adapté en métrique rang. Cependant, le protocole de Véron en métrique rang proposé par Bellini *et al.* dans [Bel+18] laisse échapper des informations sur le secret. Nous allons exploiter cette faille du protocole pour réaliser une attaque sur le support du secret avec l'algorithme de Gaborit, Ruatta, et Schreck (ou GRS) [GRS15] dans la Section 6.3.

6.2 Algorithme GRS

Dans cette section, nous présentons l'algorithme GRS [GRS15] pour résoudre le problème RSD. L'idée générale est de trouver un sous-espace vectoriel $\mathcal{F}$ contenant le support $Supp(\boldsymbol{e})$ du vecteur $\boldsymbol{e}$ de poids w puis d'exprimer ses coordonnées dans une base de $\mathcal{F}$ avant d'utiliser les équations de parité pour résoudre un système linéaire. Cet algorithme est basé sur la recherche du support de $\boldsymbol{e}$. Soit une instance du problème RSD définie par

$$\boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{s}. \tag{6.1}$$

Supposons qu'on connait un sous-espace vectoriel $\mathcal{F}$ de dimension w' et $(\mathcal{F}_1,\ldots,\mathcal{F}_{w'})$, une base de $\mathcal{F}$ tel que $Supp(\boldsymbol{e}) \subset \mathcal{F}$. Si nous exprimons les coordonnées $\boldsymbol{e}_i$ $(i=1,\ldots,n)$ de $\boldsymbol{e}$ dans cette base, on a

$$\boldsymbol{e}_i = \sum_{j=1}^{w} \upsilon_{i,j}\mathcal{F}_j, \text{avec } \upsilon_{i,j} \in \mathbb{F}_q^{nw'}. \tag{6.2}$$

En écrivant les équations de parité 6.1 sur $\mathbb{F}_{q^m}$, on obtient

$$\begin{cases} \sum_{j=1}^{w}(\upsilon_{i,1}\boldsymbol{H}_{1,1}\mathcal{F}_j + \cdots + \upsilon_{i,n}\boldsymbol{H}_{1,n}\mathcal{F}_j) = \boldsymbol{s}_1 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ \sum_{j=1}^{w}(\upsilon_{i,1}\boldsymbol{H}_{n-k,1}\mathcal{F}_j + \cdots + \upsilon_{i,n}\boldsymbol{H}_{n-k,n}\mathcal{F}_j) = \boldsymbol{s}_{n-k} \end{cases} \tag{6.3}$$

En projetant 6.3 sur $\mathbb{F}_q$, on obtient un système composé de $(n-k)m$ équations à nw' inconnus. Ce système admet au moins une solution en supposant que $\mathcal{F}$ contient le support de $\boldsymbol{e}$. Pour que cette solution soit unique, il faut avoir plus d'équations que d'inconnues. Ce qui implique la condition suivante

$$w' \leq \left\lfloor \frac{(n-k)m}{n} \right\rfloor = m + \left\lfloor \frac{-km}{n} \right\rfloor. \tag{6.4}$$

Soit p , la probabilité d'obtenir $Supp(\boldsymbol{e}) \subset \mathcal{F}$. On sait qu'un sous-espace de $\mathbb{F}_q^m$ de dimension w' contient $\begin{bmatrix} w' \\ w \end{bmatrix}_q$ sous-espaces de dimension w, alors

$$p(Supp(\boldsymbol{e}) \subset \mathcal{F}) = \frac{\begin{bmatrix} w' \\ w \end{bmatrix}_q}{\begin{bmatrix} m \\ w \end{bmatrix}_q} \approx \frac{q^{w(w'-w)}}{q^{w(m-w)}} = q^{-w(m-w')}. \tag{6.5}$$

La complexité moyenne de l'algorithme GRS est égale à l'inverse de p fois le coût de la résolution du système linéaire dans $\mathbb{F}_q$. En tenant en compte l'Équation 6.4, on obtient

$$\mathcal{O}\left((n-k)^3 m^3 q^{w\left\lceil \frac{km}{n} \right\rceil}\right) \text{ opérations dans } \mathbb{F}_q.$$

Considérons le sous-espace $\boldsymbol{e}^{-1}Supp(\boldsymbol{e})$ de dimension w contenant le vecteur 1 et appliquons la même méthode que précédemment. Le nombre d'équations du système linéaire sur $\mathbb{F}_q$ est maintenant $(n-k-1)m$ et nous connaissons un élément de $Supp(\boldsymbol{e})$. L'équation 6.4 devient

$$w' = m + \left\lfloor \frac{-(k+1)m}{n} \right\rfloor, \tag{6.6}$$

et la probabilité d'avoir $\boldsymbol{e}^{-1}Supp(\boldsymbol{e}) \subset \mathcal{F}$ est alors

$$p(\boldsymbol{e}^{-1}Supp(\boldsymbol{e}) \subset \mathcal{F}) = \frac{\begin{bmatrix} w' \\ w-1 \end{bmatrix}_q}{\begin{bmatrix} m \\ w-1 \end{bmatrix}_q} \approx q^{-(w-1)(m-w')}. \tag{6.7}$$

Puisque $1 \in Supp(\boldsymbol{e})$, la probabilité d'avoir les $w-1$ éléments restants de $Supp(\boldsymbol{e})$ dans $\mathcal{F}$ est donnée par

$$p(\boldsymbol{e}^{-1}Supp(\boldsymbol{e}) \subset \mathcal{F}) = q^{(w-1)\left\lceil \frac{(k+1)m}{n} \right\rceil}, \tag{6.8}$$

et la complexité moyenne de l'algorithme GRS devient

$$\mathcal{O}\left((n-k)^3 m^3 q^{(w-1)\left\lceil \frac{(k+1)m}{n} \right\rceil}\right). \tag{6.9}$$

Ainsi, on peut constater que la connaissance d'un vecteur de la base de $Supp(\boldsymbol{e})$ permet d'améliorer la complexité de cet algorithme.

6.3 Attaque du protocole dans [Bel+18]

Dans cette section, nous allons présenter l'attaque réalisée sur le support du secret dans le protocole d'identification présenté à CANS 2018 [Bel+18]. Dans ce protocole, Bellini *et al.* ont instancié le protocole d'identification zero-knowledge de Véron en métrique de Hamming [Vér97] dans la métrique rang. Ils utilisent une matrice génératrice $\boldsymbol{G} \in \mathbb{F}_{q^m}$ d'un code linéaire aléatoire comme clef publique et un vecteur d'erreur $\boldsymbol{e}$ de poids w comme secret. C'est un protocole d'identification composé de 3 passes avec une probabilité de triche de 2/3 comme dans la version en métrique de Hamming mais avec des tailles de données plus compactes (Table 6.1). Ce qui a pour conséquence, entre autres, d'accélérer la vérification des réponses du prouveur par le vérifieur. Cependant, le point crucial dans un schéma d'identification à divulgation nulle de connaissance est que le vérifieur ne doit obtenir aucune information sur le secret durant les étapes de réponses (Section 2.5, page 2.5).

TABLE 6.1 – Propriétés du protocole de Véron dans [Bel+18].

Données	Taille en bits
Matrice génératrice	$m \times k \times n$
Publique	$mn + \log_2(w)$
Secrète	$mk + mn$
Nombre de bits échangés	$\vartheta(3\mathtt{hash}+2+\frac{2}{3}(n+m(k+n)))$

Ainsi, pour bien masquer le secret $\boldsymbol{e}$ de poids w dans ce protocole, il faut le transformer en un autre vecteur aléatoire tout en préservant le poids w. Pour mettre en œuvre leur protocole, Bellini *et al.* utilisent une permutation aléatoire $\sigma(\boldsymbol{e})$ du secret lorsque $b = 1$ (Figure 6.1). Cependant, cette permutation n'est pas suffisante pour masquer correctement le secret $\boldsymbol{e}$ dans $\mathbb{F}_{q^m}$ et laisse échapper de l'information dans le protocole.

Prouveur		**Vérifieur**
$(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{y}, \boldsymbol{G}, w), (\boldsymbol{x}, \boldsymbol{e}))$		$\mathsf{pk} = (\boldsymbol{y}, \boldsymbol{G}, w)$
$\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_{q^m}^k$, $\sigma \xleftarrow{\$} \mathcal{S}_n$		
$c_1 \leftarrow \mathtt{hash}(\sigma)$		
$c_2 \leftarrow \mathtt{hash}(\sigma((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G}))$		
$c_3 \leftarrow \mathtt{hash}(\sigma(\boldsymbol{u}\boldsymbol{G}+\boldsymbol{y}))$	$\xrightarrow{c_1, c_2, c_3}$	
	$\xleftarrow{b}$	$b \xleftarrow{\$} \{0,1,2\}$
Si $b = 0$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow (\boldsymbol{u}+\boldsymbol{x})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_2 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2\boldsymbol{G}))$
Si $b = 1$		
$\mathbf{rp}_1 \leftarrow \sigma((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G})$ $\mathbf{rp}_2 \leftarrow \sigma(\boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_2 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1 + \mathbf{rp}_2)$ ainsi que $\mathtt{rang}(\sigma(\boldsymbol{e})) \stackrel{?}{=} w$
Si $b = 2$		
$\mathbf{rp}_1 \leftarrow \sigma$ $\mathbf{rp}_2 \leftarrow \boldsymbol{u}$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1(\mathbf{rp}_2\boldsymbol{G}+\boldsymbol{y}))$

FIGURE 6.1 – Protocole de Véron dans [Bel+18].

En effet, cette permutation aléatoire $\sigma(\boldsymbol{e})$ de $\boldsymbol{e}$ dans $\mathbb{F}_{q^m}$ révèle les valeurs non nulles dans $\boldsymbol{e}$ même si leurs positions restent secrètes. Bien sûr, cette information est inutile

dans le cas des codes binaires comme dans le protocole d'identification de Stern. Par exemple, dans ce protocole, une permutation aléatoire de coordonnées sur $\mathbb{F}_2$ a la propriété de transformer n'importe quel mot avec un poids de Hamming donné en n'importe quel autre mot de même poids. Ce point est essentiel pour la preuve à divulgation nulle de connaissance dans le protocole. Par contre, la notion de permutation dans $\mathbb{F}_{q^m}$, de manière générale, n'est pas équivalente à celle dans $\mathbb{F}_2$ et la connaissance des valeurs non nulles de $\boldsymbol{e}$ dans $\mathbb{F}_{q^m}$ donne un avantage considérable à Eve (attaquante). La permutation $\sigma(\boldsymbol{e})$ permet de changer les positions des coordonnées dans $\boldsymbol{e}$ mais pas le support $Supp(\boldsymbol{e})$. Par conséquent, les coordonnées de $\sigma(\boldsymbol{e})$ et celles de $\boldsymbol{e}$ vont générer le même espace vectoriel sur $\mathbb{F}_q$. Nous allons maintenant exploiter cette information pour réaliser une attaque du support en utilisant l'algorithme GRS.

On rappelle que cette attaque sur le support de $\boldsymbol{e}$ a été utilisée pour d'autres scénarios dans [GSZ11] et [LTP19]. Dans ce protocole interactif, le prouveur doit prouver sa connaissance du secret $\boldsymbol{e}$ par des séries propositions sans révéler aucune information sur $\boldsymbol{e}$ mais seulement la véracité de sa proposition. L'attaque que nous présentons ici cible le moment où le prouveur envoie la permutation $\sigma(\boldsymbol{e})$ ($b = 1$) dans la Figure 6.1. L'idée est de récupérer le support de $\boldsymbol{e}$ à partir de $\sigma(\boldsymbol{e})$ avec une faible complexité. Pour éviter une recherche exponentielle de bases comme dans [CS96], nous allons d'abord générer une base $(\mathcal{F}_1, \ldots, \mathcal{F}_w)$ de w vecteurs avant d'exprimer chaque coordonnée $\boldsymbol{e}_i$ (ou $\sigma(\boldsymbol{e}_i)$) de $\boldsymbol{e}$ (ou $\sigma(\boldsymbol{e})$) dans cette base comme dans la section 6.2. Ensuite, on construit une base $(\mathcal{F}_1, \ldots, \mathcal{F}_w, \mathcal{F}_{w+1}, \ldots, \mathcal{F}_m)$ de $\mathbb{F}_{q^m}$ sur $\mathbb{F}_q$ dont les w premiers vecteurs correspondent à $\mathcal{F}_1, \ldots, \mathcal{F}_w$. On écrit les équations de parité (Équation 6.1) dans cette base et on obtient ainsi un système linéaire de $(n-k) \times m$ équations et $n \times w$ inconnues sur $\mathbb{F}_q$. Enfin, la complexité de l'algorithme GRS pour trouver le secret $\boldsymbol{e}$ se réduit en une résolution de système linéaire. Les paramètres proposés dans [Bel+18] ($q = 2, n = 64, w = 9, k = 30, m = 80$) ont permis par élimination de Gauss d'une matrice de taille $(n-k) \times m$, de récupérer le secret $\boldsymbol{e}$. L'attaque a été réalisée sur un Intel® Core™ i7-7700 CPU @ 3.60 GHz et le temps d'exécution avec les paramètres proposés dans [Bel+18] sont rapportés dans la Table 6.2.

TABLE 6.2 – Temps d'exécution de notre attaque.

Schéma	n	m	w	k	Temps (en ms)
Véron80	35	48	5	16	14
Véron128	64	80	9	30	26

6.4 Nouveau schéma de signature

Le problème dans le protocole d'identification de Véron en métrique rang proposé par Bellini *et al.* dans [Bel+18], est un masquage défectueux du secret $\boldsymbol{e}$. Pour réparer ce protocole d'identification avec les propriétés de la métrique rang, nous allons utiliser la transformation de la Définition 6.4. Cette permutation spéciale appliquée au secret $\Pi_{\Gamma,\Sigma}(\boldsymbol{e})$ lorsque $b = 1$ (Figure 6.1) permet de le masquer correctement par transformation linéaire associée à une permutation de ses coordonnées et a également la propriété de préserver son rang w dans $\mathbb{F}_{q^m}$. Ainsi, à l'aide de $\Pi_{\Gamma,\Sigma}$, nous proposons un nouveau protocole d'identification à divulgation nulle de connaissance de Véron en métrique rang dans la Figure 6.2.

Prouveur		**Vérifieur**
$(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{y}, \boldsymbol{G}, w), (\boldsymbol{x}, \boldsymbol{e}))$		$\mathsf{pk} = (\boldsymbol{y}, \boldsymbol{G}, w)$
$\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_{q^m}^k$, $\Sigma \xleftarrow{\$} \mathcal{S}_n$, $\Gamma \xleftarrow{\$} \mathbb{F}_q^{m\times m}$		
$c_1 \leftarrow \mathtt{hash}(\Gamma\|\Sigma)$		
$c_2 \leftarrow \mathtt{hash}(\Pi_{\Gamma,\Sigma}((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G}))$		
$c_3 \leftarrow \mathtt{hash}(\Pi_{\Gamma,\Sigma}(\boldsymbol{u}\boldsymbol{G}+\boldsymbol{y}))$	$\xrightarrow{c_1, c_2, c_3}$	
	$\xleftarrow{b}$	$b \xleftarrow{\$} \{0,1,2\}$
Si $b = 0$		
$\mathbf{rp}_1 \leftarrow \Gamma\|\Sigma$ $\mathbf{rp}_2 \leftarrow (\boldsymbol{u}+\boldsymbol{x})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_2 = \mathtt{hash}(\Pi_{\mathbf{rp}_1}(\mathbf{rp}_2\boldsymbol{G}))$
Si $b = 1$		
$\mathbf{rp}_1 \leftarrow \Pi_{\Gamma,\Sigma}((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G})$ $\mathbf{rp}_2 \leftarrow \Pi_{\Gamma,\Sigma}(\boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_2 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1 + \mathbf{rp}_2)$ ainsi que $\mathtt{rang}(\Pi_{\Gamma,\Sigma}(\boldsymbol{e})) \stackrel{?}{=} w$
Si $b = 2$		
$\mathbf{rp}_1 \leftarrow \Gamma\|\Sigma$ $\mathbf{rp}_2 \leftarrow \boldsymbol{u}$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\Pi_{\mathbf{rp}_1}(\mathbf{rp}_2\boldsymbol{G}+\boldsymbol{y}))$

FIGURE 6.2 – Protocole de Véron dans [Bel+18] réparé.

En remplaçant la permutation $\sigma(\boldsymbol{e})$ par $\Pi_{\Gamma,\Sigma}(\boldsymbol{e})$ dans [Bel+18] on obtient la preuve du zero-knowledge basée sur le RSD avec les mêmes deux ensembles de paramètres (Table 6.1). Ainsi, le choix de paramètres, notamment pour m, n, k et w, doit satisfaire les deux conditions suivantes.

1. Pour éviter l'attaque algébrique dans [GRS15], il faut choisir les paramètres tels que

$$\log_2(w^3k^3q^{\lceil\frac{(k+1)(w+1)-(n+1)}{w}\rceil}) \geq \ell, \text{ avec } \ell, \text{ le niveau de sécurité souhaité.}$$

2. Pour éviter la meilleure attaque combinatoire générique dans [Ara+18], il faut que

$$\log_2((n-k)^3m^3q^{w\lceil\frac{(k+1)m}{n}\rceil-m}) \geq \ell.$$

En outre, nous proposons un protocole d'identification avec un faible coût de communication par rapport à la version réparée de Bellini *et al.* (Figure 6.2).

Ce coût correspond au nombre de bits échangés entre les deux entités (prouveur et vérifieur) durant le protocole. Ainsi, dans notre protocole, le prouveur envoie N_Σ (avec $N_\Sigma < n$) et N_Γ (avec $N_\Gamma < m$) bits respectivement lorsque $b = 0$ et $b = 2$ (Figure 6.3) au lieu de la permutation aléatoire $\Sigma \in \mathcal{S}_n$ et de la matrice $\Gamma \in \mathbb{F}_q^{m\times m}$ comme dans la Figure 6.2. En fait, N_Γ et N_Σ sont les graines (Remarque 2.14, page 38) qui permettront au vérifieur de générer respectivement la permutation aléatoire Σ et la matrice Γ de poids rang w. De plus, ces graines sont choisies telles que $N_\Gamma + N_\Sigma < n$. Par conséquent, en plus de proposer un schéma d'identification zero-knowledge de Véron en métrique rang, nous avons réduit nombre de bits échangés entre le prouveur et le vérifieur.

Prouveur		**Vérifieur**
$(\mathsf{pk}, \mathsf{sk}) = ((\boldsymbol{y}, \boldsymbol{G}, w), (\boldsymbol{x}, \boldsymbol{e}))$		$\mathsf{pk} = (\boldsymbol{y}, \boldsymbol{G}, w)$
$\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_{q^m}^k$, $\Sigma \xleftarrow{\$} \mathcal{S}_n$, $\Gamma \xleftarrow{\$} \mathbb{F}_q^{m\times m}$		
$c_1 \leftarrow \mathtt{hash}(\Gamma\|\Sigma)$		
$c_2 \leftarrow \mathtt{hash}(\Pi_{\Gamma,\Sigma}((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G}))$		
$c_3 \leftarrow \mathtt{hash}(\Pi_{\Gamma,\Sigma}(\boldsymbol{u}\boldsymbol{G}+\boldsymbol{y}))$	$\xrightarrow{c_1, c_2, c_3}$	
	$\xleftarrow{b}$	$b \xleftarrow{\$} \{0,1,2\}$
Si $b = 0$		
$\mathbf{rp}_1 \leftarrow N_\Gamma\|N_\Sigma$ $\mathbf{rp}_2 \leftarrow (\boldsymbol{u}+\boldsymbol{x})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_2 = \mathtt{hash}(\Pi_{\mathbf{rp}_1}(\mathbf{rp}_2\boldsymbol{G}))$
Si $b = 1$		
$\mathbf{rp}_1 \leftarrow \Pi_{\Gamma,\Sigma}((\boldsymbol{u}+\boldsymbol{x})\boldsymbol{G})$ $\mathbf{rp}_2 \leftarrow \Pi_{\Gamma,\Sigma}(\boldsymbol{e})$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_2 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\mathbf{rp}_1 + \mathbf{rp}_2)$ ainsi que $\mathtt{rang}(\Pi_{\Gamma,\Sigma}(\boldsymbol{e})) \stackrel{?}{=} w$
Si $b = 2$		
$\mathbf{rp}_1 \leftarrow N_\Gamma\|N_\Sigma$ $\mathbf{rp}_2 \leftarrow \boldsymbol{u}$	$\xrightarrow{\mathbf{rp}_1, \mathbf{rp}_2}$	**Vérifie** $c_1 = \mathtt{hash}(\mathbf{rp}_1)$ et $c_3 = \mathtt{hash}(\Pi_{\mathbf{rp}_1}(\mathbf{rp}_2\boldsymbol{G}+\boldsymbol{y}))$

FIGURE 6.3 – Protocole de Véron dans [Sec+23a].

Cette réduction du cout de communication constitue une avancée importante pour

l'identification, notamment avec les codes correcteurs d'erreurs. La taille des données de notre schéma est indiquée dans la table 6.3.

TABLE 6.3 – Propriétés du protocole de Véron dans [Sec+23a].

Données	Taille en bits
Matrice génératrice	$m \times k \times n$
Publique	$mn + \log_2(w)$
Secrète	$mk + mn$
Nombre de bits échangés	$\vartheta(3\mathtt{hash}+2+\frac{2}{3}(N_\Gamma+N_\Sigma+m(k+n)))$

Nous pouvons maintenant construire un schéma de signature à partir de notre schéma d'identification à divulgation nulle de connaissance dans la métrique rang en utilisant la transformation de Fiat-Shamir [FS86]. Pour rappel, un schéma de signature numérique est utilisé pour garantir l'intégrité d'un document, c'est-à-dire qu'il n'a pas été modifié depuis la création de la signature ou bien pour authentifier l'entité ayant apposé une signature à un document. Cette signature est calculée en utilisant le message ainsi qu'un secret connu uniquement du signataire et doit ensuite pouvoir être vérifiée grâce à une procédure publique. Les étapes de création de signature et de vérification sont détaillées dans la Section 2.5 (page 38). Dans la Table 6.4, nous comparons notre schéma de signature pour un niveau de sécurité de 128 bits utilisant des graines de 24 bits et une fonction de hachage de 256 bits, avec les meilleurs schémas de signature post-quantique actuels. Des schémas tels que CRYSTALS-Dilithium, FALCON et SPHINCS+ sont considérés aujourd'hui comme les meilleurs schémas de signature post-quantique par rapport aux exigences du NIST en termes de sécurité, de taille de signature et de rapidité. Les candidats pour cette campagne supplémentaire du NIST doivent être, soit plus performants que ces futures normes, ou fournir des propriétés de sécurité substantielles. Nous comparons également notre schéma avec les candidats pour la nouvelle campagne du NIST pour la signature post-quantique. On peut citer, d'abord, le schéma de signature MIRA [Ara+23] est basé sur le problème du MinRank et est construit sur le principe du calcul multipartite sécurisé (multi-party computation ou MPC). Ensuite, LESS [Bal+23] est un schéma de signature basé sur le problème par équivalence linéaire et est obtenu à partir de la transformation de Fiat-Shamir [FS86] d'un schéma d'identification à divulgation nulle de connaissance. Enfin, Wave [DAST19] qui est le premier schéma de signature en métrique rang basé sur le principe du «hache et signe».

TABLE 6.4 – Comparaison (taille des données en octets) avec d'autres schémas de signature post-quantiques.

Schéma	Clef publique	Signature	Famille
SPHINCS+	64	17 088	Hash
MIRA	84	7 376 000	Codes
FALCON	897	666	Lattice
CRYSTALS-Dilithium	1 312	2 420	Lattice
Notre schéma	9 640	334 924	Codes
LESS	13 700	8 400	Codes
Wave	3 677 390	822	Codes

La table 6.4 montre, d'abord, que notre schéma de signature a une taille de clef publique plus faible comparé aux autres schémas de signature basés sur les codes et candidats pour le nouveau processus du NIST tels que Wave et LESS. Une petite taille de clef publique permet notamment d'accélérer l'étape de vérification. Mais nous avons une taille de signature qui est largement plus grande que par rapport à ces deux candidats. Ensuite, nous avons une taille de signature largement plus petite par rapport à MIRA qui a la plus petite taille de clef publique parmi les schémas à base de codes présentés dans ce tableau. Enfin, par rapport aux futures normes de signature post-quantiques, nous avons des tailles de données plus grandes (notre clef publique est 9 fois plus grande que celle de CRYSTALS-Dilithium par exemple) et des améliorations sont notamment attendues dans ce sens.

6.5 Conclusion

Dans ce chapitre, nous avons d'abord réalisé une cryptanalyse complète du schéma d'identification de Véron en métrique rang proposé par Bellini *et al.* Notre attaque exploite une fuite d'information dans le protocole relative à un défaut de masquage du secret avec une simple permutation aléatoire sur $\mathbb{F}_{q^m}$ durant la phase de réponse du prouveur. Cette faiblesse dans ce schéma d'identification nous a permis de récupérer le secret par une attaque sur le support de l'erreur en utilisant l'algorithme de Gaborit, Ruatta et Schreck. Ensuite, nous avons réparé le protocole de Bellini *et al.* à l'aide d'une permutation spéciale qui permet de masquer correctement le secret dans $\mathbb{F}_{q^m}$. Ce qui nous a permis de proposer un schéma d'identification à divulgation nulle de connaissance de Véron en métrique rang avec un faible cout de communication et résistant aux meilleures attaques connues. Enfin, nous avons proposé un schéma de signature avec des tailles de clefs publique et de signature relativement correcte par rapport aux nou-

veaux candidats de signatures post-quantiques. Par exemple, notre schéma a une clef publique plus petite que certains schémas de signature à bases de codes. Ce qui permet entre autres d'accélérer la vérification qui est une exigence importante pour le processus de normalisation supplémentaire du NIST pour la signature post-quantique. Dans nos travaux futurs, nous allons essayer de faire un choix de paramètres plus «agressif» tout en faisant attention aux attaques existantes afin de réduire la taille des données, notamment la taille de la signature et de proposer une implantation.

Conclusion et perspectives

Dans cette thèse, nous avons d'abord présenté le mécanisme d'encapsulation de clef (ou KEM), DAGS qui était l'une des deux propositions pour le processus de normalisation du NIST basées sur des codes structurés utilisant les codes de Srivastava généralisés sous forme quasi-dyadiques. L'approche dans DAGS original était de faire des choix de paramètres assez extrêmes pour obtenir de petites tailles de données notamment pour la clef publique. Malheureusement, cela a conduit aux attaques de Barelli-Couvreur et à la décision du NIST de l'éliminer pour le second tour du processus. Néanmoins, grâce à l'analyse de l'attaque de Bardet *et al.*, les auteurs de la soumission ont proposé de nouveaux paramètres résistants à toutes les attaques structurelles existantes. Ils ont également proposé une approche similaire à celle dans *Classic McEliece* en plus d'un nouvel ensemble de paramètres binaires dont nous avons réalisé une implantation logicielle. Cette variante binaire de DAGS (ou DAGS binaire) offre des avantages considérables, notamment un compromis entre sécurité (hors de portée des attaques susmentionnées) et performance (en temps d'exécution) intéressante. Cette mise en œuvre logicielle a permis de réduire le temps d'exécution des différents algorithmes par rapport aux implantations précédentes de DAGS grâce à l'adaptation de certaines techniques de calculs (multiplication de Karatsuba et la décomposition $\boldsymbol{LUP}$) dans le cas des matrices dyadiques. Cependant, des efforts restent à faire dans l'utilisation de ces techniques et l'élimination de certaines opérations et/ou fonctions factices dans l'implantation afin d'être plus proche des performances des KEM finalistes du NIST basés sur les codes correcteurs d'erreurs. De plus, nous pouvons faire un choix de paramètres binaires plus efficient tout en étant hors de portée des attaques structurelles pour avoir des tailles de données encore plus petites. Par ailleurs, nous prévoyons une implantation matérielle avec des contre-mesures efficaces pour résister aux attaques par canaux cachés. Ce qui serait un atout majeur pour rendre davantage cette construction plus crédible, même à la fin du processus.

Ensuite, nous avons montré qu'en suivant simplement le pas d'exécution de l'algorithme de la décapsulation dans l'implantation référence de *Classic McEliece*, nous étions en mesure d'identifier le moment où les coefficients du polynôme de Goppa (une partie de la clef privée) sont chargés dans la mémoire. En faisant une analyse par canal caché (avec des templates notamment), nous avons réussi à trouver les poids de Hamming de ses coefficients. Cette information secrète sur le polynôme de Goppa, nous a permis de trouver directement ce polynome dans le cas des clefs faibles de Sendrier et Loidreau et d'améliorer la complexité de la recherche exhaustive de ce polynôme sur F_{2^m}. De plus, notre attaque peut être appliquée à différents endroits dans la décapsulation, en particulier lors du chargement des points d'évaluations (qui font partie de la clef privée) pour le polynôme localisateur d'erreur. Ce qui va considérablement améliorer notre connaissance sur la clef privée dans *Classic McEliece*. Clairement, cette fonction de chargement dans cette implantation n'est pas adaptée pour manipuler des données sensibles et la meilleure technique de contre-mesure connue ne permet pas en théorie de mettre en échec notre attaque.

Nous avons également proposé dans cette thèse, le problème du produit matrice-vecteur,

un nouveau formalisme en cryptographie à base de codes dont la résolution permet de faire une attaque générique dans le déchiffrement des schémas à base de codes. La multiplication d'une matrice secrète avec un vecteur (texte chiffré) constitue souvent la première étape du déchiffrement, notamment dans McEliece original et Niederreiter. Ce formalisme exploite le fait que l'on peut retrouver durant la première étape du déchiffrement dans Niederreiter les entrées du résultat de ce produit dans $\mathbb{N}$ avec une analyse de consommation avant de résoudre un système linéaire de faible complexité pour trouver la matrice secrète $\boldsymbol{Q}$. Nous avons aussi montré que notre attaque sur le problème du produit matrice-vecteur nécessite moins de textes chiffrés par rapport aux attaques de Strenzke sur le déchiffrement de McEliece original pour trouver la matrice de permutation secrète $\boldsymbol{P}$. De plus, nous n'avons pas de contraintes sur les poids de Hamming des chiffrés pour le succès de notre attaque. Une piste de réflexion intéressante serait de voir comment améliorer la détection des erreurs dans la matrice secrète après résolution avant de proposer une contre-mesure.

Enfin, nous avons montré une fuite d'information secrète dans l'exécution du protocole d'identification de Véron en métrique proposé par Bellini *et al.* Pour rappel, l'utilisation de cette métrique en cryptographie à base de codes permet notamment d'obtenir des tailles de données plus petites. La faille dans ce protocole est liée à un masquage défectueux du secret $\boldsymbol{e}$ par une simple permutation aléatoire sur $\mathbb{F}_{q^m}$ qui n'a pas les mêmes propriétés qu'en métrique de Hamming. En utilisant une permutation spéciale, nous avons réussi à masquer correctement le secret dans $\mathbb{F}_{q^m}$ avant de proposer le schéma d'identification zero-knowledge de Véron en métrique rang. Nous avons ensuite transformé ce schéma d'identification en un schéma de signature avec des tailles de données compétitives par rapport aux candidats du NIST pour la nouvelle campagne de signatures post-quantiques. Notre schéma a une clef publique plus petite que la plupart des schémas de signatures basées sur les codes correcteurs d'erreurs. Cependant, nous avons des axes d'amélioration sur les tailles de données, notamment en faisant un choix de paramètres plus judicieux, nous espérons nous rapprocher un peu plus par rapport aux nouveaux standards de signatures. Une implantation de ce schéma est en perspective pour évaluer les temps de vérification de notre signature qui est un paramètre important pour le processus de normalisation du NIST.

Annexe A

Algorithme de Berlekamp-Massey

C'est un algorithme qui prend en entrée la capacité de correction du code de Goppa t (ou degré du polynôme de Goppa) ainsi que le polynôme syndrome $s(x)$ et retourne le polynôme localisateur d'erreur $\sigma(x)$.

Algorithme 29 Algorithme de Berlekamp-Massey pour le déchiffrement

Entrée: t, degré du polynôme de Goppa et $s(x)$, le polynôme syndrome.
Sortie: $\sigma(x)$, le polynôme localisateur d'erreur.

$\sigma(x) = 1$
$\beta(x) = x$
$l = 0$
$\delta = 1$
// initialisation.
pour $k \leftarrow 0$ jusqu'à $(2t - 1)$ **faire**
    $d = \sum_{i=0}^{t} \sigma_i s_{lk-i}$.
    si $d = 0$ ou $k < 2l$ **alors**
        Calculer
        $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\beta(x), l, \delta\}$.
    sinon
        Calculer
        $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\sigma(x), k - l + 1, d\}$.
    fin si
fin pour
Retourner $\sigma(x)$.

Annexe B

Exemple simplifié de l'attaque dans [Sec+23b]

Cas 1 : pas d'erreurs dans $\boldsymbol{S}^*$ Soient $\boldsymbol{S} \in \mathbb{F}_2^{(n-k)\times(n-k)}$, une matrice inversible construite à l'**Étape 1** de notre attaque et $\boldsymbol{S}^* \in \mathbb{N}^{(n-k)\times(n-k)}$, une matrice récupérée à l'aide de l'attaque profilée (**Étape 2**). On choisit $n-k=3$ et on a

$$\boldsymbol{S} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

, son inverse

$$\boldsymbol{S}^{-1} = \begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

et

$$\boldsymbol{S}^* = \begin{pmatrix} 2 & 2 & 1 \\ 1 & 0 & 1 \\ 2 & 1 & 2 \end{pmatrix}.$$

Avec l'Équation 5.2, $\boldsymbol{R} = \boldsymbol{S}^*\boldsymbol{S}^{-1}$, on obtient

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Cas 2 : erreurs corrigibles dans $\boldsymbol{S}^*$ On garde les mêmes matrices, $\boldsymbol{S}$ son inverse $\boldsymbol{S}^{-1}$ et $\boldsymbol{S}^*$ comme dans le **premier cas**.
D'après l'Équation 5.3, $\boldsymbol{S}^{**} = \boldsymbol{S}^* + \boldsymbol{E}$, on a

$$\boldsymbol{E} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

En utilisant les Équations 5.2 et 5.3, on obtient la matrice $\boldsymbol{R}'$

$$\boldsymbol{R}' = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

On voit que la matrice $\boldsymbol{R}'$ n'est pas binaire et on peut en déduire que la première ligne contient potentiellement une erreur. En appliquant l'Algorithme 28 (page 103) de la Section 5.3, on détermine la liste des candidats suivants pour retrouver $\boldsymbol{R}$

$$\boldsymbol{R}'[1] - \boldsymbol{S}^{-1}[1] = [1 \;\; 0 \;\; 1]$$

$$\boldsymbol{R}'[1] - \boldsymbol{S}^{-1}[2] = [-1 \;\; 2 \;\; 2]$$
$$\boldsymbol{R}'[1] - \boldsymbol{S}^{-1}[3] = [-1 \;\; 1 \;\; 3].$$

Seul le premier cas donne un vecteur binaire, on en déduit $\boldsymbol{R}[1] = \boldsymbol{R}'[1] - \boldsymbol{S}^{-1}[1]$ et on obtient

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Cas 3 : erreurs dans $\boldsymbol{S}^*$ On considère la matrice d'erreur suivante

$$\boldsymbol{E} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

D'après les Équations 5.2 et 5.3, on obtient la matrice $\boldsymbol{R}'$

$$\boldsymbol{R}' = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

On constate que la matrice $\boldsymbol{R}'$ est binaire et son déterminant $det(\boldsymbol{R}')$ = 1. Dans ce cas, nous ne parvenons pas à détecter les erreurs dans $\boldsymbol{R}'$.

Annexe C

Bibliographie de l'auteur

Publications

- **Boly Seck**, Pierre-Louis Cayrel, Vlad-Florin Dragoi, Idy Diop, Jean Belo Klamti, Morgan Barbier, Jean Belo Klamti, Vincent Grosso and Brice Colombier. *«Side-Channel Attack against Classic McEliece when loading the Goppa Polynomial»*. In International Conference on Cryptology in Africa. Springer. 2023, p. 105-125.
- **Boly Seck**, Pierre-Louis Cayrel, Idy Diop, Vlad-Florin Dragoi, Kalen Couzon, Brice Colombier and Vincent Grosso. *«Key-Recovery by Side-Channel Information on the Matrix-Vector Product in Code-Based Cryptosystems»*. In Information Security and Cryptology. Springer. 2023, p. 219-234.
- **Boly Seck**, Cheikh Thiécoumba Gueye, Gilbert Ndollane Dione, Jean Belo Klamti, Pierre-Louis Cayrel, Idy Diop and Ousmane N'diaye. *« Software Implementation of a Code-Based Key Encapsulation Mechanism from Binary QD Generalized Srivastava Codes»*. In Code Based Cryptography. Springer. 2023, p. 77-89.
- **Boly Seck**, Pierre-Louis Cayrel, Idy Diop and Morgan Barbier. *« Cryptanalysis of a Code-Based Identification Scheme Presented in CANS 2018»*. In Cryptography, Codes and CyberSecurity : First International Conference. Springer. 2023, p. 3-19.
- Doudou Dione, **Boly Seck**, Idy Diop, Pierre-Louis Cayrel, Demba Faye and Ibrahima Gueye. *« Hardware Security for IoT in the Quantum Era : Survey and Challenges»*. Journal of Information Security, 2023, vol. 14, no 4, p. 227-249.

Communications

Conférences internationales

- **Key-Recovery by Side-Channel Information on the Matrix-Vector Product in Code-Based Cryptosystems.**
  25th Annual International Conference on Information Security and Cryptology. Seoul (Korea), November-December, 2022.
  http://www.icisc.org/mainp

— **Cryptanalysis of a Code-Based Identification Scheme Presented in CANS 2018.**
International Conference on Cryptology, Coding Theory and CyberSecurity, 2022.
Casablanca (Morocco) October, 2022.
https://i4cs22.univh2c.ma/

— **Software Implementation of a Code-Based Key Encapsulation Mechanism from Binary QD Generalized Srivastava Codes.**
International Workshop on Code-Based Cryptography.
Trondheim (Norway), May, 2022.
https://www.cb-crypto.org/previous-editions/cbcrypto2022

Conférences nationales

— **Cryptographie post-quantique : de la théorie vers une normalisation du NIST.**
Rencontres des jeunes chercheurs africains en France (Cinquième Édition).
Institut Henri Poincaré de Paris (France), Décembre, 2022.
http://scienceafrique.fr/rjcaf/communications.php

— **Enjeux des attaques physiques dans les pays en voie développement.**
Workshop CFAO Technologies-EDMI.
Université Cheikh Anta Diop de Dakar (Sénégal), Juin, 2022.

Séminiares

— **Cryptographie basée sur les codes et attaque template sur *Classic McEliece*.**
Séminaire Cryptologie et Sécurité du laboratoire GREYC, Caen (France), Octobre, 2021.
https://barbierm01.users.greyc.fr/seminaire_crypto/seminaire2021.html

— **Leakage on *Classic McEliece* decryption.**
Séminaire Cryptologie du laboratoire de Cryptographie de Géométrie Algébrique et Applications, Dakar (Sénégal), Juillet 2021.

Posters

— **Side-Channel Analysis on Code-Based Cryptography.**
Journée de la recherche (JDR) de l'École Doctorale Sciences - Ingénierie - Santé, juin 2023.

— **Template attack and SSA against *classic mceliece*.**
AFRICACRYPT, Fes (Maroc), Juillet, 2022.
https://africacrypt2022.cs.ru.nl/index.html

Bibliographie

[Alb+20] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen et al. « Classic McEliece ». In : *NIST Post-Quantum Cryptography Standardization Project (Round 3)*, https://classic.mceliece.org/ (2020).

[Ara+18] N. Aragon, P. Gaborit, A. Hauteville et J.-P. Tillich. « A new algorithm for solving the rank syndrome decoding problem ». In : *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2018, p. 2421-2425.

[Ara+23] N. Aragon, L. Bidoux, J.-J. Chi-Domínguez, T. Feneuil, P. Gaborit, R. Neveu et M. Rivain. « MIRA : a Digital Signature Scheme based on the MinRank problem and the MPC-in-the-Head paradigm ». In : *arXiv preprint arXiv :2307.08575* (2023).

[Arc+06] C. Archambeau, E. Peeters, F.-X. Standaert et J.-J. Quisquater. « Template attacks in principal subspaces ». In : *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, p. 1-14.

[Bal+23] M Baldi, A Barenghi, L Beckwith, J Biasse, A Esser, K Gaj, K Mohajerani, G Pelosi, E Persichetti et al. « LESS ». In : *Round 1 Additional Signatures* , https://www.less-project.com/ (2023).

[Ban+19] G. Banegas, P. S. Barreto, B. O. Boidje, P.-L. Cayrel, G. N. Dione, K. Gaj, C. T. Gueye, R. Haeussler, J. B. Klamti, O. N'diaye et al. « Dags : Reloaded revisiting dyadic key encapsulation ». In : *Code-Based Cryptography : 7th International Workshop*. 2019, p. 69-85.

[Ban+20] G. Banegas, P. S. Barreto, E. Persichetti et P. Santini. « Designing efficient dyadic operations for cryptographic applications ». In : *Journal of Mathematical Cryptology* (2020), p. 95-109.

[Bar+19] M. Bardet, M. Bertin, A. Couvreur et A. Otmani. « Practical algebraic attack on DAGS ». In : *Code-Based Cryptography*. Springer. 2019, p. 86-101.

[BC18] E. Barelli et A. Couvreur. « An efficient structural attack on NIST submission DAGS ». In : *ASIACRYPT 2018*. Springer. 2018, p. 93-118.

[BCS13] D. J. Bernstein, T. Chou et P. Schwabe. « McBits : fast constant-time code-based cryptography ». In : *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2013, p. 250-272.

[BDL97] D. Boneh, R. A. DeMillo et R. J. Lipton. « On the importance of checking cryptographic protocols for faults ». In : *International conference on the theory and applications of cryptographic techniques*. 1997, p. 37-51.

[Bec+12] A. Becker, A. Joux, A. May et A. Meurer. « Decoding Random Binary Linear Codes in $2^{n/20}$: How 1 + 1 = 0 Improves Information Set Decoding ». In : *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, p. 520-536.

[Bel+18] E. Bellini, F. Caullery, A. Hasikos, M. Manzano et V. Mateu. « Code-based signature schemes from identification protocols in the rank metric ». In : *International Conference on Cryptology and Network Security*. Springer. 2018, p. 277-298.

[Bih97] E. Biham. « A fast new DES implementation in software ». In : *International Workshop on Fast Software Encryption*. Springer. 1997, p. 260-272.

[BMVT78] E. Berlekamp, R. McEliece et H. Van Tilborg. « On the inherent intractability of certain coding problems (corresp.) » In : *IEEE Transactions on Information Theory* 24.3 (1978), p. 384-386.

[Bre01] L. Breiman. « Random forests ». In : *Machine learning* (2001), p. 5-32.

[Buc+16] D. Bucerzan, P.-L. Cayrel, V. Dragoi et T. Richmond. « Improved timing attacks against the secret permutation in the McEliece PKC ». In : *International Journal of Computers Communications & Control* 12.1 (2016), p. 7-25.

[Cay+21] P.-L. Cayrel, B. Colombier, V.-F. Drăgoi, A. Menu et L. Bossuet. « Message-recovery laser fault injection attack on the classic McEliece cryptosystem ». In : *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, p. 438-467.

[CC21] M.-S. Chen et T. Chou. « Classic McEliece on the ARM cortex-M4 ». In : *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), p. 125-148.

[Che95] K. Chen. « A new identification algorithm ». In : *International Conference on Cryptography : Policy and Algorithms*. Springer. 1995, p. 244-249.

[Cho17] T. Chou. « McBits revisited ». In : *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2017, p. 213-231.

[CMJ22] Z. Chen, Y. Ma et J. Jing. « Low-Cost Shuffling Countermeasures Against Side-Channel Attacks for NTT-Based Post-Quantum Cryptography ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.1 (2022), p. 322-326.

[Col+22] B. Colombier, V.-F. Drăgoi, P.-L. Cayrel et V. Grosso. « Profiled Side-channel Attack on Cryptosystems based on the Binary Syndrome Decoding Problem ». In : *IEEE Transactions on Information Forensics and Security* (2022).

[CRR02] S. Chari, J. R. Rao et P. Rohatgi. « Template attacks ». In : *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2002, p. 13-28.

[CS96] F. Chabaud et J. Stern. « The cryptographic security of the syndrome decoding problem for rank distance codes ». In : *Advances in Cryptology—ASIACRYPT*. Springer. 1996, p. 368-381.

[CVEYA10] P.-L. Cayrel, P. Véron et S. M. El Yousfi Alaoui. « A zero-knowledge identification scheme based on the q-ary syndrome decoding problem ». In : *International Workshop on Selected Areas in Cryptography*. Springer. 2010, p. 171-186.

[DAST19] T. Debris-Alazard, N. Sendrier et J.-P. Tillich. « Wave : A new family of trapdoor one-way preimage sampleable functions based on codes ». In : *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2019, p. 21-51.

[DH76] W. Diffie et M. Hellman. « New directions in cryptography ». In : *IEEE Transactions on information theory* 22.6 (1976), p. 644-654.

[EPW10] T. Eisenbarth, C. Paar et B. Weghenkel. « Building a side channel based disassembler ». In : *Transactions on computational science X*. Springer, 2010, p. 78-99.

[Fau+10a] J.-C. Faugère, A. Otmani, L. Perret et J.-P. Tillich. « Algebraic cryptanalysis of McEliece variants with compact keys ». In : *EUROCRYPT 2010*. Springer. 2010, p. 279-298.

[Fau+10b] J.-C. Faugère, A. Otmani, L. Perret et J.-P. Tillich. « Algebraic Cryptanalysis of McEliece Variants with Compact Keys–Toward a Complexity Analysis ». In : *SCC'10*. 2010, p. 45-55.

[Fau+13] J.-C. Faugere, V. Gauthier-Umana, A. Otmani, L. Perret et J.-P. Tillich. « A distinguisher for high-rate McEliece cryptosystems ». In : *IEEE Transactions on Information Theory* (2013), p. 6830-6844.

[Fau+16] J.-C. Faugère, A. Otmani, L. Perret, F. De Portzamparc et J.-P. Tillich. « Structural cryptanalysis of McEliece schemes with compact keys ». In : *DCC* 79.1 (2016), p. 87-112.

[FS86] A. Fiat et A. Shamir. « How to Prove Yourself : Practical Solutions to Identification and Signature Problems. » In : *Crypto*. T. 86. 1986, p. 186-194.

[Gab85] E. M. Gabidulin. « Theory of codes with maximum rank distance ». In : *Problemy Peredachi Informatsii* (1985), p. 3-16.

[GJJ22] Q. Guo, A. Johansson et T. Johansson. « A Key-Recovery Side-Channel Attack on Classic McEliece Implementations ». In : *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), p. 800-827.

[GLRP06] B. Gierlichs, K. Lemke-Rust et C. Paar. « Templates vs. stochastic methods ». In : *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, p. 15-29.

[Gop70] V. D. Goppa. « A new class of linear error-correcting codes ». In : *Probl. Inf. Transm.* 6 (1970), p. 300-304.

[Gro96] L. K. Grover. « A fast quantum mechanical algorithm for database search ». In : *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, p. 212-219.

[GRS15] P. Gaborit, O. Ruatta et J. Schrek. « On the complexity of the rank syndrome decoding problem ». In : *IEEE Transactions on Information Theory* 62.2 (2015), p. 1006-1019.

[GSZ11] P. Gaborit, J. Schrek et G. Zémor. « Full cryptanalysis of the Chen identification protocol ». In : *Post-Quantum Cryptography*. Springer. 2011, p. 35-50.

[Gus+17] B. Gustavo, S. Paolo, M Barreto, B. O. Boidje et P.-L. Cayrel. « DAGS : Key Encapsulation using Dyadic GS Codes ». In : *NIST Submission* (2017).

[Ham50] R. W. Hamming. « Error detecting and error correcting codes ». In : *The Bell system technical journal* 29.2 (1950), p. 147-160.

[Hel72] H Helgert. « Srivastava codes ». In : *IEEE Transactions on Information Theory* 18.2 (1972), p. 292-297.

[HGG20] B. Hettwer, S. Gehrer et T. Güneysu. « Applications of machine learning techniques in side-channel attacks : a survey ». In : *Journal of Cryptographic Engineering* (2020), p. 135-162.

[HHK17] D. Hofheinz, K. Hövelmanns et E. Kiltz. « A modular analysis of the Fujisaki-Okamoto transformation ». In : *Theory of Cryptography Conference*. 2017, p. 341-371.

[JMV01] D. Johnson, A. Menezes et S. Vanstone. « The elliptic curve digital signature algorithm (ECDSA) ». In : *International journal of information security* 1 (2001), p. 36-63.

[Jou13] A. Joux. « A new index calculus algorithm with complexity $l(1/4 + o(1))$ in small characteristic ». In : *International Conference on Selected Areas in Cryptography*. 2013, p. 355-379.

[Ker83] A. Kerckhoffs. « La cryptographie militaire ». In : *Journal des sciences militaires*. 1883, vol. IX, p. 5-38.

[KJJ99] P. Kocher, J. Jaffe et B. Jun. « Differential power analysis ». In : *Annual international cryptology conference*. 1999, p. 388-397.

[KM22] E. Kirshanova et A. May. « Decoding McEliece with a Hint–Secret Goppa Key Parts Reveal Everything ». In : *Security and Cryptography for Networks*. Springer. 2022, p. 3-20.

[Koc96] P. C. Kocher. « Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems ». In : *Annual International Cryptology Conference*. 1996, p. 104-113.

[LB88] P. J. Lee et E. F. Brickell. « An Observation on the Security of McEliece's Public-Key Cryptosystem ». In : *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1988, p. 275-280.

[Leo88] J. S. Leon. « A probabilistic algorithm for computing minimum weights of large error-correcting codes ». In : *IEEE Transactions on Information Theory* (1988), p. 1354-1359.

[Loi06] P. Loidreau. « Properties of codes in rank metric ». In : *arXiv preprint cs/0610057* (2006).

[LS01] P. Loidreau et N. Sendrier. « Weak keys in the McEliece public-key cryptosystem ». In : *IEEE Transactions on Information Theory* 47.3 (2001), p. 1207-1211.

[LTP19] T. S. C. Lau, C. H. Tan et T. F. Prabowo. « Key recovery attacks on some rank metric code-based signatures ». In : *IMACC*. Springer. 2019, p. 215-235.

[LW01] J. H. van Lint et R. M. Wilson. *A Course in Combinatorics*. Cambridge, U.K. ; New York : Cambridge University Press, 2001.

[MB09] R. Misoczki et P. S. Barreto. « Compact McEliece keys from Goppa codes ». In : *SAC 2009*. Springer. 2009, p. 376-392.

[McE78] R. J. McEliece. « A public-key cryptosystem based on algebraic ». In : *Coding Thv* 4244 (1978), p. 114-116.

[MMT11] A. May, A. Meurer et E. Thomae. « Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$ ». In : *International Conference on the Theory and Application of Cryptology and Information Securitys*. Springer, 2011, p. 107-124.

[MO15] A. May et I. 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*. Springer, 2015, p. 203-228.

[MS77] F. MacWilliams et N. Sloane. *The Theory of Error-Correcting Codes North-Holland*. 1977.

[Nie86] H. Niederreiter. « Knapsack-type cryptosystems and algebraic coding theory ». In : *Prob. Contr. Inform. Theory* 15.2 (1986), p. 157-166.

[Nis] https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

[Noj+08] R. Nojima, H. Imai, K. Kobara et K. Morozov. « Semantic security for the McEliece cryptosystem without random oracles ». In : *Designs, Codes and Cryptography* 49 (2008), p. 289-305.

[OC14] C. O'Flynn et Z. D. Chen. « Chipwhisperer : An open-source platform for hardware embedded security research ». In : *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2014, p. 243-260.

[OJ02] A. V. Ourivski et T. Johansson. « New technique for decoding codes in the rank metric and its cryptography applications ». In : *Problems of Information Transmission* (2002), p. 237-246.

[Pat75] N. Patterson. « The algebraic decoding of Goppa codes ». In : *IEEE Transactions on Information Theory* 21.2 (1975), p. 203-207.

[Per12] E. Persichetti. « Compact McEliece keys based on quasi-dyadic Srivastava codes ». In : *Journal of Mathematical Cryptology* 6.2 (2012), p. 149-169.

[Pra62] E. Prange. « The use of information sets in decoding cyclic codes ». In : *IRE Transactions on Information Theory* 8.5 (1962), p. 5-9.

[RO04] C. Rechberger et E. Oswald. « Practical template attacks ». In : *International Workshop on Information Security Applications*. Springer. 2004, p. 440-456.

[RS60] I. S. Reed et G. Solomon. « Polynomial codes over certain finite fields ». In : *Journal of the society for industrial and applied mathematics* 8.2 (1960), p. 300-304.

[RSA77] R. L. Rivest, A. Shamir et L. Adleman. *On Digital Signatures and Public-Key Cryptosystems*. Rapp. tech. Massachusetts Inst of Tech Cambridge Lab for Computer Science, 1977.

[RSA78] R. L. Rivest, A. Shamir et L. Adleman. « A method for obtaining digital signatures and public-key cryptosystems ». In : *Communications of the ACM* 21.2 (1978), p. 120-126.

[SA08] F.-X. Standaert et C. Archambeau. « Using subspace-based template attacks to compare and combine power and electromagnetic information leakages ». In : *Cryptographic Hardware and Embedded Systems*. Springer. 2008, p. 411-425.

[Sec+23a] B. Seck, P.-L. Cayrel, I. Diop et M. Barbier. « Cryptanalysis of a Code-Based Identification Scheme Presented in CANS 2018 ». In : *Cryptography, Codes and Cyber Security : First International Conference, I4CS 2022*. Springer. 2023, p. 3-19.

[Sec+23b] B. Seck, P.-L. Cayrel, I. Diop, V.-F. Dragoi, K. Couzon, B. Colombier et V. Grosso. « Key-Recovery by Side-Channel Information on the Matrix-Vector Product in Code-Based Cryptosystems ». In : *Information Security and Cryptology*. Springer. 2023, p. 219-234.

[Sec+23c] B. Seck, P.-L. Cayrel, V.-F. Dragoi, I. Diop, M. Barbier, J. B. Klamti, V. Grosso et B. Colombier. « A Side-Channel Attack Against Classic McEliece When Loading the Goppa Polynomial ». In : *International Conference on Cryptology in Africa*. Springer. 2023, p. 105-125.

[Sec+23d] B. Seck, C. T. Gueye, G. N. Dione, J. B. Klamti, P.-L. Cayrel, I. Diop et O. Ndiaye. « Software Implementation of a Code-Based Key Encapsulation Mechanism from Binary QD Generalized Srivastava Codes ». In : *Code-Based Cryptography*. Springer. 2023, p. 77-89.

[Sha48] C. E. Shannon. « A mathematical theory of communication ». In : *The Bell system technical journal* 27.3 (1948), p. 379-423.

[Sho94] P. W. Shor. « Algorithms for quantum computation : discrete logarithms and factoring ». In : *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, p. 124-134.

[Ste88] J. Stern. « A method for finding codewords of small weight ». In : *International Colloquium on Coding Theory and Applications*. T. 388. 1988, p. 106-113.

[Str10] F. Strenzke. « A timing attack against the secret permutation in the McEliece PKC ». In : *International Workshop on Post-Quantum Cryptography*. Springer. 2010, p. 95-107.

[Str13] F. Strenzke. « Timing attacks against the syndrome inversion in code-based cryptosystems ». In : *International Workshop on Post-Quantum Cryptography*. Springer. 2013, p. 217-230.

[Tan+10] A. Tanatmis, S. Ruzika, H. W. Hamacher, M. Punekar, F. Kienle et N. Wehn. « A separation algorithm for improved LP-decoding of linear block codes ». In : *IEEE Transactions on Information Theory* (2010), p. 3277-3289.

[Vér97] P. Véron. « Improved identification schemes based on error-correcting codes ». In : *Applicable Algebra in Engineering, Communication and Computing* 8 (1997), p. 57-69.

[WSN18] W. Wang, J. Szefer et R. Niederhagen. « FPGA-based Niederreiter cryptosystem using binary Goppa codes ». In : *International Conference on Post-Quantum Cryptography*. Springer. 2018, p. 77-98.