RAPPORT DE STAGE DE FIN D'ÉTUDES

Master 2 Algèbre Appliquée

Signatures fondées sur les codes et application à la construction de primitives cryptographiques évoluées

---

Par : El-Asri Noureddine

Sous la direction de : Ayoub Otmani et Adeline Roux-Langlois

Structure : LITIS, Université de Rouen Normandie

11/09/2024

Remerciements

Avant de commencer, je tiens à remercier Monsieur Ayoub Otmani, mon tuteur de stage, pour ses conseils et son suivi tout au long de cette expérience. Sa disponibilité et ses remarques m'ont beaucoup aidé à avancer dans mon travail et à mieux comprendre la cryptographie basée sur les codes correcteurs. Je suis vraiment reconnaissant pour tout ce qu'il m'a appris durant ces mois de stage.

Je remercie également Madame Adeline Roux-Langlois, qui m'a accueilli au laboratoire GREYC pendant une semaine. Elle a pris le temps de m'expliquer les bases de la cryptographie basée sur les réseaux (lattices), ce qui m'a permis d'élargir mes connaissances dans ce domaine.

Enfin, je souhaite aussi remercier mes deux parents pour leur soutien inconditionnel durant toute l'année universitaire.

Table des matières

Introduction

L'avènement possible des ordinateurs quantiques menace les fondements de la cryptographie actuelle, rendant vulnérables les algorithmes traditionnels basés sur des problèmes non difficiles sur une machine quantique. Un exemple étant l'algorithme quantique de Shor [18], qui factorise efficacement les entiers. C'est là qu'intervient la cryptographie post-quantique qui se présente comme un rempart essentiel pour préserver la sécurité des communications et des données sensibles face aux machines quantiques.

Le **NIST** (*National Institute of Standards and Technology, organisme de standardisation américain*) organise depuis 2016 un concours international en vue de la standardisation d'algorithmes cryptographiques post-quantiques. Il s'avère que les problèmes venant de la théorie codes correcteurs d'erreurs, comme le SDP (Problème du Décodage par Syndrome) A.2, peuvent être utilisés pour faire des algorithmes cryptographiques post-quantiques.
Parmi les algorithmes cryptographiques post-quantiques souhaités par le **NIST**, on trouve les schémas de signatures B.4. Ces schémas sont cruciaux pour assurer l'authenticité et l'intégrité des informations dans divers domaines, tels que les transactions financières en ligne, la technologie de la blockchain, les mises à jour logicielles, et les communications sécurisées entre objets connectés. Il est donc primordial de disposer de schémas sécurisés contre les menaces pour prévenir les attaques qui pourraient compromettre l'identité numérique et la confiance dans les systèmes cryptographiques actuels.

But du stage

L'objectif principal de mon stage était de tester la faisabilité d'un possible nouveau schéma de signature suggéré par Ayoub Otmani (voir 2.2).

Dans un premier temps, j'ai réalisé un état de l'art sur la théorie des codes correcteurs d'erreurs (voir annexe A), et j'ai également implémenté les algorithmes de décodage de chaque code présenté dans cette annexe (voir annexe C).

Le problème sur lequel repose le schéma 2.1 fait intervenir le smoothing parameter [1] qu'on trouve dans la cryptographie basée sur les lattices [2]. Ainsi, j'ai passé une semaine au sein du laboratoire **GREYC** à l'Université de Caen Normandie sous la direction d'Adeline Roux-Langlois pour comprendre et me familiariser avec la cryptographie basée sur les lattices.

Puis je me suis concentré sur le but de mon stage : vérifier un possible schéma de signature. Ce schéma de signature est basé sur un problème difficile de la théorie des codes correcteurs (voir 2.1). Pour tester sa validité, j'ai effectué plusieurs tests en utilisant le logiciel **Magma**, et en utilisant les algorithmes que j'ai implémentés en annexe *C*. Nous avons essayé plusieurs adaptations intéressantes, et en conséquence, plusieurs versions de ce possible schéma ont vu le jour (voir 2).

Puis, à fur et mesure, j'ai commencé à m'intéresser au **décodage statistique**, en particulier, des codes **QC-LDPC** [3] (voir 2.6).

1. paramètre de lissage
2. réseaux
3. Quasi-Cyclic Low-Density Parity-Check

Contributions et perspectives

J'ai d'abord essayé une approche utilisant les matrices de parité d'un code de **Reed-Solomon**. Après avoir effectué plusieurs tests, bien que cette approche n'ait pas abouti, elle nous a permis de comprendre pourquoi cela ne fonctionnait pas.
J'ai proposé ensuite de travailler avec des codes **LDPC**, qui sont facilement décodables en utilisant de l'algorithme bit flipping. J'ai alors implémenté ces tests en utilisant **Magma**. Malheureusement, un problème de ces codes est qu'il est difficile de leur trouver une **matrice génératrice** qui correspondait à nous besoin, ce qui limite leur efficacité.
On a ensuite utilisé des codes LDPC **quasi-circulants** (**QC-LDPC**), mais les tests ont montré que cette approche n'était pas viable non plus.
En travaillant sur ces codes **QC-LDPC**, nous avons découvert qu'un ensemble particulier de ces codes pouvait être décodé en utilisant le décodage statistique. J'ai implémenté donc des tests pour démontrer cela.
Un autre schéma potentiel de signature, basé sur le problème mentionné en 2.1, a été suggéré par Ayoub Otmani. Ce schéma prend en compte les résultats qu'on a obtenus durant mon stage. Malheureusement, je n'ai pas eu le temps de tester cette approche.
En ce qui concerne le **décodage statistique**, il est intéressant de savoir si on peut généraliser l'approche à l'ensemble des codes **QC-LDPC** ou pas. Pour une question de temps, je n'ai pas exploré ce chemin.

Chapitre 1

Contexte général du stage

J'ai effectué mon stage au sein du **LITIS** (**Laboratoire d'informatique, de traitement de l'information et des systèmes**) à l'université de Rouen Normandie dans l'équipe **CA** [1], sous la direction de Ayoub Otmani et Adeline Roux-Langlois. Mon stage a été financé par la *Fédération Normande de Recherche en Sciences et Technologies de l'Information et de la Communication* **Normastic**.

Normastic en quelques mots

Normastic est une fédération de recherche **CNRS** (FR n°3638) associant le **GREYC** et le **LITIS** et ayant vocation à réunir les chercheurs normands dans le domaine des **STIC** [2].
Le but de la fédération est de promouvoir et de développer les synergies scientifiques entre laboratoires de la fédération et de contribuer à améliorer la qualité, la visibilité et l'attractivité des recherches normandes dans le domaine des sciences et technologies de l'information.
Normastic est rattaché principalement à l'*Institut des sciences de l'information et de leurs interactions* (**INS2I**) du **CNRS**. Elle dépend secondairement de l'*Institut des sciences de l'ingénierie et des systèmes* (**INSIS**). Les sections **CNRS** de rattachement sont : 6, 7, 8.

LITIS

Le **LITIS**, **Laboratoire d'informatique, de traitement de l'information et des systèmes**, est une unité de recherche (UR 4108) en sciences et technologies de l'information rattachée à l'*Université de Rouen Normandie* (**URN**), l'*Université Le Havre Normandie* (**ULHN**) et l'*Institut National des Sciences Appliquées de Rouen Normandie* (**INSARN**). Le **LITIS** est né en janvier 2006 de la volonté des membres des laboratoires **STIC** de Haute-Normandie, et des trois établissements de tutelle, d'unir leurs forces pour valoriser les synergies existantes et renforcer leur visibilité (historique).

Comprendre la nature profonde de l'information et sa représentation est au cœur du projet scientifique du **LITIS**, qui couvre un large spectre des **STIC**, de la recherche fondamentale aux domaines appliqués, la démarche du **LITIS** est résolument pluridisciplinaire, associant praticiens et théoriciens à la jonction de l'informatique, de l'intelligence artificielle, du traitement du signal et des images et des mathématiques, avec des applications dans les systèmes de mobilité intelligents, le traitement de l'information en santé et la valorisation du patrimoine.
Depuis sa création, le **LITIS** est structuré en sept équipes dont l'équipe Combinatoire et algorithmes (**CA**) au sein duquel j'ai effectué le stage.

1. Combinatoire et Algorithmes
2. Sciences et technologies de l'information et de la communication

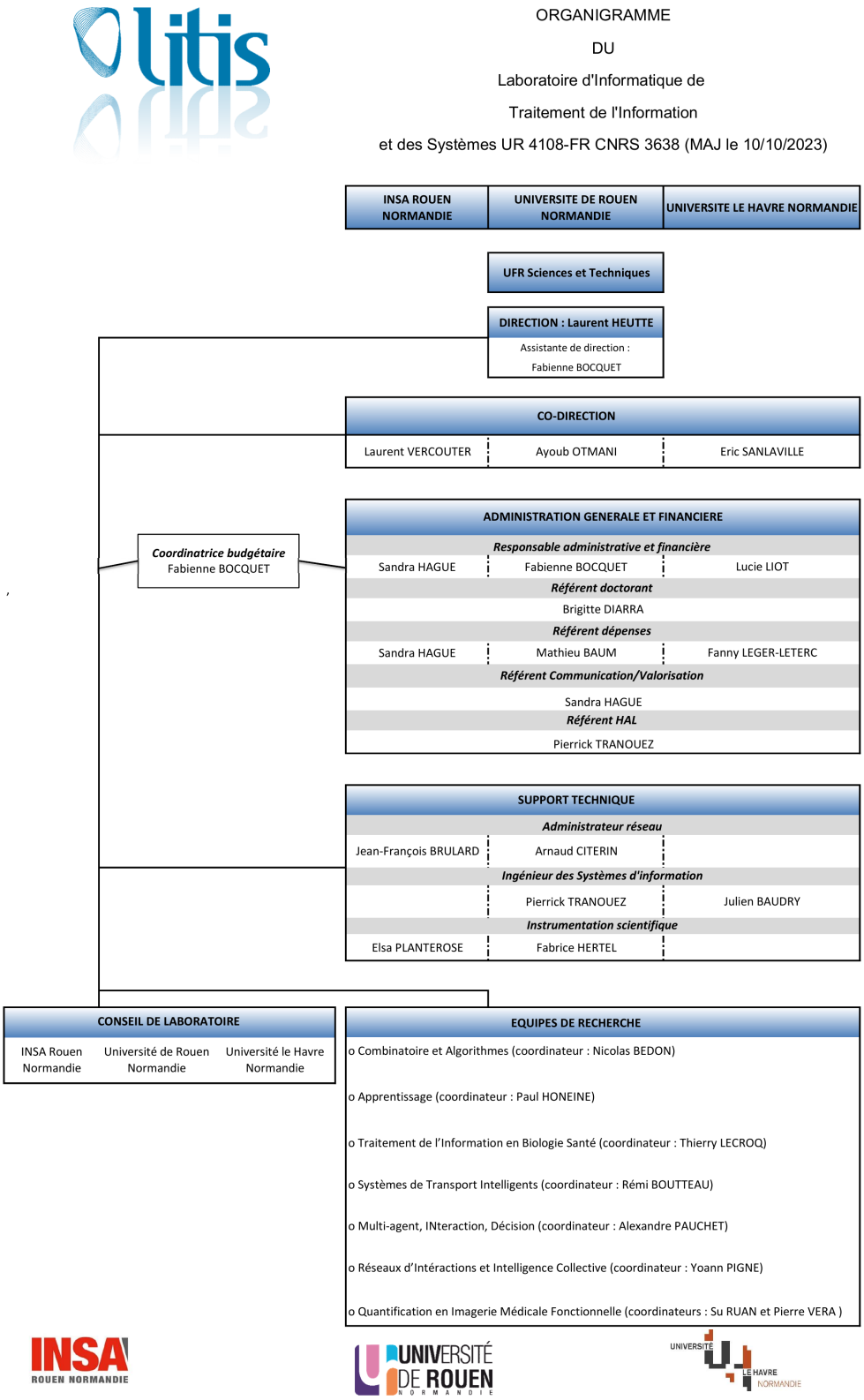


FIGURE 1.1 – Organigramme du **LITIS** [2]

Équipe Combinatoire et Algorithmes [2]

Responsable d'équipe : Magali BARDET

L'équipe « Combinatoire et algorithmes » du **LITIS** s'intéresse à l'étude des aspects fondamentaux des algorithmes ou problèmes informatiques : le travail est centré autour de l'étude combinatoire et algorithmique de modèles de nature algébrique utilisés pour le traitement de l'information (mots, monoïdes libres, automates, séries génératrices, systèmes polynomiaux). L'équipe applique aussi ses concepts, méthodes et résultats en cryptographie, traitement des documents arborescents, validation temps-réel, etc.

Les résultats de l'équipe se répartissent dans ses principales thématiques :

- Combinatoire
- Cryptographie et calcul formel
- Théorie des langages et automates

Approches :

- Étude structurelle et combinatoire des modèles algébriques (mots, monoïde libre, polynômes) ; classification
- Étude algorithmique effective (automates finis, algèbres, logique formelle, systèmes de calcul symbolique)
- Extension des modèles existants et construction de nouveaux modèles (automates d'arbres, d'ordres, algèbres de Hopf combinatoire, information quantique)
- Codes correcteurs d'erreurs, cryptographie

L'équipe spécialisée dans l'étude des codes correcteurs d'erreurs et de la cryptographie s'intéresse à la conception et à la cryptanalyse de diverses primitives cryptographiques. Ce qui est intéressant, c'est que la cryptographie basée sur les codes correcteurs fait partie de ce qu'on appelle la cryptographie post-quantique.

Méthode et organisation du rapport

La présentation de mes travaux durant le stage (2) est organisée de la manière suivante :

- Dans un premier temps, j'ai introduit le problème sur lequel nous souhaitions construire un schéma de signature.
- Ensuite, j'ai présenté les différentes approches que nous avons tentées durant ce stage. Chaque approche est organisée de la même manière, à savoir :
  1. Une présentation théorique de l'approche.
  2. Implémentation des tests réalisés sur **Magma**.
  3. Les résultats des tests et leur interprétation.
  4. Conclusion de l'approche.

Les annexes A, C et B peuvent être ignorées si le lecteur est un peu à l'aise avec la théorie des codes correcteurs et le cryptosystème de McEliece. Dans l'annexe D j'ai présenté quelques algorithmes en **Magma** que j'ai utilisés pour les tests, par exemple un algorithme qui calcule la distance de Gilbert-Varshamov. Cette dernière peut être ignorée aussi.
Pour chaque approche, j'ai intégré le code dans **Magma** afin de réaliser les tests. Voir la table des matières ou le début du chapitre 2 pour un peu plus de détails sur les annexes.

Notations

— n, k, L, w et t désignent des entiers naturels ;
— $\mathbb{F}_q$: désigne un corps fini à q éléments ;
— $\mathbb{F}_q^{k\times n}$: désigne l'espace des matrices à k lignes et n colonnes sur le corps $\mathbb{F}_q$;
— Les vecteurs seront notés avec des lettres grasses et en notation ligne : $\mathbf{a} := (a_1, a_2, .., a_n) \in \mathbb{F}_q^n$;
— Pour un vecteur $\mathbf{a} = (a_1, a_2, .., a_n) \in \mathbb{F}_q^n$, on désigne par $\mathbf{a}^T$ son transposé $\mathbf{a}^T := \begin{pmatrix} a_1 \\ a_2 \\ : \\ a_n \end{pmatrix}$;
— Si $f \in \mathbb{F}_q[X]$ est un polynôme, et $\mathbf{a} \in \mathbb{F}_q^n$, on note $f(\mathbf{a}) := (f(a_1), f(a_2), .., f(a_n)) \in \mathbb{F}_q^n$;
— Pour une matrice $A = (a_{i,j})_{\substack{1\leq i\leq k \\ 1\leq j\leq n}} \in \mathbb{F}_q^{k\times n}$, on désigne par A^T sa transposée

$$A^T := (a_{j,i})_{\substack{1\leq i\leq k \\ 1\leq j\leq n}} \in \mathbb{F}_q^{n\times k}$$

— $x := y$ signifie que x est défini comme étant égale à y ;
— $x \leftarrow y$ signifie que x prend la valeur de y ;
— Pour deux entiers $n \leq m$, $[|n, m|]$ est définie comme l'ensemble $\{n, n+1, n+2, .., m-1, m\}$;
— $h_q(x) := -(1-x)\log_q(1-x) - x\log_q(\frac{x}{q-1})$ est la fonction entropie en base q ;
— Δ est utilisé pour la distance statistique définie ici 2.1 ;
— Pour deux fonctions $f, g : \mathbb{N} \to \mathbb{R}$, on a :

1. $f = O(g)$ ou $g = \Omega(f)$ lorsque $\exists N \in \mathbb{N}$ et $c > 0$ constante tels que $\forall n \geq N,\ |f(n)| \leq c.|g(n)|$;
2. $f(n) = poly(n)$ si $\exists c > 0$ constante tel que $f = O(n^c)$;

— Étant donné un ensemble Ω, on désigne par $x \leftarrow U(\Omega)$ que x est choisi uniformément dans Ω ;
— $\mathbb{P}$: désigne une mesure de probabilité ;
— Pour un vecteur $\mathbf{e} \in \mathbb{F}_q^n$, on désigne par $w_H(\mathbf{e})$ son poids de Hamming (voir A) ;
— Pour $E = (E_{i,j})_{i,j} \in \mathbb{F}_q^{k\times n}$, on note $w_H(E) = \max_{j\in[|1,n|]}(w_H((E_{1,j}, E_{2,j}, .., E_{k,j})))$
— I_n est la matrice identité de $\mathbb{F}_q^{n\times n}$;
— Pour $A \in \mathbb{F}_q^{k\times n}$, $Rank(A)$ ou $rg(A)$ désignent le rang de la matrice A ;
— Pour un vecteur $\mathbf{a} \in \mathbb{F}_q^n$, on note $a_1, a_2, .., a_n$ ses coordonnées ;
— Pour un vecteur $\mathbf{a} \in \mathbb{F}_q^n$, on note $Supp(\mathbf{a}) := \{i \in [|1, n|] \mid a_i \neq 0\}$ son support ;
— Pour un ensemble Ω, on note $Card(\Omega)$, $card(\Omega)$ ou $|\Omega|$ son cardinal (un élément de $\mathbb{N} \cup \{+\infty\}$) ;
— $d_{GV}(n, k, q)$ (ou $d_{GV}(n, k)$ lorsque $q = 2^s$, $s \in \mathbb{N}^*$) est la distance de Gilbert-Varshamov défini ici A.1. Dans les expériences en **Magma**, on utilise **binary_gv**(n, k), car on travaille dans un corps de caractéristique pair. Voir algorithme (D.1) qui le calcule ;
— Pour $x \in \mathbb{R}$, $\lfloor x \rfloor$ désigne la partie entière inférieure de x ;
— Pour $G \in \mathbb{F}_q^{k\times n}$ avec $k \leq n$, on note C_G ou $C(G)$ le code généré par G : $C_G := \{\mathbf{x}.G \mid \mathbf{x} \in \mathbb{F}_q^k\}$.
— Pour $\mathbf{x}$ et $\mathbf{y}$ deux vecteurs de $\mathbb{F}_q^n$, $< \mathbf{x}|\mathbf{y} >:= \sum_{i=1}^n x_i.y_i$;
— En ce qui concerne les commandes en **Magma**, consultez ([3]) ;

Chapitre 2

Travaux effectués

Avant d'aborder le cœur du sujet, je souhaite vous informer que j'ai intégré :

— En annexe A, une introduction aux codes correcteurs d'erreurs, aux différents codes ainsi que leurs algorithmes de décodage comme : les codes de **Reed-Solomon généralisés**, les codes **BCH**, les codes de **Goppa** ainsi que les codes **LDPC** ;

— En annexe C, les implémentations en **Magma** des codes présentés ci-dessus ainsi que certains codes que je vais introduire dans ce chapitre comme le **décodage complet** (2.4.3) ;

— En annexe B, une introduction aux notions cryptographiques comme les algorithmes de **chiffrement** et les algorithmes de **signature**. J'ai aussi présenté les différentes notions de cryptographie à base de codes correcteurs ;

— En annexe D, les implémentations en **Magma** de différents algorithmes utilisés dans ce rapport ou utilisés durant mon stage.

2.1 Un problème difficile

Définition 1 (Distance statistique)**.** *Soient D_1 et D_2 deux **distributions** de probabilités définies sur un ensemble Ω. Alors, on définit leur **distance statistique**, notée $\Delta(D_1, D_2)$, par :*

$$\Delta(D_1, D_2) := \frac{1}{2} \sum_{x \in \Omega} |D_1(x) - D_2(x)|$$

Fait : La distance statistique est bien une distance sur l'ensemble des distributions définies sur Ω, i.e, elle vérifie :

— $\Delta(D_1, D_2) \geq 0$ (Positive)

— $\Delta(D_1, D_2) = \Delta(D_2, D_1)$ (Symétrique)

— $\Delta(D_1, D_3) \leq \Delta(D_3, D_2) + \Delta(D_1, D_2)$ (Inégalité triangulaire)

Définition 2 (Fonction négligeable)**.** *Une **fonction négligeable** est une fonction $f : \mathbb{N} \to \mathbb{R}$ telle que $\forall c \in \mathbb{N},\ \exists N \in \mathbb{N}$ tel que $\forall n \geq N,\ |f(n)| \leq \frac{1}{n^c}$*

Définition 3. *Deux ensembles de distributions $\{D_1^n\}_n$ et $\{D_2^n\}_n$ sont dits **statistiquement indistinguables** lorsque $\Delta(D_1^n, D_2^n)$ est une fonction négligeable en n.*

L'hypothèse d'indistinguabilité statistique est très forte. En cryptographie, on préfère utiliser une autre notion :

Définition 4 (indistinguabilité calculatoire)**.** *Deux familles de distributions $\{D_1^n\}_n$ et $\{D_2^n\}_n$ sur un ensemble Ω sont dites **calculatoirement indistinguables**, et on note $D_1 \sim D_2$, si, pour tout algorithme*

probabiliste en temps polynomial $\mathcal{A}$ retournant 0 ou 1, $|\mathbb{P}_{x\leftarrow D_1^n}(\mathcal{A}(x) = 1) - \mathbb{P}_{x\leftarrow D_2^n}(\mathcal{A}(x) = 1)|$ est négligeable en n.

Toute la construction qui va suivre se basera sur deux choses. D'abord, le **problème de décodage par syndrome** :

Définition 5 (Problème de décodage par syndrome (SDP))**.** *Le problème de décodage par syndrome $SDP_{n,k,w}$ est le suivant :*

- ***Entrée :*** *$H \in \mathbb{F}_q^{(n-k)\times n}$ de rang plein, $\mathbf{s} \in \mathbb{F}_q^{n-k}$ et $w \in [|0, n|]$ un poids ;*
- ***Sortie :*** *Existe-t-il $\mathbf{e} \in \mathbb{F}_q^n$ tel que $w_H(\mathbf{e}) \leq w$ et $H.\mathbf{e}^T = \mathbf{s}^T$.*

Voir A.2 pour en savoir plus.

Puis sur l'hypothèse suivante :
Soient $0 < k \leq n < L$ des entiers naturels, $H \in \mathbb{F}_q^{k\times L}$ une matrice quelconque et $E \in \mathbb{F}_q^{L\times n}$ une matrice dont chaque colonne est de poids $w \in [|0, L|]$. Alors, à condition que w est assez élevé, la **distribution** des matrices $U := H.E$ est calculatoirement indistinguable d'une distribution uniforme.
Remarque : Cette proposition résulte d'un travail sur les codes correcteurs et les réseaux (lattices), où les propriétés du paramètre de lissage (smoothing parameter) ont été exploitées [8].

Définition 6 (Schéma de signature)**.** *Soit λ un paramètre de sécurité. Un schéma de signature est l'ensemble de trois algorithmes $(KeyGen, Sign, Verify)$ tel que :*

- *$KeyGen(\lambda)$: est un algorithme qui prend en entrée un paramètre de sécurité λ et qui sort deux clés, une clé de signature sk et une clé de vérification vk ;*
- *$Sign(sk, m)$: est un algorithme qui prend en entrée un message m et le secret sk et qui sort une signature σ de m ;*
- *$Verify(pk, m, \sigma)$: est un algorithme qui prend en entrée une signature σ, le message signé m et la clé de vérification vk et qui sort $v \in \{0, 1\}$.*

Pour en connaitre plus sur les schémas de signature, voir B.4.

Définition 7 (Signature de type hache et signe)**.** *On dispose de $f : \mathcal{A} \to \mathcal{D}$, une fonction à sens unique à trappe, de trappe sk, c'est-à-dire :*

- *Pour toute entrée $x \in \mathcal{A}$, $f(x)$ se calcule facilement (en temps polynomial),*
- *Aucun algorithme ne connaissant sk ne peut trouver un élément de $f^{-1}(\{f(x)\})$ en temps polynomial,*
- *On calcule facilement un élément de $f^{-1}(f(x))$ avec sk,*

Soit $\mathcal{H}$ une fonction de hachage cryptographique. La signature d'un message m et sa vérification consistent simplement en :

- *La signature de m est un élément de $f^{-1}(\{\mathcal{H}(m)\})$ qui se calcule facilement en connaissant sk,*
- *La vérification consiste simplement à vérifier qu'on a bien $f(\sigma) = \mathcal{H}(m)$.*

Le but est d'exploiter le problème de **décodage par syndrome** et l'hypothèse ci-dessus, pour aboutir à un nouveau schéma de signature, un schéma de signature de type hache et signe. Le schéma général est le suivant :

— On génère des entiers naturels $0 < k \leq n < L$, et un poids $w \in [|0, L|]$ assez élevé,
— Pour $H \in \mathbb{F}_q^{(n-k)\times L}$ on prend une matrice de parité d'un code $[L, L-n+k]_q$ qu'on sait décoder (Un code de **Reed-Solomon** par exemple). Pour $E \in \mathbb{F}_q^{(L+n)\times n}$, on choisit une matrice aléatoire dont les colonnes sont de poids w,
— (E, H) constituent la clé de signature, et $U := H.E$ la clé de vérification.

Soit $\mathcal{H}$ une fonction de hachage cryptographique à valeurs dans $\mathbb{F}_q^{n-k}$. Pour signer un message m, on fait simplement :

— On calcule $\mathbf{s} := \mathcal{H}(m)$,
— On trouve, en utilisant un algorithme de décodage $\mathbf{e}' \in \mathbb{F}_q^L$ tel que $H.\mathbf{e}'^T = s^T$ et de poids petit,
— puis on trouve $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{e}'^T = E.\mathbf{e}^T$ et $t := w_H(\mathbf{e})$ assez élevé,
— La signature de m est $\sigma := (\sigma_1, \sigma_2) = (\mathbf{e}, t)$,

Pour vérifier la signature, il suffit de vérifier que $U.{\sigma_\mathbf{1}}^T = s^T$ et $w_H(\sigma_\mathbf{1}) \leq \sigma_\mathbf{2}$,

2.2 Première approche

Dans cette approche, nous avons essayé la construction suivante :

Les deux matrices (H, E) de la **clé de signature** [1] seront construites de la manière suivante :

- Pour H nous avons pris un code de **Reed-Solomon** [2] de type $[n+L, k+L]_q$ où $k \leq n << L$ (H est une matrice de parité).
- Pour E nous avons pris une matrice de la forme $E = \begin{bmatrix} E^* \\ I_n \end{bmatrix}$ où $E^* \in \mathbb{F}_q^{L \times n}$ est aléatoire dont chaque colonne est de poids de Hamming $w := L^a - 1$ avec $0 < a < 1$.

Puis, on prend $U := H.E$ et $t \gtrsim d_{GV}(n, k, q)$ [3] comme **clé de vérification**.

Pour **signer** $\mathbf{s} \in \mathbb{F}_q^{n-k}$, on procède comme la suite :

1. On utilise l'**algorithme de décodage** des codes de **Reed-Solomon** (C.1) pour trouver $\mathbf{e}' \in \mathbb{F}_q^{L+n}$ tel que $H.\mathbf{e}'^T = \mathbf{s}^T$ et $w_H(\mathbf{e}') \leq w.t$;
2. Puis on trouve $\mathbf{e} \in \mathbb{F}_q^n$ tel que $\mathbf{e}' = \mathbf{e}.E^T$ et $w_H(\mathbf{e}) \leq t$;

Pour **vérifier** la signature, il suffit de montrer que $U.\mathbf{e}^T = \mathbf{s}^T$ et $w_H(\mathbf{e}) \leq t$.

Or, nous avons démontré qu'avec de tels paramètres, il est pas possible d'aboutir à une signature.

Notons $D \in \mathbb{F}_q^{(L-n)\times L}$ une matrice de parité du code généré par E^T. Pour trouver $\mathbf{e}$ que l'on cherche, on va essayer de trouver un vecteur $\mathbf{e}'$ qui soit un mot du code généré par E^T. Autrement dit, $\mathbf{e}' \in \mathbb{F}_q^{n+L}$ doit vérifier :

$$H.\mathbf{e}'^T = \mathbf{s}^T \tag{2.1}$$

$$w_H(\mathbf{e}') \leq w.t \tag{2.2}$$

$$D.\mathbf{e}'^T = \mathbf{0}^T \tag{2.3}$$

i.e (2.1, 2.3) :

$$\begin{bmatrix} H \\ D \end{bmatrix} .\mathbf{e}'^T = \begin{bmatrix} \mathbf{s}^T \\ \mathbf{0}^T \end{bmatrix}$$

et (2.2) :

$$w_H(\mathbf{e}') \leq w.t$$

On remarque : Trouver $\mathbf{e}'$ revient au problème de décodage par syndrome avec la matrice de parité $\begin{bmatrix} H \\ D \end{bmatrix}$. Comme le montre l'analyse suivante, on ne peut trouver $\mathbf{e}'$ par la méthode classique (décodage des codes de Reed Solomon).

Comme H est une matrice de parité d'un code de Reed-Solomon $RS(n+L, k+L)$, alors la distance minimale de ce code est $d_{min} = n - k + 1$.

1. Voir définition d'un schéma de signature ici B.4
2. Voir la définition d'un code de Reed-Solomon ici A.3.1
3. Voir définition de la distance de Gilbert-Varshamov ici A.1

Ainsi, pour bien trouver $\mathbf{e}'$, on doit [4] avoir :

$$t.w < \frac{n-k}{2} \tag{2.4}$$

D'autre part :

$$\binom{n}{d_{GV}(n,k,q)} (q-1)^{d_{GV}(n,k,q)} \sim q^{n-k} \qquad \text{voir } A.1$$

Ce qui correspond approximativement [5] à :

$$2^{n.h_2(\delta)}.2^{d_{GV}(n,k,q).log_2(q-1)} \sim 2^{(n-k)log_2(q)} \qquad \text{avec } \delta = \frac{d_{GV}(n,k,q)}{n}$$

On a donc :

$$n.h_2(\delta) \sim -d_{GV}(n,k,q).log_2(q-1) + (n-k).log_2(q)$$

Si on pose $R = \frac{k}{n}$, on aura :

$$h_2(\delta) \sim -\delta.log_2(q-1) + (1-R).log_2(q)$$

i.e

$$1 - R \sim \delta.\frac{log_2(q-1)}{log_2(q)} + \frac{h_2(\delta)}{log_2(q)}$$

Qui ne dépend pas de L, et comme (2.4) $\implies \delta.n.w < \frac{n-k}{2}$, on a :

$$w < \frac{n-k}{2}.\frac{1}{\delta.n}$$

i.e

$$w < \frac{1}{2.\delta}.(1-R)$$

Ce qui veut dire que $w = O(1)$ et qu'il ne dépend pas de L. Cela contredit le fait que $w = L^a - 1$ avec $0 < a < 1$. ■

Conclusion : Il n'est donc pas possible de trouver e' lorsque $t \gtrsim d_{GV}(n,k,q)$ avec cette approche.

4. Voir la définition de la capacité de décodage d'un code ici A.1
5. Voir https ://fr.wikipedia.org/wiki/Coefficient_binomial section Encadrement et approximations

2.3 Deuxième approche

Ici, nous avons essayé d'exploiter la forme de $E = \begin{bmatrix} E^* \\ I_n \end{bmatrix}$ (plus précisément, nous exploitons le fait que E^T est de rang plein).
Comme $E = ({E^*}^T | I_n)$, alors on peut prendre $D := (-I_L | E^*)$ (A.1).

En analysant les deux équations (2.1) et (2.3), et en notant $\mathbf{e'} := (e'_1, e'_2, .., e'_{n+L})$ on trouve :

— (2.3) : On peut trouver $\epsilon_\mathbf{1} := (e'_1, .., e'_L)$ en fonction des $\epsilon_\mathbf{2} := (e'_{L+1}, .., e'_{L+n})$. Autrement dit, il suffit de trouver les n coordonnées $e'_{L+1}, .., e'_{L+n}$, et on trouve les autres avec :

$$\begin{bmatrix} e'_1 \\ : \\ e'_L \end{bmatrix} = E^* . \begin{bmatrix} e'_{L+1} \\ : \\ e'_{L+n} \end{bmatrix} \tag{2.5}$$

— (2.1) : Si on note $H = (H_1 | H_2)$, avec $H_1 \in \mathbb{F}_q^{(n-k)\times L}$ et $H_2 \in \mathbb{F}_q^{(n-k)\times n}$, alors on a

$$H_1 . {\epsilon_\mathbf{1}}^T + H_2 . {\epsilon_\mathbf{2}}^T = s^T$$

de (2.5) on a :

$$H_1 . E^* . {\epsilon_\mathbf{2}}^T + H_2 . {\epsilon_\mathbf{2}}^T = s^T$$

i.e :

$$(H_1 . E^* + H_2) . {\epsilon_\mathbf{2}}^T = s^T \tag{2.6}$$

On note $\Sigma := H_1 . E^* + H_2 \in \mathbb{F}_q^{(n-k)\times n}$.

Ce qu'il faut maintenant, c'est de trouver un $\mathbf{e'} := (\epsilon_\mathbf{1}, \epsilon_\mathbf{2})$ vérifiant l'inégalité (2.2). À remarquer que, d'après le **théorème du rang**, il existe q^k possibilité pour $\epsilon_\mathbf{2}$ vérifiant (2.6), i.e, q^k possibilités pour e'. Il sera donc difficile de trouver un tel e' vérifiant en plus l'inégalité (2.2).
D'autre part, on a :

$$\begin{aligned} w_H(e') &= w_H(\epsilon_\mathbf{1}) + w_H(\epsilon_\mathbf{2}) \\ &= w_H(E * . {\epsilon_\mathbf{2}}^T) + w_H(\epsilon_\mathbf{2}) && \text{par (2.5)} \\ &\leq w_H(E^*) . w_H(\epsilon_\mathbf{2}) + w_H(\epsilon_\mathbf{2}) \\ &= w . w_H(\epsilon_\mathbf{2}) && \text{car } w = w_H(E^*) + 1 \end{aligned}$$

Si on veut un $\mathbf{e'}$ vérifiant (2.2), il suffit de trouver un $\epsilon_\mathbf{2}$ vérifiant (2.6) et $w_H(\epsilon_\mathbf{2}) \leq t$. Or cela revient au problème de décodage par syndrome avec la matrice Σ, qui est un problème difficile pour des $t \gtrsim d_{GV}(n, k, q)$. (voir A.2)

Pour remédier à ce problème, on a choisi de procéder comme dans l'**algorithme de Peterson** (A.3.3.3). H étant une matrice de parité d'un code de Reed-Solomon, on a essayé de trouver un polynôme localisateur d'erreurs, et la valeur des erreurs en exploitant la dépendance linéaire entre $\epsilon_\mathbf{1}$ et $\epsilon_\mathbf{2}$ (2.5).
Soit $\mathbf{x} := (x_1, .., x_{n+L}) \in \mathbb{F}_q^{L+n}$ tel que $x_i := \beta^{i-1}$ où β est un élément d'ordre $n + L$ de $\mathbb{F}_q$. On définit H comme étant une matrice de parité du code de **Reed-Solomon** $RS(n + L, k + L, \mathbf{x})_q$ (A.3.1).

i.e, on peut prendre :

$$H := \begin{bmatrix} 1 & x_1 & .. & x_1^{n+L-1} \\ 1 & x_2 & .. & x_2^{n+L-1} \\ : & : & .. & : \\ 1 & x_{n-k} & .. & x_{n-k}^{n+L-1} \end{bmatrix}$$

Supposons qu'on a reçu un mot bruité $\mathbf{r} = g(\mathbf{x}) + \mathbf{e'}$ avec $g = \sum_{i=0}^{k+L-1} g_i.X^i \in \mathbb{F}_q[X]_{\leq L+k-1}$ et $w_H(\mathbf{e'}) \leq w.t$.
Notre but est de retrouver g malgré le fait que t dépasse la capacité de correction de notre code ! Pour y parvenir, soit $P(X) := P_0 + P_1.X + ..P_{t.W}.X^{t.w-1}$ un polynôme localisateur d'erreurs qu'on cherche à trouver. Alors P et g vérifient

$$\begin{cases} P(x_i)(r_i - g(x_i)) = 0 \quad , \forall i \in [|1, n+L|] \\ deg(P) \leq t.w \\ deg(g) \leq k + L - 1 \end{cases} \tag{2.7}$$

(2.7) implique que $\forall\ i \in [|1, n+L|]$ on a $P(x_i)(r_i - g(x_i)) = 0$, i.e :

$$P(x_i).r_i - P(x_i).g(x_i) = 0$$

i.e :

$$\sum_{u=0}^{w.t-1} (r_i.x_i^u).P_u - \sum_{u=0}^{w.t-1} \sum_{v=0}^{k+L-1} (x_i^{u+v}).(g_v.P_u) = 0 \tag{2.8}$$

Or, on veut $D.\mathbf{e'}^T = 0$, i.e, $D.\mathbf{r}^T = D.g(\mathbf{x})$. Autrement dit,

$$D.\mathbf{r}^T = D.\sum_{u=0}^{k+L-1} g_u.x^u \qquad \text{avec } x^u = \begin{bmatrix} x_1^u \\ : \\ x_{n+L}^u \end{bmatrix}$$

$$= \sum_{u=0}^{k+L-1} g_u.(D.x^u)$$

i.e, en notant $\mathbf{b} := D.\mathbf{r}^T$ et $\mathbf{S_u} := D.\mathbf{x}^u$, on a :

$$\begin{aligned}
\mathbf{b} &= \sum_{u=0}^{k+L-1} S_u.g_u \\
&= (\mathbf{S_0}|..|\mathbf{S_{k+L-1}}).\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix} \\
&= (D.\mathbf{x}^0|D.\mathbf{x}|..|D.\mathbf{x}^{k+L-1}).\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix} \\
&= D.(\mathbf{x}^0|\mathbf{x}^1|..|\mathbf{x}^{k+L-1}).\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix}
\end{aligned}$$

Notons $H^* := (\mathbf{x}^0|\mathbf{x}^1|..|\mathbf{x}^{k+L-1})$ et $S := D.H^* \in \mathbb{F}_q^{L\times(L+k)}$. Alors, on aurait

$$\mathbf{b} = S.\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix}$$

En faisant une **élimination de Gauss**, on trouve un système équivalent $\mathbf{b'} = S'.\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix}$ où $S = (I_L|A)$ avec $A = (a_{i,j})_{i,j} \in \mathbb{F}_q^{L\times k}$. Autrement dit,

$$\begin{aligned}
\mathbf{b'} = S'.\begin{bmatrix} g_0 \\ g_1 \\ : \\ g_{k+L-1} \end{bmatrix} &\iff b'_j = g_j + \sum_{i=0}^{k-1} a_{j,i}.g_{L+i} && \forall j \in [|0, L-1|] \\
&\iff g_j = b'_j - \sum_{i=0}^{k-1} a_{j,i}.g_{L+i} && \forall j \in [|0, L-1|]
\end{aligned}$$

Maintenant, si on remplace dans (2.8), on se trouve avec, pour $j \in [|1, n+L|]$:

$$\begin{aligned}
0 &= \sum_{u=0}^{w.t-1} (r_j.x_j^u).P_u + \sum_{u=0}^{w.t-1}\sum_{v=0}^{L-1}(-x_j^{u+v}).(g_v.P_u) + \sum_{u=0}^{w.t-1}\sum_{v=L}^{k+L-1}(x_j^{u+v}).(g_v.P_u) \\
\iff \\
0 &= \sum_{u=0}^{w.t-1} (r_j.x_j^u).P_u + \sum_{u=0}^{w.t-1}\sum_{v=0}^{L-1}(-x_j^{u+v}).(P_u).(b'_v - \sum_{i=0}^{k-1} a_{v,i}.g_{L+i}) + \sum_{u=0}^{w.t-1}\sum_{v=L}^{k+L-1}(x_j^{u+v}).(g_v.P_u) \\
\iff \\
0 &= \sum_{u=0}^{w.t-1} (r_j.x_j^u).P_u + \sum_{u=0}^{w.t-1}\sum_{v=0}^{L-1}(-b'_v.x_j^{u+v}).P_u + \sum_{u=0}^{w.t-1}\sum_{i=0}^{k-1}\sum_{v=0}^{L-1}(a_{v,i}.x_j^{u+v}).(g_{L+i}.P_u) + \sum_{u=0}^{w.t-1}\sum_{v=L}^{k+L-1}(x_j^{u+v}).(g_v.P_u)
\end{aligned}$$

$$\iff$$

$$0 = \sum_{u=0}^{w.t-1}(r_j.x_j^u).P_u + \sum_{u=0}^{w.t-1}\sum_{v=0}^{L-1}(-b'_v.x_j^{u+v}).P_u + \sum_{u=0}^{w.t-1}\sum_{i=0}^{k-1}\sum_{v=0}^{L-1}(a_{v,i}.x_j^{u+v}).(g_{L+i}.P_u) + \sum_{u=0}^{w.t-1}\sum_{f=0}^{k-1}(x_j^{u+f+L}).(g_{f+L}.P_u)$$

$$\iff$$

$$0 = \sum_{u=0}^{w.t-1}[r_j.x_j^u - \sum_{v=0}^{L-1}b'_v.x_j^{u+v}].P_u + \sum_{i=0}^{k-1}\sum_{u=0}^{w.t-1}[\sum_{v=0}^{L-1}a_{v,i}.x_j^{u+v} - x_j^{u+i+L}].(g_{i+L}.P_u)$$

$$\iff$$

$$0 = \sum_{u=0}^{w.t-1}T_{j,u}.P_u + \sum_{i=0}^{k-1}\sum_{u=0}^{w.t-1}F_{j,u,i}.(g_{i+L}.P_u)$$

Notons : $M := (T|F^{(0)}|..|F^{(wt-1)})$ où

$$T = \begin{bmatrix} T_{0,0} & .. & T_{0,wt-1} \\ : & .. & : \\ T_{L+n-1,0} & .. & T_{L+n-1,wt-1} \end{bmatrix}$$

$$F^{(m)} = \begin{bmatrix} F_{0,m,0} & F_{0,m,1} & .. & F_{0,m,k-1} \\ F_{1,m,0} & F_{1,m,1} & .. & F_{1,m,k-1} \\ : & : & .. & : \\ F_{L+n-1,m,0} & F_{L+n-1,m,1} & .. & F_{L+n-1,m,k-1} \end{bmatrix} \qquad \forall m \in [|0, wt-1|]$$

Ainsi, M est une matrice avec :

- $L+n$ lignes.
- $w.t + w.t.k = w.t.(k+1)$ colonnes.

En notant encore :

$$X := (P_0, .., P_{tw-1}, P_0.g_L, .., P_0.g_{L+k-1}, .., P_{tw-1}.g_L, .., P_{tw-1}.g_{L+k-1})$$

On a :

$$M.X^T = \mathbf{0} \tag{2.9}$$

Pour espérer avoir une solution non triviale à ce système linéaire, on impose la condition :

$$L + n < w.t.(k+1)$$

i.e :

$$w > \frac{L+n}{t.(k+1)}$$

On a testé cette approche en utilisant **Magma**, pour plusieurs paramètres n, k et L, et pour lesquels on a fixé $w := \lfloor \frac{L+n}{t.(k+1)} \rfloor + 1$ le plus petit possible : Comme ça on trouve directement un tel $\mathbf{e}'$ ou de minimiser le cardinal de l'espace auquel appartiennent les éléments de la forme :

$$X := (P_0, .., P_{tw-1}, P_0.g_L, .., P_0.g_{L+k-1}, .., P_{tw-1}.g_L, .., P_{tw-1}.g_{L+k-1})$$

Expériences (sur Magma) et résultats :

Spécifions d'abord les paramètres :

```
q :=2^16                          ;
n :=32                            ;
k :=16                            ;
L :=225                           ;
t := binary_gv(n,k)               ;
w := Floor((L+n) / (t*(k+1))) +1 ;
```

On a choisi les paramètres de sorte que $L+n$ divise $q-1$. Dans ce cas, on sait que, si α est un **élément primitif** du corps $\mathbb{F}_q$, alors $\beta^{\frac{q-1}{L+n}}$ est exactement d'ordre $n+L$.

```
alpha := PrimitiveElement(GF(q))    ; // Un element primitif du corps F_q
beta  := alpha^((q-1) div (n+L))    ; // Un element d'ordre n+L du corps F_q
x     := [beta^i : i in {0..L+n-1}] ; //
```

Et nous avons pris $\mathbf{x} := (1, \beta, \beta^2, .., \beta^{n+L-1})$ comme support de notre code de Reed-Solomon de type $[n+L, k+L, n-k+1]_q$.
On construit en suite les matrices H, E, D, H^* ainsi que G qui est une matrice génératrice du code de Reed-Solomon :

```
G := KMatrixSpace(GF(q),k+L,n+L) ! [[ x[i]^j : i in {1..n+L}] : j in {0..L+k-1}] ;
H := KMatrixSpace(GF(q),n-k,n+L) ! [[ x[j]^i : i in {0..n+L-1}] : j in {2..n-k+1}] ;

print(H*Transpose(G) eq 0); // S'assurer que G.H^T=0

H_k := KMatrixSpace(GF(q),n+L,k+L) ! [[ x[j]^i : i in {0..k+L-1}] : j in {1..n+L}] ;
    \\ H^*

E_s := KMatrixSpace(GF(q),L,n) ! 0; // E*
for j in {1..n} do
       T := RandomSubset({1..L},w-1); // Sous partie aleatoire de {1,..,L} de
           cardinal w-1
       for i in T do
              E_s[i][j] := 1;
       end for;
end for;
E   := VerticalJoin(E_s,IdentityMatrix(GF(q),n)) ; // E = (E*^T|I_n)
C_E := LinearCode(transpose(E)); // Code defini par E^T

D   := HorizontalJoin(-IdentityMatrix(GF(q),L),E_s) ; // D = (-I_L|E*)
```

Puis on répète un procédé :

```
List := [];
for o in {1..100} do
       c := Random(C); // Un mot aleatoire du code genere par G

       _e_ := Random(C_E,w*t-1); // choisir e' tel que D.e'^T=0^T

       r := c + _e_ ;
       b := r*Transpose(D); // b=D.r^T

       S := D*H_k;          // D=D.H^*
```

```
1
2         ///// Elimination de Gauss /////
3         P := Submatrix(S,1,1,L,L)^(-1);
4         b := b*Transpose(P); //b'
5         S := P*S; // S'
6         ////////////////////////////////
7         A := Submatrix(S,1,L+1,L,k); // S' =(I_L|A)
8         M := KMatrixSpace(GF(q),L+n,w*t) ! [[ r[j]*x[j]^(u-1) - &+[b[v+1]*x[j]^(u-1+v) :
              v in {0..L-1}] : u in {1..w*t} ] : j in {1..L+n}] ;
9         for m in {0..w*t-1} do
10                M := HorizontalJoin(T,dala1(n,k,L,m,x,A,t,w,q));
11        end for;
12        // M = (T|F^(0)|F^(1)|..|F^(wt-1))
13        Append(~List,Rank(M)); // List contiendra les Rank(M)
14 end for;
```

On trouve dans ce cas-là :

```
L = [ 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
      62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
     62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
     62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
     62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62 ]
```

Ce phénomène se répète en prenant d'autres valeurs de n, k et L.
En fait, en fixant E, et en itérant ℓ fois la procédure suivante :

1. $c \leftarrow U(C)$;
2. $e \leftarrow U(\mathbb{F}_q^n)$ tel que $w_H(e) \leq wt - 1$ et $D.e^T = 0$;
3. $r \leftarrow c + e$;
4. Trouver $M := (T|F^{(0)}|F^{(1)}|..|F^{(wt-1)})$ et calculer $R_\ell := Rank(M)$;

On trouve que la suite $(R_\ell)_\ell$ est constante. $Rank(M)$ est le même, et ne dépend pas du choix du mot de code c.

On peut en conséquent faire un test plus général :

1. $E_\ell^* \leftarrow U(\mathbb{F}_q^{L \times n})$ de poids $w - 1$ par colonne ;
2. Construire $E_\ell := (E^{*T}|I_n)$;
3. $c \leftarrow U(C)$;
4. $e \leftarrow U(\mathbb{F}_q^n)$ tel que $w_H(e) \leq wt - 1$ et $D.e^T = 0$;
5. $r \leftarrow c + e$;
6. Trouver $M_\ell := (T|F^{(0)}|F^{(1)}|..|F^{(wt-1)})$ et stocker $R_\ell := Rank(M_\ell)$ dans une liste $List$;
7. À E_ℓ on associe $R_\ell := Rank(M_\ell)$;

On trace le graphe $\{(l, R_\ell)\}$, et on trouve les résultats suivants :

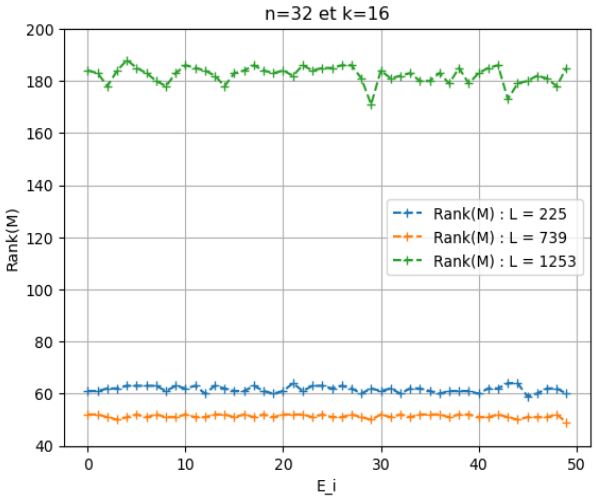


FIGURE 2.1 – $n = 32$, $k = 16$ et $(L, wt(k+1)) \in \{(225, 340), (739, 850), (1253, 1360)\}$

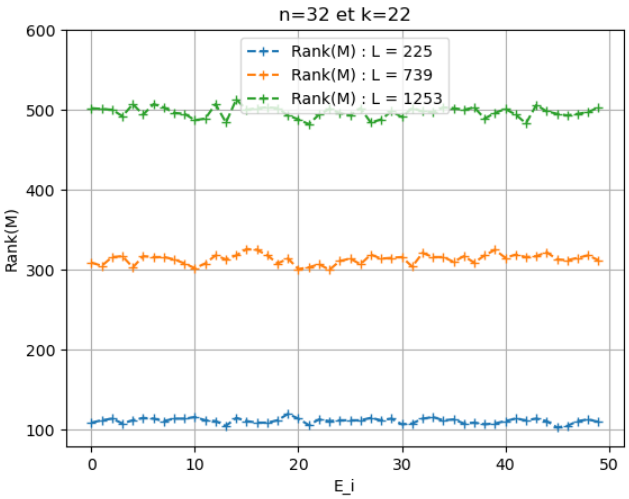


FIGURE 2.2 – $n = 32$, $k = 22$ et $(L, wt(k+1)) \in \{(225, 276), (739, 828), (1253, 1311)\}$

Même si on prend k très proche de n, on trouve :

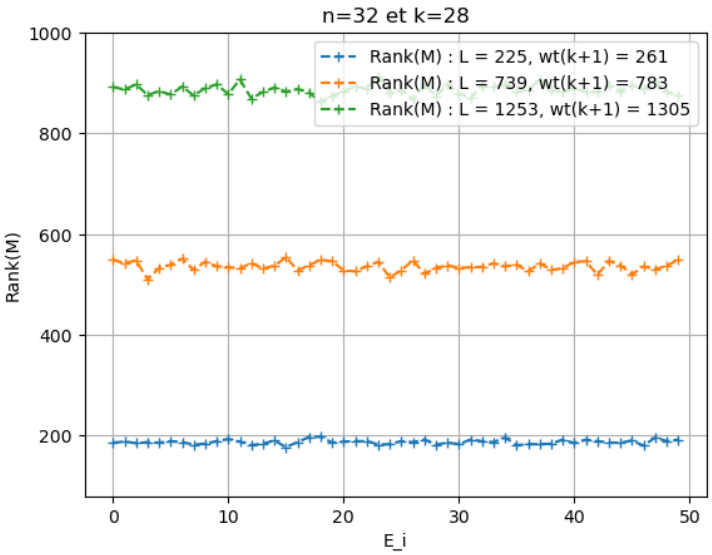


FIGURE 2.3 – $n = 32$ et $k = 28$

Conclusion : L'approche n'est pas intéressante d'un point de vue cryptographique. On ne peut utiliser une matrice de parité d'un code de Reed-Solomon pour H.

2.4 Troisième approche

Donnons d'abord quelques définitions et propriétés :

2.4.1 Définitions et généralités

Définition 8 (Matrice circulante)**.** *Une matrice carrée $M \in \mathbb{F}_q^{n\times n}$ est **circulante** si elle est de la forme :*

$$M := \begin{bmatrix} a_1 & a_2 & .. & a_n \\ a_n & a_1 & .. & a_{n-1} \\ : & : & .. & : \\ a_2 & a_3 & .. & a_1 \end{bmatrix}$$

avec $(a_1, a_2, .., a_n) \in \mathbb{F}_q^n$.

Une propriété cruciale de ces matrices est la **commutativité** de leur produit :

Proposition 1. [6] *Soient A et B deux matrices carrées circulantes de $\mathbb{F}_q^{n\times n}$. Alors $A.B = B.A$.*

Définition 9 (Matrice quasi-circulante)**.** *Soient p et ℓ deux entiers naturels. Une matrice M est **quasi-circulante** si elle est de la forme :*

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} & .. & M_{1,p} \\ M_{2,1} & M_{2,2} & .. & M_{2,p} \\ : & : & .. & : \\ M_{\ell,1} & M_{\ell,2} & .. & M_{\ell,p} \end{bmatrix}$$

Où chaque M_i est une matrice circulante.

Définition 10 (QC-LDPC)**.** *Un code binaire C est dit **LDPC quasi-circulant** (**QC-LDPC**[7]) si sa matrice de parité est de la forme*

$$H = \begin{bmatrix} H_{1,1} & H_{1,2} & .. & H_{1,p} \\ H_{2,1} & H_{2,2} & .. & H_{2,p} \\ : & : & .. & : \\ H_{\ell,1} & H_{\ell,2} & .. & H_{\ell,p} \end{bmatrix}$$

*où chaque matrice H_i est circulante et représente une matrice de parité d'un code **LDPC**[8] (A.3.4).*

6. https ://fr.wikipedia.org/wiki/Matrice_circulante
7. Quasi-Cyclic LDPC Codes
8. Low-density parity-check

2.4.2 Cas où E est quasi-circulante avec deux blocs et H matrice de parité d'un code LDPC

Dans cette approche, on a choisi D de la forme $D = (D_1|D_2)$, avec D_1 et D_2 des matrices de $\mathbb{F}_2^{L\times L}$ circulante de poids faible.
Pour rappel : La matrice D est une matrice de parité associée à la matrice E^T ici 2.2.

Si on choisit $D_i \in \mathbb{F}_2^{L\times 2L}$, alors la matrice E est de la forme $E = (-D_2^T|D_1^T)^T$. En effet :

$$\begin{aligned} D.E^T &= (D_1|D_2).\begin{pmatrix} -D_2 \\ D_1 \end{pmatrix} \\ &= -D_1.D_2 + D_2.D_1 \\ &= 0 \qquad\qquad \text{par commutativité des matrices circulantes} \end{aligned}$$

On a choisi cette forme pour E pour deux raisons :

— La matrice E ainsi que sa matrice de parité D sont toutes les deux des matrices creuses, avec des poids par colonne (ligne) fixe :
Proposition 2. *Si la matrice $D = (D_1|D_2)$ a un poids μ pour chaque colonne, alors $E := (-D_2^T|D_1^T)^T$ a un poids 2μ de pour chaque colonne.*
Preuve 1. *Si D_i est de poids μ par colonnes, alors en notant x le poids par lignes de D_i, on a que $L.x = L.\mu$, i.e, $x = \mu$. Comme $E = \begin{bmatrix} -D_2 \\ D_1 \end{bmatrix}$, on a le résultat.* ■

— Il est facile de trouver D à partir de E et inversement.

Et pour la matrice $H \in \mathbb{F}_2^{(L-k)\times 2L}$, nous avons pris la matrice de parité d'un code **LDPC** (A.3.4). Plus précisément :

— w_D le poids par colonnes de D ;
— w_H le poids par colonnes de H ;
— On pose :

$$H^* := \begin{bmatrix} H \\ D \end{bmatrix} \tag{2.10}$$

et posons $w := w_H + w_D$. H^* est une matrice creuse de poids w par colonnes, i.e, elle représente une matrice de parité (redondante) d'un code **LDPC**. D'après la section A.3.4, le code représenté par H^* est $\lfloor \frac{w}{2} \rfloor$-correcteur (A.1).

Ainsi, avec ces notations, les conditions (2.1), (2.2) et (2.3) ici 2.2 sont équivalentes à trouver $\mathbf{e}' \in \mathbb{F}_2^{2L}$ tel que $H^*.\mathbf{e}'^T = \begin{bmatrix} \mathbf{s}^T \\ \mathbf{0} \end{bmatrix}$ et $w_H(\mathbf{e}') \leq 2.w_D.t$ avec $t \gtrsim d_{GV}(L,k)$.
Le code **LDPC** représenté par H^* étant $\lfloor \frac{w}{2} \rfloor$-**correcteur** [9], alors :

$$\begin{aligned} w_H(\mathbf{e}') \leq 2.w_D.t &\iff \lfloor \frac{w}{2} \rfloor \geq 2w_d t \\ &\iff w_H + w_D \geq 4.w_D.t \qquad\qquad \text{car : } w = w_H + w_D \\ &\iff w_D.(4t-1) \leq w_H \leq L - k \end{aligned}$$

9. Peut corriger jusqu'à $\lfloor \frac{w}{2} \rfloor$: voir A pour plus de précisions.

Expériences (sur Magma) et résultats :

Comme L est censé être grand et que les deux matrices D et H sont creuses et binaires, on a choisi de les représenter comme la suite :
Soit $A \in \mathbb{F}_2^{k\times n}$ une matrice. Alors, on peut représenter A de deux manières :

— **Représentation par colonnes K_A de A :** $K_A := [T_1, .., T_n]$ où pour $j \in [|1, n|]$,

$$T_j := \{i \in [|1, k|] \mid A_{i,j} = 1\}$$

— **Représentation par lignes R_A de A :** $R_A := [T_1, .., T_k]$ où pour $i \in [|1, k|]$,

$$T_i := \{j \in [|1, n|] \mid A_{i,j} = 1\}$$

On construit les deux matrices H et D avec leur représentation par colonne, l'algorithme **LDPC** ici D.2.
On commence par choisir les paramètres :

```
//////// PARAMETRES //////////////
L      := 250;
k      := 200;
r      := L - k;
d      := binary_gv(L,k); // Voir l'annexe D pour trouver l'algorithme binary_gv
w      := d     ;
```

En suite, on construit les matrices H et D :

```

K_D, K_E := LDPCC(L,2*L,w);

E := KMatrixSpace(GF(2),2*L,L) ! 0;

for j in {1..L} do
        for i in K_E[j] do
                E[i][j] := 1;
        end for;
end for;

D := KMatrixSpace(GF(2),L,2*L) ! 0;

for j in {1..2*L} do
        for i in K_D[j] do
                D[i][j] := 1;
        end for;
end for;


H := KMatrixSpace(GF(2),r,2*L) ! 0;

K_H := [];
for j in {1..2*L} do
        T := RandomSubset({1..r},w*(4*d-1)+1);
        Append(~K_H,T);
        for i in T do
                H[i][j] := 1;
        end for;
end for;

K := [[j : j in K_H[i]] cat [L-r + j : j in K_D[i]] : i in {1..2*L}];
```

On effectue en suite le test suivant :

1. On génère $\mathbf{e} \in \mathbb{F}_2^L$ de poids $\leq t$;
2. $\mathbf{e}_1^T \leftarrow E.\mathbf{e}^T$ et $\mathbf{s}^T \leftarrow H.\mathbf{e}_1^T$;
3. ${\mathbf{s_h}}^T \leftarrow \begin{bmatrix} \mathbf{s}^T \\ \mathbf{0} \end{bmatrix}$;
4. On utilise le décodage des **LDPC** avec la matrice H^* pour trouver $\_\mathbf{e}\_$ tel que $H^*.{\_\mathbf{e}\_}^T = {\mathbf{s_h}}^T$;
5. On calcule $\_\mathbf{s}\_ \leftarrow H^*.{\_\mathbf{e}\_}^T$;
6. Vérifier si $\mathbf{s} = \_\mathbf{s}\_$ (renvoie 1) ou pas (renvoie 0).

Sur **Magma**, cela donne :

```
List := [];
for o in {1..500} do
    /////////// Etape 1 ////////////
        T := RandomSubset({1..L},d+1);
        e := VectorSpace(GF(2),L) ! 0;
        for i in T do
                e[i] := 1;
        end for;
        /////////// Etape 2 ////////////
        _e_ := e*Transpose(E);
```

```
        s := _e_*Transpose(H);
        ////////// Etape 3 ////////////
        S := VectorSpace(GF(2),L+r) ! 0;
        for i in {1..r} do
                S[i] := s[i] ;
        end for;

    //////// Etape 4 ////////////
        test := Decodage_LDPC(K,2*L,L+r,S);
        //////// Etape 5 ////////////
        _s_ := test*Transpose(H);
        /////// Etape 6 ///////////
        if _s_ eq s then
                Append(~List,1);
        else
                Append(~List,0);
        end if;

end for;
```

L'algorithme de signature **sign(K,r,L,s)** peut être n'importe quel algorithme parmi les trois ici C.3.

On trouve que :

```
List = [ 0, 0, 0, 0, 0,..,0]
```

Autrement dit, $\mathbf{s}$ n'est jamais égale à $\_\mathbf{s}\_$. Le résultat ne dépend pas de l'algorithme de signature choisi. On n'arrive donc pas à signer avec cette approche.

Conclusion : Les résultats pratiques que nous avons obtenus montrent que, avec la forme prise pour E et H, on n'arrive pas à trouver le résultat qu'on espérait avoir.

2.4.3 Utilisation du décodage complet (Full decoding)

Ici, on ne change pas E, mais, pour H on choisit une forme particulière de matrices :
Soit p un diviseur commun à $2.L$ et $L-k$. Notons $k_0 = \frac{L-k}{p}$ et $n_0 = \frac{2L}{p}$

$$H := \begin{bmatrix} A_1 & 0 & 0 & .. & 0 \\ 0 & A_2 & 0 & .. & 0 \\ 0 & 0 & A_3 & .. & 0 \\ : & : & : & .. & : \\ 0 & 0 & 0 & .. & A_p \end{bmatrix} \in \mathbb{F}_2^{(L-k)\times 2L} \tag{2.11}$$

Où pour $i \in [|1,p|]$, $A_i = (I_{k_0}|A_i^*)$ avec $A_i^* \in \mathbb{F}_2^{k_0\times(n_0-k_0)}$ ou plus généralement, on prend A_i de rang plein.
On choisit les matrices A_i pour qu'elles soient de la plus petite taille possible, i.e, $d = pgcd(L-k, 2L)$.
L'intérêt de cette construction est la suivante :
Étant donné $\mathbf{s} \in \mathbb{F}_q^{L-k}$ et $t \in [|1, 2L|]$, trouver $\mathbf{e} \in \mathbb{F}_2^{2L}$ tel que $A.\mathbf{e}^T = \mathbf{s}^T$ et $w_H(\mathbf{e}) \leq t$ est assez facile. en effet, en séparant $\mathbf{s}$ et $\mathbf{e}$ en p blocs, on aura

$$\begin{aligned} &\mathbf{s} = (\mathbf{S_1}|\mathbf{S_2}|..|\mathbf{S_p}) && \text{Avec } \mathbf{S_i} \in \mathbb{F}_2^{k_0} \\ &\mathbf{e} = (\mathbf{E_1}|\mathbf{E_2}|..|\mathbf{E_p}) && \text{Avec } \mathbf{E_i} \in \mathbb{F}_2^{n_0} \end{aligned}$$

et

$$H.\mathbf{e}^T = \mathbf{s}^T \iff \forall i \in [|1,p|],\ A_i.\mathbf{E_i}^T = \mathbf{S_i}^T$$

Comme les A_i sont de tailles petites, alors on peut trouver de tels $\mathbf{E_i}$ avec un algorithme générique (par force brute). On peut choisir les E_i de sorte que $\sum_{i=1}^{p} w_H(\mathbf{E_i}) \leq t$ (voir C.4).
C'est ce qu'on appelle **"décodage complet"**.

Pour vérifier la signature avec cette nouvelle construction, on a fait le test suivant :

1. On génère le mot à signer $\mathbf{s} \leftarrow \mathbb{F}_2^k$;
2. On trouve par le décodage complet un mot $\mathbf{e}' \in \mathbb{F}_2^k$ avec un poids égal à $t.w$ où w le poids des colonnes de E et $t \gtrsim d_{GV}(L,k)$ tel que $H.\mathbf{e}'^T = \mathbf{s}^T$;
3. On vérifie si $\mathbf{e}'$ est un élément de $Im(E) := \{E.\mathbf{x}^T \mid \mathbf{x} \in \mathbb{F}_2^L\}$;

Or :
Résultats : $\mathbf{e}'$ est très rarement un élément de $Im(E)$. Pourquoi ?

Si on analyse bien la situation, on peut comprendre pourquoi.

Si on note

$$\begin{aligned} f_E : \mathbb{F}_2^L &\to \mathbb{F}_2^{2L} \\ \mathbf{x} &\mapsto E.\mathbf{x}^T \end{aligned}$$

Alors la fonction f_E est injective, car E est de rang plein. Ainsi, $Im(f_E) \simeq \mathbb{F}_q^L$.
D'autre part, si on note

$$\begin{aligned} f_H : \mathbb{F}_2^{2L} &\to \mathbb{F}_2^{L-k} \\ \mathbf{x} &\mapsto H.\mathbf{x}^T \end{aligned}$$

Alors $f_H^{-1}(\{s\}) \simeq \mathbb{F}_2^{L+k}$ car d'après le théorème du rang, on a $rg(f_H) + dim(ker(f_H)) = 2L$, i.e, $dim(ker(f_H)) = 2L - (L-k) = L + k$.

Ainsi :

$$\begin{aligned} dim(ker(f_H) + im(f_E)) &= dim(ker(f_H)) + rg(E) - dim(ker(f_H) \cap im(E)) \\ &= L + k + L - dim(ker(f_H) \cap im(E)) \\ &\geq 2L \qquad \text{car : } dim(ker(f_H) \cap im(E)) \leq k \end{aligned}$$

Autrement dit : $ker(f_H) + im(f_E) = \mathbb{F}_2^{2L}$ mais rien ne nous assure que $dim(ker(f_H) \cap im(E)) = 0$, i.e, rien ne nous assure que $\mathbf{e}' \in Im(E)$.

Conclusion : Cette approche ne fonctionne pas, mais elle nous a permis de mieux comprendre le problème.

2.5 Quatrième approche

Ici, on va prendre $E := (E_1|E_2|..|E_p)$ avec p un nombre pair et les $E_i \in \mathbb{F}_2^{n\times n}$ sont circulantes. Autrement dit : $L = p.n$.
On choisit p assez grand pour que $L \sim n^a$ avec $a > 1$.
Maintenant que E est sous cette forme, quelle est la forme de D, une matrice de parité associée à E qui soit aussi creuse et dont les colonnes sont de poids fixé ? Lorsque $p = 2$, c'était simple. Dans le cas contraire, on a trouvé le résultat suivant :

Proposition 3. *Soit $E := (E_1|E_2|..|E_p)$ où p est paire et les $E_i \in \mathbb{F}_2^{n\times n}$ circulantes de rang plein. Notons*

$$D := \begin{bmatrix} E_2^T & E_1^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_3^T & E_4^T & E_1^T & E_2^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_4^T & E_3^T & E_2^T & E_1^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_5^T & E_6^T & E_4^T & E_3^T & E_1^T & E_2^T & .. & E_p^T & E_{p-1}^T \\ E_6^T & E_5^T & E_4^T & E_3^T & E_2^T & E_1^T & .. & E_p^T & E_{p-1}^T \\ : & : & : & : & : & : & .. & : & : \\ E_{p-1}^T & E_p^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_1^T & E_2^T \\ E_p^T & E_{p-1}^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_2^T & E_1^T \end{bmatrix}$$

Alors $D.E^T = 0$.

Preuve 2. *Cela est facile à vérifier (et à voir), par exemple on a, pour la ligne 1 des blocs de D :*

$$E_2^T.E_1^T + E_1^T.E_2^T + E_4^T.E_3^T + E_3^T.E_4^T + .. + E_p^T.E_{p-1}^T + E_{p-1}^T.E_p^T = 0$$

car la caractéristique est paire et que les matrices circulante commutent.

Remarque : D ci-dessus n'est pas forcément une matrice de parité associé à E. Mais en testant cela sur **Magma**, D est de rang plein dans la majore partie des cas. Ainsi, on va générer une matrice E de sorte que D est de rang plein, i.e, de sorte que D est une matrice de parité associé à E.

En répétant l'expérience qu'on a fait lorsque $p = 2$, i.e :

1. On génère le mot à signer $\mathbf{s} \leftarrow \mathbb{F}_2^{L-k}$;
2. On trouve par le décodage complet un mot $\mathbf{e}' \in \mathbb{F}_2^k$ avec un poids égal à $t.w$ où w le poids des colonnes de E et $t \gtrsim d_{GV}(L, k)$ tel que $H.\mathbf{e}'^T = \mathbf{s}^T$;
3. On vérifie si $\mathbf{e}'$ est un élément de $Im(E) := \{E.\mathbf{x}^T \mid \mathbf{x} \in \mathbb{F}_2^L\}$;

On trouve exactement le même problème qu'avec le cas $p = 2$: $\mathbf{e}'$ n'est que très rarement un élément de $Im(E)$.
Ainsi, on a utilisé un point de vue différent dans la suite. En fait, on remarque que : Pour $\mathbf{e}' \in \mathbb{F}_2^L$, $\exists \mathbf{e} \in \mathbb{F}_2^n$ et $\mathbf{a} \in Ker(f_H)$ tels que $\mathbf{a}'^T = E.\mathbf{e}^T + \mathbf{a}^T$.

Remarque : Dans ce cas, signer revient à trouver un $\mathbf{e}$ et un $\mathbf{a}$ vérifiant les bonnes conditions, i.e, $w_H(\mathbf{e}) \gtrsim d_{GV}(n, k)$, $H.\mathbf{a}^T = \mathbf{0}$ et $D.\mathbf{e}'^T = D.\mathbf{a}^T$.
Donc en notant :

$$H^* := \begin{bmatrix} D \\ H \end{bmatrix}$$

signer revient à trouver $\mathbf{a} \in \mathbb{F}_2^L$ tel que :

$$H^*.\mathbf{a}^T = \begin{bmatrix} D.\mathbf{e}'^T \\ \mathbf{0} \end{bmatrix}$$

et puis trouver par l'algèbre linéaire $\mathbf{e} \in \mathbb{F}_2^n$ tel que $E.\mathbf{e}^T = \mathbf{e}'^T + \mathbf{a}^T$ et $w_H(\mathbf{a} + \mathbf{e}') = wt$.
Pour la simplicité du décodage des codes **LDPC**, on a intérêt que H^* soit une matrice de parité d'un code LDPC. La matrice H suivante

$$H := \begin{bmatrix} A_1 & 0 & 0 & .. & 0 \\ 0 & A_2 & 0 & .. & 0 \\ 0 & 0 & A_3 & .. & 0 \\ : & : & : & .. & : \\ 0 & 0 & 0 & .. & A_p \end{bmatrix}$$

sera définie en prenant les A_i de rang plein, et de poids faible fixé par colonnes.

Sur **Magma**, on a effectué le test suivant :

- Génère $\mathbf{s} \leftarrow U(\mathbb{F}_2^{n-k})$;
- Trouver $\mathbf{e}' \in \mathbb{F}_2^L$ tel que $H.\mathbf{e}'^T = \mathbf{s}^T$ et $w_H(\mathbf{e}') \sim w.t$ par le décodage complet ;
- calculer $\mathbf{s}' := D.\mathbf{e}'^T$ et $\mathbf{S} \leftarrow \begin{bmatrix} \mathbf{s}' \\ \mathbf{0} \end{bmatrix}$;
- en utilisant un algorithme de décodage des LDPC, on trouve $\mathbf{a}$ tel que $H^*.\mathbf{a}^T = \mathbf{S}$;

Le programme est le suivant :

Expériences (sur Magma) et résultats :

On commence par créer les matrices :

- $E = (E_1|E_2|..|E_b)$;
- D de la forme ci-dessus qui est de rang plein (i.e : $rg(D) = (b-1) * n$) ;
- La matrice H (nommée A dans mon code) de la forme (3.11) ;

```
////////////////////  Parameters  ////////////////////
R := 1/2                    ;
b := 18                     ; // b pair
n := 416                    ;
r := 12*b                    ;
k := n-r                    ;
L := b*n                    ;
w := 19                     ;
t := binary_gv(n,k)         ;
d := binary_gv(L,L-n)       ;
////////////////////////////////////////////////
L0 := L div b;
r0 := r div b;
///////////////
```

```
repeat
        K_E := [];
        ech := LDPC(n,w);

        while #K_E ne b do
                //print(false);
                if ech in K_E then
                        ech := LDPC(n,w);
                else
                        Append(~K_E,ech);
                end if;
        end while;

        E := [];

        for bloc in {1..b} do
                M := KMatrixSpace(GF(2),n,n) ! 0;

                for j in {1..n} do
                        for i in K_E[bloc][j] do
                                M[i][j] := 1;
                        end for;
                end for;
                Append(~E,M);
        end for;

        EE := E[1];
        for i in {2..b} do
                EE := HorizontalJoin(EE,E[i]); // La matrice E
        end for;

        M := KMatrixSpace(Rationals(),b-1,b) !
 [[2,1,4,3,6,5,8,7,10,9,12,11,14,13,16,15,18,17],
 [3,4,1,2,6,5,8,7,10,9,12,11,14,13,16,15,18,17],
 [4,3,2,1,6,5,8,7,10,9,12,11,14,13,16,15,18,17],
 [5,6,4,3,1,2,8,7,10,9,12,11,14,13,16,15,18,17],
 [6,5,4,3,2,1,8,7,10,9,12,11,14,13,16,15,18,17],
 [7,8,4,3,6,5,1,2,10,9,12,11,14,13,16,15,18,17],
 [8,7,4,3,6,5,2,1,10,9,12,11,14,13,16,15,18,17],
 [9,10,4,3,6,5,8,7,1,2,12,11,14,13,16,15,18,17],
 [10,9,4,3,6,5,8,7,2,1,12,11,14,13,16,15,18,17],
 [11,12,4,3,6,5,8,7,10,9,1,2,14,13,16,15,18,17],
 [12,11,4,3,6,5,8,7,10,9,2,1,14,13,16,15,18,17],
 [13,14,4,3,6,5,8,7,10,9,12,11,1,2,16,15,18,17],
 [14,13,4,3,6,5,8,7,10,9,12,11,2,1,16,15,18,17],
 [15,16,4,3,6,5,8,7,10,9,12,11,14,13,1,2,18,17],
 [16,15,4,3,6,5,8,7,10,9,12,11,14,13,2,1,18,17],
 [17,18,4,3,6,5,8,7,10,9,12,11,14,13,16,15,1,2],
 [18,17,4,3,6,5,8,7,10,9,12,11,14,13,16,15,2,1]] ;
    D := KMatrixSpace(GF(2),(b-1)*n,b*n) ! 0; // La matrice D
        for bloc1 in {1..b-1} do
                for bloc2 in {1..b} do
                        if M[bloc1,bloc2] ne 0 then
                                InsertBlock(~D , Transpose(E[Integers() ! M[bloc1,bloc2]
                                    ]) ,1 + n*(bloc1-1) , 1 + n*(bloc2-1) );
                        end if;
                end for;
        end for;

until ((b-1)*n-Rank(D)) eq (b-1)*n ;
```

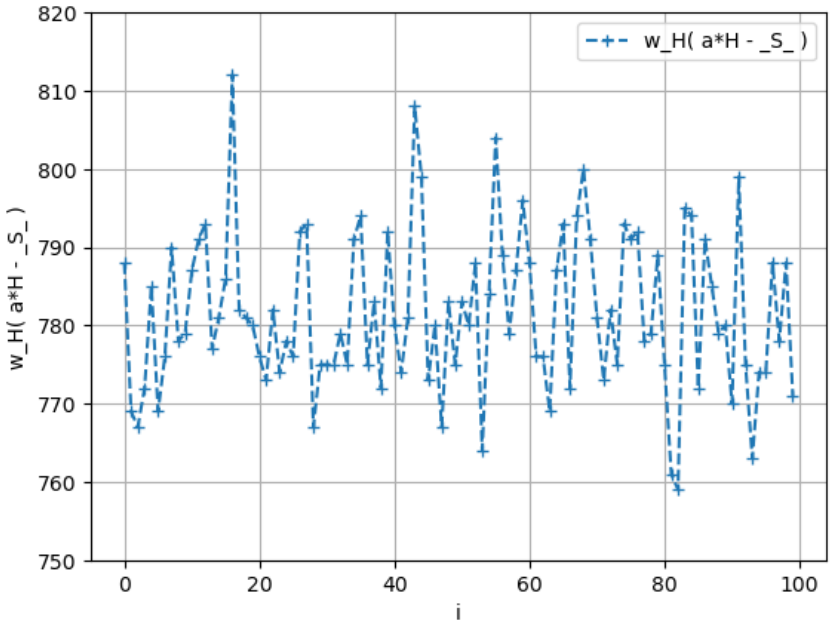
```
AA := bloc_LDPC(L0,r0,b,w div b);


K_A := [[] : i in {1..L}] ;

for bloc in {1..b} do
        for j in {1..L0} do
                for i in {1..r0} do
                        if AA[bloc][i][j] ne 0 then
                                Append(~K_A[j+(bloc-1)*L0], i + (bloc-1)*r0);
                        end if;
                end for;
        end for;
end for;


A := KMatrixSpace(GF(2),r,L) ! 0 ; // La matrice A

for bloc in {1..b} do
        for i in {1..r0} do
                for j in {1..L0} do
                        A[i+(bloc-1)*r0][j+(bloc-1)*L0] := AA[bloc][i][j] ;
                end for;
        end for;
end for;

C_E := LinearCode(EE); // Le code engendre par E
//////
dual_E := Dual(C_E); // Le dual de C_E
/////


H_etoile := VerticalJoin(D,A); la matrice H^*

K := [[] : i in {1..L}]; // La representation par colonnes de H^*

for j in {1..L} do
        for i in {1..L-n+r} do
                if H_etoile[i,j] ne 0 then
                        Append(~K[j],i);
                end if;
        end for;
end for;
```

Le test en **Magma** est :

```
List := [];
for o in {1..100} do
        o;
        s := [Random(VectorSpace(GF(2),r0)) : i in {1..b}];
        S := s[1];
        for i in {2..b} do
                S := HorizontalJoin(S,s[i]);
        end for;

        // e := Decodagecomplet(AA,s,L0,r0,b,b);
        e := complet_prangedecode(AA,s,L0,r0,b);

        _S_ := e*Transpose(D);
```

```
        _S_ := VectorSpace(GF(2),L-n+r) ! HorizontalJoin(_S_,VectorSpace(GF(2),r) ! 0);

        a := decode_LDPC(K,L,L-n+r,_S_,50);

        aH := a*Transpose(H_etoile);

        Append(~List,Weight((VectorSpace(GF(2),L-n+r)! aH) - _S_));
end for;
```

On trouve :



On voit que le résultat est loin de ce qu'on attendait. Normalement, on doit avoir $w_H(\mathbf{a}.H^{*T} - \mathbf{S}) = \mathbf{0}$, or dans ce qu'on a trouvé, cette valeur tourne autour de 785.

Conclusion : L'approche ne fonctionne pas. Supposer que H^* est une matrice de parité d'un **LDPC** ne nous aide pas à décoder.

2.6 Décodage statistique des codes QC-LDPC

Pour en connaître plus sur le décodage statistique, je recommande de voir la thèse de Thomas Debris-Alazard ici ([7]).

2.6.1 Généralités

Le décodage statistique fait partie d'une famille d'approches qui cherchent à résoudre le problème de décodage générique présentée dans l'annexe A.2. L'idée est proposée par A. Al Jabri dans son article [13], et est la suivante (comme présenté dans [7]) :

Soient C un code de type $[n,k]_2$, $\mathbf{y} := \mathbf{c} + \mathbf{e}$ avec $\mathbf{c} \in C$, $w_H(\mathbf{e}) = w \in [|0,n|]$ et $\mathbf{h} \in C^\perp$. Dans ce cas, on a :

$$< \mathbf{y}|\mathbf{h} >=< \mathbf{e}|\mathbf{h} >$$

Supposons que pour un $i_0 \in [|1,n|]$, on a $h_{i_0} = 1$. Alors, on a deux cas possibles :

- si $e_{i_0} = 1 : < \mathbf{e}|\mathbf{h} >= 1 \iff Card(Supp(\mathbf{e}) \cap Supp(\mathbf{h}) \backslash \{i_0\})$ est pair.
- si $e_{i_0} = 0 : < \mathbf{e}|\mathbf{h} >= 1 \iff Card(Supp(\mathbf{e}) \cap Supp(\mathbf{h}) \backslash \{i_0\})$ est impair.

Faisons-nous l'hypothèse suivante :

1 (Hypothèse). *La distribution du produit scalaire* $\mathbf{y}.\mathbf{h}^T$ *avec* $\mathbf{h} \leftarrow U(\{\mathbf{x} \in C^\perp \mid h_i = 1$ ***et*** $w_H(\mathbf{h}) = t\})$ *est approchée par la distribution de* $\mathbf{y}.\mathbf{h}^T$ *avec* $\mathbf{h} \leftarrow U(\{\mathbf{x} \in \mathbb{F}_2^n \mid h_i = 1$ ***et*** $w_H(\mathbf{h}) = t\}$.

Alors, pour $\mathbf{e} \in \mathbb{F}_2^n$ de poids s et $\mathbf{h} \in \mathbb{F}_2^n$ de poids t tel que $h_i = 1$ on a :

—

$$\begin{aligned} q_1(\mathbf{e},t,i) &:= \mathbb{P}_{\mathbf{h}}(\mathbf{e}.\mathbf{h}^T = 1 \mid e_i = 1) \\ &= \frac{\sum_{j\ pair}^{t-1} \binom{w-1}{j}\binom{n-w}{t-1-j}}{\binom{n-1}{t-1}} \end{aligned}$$

—

$$\begin{aligned} q_0(\mathbf{e},t,i) &:= \mathbb{P}_{\mathbf{h}}(\mathbf{e}.\mathbf{h}^T = 1 \mid e_i = 0) \\ &= \frac{\sum_{j\ impair}^{t-1} \binom{w}{j}\binom{n-w-1}{t-1-j}}{\binom{n-1}{t-1}} \end{aligned}$$

En traçant la courbe de $q_1(\mathbf{e},t,i) - q_0(\mathbf{e},t,i)$, pour différents t et $\mathbf{e}$, on trouve :

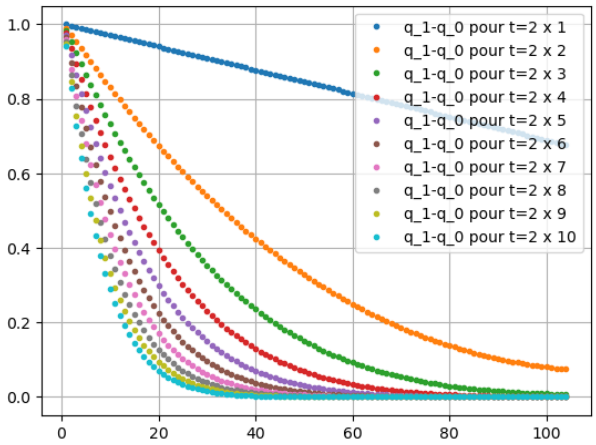


FIGURE 2.4 – $w_H(\mathbf{e})$ en abscisses et $q_1(\mathbf{e}, t, i_0) - q_0(\mathbf{e}, t, i_0)$ en coordonnées

Autrement dit, la probabilité (étant donné $\mathbf{h}$) que $< \mathbf{y}|\mathbf{h} >= 1$ dépend de la présence ou pas d'une erreur à la position i_0 de $\mathbf{y}$. Plus précisément, on a : $q_1(\mathbf{e}, t, i_0) > q_0(\mathbf{e}, t, i_0)$, et , plus t est grand, moins on arrive à distinguer ces deux quantités.
Le décodage statistique est donc un algorithme en deux étapes :

— Pour $i \in \{1..n\}$ fait :
  — Étape 1 : Calculer $S_i \subset \{\mathbf{h} \in C^\perp \mid w_H(\mathbf{h}) = t \text{ et } h_i = 1\}$ avec t petit.
  — Étape 2 : Calculer la valeur du compteur $V_i := \sum_{\mathbf{h} \in S_i} \mathbf{y}.\mathbf{h}^T$.
— Soit $\mathcal{U}(\mathbf{y}) := [V_1, V_2, .., V_n]$.

  — Analyser $\mathcal{U}(\mathbf{y})$, trouver les positions de $\mathbf{y}$ qui sont probablement des erreurs.
  — Corriger ces erreurs et obtenir un vecteur $\mathbf{y}'$.
  Refaire la même chose avec $\mathbf{y}'$, jusqu'à obtenir le mot $\mathbf{c} \in C$.

Remarque : Les S_i peuvent être pré-calculés en avance.

On a donc intérêt à prendre un ensemble S le plus grand possible et d'espérer que $w \leq d_{GV}(n, k)$, comme ça, par définition de la distance de Gilbert-Varshamov (A.1), pour tout $\mathbf{y} \in \mathbb{F}_2^n$, on peut trouver une décomposition de y de la forme $\mathbf{y} = \mathbf{c} + \mathbf{e}$ avec $\mathbf{c} \in C$ et $w_H(\mathbf{e}) = w$.
Sans rentrer dans les détails (qu'on peut trouver facilement dans la thèse [7] avec la complexité de l'algorithme), on voit bien que si on arrive à récolter assez de vecteurs $\mathbf{h} \in C^\perp$, d'un poids fixe, alors on arrivera à résoudre le problème de décodage. Or, **il est difficile de récolter de tels vecteurs**. En effet, c'est un problème qui est aussi difficile que le **problème de décodage**.
Ce dernier point essentiel, n'est plus difficile lorsque C est un code **QC-LDPC** qui a une matrice génératrice de la forme : $E := (E_1|E_2|..|E_p)$ où p est pair et les E_i sont des matrices circulantes.

2.6.2 Décodage statistique d'un type particulier de codes $QC - LDPC$

Dans la section précédente, on a travaillé avec un code C de type $[np, p]_2$ généré par une matrice génératrice de la forme :

$$E := (E_1|E_2|..|E_p)$$

où p est pair et les $E_i \in \mathbb{F}_2^{n \times n}$ sont des matrices circulantes de poids w par ligne.
Lors de mon stage, j'ai trouvé que, avec une grande probabilité, une matrice de parité associée au code

C est de la forme :

$$D := \begin{bmatrix} E_2^T & E_1^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_3^T & E_4^T & E_1^T & E_2^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_4^T & E_3^T & E_2^T & E_1^T & E_6^T & E_5^T & .. & E_p^T & E_{p-1}^T \\ E_5^T & E_6^T & E_4^T & E_3^T & E_1^T & E_2^T & .. & E_p^T & E_{p-1}^T \\ E_6^T & E_5^T & E_4^T & E_3^T & E_2^T & E_1^T & .. & E_p^T & E_{p-1}^T \\ : & : & : & : & : & : & .. & : & : \\ E_{p-1}^T & E_p^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_1^T & E_2^T \\ E_p^T & E_{p-1}^T & E_4^T & E_3^T & E_6^T & E_5^T & .. & E_2^T & E_1^T \end{bmatrix}$$

Le code C est donc un code **QC-LDPC** (par définition). Et remarquons que chaque ligne de D est de poids fixe $\mu := p.w$. Ainsi, on obtient $(p-1)n$ vecteurs appartenant à $C^\perp$ (ce sont les vecteurs ligne de D).

Le problème est que $(p-1)n$ est assez petit. Un autre résultat qu'on peut trouver facilement est que

$$D^* := \begin{bmatrix} E_2^T & E_1^T & 0 & 0 & 0 & 0 & .. & 0 & 0 \\ E_3^T & 0 & E_1^T & 0 & 0 & 0 & .. & 0 & 0 \\ E_4^T & 0 & 0 & E_1^T & 0 & 0 & .. & 0 & 0 \\ E_5^T & 0 & 0 & 0 & E_1^T & 0 & .. & 0 & 0 \\ E_6^T & 0 & 0 & 0 & 0 & E_1^T & .. & 0 & 0 \\ : & : & : & : & : & : & .. & : & : \\ E_{p-1}^T & 0 & 0 & 0 & 0 & 0 & .. & E_1^T & 0 \\ E_p^T & 0 & 0 & 0 & 0 & 0 & .. & 0 & E_1^T \end{bmatrix}$$

est aussi une matrice de parité associée à C de poids fixé $2w$ par ligne.

De la matrice D^* on conclut que toute matrice de la forme $(0|0|..|E_i^T|0|..|0|E_j^T|0|..|0)$ avec $p \geq i > j$ vérifie :

$$(0|0|..|E_i^T|0|..|0|E_j^T|0|..|0).E^T = 0$$

i.e, les lignes de la matrice $(0|0|..|E_i^T|0|..|0|E_j^T|0|..|0)$ sont des éléments de $C^\perp$ de poids $2.w$.
En dénombrant ces matrices, on trouve $\binom{p}{2}$ choix possibles : d'abord On choisit $j \in [|1, p-1|]$, puis on choisit $i \in [|i+1, p|]$.
Sur **Magma**, on effectue le test suivant :

- On génère aléatoirement une matrice $E := (E_1|E_2|..|E_p)$ avec p pair ;
- On construit un ensemble $S \subset C^\perp$ contenant les matrices de la forme $(0|0|E_i^T|0|..|0|E_j|0|..|0)$ avec $p \geq i > j$;
- Puis on répète :
  1. $\mathbf{c} \leftarrow U(C)$;
  2. $\mathbf{e} \leftarrow \mathbb{F}_2^L$ de poids $w \leq d_{GV}(L, n)$;
  3. $\mathbf{r} \leftarrow \mathbf{c} + \mathbf{e}$;
  4. Pour $i \in [|1, n|]$ on fait :
     (a) Choisir $S_i \subset S$ contenant des vecteurs $\mathbf{x}$ avec $x_i = 1$;
     (b) On calcule $V_i := \sum_{\mathbf{h} \in S_i} < \mathbf{r}|\mathbf{h} >$;
  5. Comme on connaît $\mathbf{e}$, il suffit de comparer $V_1, V_2, .., V_n$ et voir s'il y a une différence entre les V_i tels que $e_i = 1$ et les V_j tels que $e_j = 0$;

Expériences (sur Magma) et résultats :

```
////////////////// Parameters /////////////////
R := 1/2                    ;
b := 10                     ;
n := 32                     ;
k := 16                     ;
r := n-k                    ;
L := b*n                    ;
w := 3                      ;
d := binary_gv(L,n)         ;
```

On construit en suite l'ensemble S de la manière suivante :

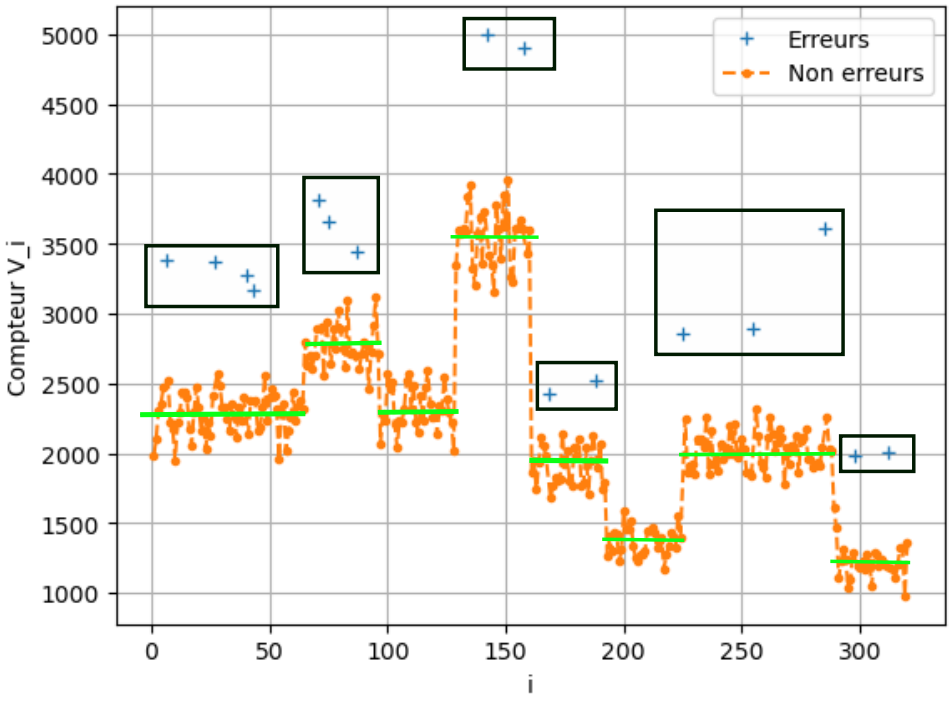
```
pip := 0 ;
repeat
        pip;
        i1 := Random({1..b div 2});
        i2 := Random({i1+1..b});
        left := KMatrixSpace(GF(2),n,(i1-1)*n) ! 0;
        middle := KMatrixSpace(GF(2),n,(i2-i1-1)*n) ! 0;
        right := KMatrixSpace(GF(2),n,(b-i2)*n) ! 0;
        Append(~Vide, HorizontalJoin(left,HorizontalJoin(Transpose(E[i2]),HorizontalJoin(
            middle,HorizontalJoin(Transpose(E[i1]),right)))));
        pip +:= 1;
until pip ge (Factorial(b-1)*3) div 2 ;
```

On essaye le test suivant pour plusieurs poids $w_H(\mathbf{e})$ (par exemple, dans le test suivant, on prend : $w_H(\mathbf{e}) = 25 \sim \lfloor \frac{d_{GV}(L,n)}{6} \rfloor$) :

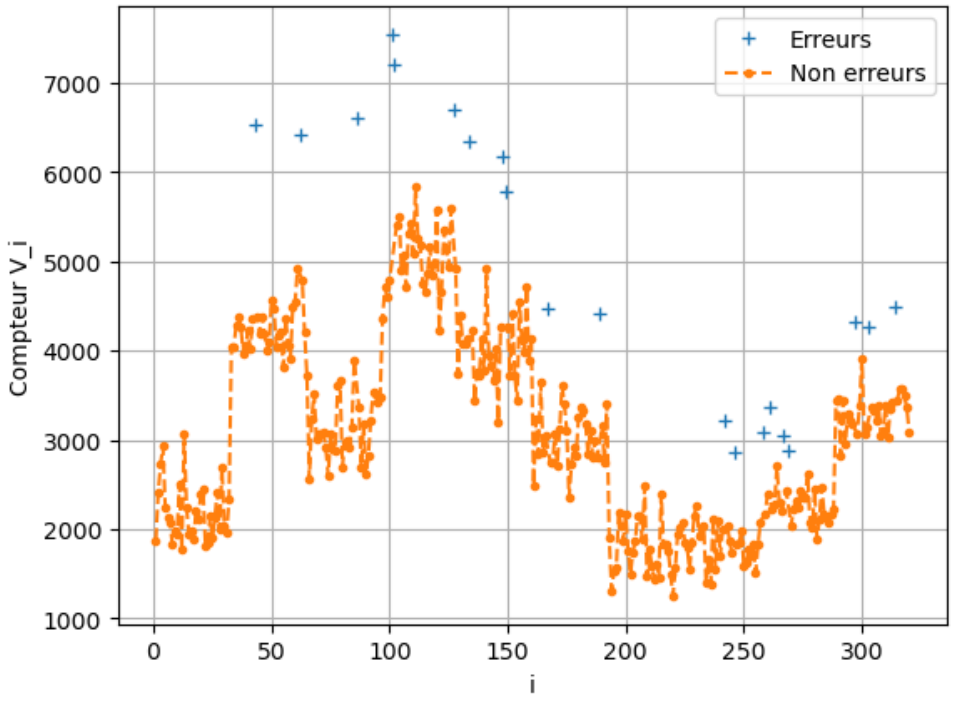
```
c := Random(C_E);
e := VectorSpace(GF(2),L) ! 0;
T := RandomSubset({1..L},d div 22);
T := [i : i in T];
for i in T do
        e[i] := 1;
end for;

x := c+e;

bar_T := [i : i in {1..L} | i notin T];

O1 := [];

for com in T do
        com;
        S_com := [];
        for v in {1..#Vide} do
                for ligne in {1..n} do
                        if Vide[v][ligne][com] ne 0 then
                                Append(~S_com,Vide[v][ligne]);
                        end if;
                end for;
        end for;
        Append(~O1, [&+[x[m]*S_com[h][m] : m in {1..L}] : h in {1..#S_com} ]);
end for;

FA := [#[i : i in O1[j]|i eq 1 ] : j in {1..#O1} ];
```

```
O2 := [];

for com in bar_T do
        com;
        S_com := [];
        for v in {1..#Vide} do
                for ligne in {1..n} do
                        if Vide[v][ligne][com] ne 0 then
                                Append(~S_com,Vide[v][ligne]);
                        end if;
                end for;
        end for;
        Append(~O2, [&+[x[m]*S_com[h][m] : m in {1..L}] : h in {1..#S_com} ]);
end for;
F_B := [#[i : i in O2[j]|i eq 1 ] : j in {1..#O2} ];
```

On trace les compteurs pour chaque position de **r**. On trouve :



Pour $w_H(\mathbf{e}) = 20$, on arrive quand même à distinguer les positions non nulles de $\mathbf{e}$ (Les + en bleu dans les figures) :



Par contre, dès qu'on prend des $\mathbf{e} \in \mathbb{F}_2^L$ de poids $w_H(\mathbf{e}) \geq 25$, on ne peut plus distinguer la position de certaines erreurs :

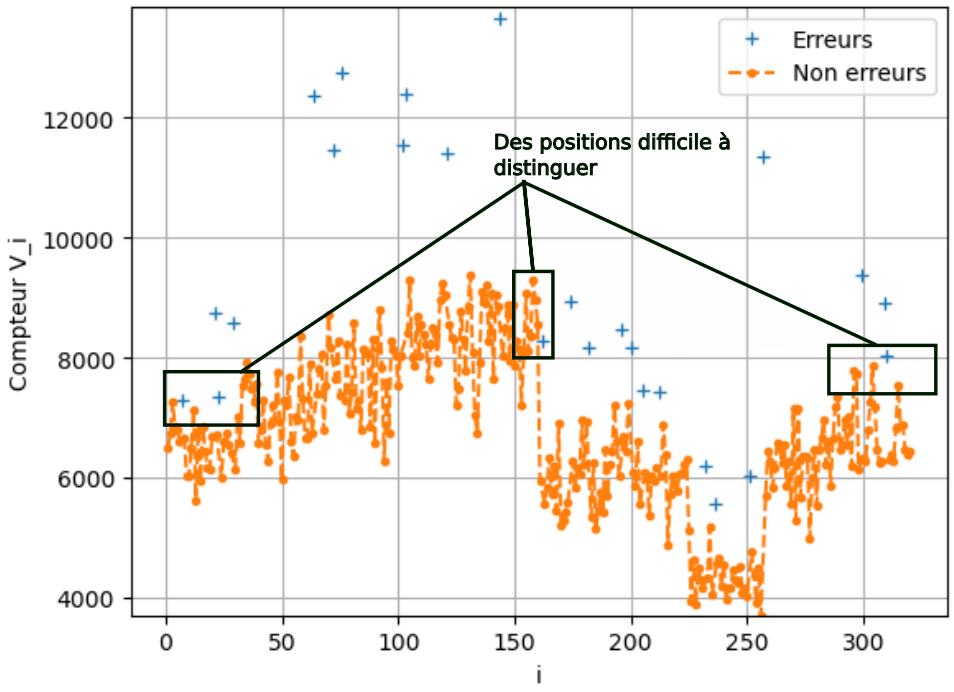


FIGURE 2.5 – $25 \sim d_{GV}(L, n)/4$ erreurs

Pour pouvoir corriger ces erreurs, une version itérative du **décodage statistique** existe. Cela consiste à corriger les positions qui sont très probablement des erreurs, puis de refaire le décodage statistique sur

le nouveau. Et ainsi de suite, jusqu'à corriger toutes les erreurs : on connaît le nombre d'erreurs sur le message. On prend l'exemple de la figure 2.5. On arrive à distinguer toutes les erreurs en 4 étapes :

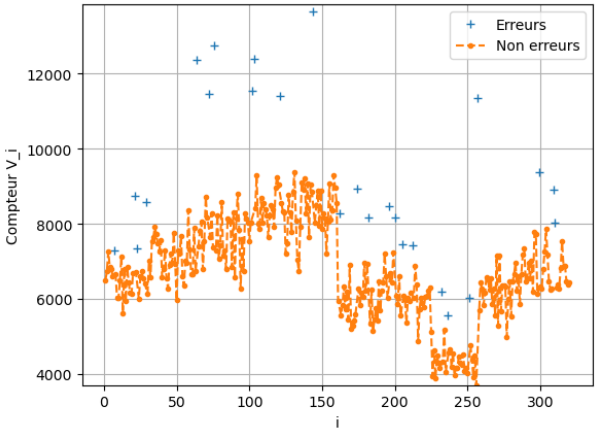


FIGURE 2.6 – Il reste 25 erreurs à corriger

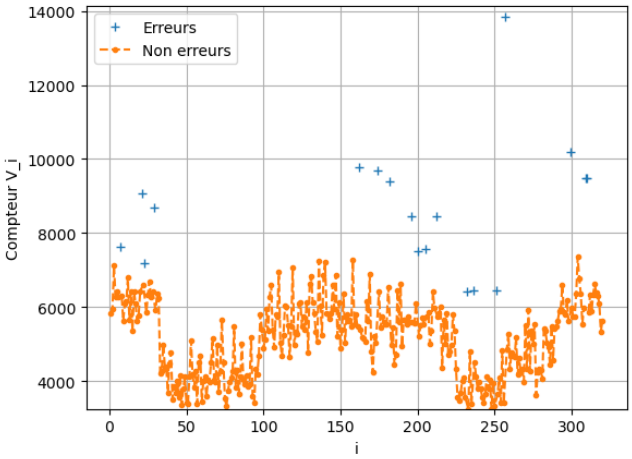


FIGURE 2.7 – Il reste 17 erreurs à corriger

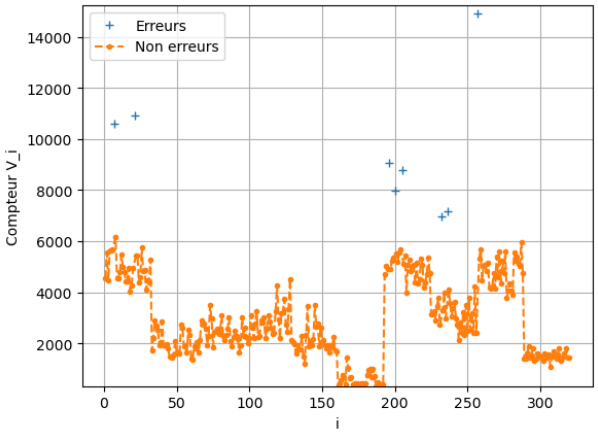


FIGURE 2.8 – Il reste 8 erreurs à corriger

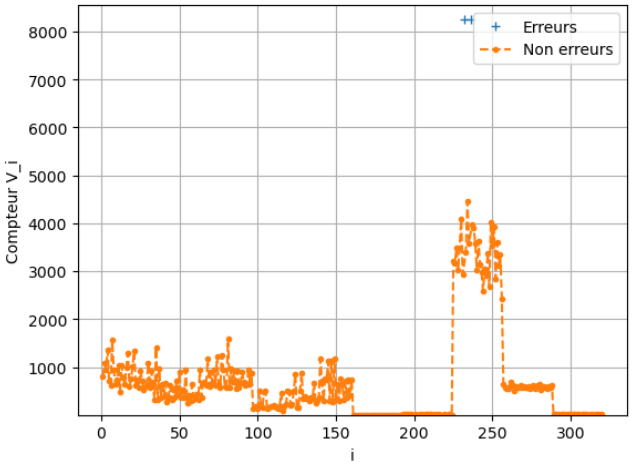


FIGURE 2.9 – Il reste 17 erreurs à corriger

À la fin, on arrive directement à trouver le mot du code et à éliminer le bruit $\mathbf{e}$.

Quitte à faire quelques itérations, on arrive à trouver l'erreur $\mathbf{e}$ à chaque fois, même pour des $\mathbf{e}$ de poids assez grand, par exemple $w_H(e) \sim \frac{d_{GV}(L,n)}{2}$:

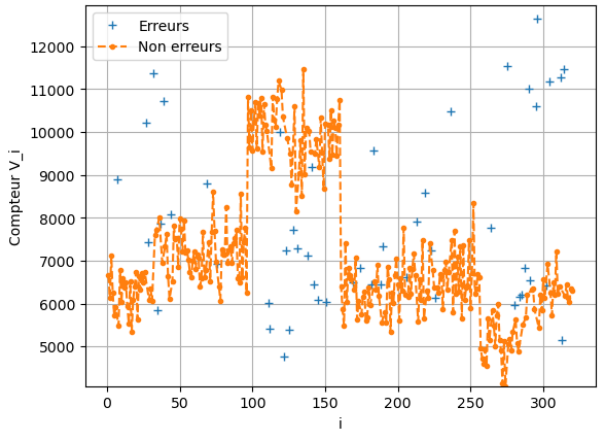


FIGURE 2.10 – Il reste 50 erreurs à corriger

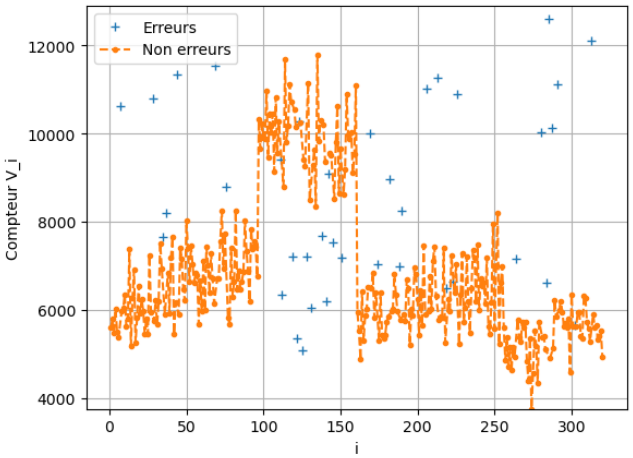


FIGURE 2.11 – Il reste 35 erreurs à corriger

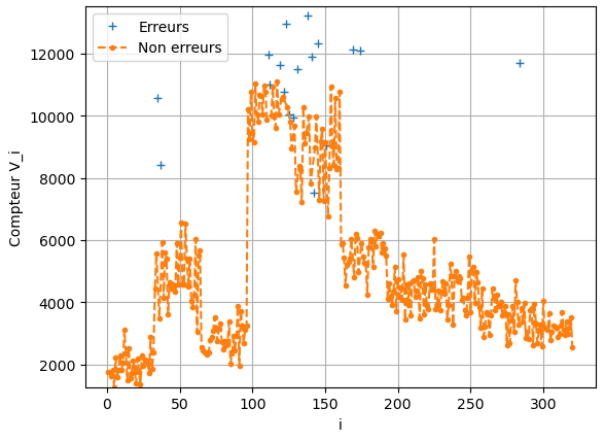


FIGURE 2.12 – Il reste 18 erreurs à corriger

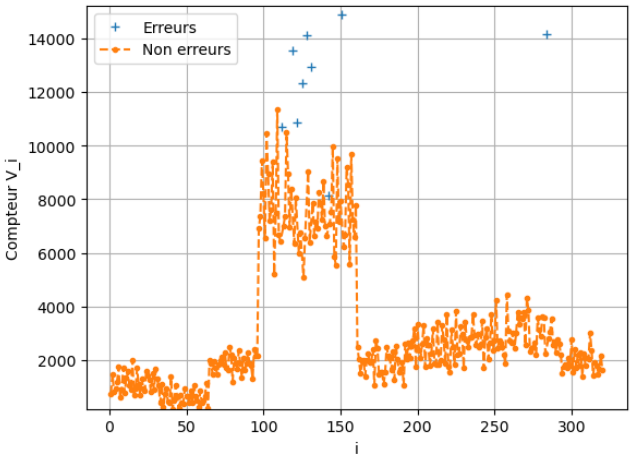


FIGURE 2.13 – Il reste 9 erreurs à corriger

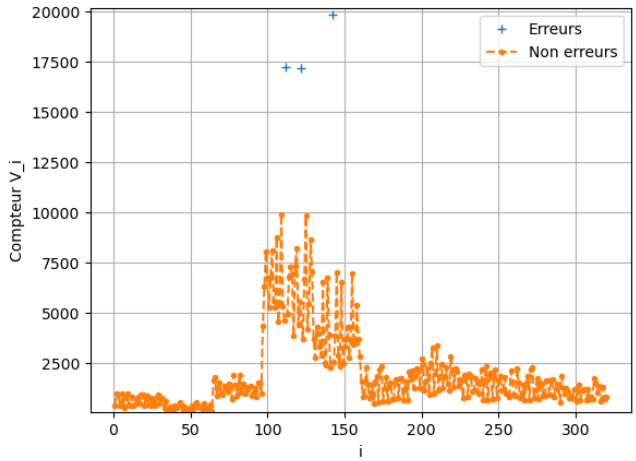


FIGURE 2.14 – Il reste trois erreurs

Remarques :

— Ce calcul prend assez de temps ($\sim$ 4 minutes). Surtout lorsqu'on doit choisir les ensembles

$$S_i := \{h \in S \mid h_i \neq 0\}$$

Pour décoder la position i du mot reçu. Si on veut décoder avec un tel code, il suffit de calculer les S_i en avance, et les garder en mémoire.
Un avantage de cette méthode est le fait de pouvoir paralléliser le calcul : Avec L processeurs, on arrivera à éliminer le bruit très vite, car chaque processeur va devoir calculer le compteur d'une position précise et les S_i on les a déjà.

— Le calcul de l'ensemble S est facile, i.e, ne prends pas de temps. C'est ce qui rend ce type de codes vulnérables d'un point de vue cryptographique, mais assez performant d'un point de vue codes correcteurs d'erreurs. Car la vraie difficulté pour faire du décodage statistique est de calculer efficacement et rapidement l'ensemble S.

Bibliographie

[1] BIKE - Bit Flipping Key Encapsulation.

[2] https ://www.litislab.fr/.

[3] Summary of New Features in Magma V2.28.

[4] Berlekamp, E., McEliece, R., and Tilborg, H. On the inherent intractability of certain coding problems (Corresp.). 384–386.

[5] Chaulet, J. Etude de cryptosystèmes à clé publique basés sur les codes mdpc quasi-cycliques.

[6] Collectif, Yger, A., and Weil, J.-A. *Mathématiques L3 appliquées : Cours complet avec 500 tests et exercices corrigés*. PEARSON.

[7] Debris, T. Décodage statistique.

[8] Debris, T., Ducas, L., Resch, N., and Tillich, J.-P. Smoothing Codes and Lattices : Systematic Study and New Bounds.

[9] Döttling, N. Cryptography based on the Hardness of Decoding.

[10] EDMOND, N. N. S. Le cryptosysteme mceliece.

[11] Gallager, R. G. Low-density parity-check codes. 21–28.

[12] Hong, J., and Vetterli, M. Simple Algorithms for BCH Decoding. 2324–2333.

[13] Jabri, A. A. A Statistical Decoding Algorithm for General Linear Block Codes. In *Cryptography and Coding*, B. Honary, Ed., Springer, pp. 1–8.

[14] Justesen, J., and Høholdt, T. *A Course in Error-Correcting Codes*, second edition ed. EMS Textbooks in Mathematics. European Mathematical Society.

[15] Lloyd R. Welch, E. R. B. error correction for algebraic block codes.

[16] Madhu, S. Decoding of reed solomon codes beyond the error-correction bound. 180–193.

[17] McEliece, R. J. A public-key cryptosystem based on algebraic. *Coding Thv 4244* (1978), 114–116.

[18] Shor, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. 1484–1509.

Annexe A

Codes Correcteurs d'erreurs

A.1 Définitions générales

On va d'abord définir quelques notions de la théorie des codes correcteurs d'erreurs :

Définition 11 (Code linéaire)**.** *Soit $\mathbb{F}_q$ un corps fini avec $q = p^s$ où p est un nombre premier et $s \in \mathbb{N}^*$. Un **code linéaire** C sur $\mathbb{F}_q$, de longueur $n \in \mathbb{N}$ et de dimension $k \in \mathbb{N}$ est un sous-$\mathbb{F}_q$-espace vectoriel de $\mathbb{F}_q^n$ de dimension k. On dit que C est un code $[n,k]_q$.*
*Les éléments de C sont appelés les mots de code. Le **rendement** du code est $R := \frac{k}{n}$.*

On peut représenter un code linéaire $[n,k]_q$ de deux manières équivalentes :

Définition 12 (Matrice génératrice)**.** *Une **matrice génératrice** d'un code C de type $[n,k]_q$ est une matrice $G \in \mathbb{F}_q^{k\times n}$ dont les vecteurs lignes constituent une base de C en tant que $\mathbb{F}_q$-espace vectoriel. Ainsi,*

$$C = \{\mathbf{x}.G \mid \mathbf{x} \in \mathbb{F}_q^k\}$$

Définition 13 (Le dual d'un code)**.** *Soit C un code de type $[|n,k|]_q$. Son **dual** est défini par :*

$$C^\perp := \{\mathbf{x} \in \mathbb{F}_q^n \mid \forall \mathbf{y} \in C,\ <\mathbf{y},\mathbf{x}>= 0\}$$

Définition 14 (Matrice de parité)**.** *Une **matrice de parité** d'un code C de type $[n,k]_q$ est une matrice $H \in \mathbb{F}_q^{(n-k)\times n}$ de rang $n-k$ telle que $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}.H^T = 0\}$. Autrement dit, H est une matrice génératrice du **code dual** $C^\perp$.*

Remarque : On peut représenter un code soit par sa matrice génératrice, soit par sa matrice de parité.

Proposition 4. *Si une matrice génératrice d'un code C est de la forme $G = (I_k|A)$, alors $H := (-A^T|I_{n-k})$ est une matrice de parité de C.*

Définition 15 (Distance de Hamming)**.** *Soit*

$$\begin{aligned} d_H : \mathbb{F}_q^n \times \mathbb{F}_q^n &\to [|0,n|] \\ (\mathbf{x},\mathbf{y}) &\mapsto Card(\{i \in [|1,n|] \mid x_i \neq y_i\}) \end{aligned}$$

*Alors $(\mathbb{F}_q^n, d_H)$ est un espace métrique, i.e, d_H est une distance sur $\mathbb{F}_q^n$, appelée **distance de Hamming**. Pour $\mathbf{x} \in \mathbb{F}_q^n$, on définit son **poids** (de Hamming) par $w_H(\mathbf{x}) := d_H(\mathbf{x},\mathbf{0})$. Autrement dit, c'est le nombre de coordonnées non nulles de $\mathbf{x}$.*

On va maintenant définir une notion fondamentale en théorie des codes correcteurs :

Définition 16 (Distance minimale d'un code)**.** *Soit C un code $[n,k]_q$. On définit la* ***distance minimale*** *de C comme $d_C := min(\{w_H(\mathbf{x}) \mid \mathbf{x} \in C \backslash \{\mathbf{0}\}\})$. On dit alors que C est un code $[n,k,d_C]_q$.*

Remarque : On a aussi $d_C := min(\{d_H(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \neq \mathbf{y} \in C\})$.

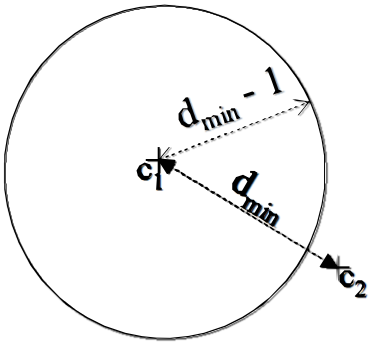


FIGURE A.1 – Boule de centre un mot du code $\mathbf{c_1}$ et de rayon $d_{min} - 1$

Définition 17 (Capacité de correction d'erreurs d'un code)**.**
Soient C un code de longueur n et $t \in [|1,n|]$. Un algorithme $\mathcal{A}$ de décodage est t-correcteur pour le code C si $\forall \mathbf{c} \in C$ et $\forall \mathbf{e} \in \mathbb{F}_q^n$ tel que $w_H(\mathbf{e}) \leq t$, on a $\mathcal{A}(\mathbf{c}+\mathbf{e}) = \mathbf{c}$.
Si un tel algorithme existe, on dit que le code C est t-correcteur.

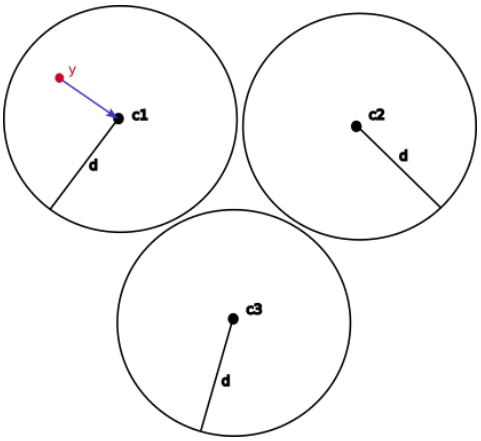


FIGURE A.2 – Décodage de $\mathbf{y}$ en $\mathbf{c_1}$ jusqu'à d erreurs

Définition 18 (Distance de Gilbert-Varshamov)**.** *Pour $k \leq n$ deux entiers, La* ***distance de Gilbert-Varshamov*** *sur $\mathbb{F}_q^n$, notée $d_{GV}(n,k,q)$, est le plus petit $d \in \mathbb{N}$ tel que*

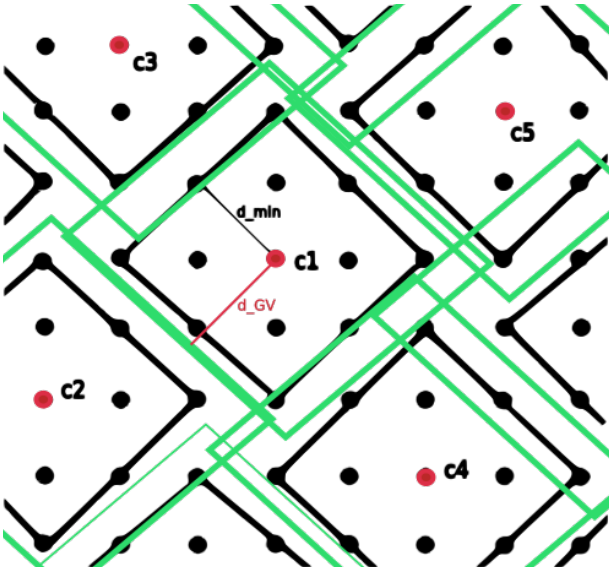
$$\sum_{i=0}^{d-1} \binom{n}{k} (q-1)^i \geq q^{n-k}$$

Proposition 5. *En conservant les notations de la définition précédente, on a l'approximation suivante :*

$$\binom{n}{d_{GV}(n,k,q)} (q-1)^{d_{GV}(n,k,q)} \underset{n \to +\infty}{\sim} q^{n-k}$$

avec un rendement $R = \frac{k}{n}$ constant.

Intéressant : Pour $0 \leq k \leq n$ deux entiers, la distance de Gilbert-Varshamov $d_{GV}(n,k,q)$ est la distance à partir de laquelle, tout **code aléatoire** C de type $[n,k]_q$ vérifie, avec une grande probabilité, la propriété :
$\mathbb{F}_q^n = \bigcup_{c \in C} B(\mathbf{c}, d)$ avec $B(\mathbf{c}, d) := \{\mathbf{x} \in \mathbb{F}_q^n \mid d_H(\mathbf{c}, \mathbf{x}) \leq d\}$.



Proposition 6 (Borne de Singleton)**.** *Soit C un code $[n,k,d]_q$. Alors $d \leq n - k + 1$.*

Preuve 3. *Soit $G \in \mathbb{F}_q^{k \times n}$ une matrice génératrice du code C. Quitte à faire des opérations élémentaires sur les lignes de G, on peut mettre G sous la forme $G' := (I_k|A)$ avec $A \in \mathbb{F}_q^{(n-k)\times(n-k)}$. Toutes les lignes, notées $\mathbf{L_1}, .., \mathbf{L_k}$ de G' sont des mots du code C, et par définition $d \leq w_H(\mathbf{L_1}) \leq 1 + (n-k)$.* ■

A.2 Problèmes difficiles en théorie des codes correcteurs

Définition 19 (Problème de décodage (DP [1]))**.** *Le problème de décodage $\mathbf{DP_{n,k,w}}$ est le suivant :*

- ***Entrée :*** *$G \in \mathbb{F}_q^{k \times n}$, $\mathbf{y} \in \mathbb{F}_q^n$ et $w \in [|0, n|]$ un poids ;*
- ***Sortie :*** *$\mathbf{m} \in \mathbb{F}_q^k$ tel que $d_H(\mathbf{y}, \mathbf{m}.G) \leq w$;*

En pratique, on cherche à résoudre plutôt le problème suivant :

Définition 20 (Problème de décodage par syndrome (SDP [2]))**.** *Le problème de **décodage par syndrome** $\mathbf{SDP_{n,k,w}}$ est le suivant :*

- ***Entrée :*** *$H \in \mathbb{F}_q^{(n-k)\times n}$ de rang plein, $\mathbf{s} \in \mathbb{F}_q^{n-k}$ et $w \in [|0, n|]$ un poids ;*
- ***Sortie :*** *$\mathbf{e} \in \mathbb{F}_q^n$ tel que $w_H(\mathbf{e}) \leq w$ et $H.\mathbf{e}^T = \mathbf{e}^T$;*

Proposition 7. *Les deux problèmes sont équivalents.*

Preuve 4.

- *Supposons qu'on peut résoudre DP.*
  H étant de rang $n-k$, on peut trouver en temps polynomial G de rang k telle que $G.H^T = 0$.
  On peut ensuite trouver (Systèmes linéaires, i.e, en temps polynomial) $\mathbf{y}$ tel que $H.\mathbf{y}^T = \mathbf{s}^T$.
  On a donc $H.(\mathbf{y} - \mathbf{e})^T = 0$ ($\mathbf{y} - \mathbf{e}$ est un mot du code C_G) avec $H.\mathbf{e}^T = \mathbf{s}^T$ et $w_H(\mathbf{e}) = t$. Donc

1. Decoding Problem en anglais
2. Syndrome Decoding Problem en anglais

$\mathbf{y} = \mathbf{c} + \mathbf{e}$ *pour un* $\mathbf{c} = \mathbf{x}.G$ *un mot de code* C_G*, et on peut donc trouver* $\mathbf{c}$ *car on peut décoder (par hypothèse), et donc on a* $\mathbf{e} = \mathbf{y} - \mathbf{c}$.

— *Supposons qu'on peut résoudre SD.*
On a $\mathbf{y} = \mathbf{c} + \mathbf{e}$ *avec* $\mathbf{c} \in C_G$*. On peut calculer une matrice de parité* H *de* G *en temps polynomial (pivot de Gauss), et on a en suite* $\mathbf{s}^T = H.\mathbf{y}^T = H.\mathbf{e}^T$ *et* $w_H(\mathbf{c}) = t$.
Par hypothèse, on trouve directement $\mathbf{e}$. ■

On ne peut parler de la difficulté sans avoir un problème de décision sous nos mains.

Définition 21 (Problème de décodage par syndrome (SDP))**.** *Le problème de décodage par syndrome* $SDP_{n,k,w}$ *est le suivant :*

— ***Entrée :*** $H \in \mathbb{F}_q^{(n-k)\times n}$ *de rang plein,* $\mathbf{s} \in \mathbb{F}_q^{n-k}$ *et* $w \in [|0, n|]$ *un poids ;*
— ***Sortie :*** *Existe-t-il* $\mathbf{e} \in \mathbb{F}_q^n$ *tel que* $w_H(\mathbf{e}) \leq w$ *et* $H.\mathbf{e}^T = \mathbf{s}^T$.

Theorem 1 ([4])**.** *Pour* $q = 2$*, le SDP est NP-Complet.*

A.3 Familles connues de codes

Ici, on va introduire les codes qu'on a utilisés pendant ce stage, et on va détailler leurs propriétés.

A.3.1 Codes de Reed-Solomon

A.3.1.1 Définitions et propriétés

Définition 22 (Code de Reed-Solomon[3] RS)**.** *Soit $\mathbb{F}_q$ un corps fini et soient $x_1, x_2, .., x_n$ des éléments deux-à-deux distincts non nuls de $\mathbb{F}_q$.*
*Pour $k \leq n$ deux entiers, on considère $RS(n,k,\mathbf{x}) := \{(f(x_1), f(x_2), .., f(x_n)) \mid f \in \mathbb{F}_q[X]_{\leq k-1}\}$ avec $\mathbf{x} := (x_1, .., x_n)$. $RS(n,k,\mathbf{x})_q$ est le code de **Reed-Solomon** de longueur n, de dimension k et de support $\mathbf{x}$ sur $\mathbb{F}_q$.*

Theorem 2. *Les codes de Reed-Solomon $RS(n,k,\mathbf{x})_q$ sont des codes $[n, k, n-k+1]_q$.*

Preuve 5.

1. *Le code est de longueur n par définition.*
2. *La linéarité du code vient de la linéarité de l'espace des polynômes de degré inférieurs ou égales à $k-1$.*
3. *Le code est de dimension k :*
   - *On a $Card(RS(n,k,\mathbf{x})_q) \leq q^k$ car à chaque polynôme de degré $\leq k$ on peut associer un mot du code ;*
   - *De plus, cette association est injective : si $(f(x_1), .., f(x_n)) = (g(x_1), .., g(x_n))$ et f, g sont deux polynômes de degré au plus $k-1$, alors le polynôme $f-g$ est de degré au plus $k-1 < n$ et s'annule en n points distincts, i.e, $f-g$ est le polynôme nul, i.e, $f = g$. Ainsi, $Card(RS(n,k,\mathbf{x})_q) = q^k$.* ■
4. *Notons d_{min} la distance minimale du code $RS(n,k,\mathbf{x})_q$.*
   - *La borne de Singleton assure que $d_{min} \leq n-k+1$. (Proposition 3)*
   - *Si f est un polynôme de degré au plus $k-1$ non nul, alors il ne peut s'annuler en plus de $k-1$ points, i.e, $w_H(f(x_1), .., f(x_n)) \geq n-(k-1) = n-k+1$.* ■

Les codes $[n,k]_q$ comme Reed-Solomon, dont la distance minimale est égale à $n-k+1$, sont appelés **codes à distance séparable maximale (MDS)**.

A.3.1.2 Décodage des codes de Reed-Solomon

Les codes de **Reed-Solomon** de type $[n,k,d]_q$ sont des codes $\lfloor \frac{n-k}{2} \rfloor$-correcteurs.
L'algorithme de décodage le plus simple est l'algorithme de **Welch-Berlekamp** ([15]) qu'on décrit ci-dessous.

Soit $\mathbf{r} = \mathbf{c} + \mathbf{e}$ le mot reçu tel que $w_H(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. En suite, on cherche un polynôme bivarié $Q(x,y) = Q_0(x) + y.Q_1(x)$ tel que :

- $Q(x_i, r_i) = 0, \ \forall i \in [|1,n|]$;
- $deg(Q_0) \leq n-1-t$;
- $deg(Q_1) \leq n-1-t-(k-1)$;

Q un polynôme d'interpolation bivarié sur les points (x_i, r_i).

Proposition 8. *Il existe au moins un polynôme $Q(x,y)$ non nul vérifiant ces conditions.*

3. Irving S. Reed et Gustave Solomon sont deux ingénieurs et mathématiciens américains connus pour avoir co-inventé les codes de Reed-Solomon en 1960.

Preuve 6. *On utilise l'algèbre linéaire.*
Si on note $Q_0(X) := \sum_{i=0}^{n-1-t} q_{i,0}.X^i$ *et* $Q_1(X) := \sum_{i=0}^{n-1-t-(k-1)} q_{i,1}.X^i$, *alors :*

$$(\forall j \in [|1,n|],\ Q(x_j,r_j)=0) \implies \begin{bmatrix} Q^0 & Q^1 \end{bmatrix} . \begin{bmatrix} q_{0,0} \\ : \\ q_{n-1-t,0} \\ q_{0,1} \\ : \\ q_{n-1-t-(k-1),1} \end{bmatrix} = 0$$

Où pour $b \in \{0,1\}$, $Q^b := (r_i^b.x_i^j)_{i,j} \in \mathbb{F}_q^{n\times(n-t)}$.
Comme $n-t+n-t-(k-1) = 2.n-2.t-(k-1)$ *et* $t = \lfloor \frac{n-k}{2} \rfloor$, *alors* $n-t+n-t-(k-1) \geq n+1$, *et donc, la matrice* $\begin{bmatrix} Q^0 & Q^1 \end{bmatrix}$ *a n lignes et plus de* $n+1$ *colonnes. Le système admet donc au moins une solution non nulle.* ■

Theorem 3. *Soit* $\mathbf{c} := (f(x_1),..,f(x_n)) = f(\mathbf{x})$ *avec* $deg(f) \leq k-1$ *un mot d'un code* $RS(n,k,\mathbf{x})_q$. *Soit* $\mathbf{r} = \mathbf{c}+\mathbf{e}$ *le mot reçu avec* $w_H(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. *Alors si on construit* Q *comme ci-dessus, on* $f = -\frac{Q_0}{Q_1}$.

Preuve 7. *On a* $Q(x_i,r_i)=0$ *pour* $i \in [|1,n|]$, *i.e,* $Q(x_i,f(x_i))=0$ *lorsque* $e_i = 0$. *Ainsi, le polynôme uni-varié* $Q(X,f(X))$ *admet au plus* $n-t$ *racines distinctes. Or* $deg_Q(Q(X,f(X)) \leq n-1-t$ *! Le polynôme* $Q(X,f(X))$ *est donc le polynôme nul.* ■

Remarque : On a $Q(X,Y) = Q_1(X).(Y-f(X))$. Ainsi, $Q_1(x_i) = 0 \iff e_i \neq 0$. Q_1 est un **polynôme localisateur d'erreurs**. Pour décoder un mot reçu r, il suffit de construire la matrice $Q^* := \begin{bmatrix} Q^0 & Q^1 \end{bmatrix}$ et trouver une solution non nulle au système $Q^*.X = 0$. C'est l'étape dur de cet algorithme.

A.3.1.3 Décodage en liste des RS

Soit C un code $[n,k,d]_q$. Sa capacité de décodage (Définition 6) est de $t := \lfloor \frac{d-1}{2} \rfloor$. Mais que se passe-t-il si un mot $\mathbf{c} \in C$ est corrompu avec une erreur $\mathbf{e}$ de poids $\tau > t$?
Si $\tau > t$, et $\mathbf{y} \in \mathbb{F}_q^n$, le décodage en liste consiste à renvoyer tous les mots de code C dans la boule de Hamming $B(\mathbf{y},\tau)$.

Je vais décrire l'algorithme de **Sudan**[4] ([16]), pour le décodage en liste des **RS**.

On prend un code $RS(n,k,\mathbf{x})_q$.
On cherche un polynôme bivarié $Q(X,Y) = Q_0(X) + y.Q_1(X) + y^2.Q_2(X) + .. + y^\ell.Q_\ell(X)$ vérifiant :

1. $\forall j \in [|1,n|],\ Q(x_j,r_j) = 0$;
2. Pour $j \in [|0,\ell|]$, $deg(Q_j(X)) \leq n-\tau-1-j.(k-1)$;
3. Q n'est pas le polynôme nul.

Remarque : ℓ est la taille de la liste de décodage renvoyée.

Proposition 9. *Si* $\tau < \frac{n}{\ell+1} - (k-1).\frac{l}{2}$ *et* $n-\tau-1-\ell.(k-1) \geq 0$, *alors* $Q(X,Y)$ *ci-dessus existe.*

Preuve 8. *Comme dans la preuve de l'algorithme de* ***Welch-Berlekamp****, on a :*
La condition (1) implique :

$$\begin{bmatrix} Q^0|Q^1|..|Q^\ell \end{bmatrix} . \begin{bmatrix} q^1 \\ : \\ q^\ell \end{bmatrix}$$

4. Madhu Sudan est un informaticien et mathématicien américain d'origine indienne, connu pour ses contributions à la théorie des codes correcteurs d'erreurs, en particulier pour avoir introduit la technique de décodage en liste des codes de Reed-Solomon

Avec $Q^b := (r_i^b.x_i^j)_{i,j} \in \mathbb{F}_q^{n\times(n-\tau-i.(k-1))}$ *et* $q^i := \begin{bmatrix} q_{i,1} \\ q_{i,2} \\ : \\ q_{i,n-\tau-1-i.(k-1)} \end{bmatrix}$ *les coefficients du polynôme* Q_i.

Notons $Q^* := \left[Q^0|Q^1|..|Q^\ell\right]$.
C'est une matrice à n lignes et $\sum_{i=0}^{\ell}(n-\tau-j.(k-1)) = (\ell+1).(n-\tau) - (k-1).\frac{\ell.(\ell+1)}{2}$ *colonnes.*
Pour que Q existe, il faut que le nombre de colonnes de Q^* *soit supérieur à* $n+1$. *Autrement dit,* $n-\tau-(k-1).\frac{\ell}{2} \geq \frac{n+1}{\ell+1} > \frac{n}{\ell+1}$.
Comme Q_ℓ *n'est pas nul (Sinon ça ne sert à rien de mentionner* Q_ℓ*..), la deuxième inégalité vient de là.* ■

Theorem 4. *Si* $Q(X,Y)$ *vérifie les conditions précédentes, et si le mot du code envoyé est* $\mathbf{c} := (f(x_1),..,f(x_n))$ *avec* $deg(f) \leq k-1$, *alors* $(Y-f(X))$ *divise* $Q(X,Y)$.

Preuve 9. *Considérons le polynôme univarié* $Q(X,f(X))$. *Ce dernier est de degré* $deg(Q(X,f(X))) \leq n-\tau-1$.
De plus, $Q(X,f(X))$ *s'annule au moins* $n-\tau$ *valeurs (par la condition* 1 *sur* Q*) ! Ainsi,* $Q(X,f(X))$ *est le polynôme nul.*
Donc $Q_0(X) = -(f(X).Q_1(X) + .. + f(X)^\ell.Q_\ell(X))$, *et en remplaçant dans* $Q(X,Y)$, *on trouve* $Q(X,Y) = \sum_{i=1}^{\ell} Q_i(X)(Y^i - F(X)^i)$.■

L'algorithme de Sudan consiste à faire :

- On commence par trouver le polynôme Q en utilisant l'algèbre linéaire ;
- Factoriser le polynôme $Q(X,Y)$ dans $(\mathbb{F}_q[X])[Y]$ pour faire apparaître les composantes $Y-f(X)$ avec $deg(f) \leq k-1$. Chacun des polynômes f représente un mots de code possible pour $\mathbf{r}$.
- Voir l'implémentation en Magma ici : C.1.

On peut généraliser la construction des codes de Reed-Solomon pour obtenir ce qu'on appelle des **Codes de Reed-Solomon généralisés**

Définition 23 (Codes de Reed-Solomon généralisés (GRS [5])). *Soient* $\mathbf{x} := (x_1,..,x_n) \in (\mathbb{F}_q\backslash\{0\})^n$ *avec les* x_i *deux-à-deux distincts et* $\lambda = (\lambda_1,..,\lambda_n) \in (\mathbb{F}_q\backslash\{0\})^n$. *Un* ***GRS*** $RS_k(x,\lambda)$ *sur* $\mathbb{F}_q$ *est défini par :*

$$RS_k(\mathbf{x},\lambda) := \{(\lambda_1.f(x_1),..,\lambda_n.f(x_n)) \mid f \in \mathbb{F}_q[X]_{\leq k-1}\}$$

Par définition, on a que :

Proposition 10. *Un code de Reed-Solomon généralisé* $RS_k(\mathbf{x},\lambda)$ *sur* $\mathbb{F}_q$ *est un code* $[n,k,n-k+1]_q$.

Preuve 10. *En effet :*

- *La dimension est bien k : Les* $\mathbf{e_i} := (\lambda_1.x_1^{i-1},..,\lambda_n.x_n^{i-1})$ *avec* $i \in [|1,k|]$, *constituent bien une base du* $\mathbb{F}_q$*-espace vectoriel.*
- *La distance minimale ne diffère pas d'un code de Reed-Solomon normal, car la multiplication des coordonnées d'un vecteur par des éléments non nuls ne change pas le poids du vecteur.*

Le décodage des **GRS** se réduit celui des codes de Reed-Solomon normaux. En effet :

- On reçoit le mot $(r_1,..,r_n) = (\lambda_1.f(x_1),..,\lambda_n.f(x_n)) + (e_1,..,e_n)$ avec $w_H((e_1,..,e_n)) \leq \lfloor\frac{n-k}{2}\rfloor$;
- On calcule $(r_1/\lambda_1,..,r_n/\lambda_n) = (f(x_1),..,f(x_n)) + (e_1/\lambda_1,..,e_n/\lambda_n) : \lambda_i \neq 0,\ \forall i \in [|1,n|]$;
- On décode le mot $(r_1/\lambda_1,..,r_n/\lambda_n)$ en utilisant un décodeur des Reed-Solomon (A.3.1.2), et on trouve f : On peut faire ça, car les λ_i ne sont pas nuls, i.e, $w_H((e_1/\lambda_1,..,e_n/\lambda_n)) \leq \lfloor\frac{n-k}{2}\rfloor$. ■

5. Generalised Reed-Solomon

A.3.2 Codes alternants et codes de Goppa

Avant de parler des codes de **Goppa** [6], on va introduire une notion sur laquelle reposent ces codes :

Définition 24 (Codes alternants [6]). *On note $RS_{n-r}(\mathbf{x}, \lambda)$ un **GRS** de type $[n, n-r, r+1]_{q^m}$.*
*Un **code alternant** est défini par : $Al_k(\mathbf{x}, \lambda) := RS_{n-r}(\mathbf{x}, \lambda) \cap \mathbb{F}_q^n$*

Proposition 11. *Un code alternant définit comme ci-dessus est un code $[n, k, d]_q$ avec $n - m.r \leq k \leq n - r$ et $d \geq r + 1$.*

Preuve 11.

— *Il faut remarquer qu'un code alternant est un code **GRS** auquel on a enlevé des points. Conséquence : La distance minimale ne peut qu'augmenter ou rester comme elle est, i.e, $r + 1 \leq d$.*

— *Soient $\{e_1, .., e_m\}$ une base du $\mathbb{F}_q$-espace vectoriel $\mathbb{F}_{q^m}$ et $H = (h_{i,j})_{i,j} \in \mathbb{F}_{q^m}^{r \times n}$ une matrice de parité du code $RS_{n-r}(\mathbf{x}, \lambda)$. Alors $h_{i,j} := \sum_{p=1}^{m} h_{i,j}^{(p)}.e_p$. On remarque ainsi que :*

$$\begin{aligned}
&\mathbf{c} = (c_1, .., c_n) \in Al_k(\mathbf{x}, \lambda) \\
&\iff H.c^T = 0 \\
&\iff \forall i \in [|1, r|], \sum_{j=1}^{n} h_{i,j}.c_i = 0 \\
&\iff \forall i \in [|1, r|], \sum_{j=1}^{n} \sum_{p=1}^{m} h_{i,j}^{(p)}.c_i = 0 \\
&\iff H'.\mathbf{c}^T = 0
\end{aligned}$$

Où $H' = \left(\begin{bmatrix} \mathbf{h_{i,j}^{(1)}} \\ \mathbf{h_{i,j}^{(2)}} \\ \vdots \\ \mathbf{h_{i,j}^{(m)}} \end{bmatrix} \right)_{i,j} \in \mathbb{F}_q^{mr \times n}$.
La matrice H' engendre le dual de notre code alternant, et donc $n - m.r \leq k \leq n - r$. ■

Remarque : On a donc une méthode pour construire des codes alternants, qui ont plusieurs avantages comparés aux codes de Reed-Solomon, parmi ces avantages, on trouve :

1. La taille de ces codes n'est plus bornée par la taille du corps sur lequel est défini le code.
2. La distance minimale de ces codes est plus grande que celle des Reed-Solomon.
3. On peut décoder *partiellement* ces codes en tant que sous codes d'un code de Reed-Solomon généralisé : En effet, si un mot du code $c \in Al_{n-r}(\mathbf{x}, \lambda)$ est corrompu avec $t \leq \lfloor \frac{n-k}{2} \rfloor$ erreurs, alors on peut le décoder avec un décodeur d'un GRS.
4. Ils sont très utiles en cryptographie vu qu'ils sont nombreux pour des paramètres donnés. D'ailleurs, McEliece a utilisé un cas particulier de ces codes pour définir son cryptosystème : Les **codes de Goppa** ci-dessous.

Définition 25 (Codes de Goppa). *Soient $L := \{x_1, x_2, .., x_n\} \subset \mathbb{F}_{q^m}$ et $G \in \mathbb{F}_{q^m}[X]$ de degré r tel que $G(x_i) \neq 0$, $\forall i \in [|1, n|]$. Un code de Goppa $\Gamma(L, G)$ est un code $Al_k(L, \lambda)$ où $\lambda := (G(x_1)^{-1}, G(x_2)^{-1}, .., G(x_n)^{-1})$.*

Remarque : En général, on prend $L := \mathbb{F}_{q^m} \backslash \{\mathbf{y} \in \mathbb{F}_{q^m} \mid G(\mathbf{y}) = 0\}$.

6. Vladimir D. Goppa est un mathématicien russe renommé pour ses contributions dans ce domaine, notamment pour l'introduction des codes de Goppa.

A.3.3 Codes de Reed-Solomon cycliques et codes BCH

Avant de parler des codes **BCH** [7], on va parler des codes cycliques, une famille de codes très utilisée en cryptographie.

A.3.3.1 Codes cycliques

Définition 26. *Un code $C\ [n,k]_q$ est cyclique lorsque $\forall\ \mathbf{c} := (c_0, c_1.., c_{n-1}) \in C$ le mot $(c_{n-1}, c_0, c_1, .., c_{n-2})$ est aussi un mot du code C.*

Autrement dit, un code cyclique est un code stable par permutation circulaire.

Considérons maintenant l'association suivante :
À chaque mot $\mathbf{c} = (c_0, c_1, .., c_{n-1})$ on associe le polynôme $c_0 + c_1.x + .. + c_{n-1}.x^{n-1} \in \mathbb{F}_q[X]/ < X^n - 1 >$ et réciproquement. On a alors le résultat important suivant :

Theorem 5. *Soit C un code $[n,k]_q$ cyclique, et regardons-le comme un sous-ensemble de $\mathbb{F}_q[X]/ < X^n - 1 >$ par l'association précédente. Alors C est un idéal de l'anneau $\mathbb{F}_q[X]/ < X^n - 1 >$.*

Remarque : Dans la suite, tout code cyclique sera considéré comme sous partie de $\mathbb{F}_q[X]/ < X^n - 1 >$ par l'association. Comme $\mathbb{F}_q[X]/ < X^n - 1 >$ est un anneau principal, on a le corollaire suivant :

Corollaire 1. *Soit C un code $[n,k]_q$ cyclique. Alors, il existe un unique $\mathbf{g} \in \mathbb{F}_q[X]/ < X^n - 1 >$ unitaire, non nul et de degré minimal, tel que :*

1. *$C =< \mathbf{g} >$;*
2. *$\mathbf{g}$ divise $X^n - 1$ dans $\mathbb{F}_q[X]$;*
3. *$deg(\mathbf{g}) = n - k$;*

Preuve 12.
— *Le (1) est conséquence directe du fait que $\mathbb{F}_q[X]/ < X^n - 1 >$ est principal ;*
— *Pour le (2). Soit $\mathbf{c} \in C$ tel que $c_{n-1} = 1$ (Comme $C \neq \{\mathbf{0}\}$, alors quitte à décaler des $\mathbf{0} \neq \mathbf{c} \in C$, on peut faire cette hypothèse). Comme $\mathbf{g}$ divise $\mathbf{c}$ et son décalé $\mathbf{c}'$, et que :*

$$\begin{aligned} x.c - c_{n-1}.(x^n - 1) &= c_0.X + c_1.X^2 + .. + c_{n-1}.X^n - c_{n-1}.X^n + c_{n-1} \\ &= c_{n-1} + c_0.X + .. + c_{n-2}.X^{n-1} \\ &= \mathbf{c}' \end{aligned}$$

Alors $\mathbf{g}$ divise forcément $c_{n-1}.(X^n - 1) = X^n - 1$
— *Par (1), tout élément $\mathbf{c} \in C$ est de la forme $\mathbf{c}(X) = \mathbf{g}(X).\mathbf{h}(X)$ avec $deg(\mathbf{c}) \leq n - 1$. Ainsi, $deg(\mathbf{h}) \leq n - 1 - deg(\mathbf{g})$ et donc il y a $q^{n-deg(\mathbf{g})}$ possibilités, i.e, $Card(C) = q^{n-deg(\mathbf{g})} = q^k$. Autrement dit, $n - deg(\mathbf{g}) = k$.* ■

Corollaire 2. *Soit C un code $[n,k]_q$ cyclique de polynôme générateur $\mathbf{g}(X) = g_0 + g_1.X + .. + g_{n-k}.X^{n-k}$. Alors $C = \{\mathbf{g}(X).\mathbf{a}(X) \mid \mathbf{a} \in \mathbb{F}_q[X],\ deg(\mathbf{a}(X)) \leq k - 1\}$, de dimension k et de matrice génératrice :*

$$G = \begin{bmatrix} g_0 & g_1 & .. & g_{n-k} & 0 & .. & 0 \\ 0 & g_0 & .. & g_{n-k-1} & g_{n-k} & .. & 0 \\ : & : & .. & : & : & .. & : \\ 0 & 0 & .. & g_0 & g_1 & .. & g_{n-k} \end{bmatrix}$$

7. Nommés d'après leurs inventeurs, R. C. Bose, D. K. Ray-Chaudhuri, et Alexis Hocquenghem

Remarque : On a maintenant tout ce qu'il faut pour construire et étudier des codes cycliques ! Pour avoir un code cyclique $[n,k]_q$, il suffit de trouver des polynômes $\mathbf{g} \in \mathbb{F}_q[X]$ qui soient unitaires, irréductibles, divisant $X^n - 1$ et de degré $n-k$. Puis, on représente ce code par la matrice dans le corollaire ci-dessus.

Proposition 12. *Soit C un code $[n,k]_q$ cyclique de polynôme générateur $\mathbf{g}(X) = g_0 + g_1.X + .. + g_{n-k}.X^{n-k}$ et soit $\mathbf{h}(X) = h_0 + h_1.X + .. + h_k.X^k \in \mathbb{F}_q[X]_k$ tel que $X^n - 1 = \mathbf{g}(X).\mathbf{h}(X)$. Alors*

$$H = \begin{bmatrix} h_k & h_{k-1} & .. & h_0 & 0 & .. & 0 \\ 0 & h_k & .. & h_1 & h_0 & .. & 0 \\ : & : & .. & : & : & .. & : \\ 0 & 0 & .. & h_k & h_{k-1} & .. & h_0 \end{bmatrix}$$

est une matrice de parité du code C.

! Le polynôme $\mathbf{h}$ dans la preuve est appelé le polynôme de contrôle du code cyclique C.

Preuve 13. *Observons que $X^n - 1 = \mathbf{g}(X).\mathbf{h}(X)$ pour $\mathbf{h} \in \mathbb{F}_q[X]_k$ car $\mathbf{g}$ divise $X^n - 1$. Ainsi, $\mathbf{g}(X).\mathbf{h}(X) = 0$ dans $\mathbb{F}_q[X]/ < X^n - 1 >$.*
Notons $\mathbf{h}(X) = h_0 + h_1.X + .. + h_{k-1}.x^{k-1} + h_k.x^k$. On a :

$$\begin{aligned} \mathbf{g}(X).\mathbf{h}(X) &= \sum_{p=0}^{n} (\sum_{i=0}^{p} g_i.h_{p-i}).X^p \\ &= 0 \end{aligned}$$

Ce qui veut dire que $\sum_{i=0}^{n-1} g_i.h_{n-1-i} = 0$ en prenant $h_i = 0$ pour $i \leq n-k-2$. ■

Remarque : Ici, on a $h_k = 1$ et $g_{n-k} = 1$.

A.3.3.2 Codes de Reed-Solomon cycliques

Soient α un **élément primitif** de $\mathbb{F}_q$ et n un diviseur de $q-1$. Notons $\beta := \alpha^{\frac{q-1}{n}}$ et soit $\mathbf{x} := (x_1, .., x_n)$ avec $x_i := \beta^{i-1}$.

Theorem 6. *Le code de Reed-Solomon $RS(n,k,\mathbf{x})_q$ sur $\mathbb{F}_q$ est cyclique de polynôme générateur $g(X) := (X-\beta)(X-\beta^2)..(X-\beta^{n-k})$ et de matrice de parité*

$$H = \begin{bmatrix} 1 & \beta & .. & \beta^{n-1} \\ 1 & \beta^2 & .. & \beta^{2.(n-1)} \\ : & : & .. & : \\ 1 & \beta^{n-k} & .. & \beta^{(n-k).(n-1)} \end{bmatrix}$$

.

Preuve 14. — *Le code est bien cyclique. En effet, si $c = (f(1), f(\beta), .., f(\beta^{n-1}))$ est un mot du code, alors son décalé est :*

$$\begin{aligned} c' &= (f(\beta^{n-1}), f(1), .., f(\beta^{n-2})) \\ &= (f(\beta^{-1}.1), f(\beta^{-1}.\beta), .., f(\beta^{-1}.\beta^{n-1})) && \text{Car } \beta^n = 1 \\ &= (h(1), h(\beta), .., h(\beta^{n-1})) && \text{Avec } h(X) := f(\beta^{-1}.X) \end{aligned}$$

— *Une matrice génératrice de ce code est*

$$G = \begin{bmatrix} 1 & 1 & .. & 1 \\ \beta & \beta^2 & .. & \beta^n \\ : & : & .. & : \\ \beta^{k-1} & \beta^{2(k-1)} & .. & \beta^{(k-1).n} \end{bmatrix}$$

Il nous reste à montrer que $G.H^T = 0$ *:*
$(G.H^T)_{i,j} = \sum_{p=0}^{n-1} \beta^{i.p}.\beta^{p.j} = \sum_{p=0}^{n-1} \beta^{(i+j).p} = \sum_{p=0}^{n-1} (\beta^{(i+j)})^p = 0$ *car* $(\beta^{i+j})^n - 1 = 0$ *et* $\beta \neq 1$.

A.3.3.3 Codes BCH et décodage ([14])

Dans la suite, on prend q de la forme : $q = 2^m$.
Notons $RS(n,k,\mathbf{x})_{bin} := \{(c_1,..,c_n) \in RS(n,k,\mathbf{x}) \mid \forall i \in [|1,n|],\ c_i \in \mathbb{F}_2\}$, les mots binaires d'un code de Reed-Solomon.

Theorem 7. *Le code* $RS(n,k,\mathbf{x})_{bin}$ *ainsi défini est un code linéaire* $[n,k,n-k+1]_2$*, cyclique et de polynôme générateur* $g(X) = \prod_{i=1}^{n-k} m_i(X)$ *où* $m_i(X) \in \mathbb{F}_2$ *est le polynôme minimal de* β^i*.*

C'est donc une autre famille de codes MDS **!**

Preuve 15.
— *La linéarité vient du fait que le corps* $\mathbb{F}_q$ *contient le corps* $\mathbb{F}_2$ *et qu'un code de Reed-Solomon est linéaire.*
— *La distance minimale est héritée du code de Reed-Solomon de départ : On a juste enlevé des poids du code de départ, et comme la borne* MDS *est optimale, et ne peut augmenter, la conclusion s'en découle naturellement.*
— β*, ..,* β^{n-k} *sont parmi les racines du polynôme générateur (Car il divise* $X^n - 1$*). Le polynôme de plus petit degré sur* $\mathbb{F}_2$ *vérifiant ça n'est rien d'autre que* $\prod_{i=1}^{n-k} m_i(X)$*.*■

Lemme 1. *Si* y *est racine d'un polynôme sur* $\mathbb{F}_2$*, alors* y^2 *l'est aussi.*

Preuve 16. *Cela vient du fait que* $f(X)^2 = f(X^2)$ *pour tout* $f \in \mathbb{F}_2[X]$*.*

Remarque : Ainsi, β^i et $\beta^{2.i}$ ont le même polynôme minimal sur $\mathbb{F}_2$, et on prend $g(X) := \prod_{\substack{i=1 \\ i\ impaire}}^{n-k} m_i(X)$.
On a la définition suivante :

Définition 27 (Codes BCH binaires)**.** $g(X) := ppcm((m_i(X))_{\substack{i=1:n-k \\ i\ impaire}})$*. Les codes linéaires cycliques engendrés par les polynômes* $g(X)$ *ainsi construits sont appelés codes* BCH *binaires et on les note* $BSH(n,k,\beta)$*.*

Remarque (!) : Il est possible de construire des codes BCH de la même manière en prenant $q = p^m$ où p est un nombre premier impair.
Ces codes ont des propriétés intéressantes :

Proposition 13. *Soit* C *un code* $BCH(n,k,\beta)$ *sur* $q(= 2^m)$*. Alors ce dernier est un code* $[n,s,d_{min}]_2$ *avec :*

1. *Le code est de dimension* $s \geq n - m.\frac{n-k}{2}$ *;*
2. *Si* $n = 2^m - 1$ $(n = q-1)$ *et* $d_{min} = 2.t+1$*, alors* $n - k \leq t.log_2(n)$ *;*

Preuve 17.
— *Pour 1, il suffit de voir que* $deg(g(X)) \leq m.\frac{n-k}{2}$ *dans le cas* $q = 2^m$ *(Remarque ci-dessus) et* $s = n - deg(g(X))$ *(car le code est cyclique).*

— *De* (1), *on a* $n - s \leq m.\frac{n-k}{2}$. ■

On vient d'introduire une famille de codes intéressante, sur des corps petits ($\mathbb{F}_p$ *avec* p *premier, ici* $p = 2$*), contrairement aux codes de Reed-Solomon où la longueur du code définit la taille du corps sur lequel on travaille. Il nous reste à donner la chose la plus importante pour terminer cette section : Un algorithme de décodage efficace pour ces codes.*

Algorithme de Peterson :([6], [14]) : [8]

Soit C un code cyclique $[n,k]_q$ et supposons que $\beta, \beta^2, .., \beta^{2t}$ sont parmi les racines de son polynôme générateur, avec $\beta \in \mathbb{F}_{q^m}$ d'ordre n.
On reçoit le mot $\mathbf{r}(X) = \mathbf{c}(X) + \mathbf{e}(X)$ avec $\mathbf{e}$ une erreur de poids $w_H(\mathbf{e}) \leq t$ et $\mathbf{c}$ le mot envoyé. Notons $\mathbf{e}(X) := e_{k_1}X^{k_1} + .. + e_{k_u}X^{k_u}$. Le problème de décodage revient à retrouver $\mathbf{e}(X)$.
Comme $\mathbf{c}(X) \in C =< \mathbf{g}(X) >$, alors $\mathbf{c}(\beta^i) = 0$ pour $i \in [|1, 2t|]$. En particulier, $\mathbf{r}(\beta^i) = \mathbf{e}(\beta^i)$ pour $i \in [|1, 2t|]$.
On en déduit que le polynôme localisateur d'erreurs, i.e, $Q(X) = \prod_{i=1}^{u}(1 - \beta^{k_i}.X) = 1 + \sum_{i=1}^{u} q_i.X^i$, vérifie :

$$Q(\beta^{-k_i}) = 1 + \beta^{-k_i}.q_1 + .. + \beta^{-u.k_i}.q_u = 0, \ \forall i \in [|1, u|]$$

On cherche d'abord une solution non triviale au système :

$$\begin{bmatrix} r(\beta) & r(\beta^2) & .. & r(\beta^{u+1}) \\ r(\beta^2) & r(\beta^3) & .. & r(\beta^{u+2}) \\ : & : & .. & : \\ r(\beta^u) & r(\beta^{u+1}) & .. & r(\beta^{2.u}) \end{bmatrix} . \begin{bmatrix} 1 \\ q_1 \\ : \\ q_u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \end{bmatrix}$$

Si on note, pour $\mu \leq t$:

$$S_\mu := \begin{bmatrix} r(\beta) & r(\beta^2) & .. & r(\beta^{\mu+1}) \\ r(\beta^2) & r(\beta^3) & .. & r(\beta^{\mu+2}) \\ : & : & .. & : \\ r(\beta^\mu) & r(\beta^{\mu+1}) & .. & r(\beta^{2.\mu}) \end{bmatrix}$$

Alors

$$S_\mu . \begin{bmatrix} 1 \\ q_1 \\ : \\ q_u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \end{bmatrix}$$

admet une solution non triviale si $\mu = u$ et n'admet pas de solutions non triviales dès que $\mu > u$ ([6]).
Ainsi, on doit chercher le premier μ, en partant de $\mu = t$, $\mu = t - 1$,..,$\mu = 1$ tel que

$$S_\mu . \begin{bmatrix} q_u \\ q_{u-1} \\ : \\ q_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \end{bmatrix}$$

admet une solution non triviale. On finit par trouver le polynôme localisateur Q.

L'étape suivante est de trouver, parmi les β^i, avec $i \in [|1, 2.t|]$, les racines de Q et ainsi, on trouve $\beta^{-k_1}, .., \beta^{-k_u}$.
Maintenant comme $\mathbf{r}(\beta^i) = \mathbf{e}(\beta^i)$ pour $i \in [|1, 2t|]$ et $\mathbf{e}(X) := e_{k_1}X^{k_1} + .. + e_{k_u}X^{k_u}$, alors on peut obtenir

8. Wesley W. Peterson est un informaticien et mathématicien américain, connu pour ses contributions à la théorie des codes correcteurs d'erreurs.

les e_{k_i} en résolvant le système suivant :

$$\begin{bmatrix} \beta^{k_1} & \beta^{k_2} & .. & \beta^{k_u} \\ (\beta^{k_1})^2 & (\beta^{k_2})^2 & .. & (\beta^{k_u})^2 \\ : & : & .. & : \\ (\beta^{k_1})^{2t} & (\beta^{k_2})^{2t} & .. & (\beta^{k_u})^{2t} \end{bmatrix} . \begin{bmatrix} e_{k_1} \\ e_{k_u} \\ : \\ e_{k_u} \end{bmatrix} = \begin{bmatrix} r(\beta) \\ r(\beta^2) \\ : \\ r(\beta^{2t}) \end{bmatrix} \tag{A.1}$$

En fin, on vient de trouver $k_1, .., k_u$ ainsi que la valeur des erreurs $e_{k_1}, .., e_{k_u}$: On vient de trouver e.

Remarque : L'étape où on trouve les valeurs des erreurs n'est pas nécessaire dans le cas binaire : $e_i \in \{0, 1\}$.

Complexité : Les étapes les plus coûteux consistent à résoudre, dans un premier temps, un système pour trouver le polynôme Q, en $O(n^3)$ opérations, et puis de trouver l'erreur en résolvant un autre système qui se fait en $O(n^3)$.
On peut améliorer cette démarche en trouvant de manière plus efficace le polynôme localisateur Q, par exemple en utilisant l'algorithme d'Euclide :

Algorithme d'Euclide :([6], [14], [12]) :

Soit, comme ci-dessus, C un code $[n, k]_q$ cyclique et dont le polynôme générateur a parmi ses racines $\beta, .., \beta^{2.t}$ avec β une racine n-ème de l'unité dans $\mathbb{F}_q$.
Pour un code de Reed-Solomon cyclique, on a n qui divise $q - 1$ et $t := \lfloor \frac{n-k}{2} \rfloor$.
Pour un code BCH binaire, on a n qui divise $2^m - 1$ et $d_{min} \geq 2.t + 1$.

D'après ce qu'on vient de voir dans l'algorithme de Peterson, le polynôme localisateur d'erreurs $Q(X) := q_0 + q_1.X + .. + q_t.X^t$ vérifie :

$$\begin{bmatrix} r(\beta) & r(\beta^2) & .. & r(\beta^{t+1}) \\ r(\beta^2) & r(\beta^3) & .. & r(\beta^{t+2}) \\ : & : & .. & : \\ r(\beta^t) & r(\beta^{t+1}) & .. & r(\beta^{2.t}) \end{bmatrix} . \begin{bmatrix} q_0 \\ q_1 \\ : \\ q_u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \end{bmatrix} \tag{A.2}$$

Avec r le mot reçu en forme polynomiale.

Rappel (Algorithme d'Euclide étendu) : Étant donné deux polynômes $a, b \in \mathbb{F}_q[X]$, le théorème d'Euclide étendu calcule $d := pgcd(a, b)$ et deux polynômes f et g réalisant l'identité de Bézout

$$a.f + b.g = d$$

avec $deg(g) < deg(a)$ et $deg(f) < deg(b)$. L'algorithme procède par itérations successives de divisions euclidiennes, comme la suite :

- **Initialisation :** $r_{-1} = a$, $f_{-1} = 1$, $g_{-1} = 0$, $r_0 = b$, $f_0 = 0$ et $g_0 = 1$;
- Pour $i \geq 1$ calcul :
  1. q_i est le quotient de la division euclidienne de r_{i-2} par r_{i-1} ;
  2. r_i est le reste de la division euclidienne de r_{i-2} par r_{i-1} ;
  3. mettre à jour $f_i \leftarrow f_{i-2} - q_i.f_{i-1}$ et $g_i \leftarrow g_{i-2} - q_i.g_{i-1}$;
- Dans chaque étape i de l'algorithme, on a :
  1. $pgcd(r_{i-1}, r_i) = pgcd(r_{i-1}, r_i)$;

2. $a.f_i + b.g_i = r_i$;
3. $deg(g_i) + deg(r_{i-1}) = deg(a)$;

Si on note $S(X) := \sum_{i=1}^{2.t} r(\beta^i).X^{i-1}$, alors d'après les relations (1.1) et (1.2), le polynôme $S(X)$ vérifie :

$$\begin{aligned} S(X) &= \sum_{i=0}^{2.t} r(\beta^i).X^{i-1} \\ &= \sum_{i=1}^{2.t}\sum_{j=1}^{u} \beta^{i.k_j}.e_{k_j}.X^{i-1} \\ &= \sum_{j=1}^{u} \beta^{k_j}.e_{k_j}.\sum_{i=1}^{2.t}(\beta^{k_j}.X)^{i-1} \\ &= \sum_{j=1}^{u} \beta^{k_j}.e_{k_j}.\frac{1}{1-\beta^{k_j}.X} \qquad \text{Faisant } t \to \infty \end{aligned}$$

Comme $Q(X) := \prod_{i=1}^{u}(1 - \beta^{k_i}.X)$, alors

$$\begin{aligned} \tau(X) &:= S(X).Q(X) \\ &= \sum_{j=1}^{u} \beta^{k_j}.e_{k_j} \prod_{\substack{i=1 \\ i\neq j}}^{u}(1-\beta^{k_i}X) \end{aligned}$$

Le degré de τ est donc inférieur à $u-1$, i.e, les termes de τ de X^u jusqu'à X^{2t-1} sont nuls. Autrement dit, $\sum_{i=0}^{u} q_i.r(\beta^{j-i}) = 0$ pour $u+1 \leq j \leq 2t$. Ainsi, on trouve (1.2) si on prend $u+1 \leq j \leq t$, et on a le résultat suivant :

Theorem 8. *Si on utilise l'algorithme d'Euclide étendu en prenant comme polynômes de départ $S(X)$ et X^{2t}, et en s'arrêtant dès que le reste $r_i(X)$ est le premier tel que $deg(r_i(X)) \leq t-1$, alors on obtient le polynôme localisateur inversé $Q'(X) := X^t.Q(X^{-1})$.*

Preuve 18. *Dans ce cas, il existe $f_i(X)$, $g_i(X)$ deux polynômes tels : $deg(f_i) < deg(S(X))$ et $deg(g_i) < deg(X^{2t}) = 2t$ et $g_i(X).S(X) + f_i(X).X^{2t} = r_i(X)$. Ainsi, $deg(g_i) \leq t$ et $S(X).g_i(X) = r_i(X)\ mod(X^{2t})$.* ■

A.3.4 Codes LDPC (Low Density Parity check)

A.3.4.1 Définitions générales ([14])

Définition 28 (Code LDPC)**.** *Un code **LDPC** de paramètres (n,i,j) (ou code (n,i,j)-LDPC) est un code de longueur n sur $\mathbb{F}_2$ représenté par une matrice de parité redondante* H dont chaque ligne est de poids de Hamming i et chaque colonne est de poids de Hamming j et tel que $Card(Supp(\mathbf{C_i^H}) \bigcap Supp(\mathbf{C_j^H})) \leq 1$, $\forall i \neq j$, avec $\mathbf{C_i^H}$ la i-ème colonne de H.*

* : Les lignes de H ne doivent pas forcément être $\mathbb{F}_2$-linéairement indépendantes.
Une propriété intéressante de ces codes est la suivante :

Lemme 2. *Le rang r d'un code LDPC de paramètres (n,i,j) vérifie : $r \geq 1 - \frac{i}{j}$*

Preuve 19. *Notons m le nombre de lignes de H. On a alors $m.i = n.j$.*
La matrice H est redondante, i.e, la dimension k du code vérifie $k \geq n-m$, i.e, $r \geq 1 - \frac{m}{n} = 1 - \frac{j}{i}$. ■

Lemme 3 ([14])**.** *Un code LDPC de paramètres (n,i,j) a une distance minimale d vérifiant : $d \geq i+1$.*

Remarque : Ainsi, un code (n, i, j)-LDPC est un code au moins $\lfloor \frac{i}{2} \rfloor$-correcteur.

A.3.4.2 Décodage itératif des codes LDPC : Bit flipping ([11])

L'algorithme le plus utile pour décoder les codes LDPC est appelé **Bit flipping**, qu'on va décrire dans la suite :
Soit H une matrice redondante d'un code (n, i, j)-LDPC

— **Entrée :** Le mot reçu $\mathbf{r}$;

1. On met $\mathbf{y} \leftarrow \mathbf{r}$;
2. Calculer $\mathbf{s}^T := H.\mathbf{y}^T$;
3. Trouver un $j \in [|1, n|]$ tel que $Card(Supp(\mathbf{C_j^H}) \bigcap Supp(\mathbf{s})) > \frac{i}{2}$ et faire $y_j \leftarrow y_j \oplus 1$;
4. Répète les étapes depuis 2 jusqu'à ne plus trouver un j vérifiant 3 ;

— **Sortie :**

1. $(\mathbf{y}, 1)$ si $s = 0$;
2. $(\mathbf{y}, 0)$ sinon ;

Pour chaque mot $\mathbf{r}$ reçu, l'algorithme termine :

Lemme 4. *Le poids de Hamming du syndrome* $\mathbf{s}$ *diminue à chaque itération des étapes* 2 *et* 3.

Preuve 20. *À chaque répétition, on compare le nombre des positions non nulles de* $\mathbf{s}$ *avec les positions non nulles des colonnes de* H*, et si ce nombre dépasse un seuil, on diminue le poids de* $\mathbf{s}$ *en flippant un bit de* $\mathbf{y}$*.*

On a ainsi le théorème suivant :

Theorem 9. *L'algorithme finira par s'arrêter, et le* $\mathbf{y}$ *retourné est soit un mot du code, et l'algorithme renvoie* $(\mathbf{y}, 1)$*, soit un* $\mathbf{y}$ *proche d'un mot du code, et l'algorithme renvoie* $(\mathbf{y}, 0)$*.*

Annexe B

Notions de cryptographie à base de codes correcteurs

La cryptographie basée sur les codes correcteurs a émergé avec les travaux de Robert McEliece [1] en 1978 ([17]), où il a exploité l'absence d'un algorithme qui s'exécute en temps polynomial pour résoudre le problème de décodage (A.2). Dans cette partie, nous introduirons quelques notions de cryptographie, discuterons du chiffrement de McEliece et aborderons le problème des signatures basées sur les codes correcteurs. On terminera cette partie par donner un état de l'art sur les signatures basées sur les codes.

B.1 La cryptographie

D'abord, qu'est-ce que la cryptographie ?

Définition 29 (Cryptographie)**.** *La cryptographie est l'ensemble des techniques mises en ouvre pour garantir la sécurité de l'information sur un canal de communication.*

On peut illustrer cela par un schéma, où une personne, nommée (Alice) veut envoyer un message à une personne nommée (Bob), en passant par un canal de communication surveillé par un espion (Ève) :

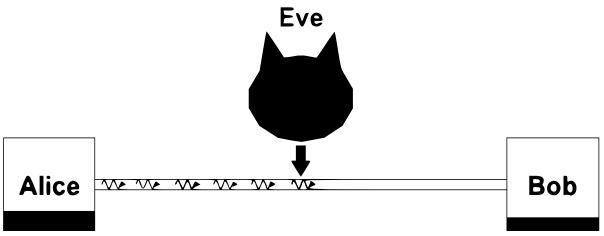


FIGURE B.1 – Schéma de communication entre Alice et Bob

Le but de la cryptographie est de protéger l'information passée par le canal de communication des personnes malveillantes comme Ève. L'outil utilisé pour garantir cela sont les algorithmes de chiffrements :

B.2 Schéma de chiffrement à clé publique et notion de sécurité

Définition 30 (Schéma de chiffrement à clé publique [9])**.** *Un schéma de chiffrement à clé publique (ou symétrique),* ***PKE****, est l'ensemble de trois algorithmes* $PKE := (KeyGen, Enc, Dec)$ *tels que :*

1. Robert J. McEliece était un mathématicien et informaticien américain.

— *$KeyGen(\lambda)$ est un algorithme, qui prend en entrée un paramètre de sécurité λ, et sort deux clés (sk, pk) :*
  1. *Clé privée sk ;*
  2. *Clé publique pk ;*
— *$Enc(pk, \mathbf{m})$ est un algorithme qui prend en entrée une clé publique pk et un message m et qui sort un chiffré message chiffré $\mathbf{c}$.*
— *$Dec(sk, \mathbf{c})$ est un algorithme qui prend en entrée une clé secrète sk et un message chiffré $\mathbf{c}$ et qui sort un message*

Définition 31 (Complétude)**.** *Un schéma de chiffrement symétrique $PKE = (KeyGen, Enc, Dec)$ est complet si, pour tout paramètre de sécurité λ, et pour tout message m, la probabilité :*

$$\mathbb{P}(Dec(sk, Enc(pk, m)) \neq m \mid (sk, pk) \leftarrow KeyGen(\lambda)) \tag{B.1}$$

est négligeable[2] *en la taille de λ.*

Les schémas de chiffrement doivent satisfaire certaines notions de sécurité. Celles-ci déterminent le degré de résistance de ces schémas aux attaques. Il existe plusieurs notions de sécurité ([9]), parmi lesquelles on trouve les trois classiques :

Définition 32 (Indistinguabilité sous des attaques à textes en clair choisis (IND-CPA))**.** *Soit $\mathcal{A}$ un adversaire (Algorithme polynomial*[3]*) et $\mathcal{V}$ un vérifieur. On définit d'abord un jeu (une expérience) qu'on nomme **IND-CPA** :*

1. *$\mathcal{V}$ génère $(sk, pk) \leftarrow KeyGen(\lambda)$ et donne pk à $\mathcal{A}$;*
2. *$\mathcal{A}$ génère deux messages m_0 et m_1 et va les donner à $\mathcal{V}$;*
3. *$\mathcal{V}$ va calculer $c_0 \leftarrow Enc(pk, m_0)$ et $c_1 \leftarrow Enc(pk, m_1)$, puis il génère $b \leftarrow U(\{0, 1\})$ et envoie m_b à $\mathcal{A}$;*
4. *$\mathcal{A}$ renvoie $b' \in \{0, 1\}$. Il gagne si et seulement si $b' = b$.*

Une version forte de cette notion existe :

Définition 33 (Indistinguabilité sous des attaques à textes en chiffré choisis (IND-CCA1))**.** *Soit $\mathcal{A}$ un adversaire (Algorithme polynomial) et $\mathcal{V}$ un vérifieur. On définit d'abord un jeu (une expérience) qu'on nomme **IND-CCA1** :*

1. *$\mathcal{V}$ génère $(sk, pk) \leftarrow KeyGen(\lambda)$ et donne pk à $\mathcal{A}$;*
2. *Uniquement dans cette étape, $\mathcal{A}$ possède un oracle de déchiffrement $\mathcal{O}_{sk}$ qui déchiffre tout message chiffré avec pk. $\mathcal{A}$ génère deux messages m_0 et m_1 et va les donner à $\mathcal{V}$;*
3. *$\mathcal{V}$ va calculer $c_0 \leftarrow Enc(pk, m_0)$ et $c_1 \leftarrow Enc(pk, m_1)$, puis il génère $b \leftarrow U(\{0, 1\})$ et envoie m_b à $\mathcal{A}$;*
4. *$\mathcal{A}$ renvoie $b' \in \{0, 1\}$. Il gagne si et seulement si $b' = b$.*

Une notion plus forte existe encore :

Définition 34 (Indistinguabilité sous des attaques à textes en chiffré choisis adaptatifs (IND-CCA2))**.** *Soit $\mathcal{A}$ un adversaire (Algorithme polynomial) et $\mathcal{V}$ un vérifieur. On définit d'abord un jeu (une expérience) qu'on nomme **IND-CCA2** :*

2. voir 2.1
3. Un algorithme qui s'exécute en temps polynomial

1. *$\mathcal{V}$ génère $(sk, pk) \leftarrow KeyGen(\lambda)$ et donne pk à $\mathcal{A}$;*
2. *Dans cette étape, $\mathcal{A}$ possède un oracle de déchiffrement $\mathcal{O}_{sk}$ qui déchiffre tout message chiffré avec pk. $\mathcal{A}$ génère deux messages m_0 et m_1 et va les donner à $\mathcal{V}$;*
3. *$\mathcal{V}$ va calculer $c_0 \leftarrow Enc(pk, m_0)$ et $c_1 \leftarrow Enc(pk, m_1)$, puis il génère $b \leftarrow U(\{0,1\})$ et envoie m_b à $\mathcal{A}$;*
4. *Ici, $\mathcal{A}$ possède l'oracle $\mathcal{O}_{sk}$, mais ne peut déchiffrer m_b avec. Après analyse, $\mathcal{A}$ renvoie $b' \in \{0,1\}$. Il gagne si et seulement si $b' = b$.*

Définition 35. *Un schéma de chiffrement symétrique $PKE = (KeyGen, Enc, Dec)$ est sûr pour la notion de sécurité S (S peut être IND-CPA, IND-CCA1 ou IND-CCA2) si, pour tout adversaire polynomial $\mathcal{A}$, la quantité :*

$$\mathbb{P}(\mathcal{A} \text{ gagne dans le jeu } S) - \frac{1}{2}$$

est négligeable en la taille de λ.
Autrement dit, la meilleure chose que peut $\mathcal{A}$ faire est de deviner.

Ces trois notions de sécurité impliquent toutes une chose : L'algorithme de chiffrement *Enc* **doit être randomisé.**

B.3 Chiffrement à base de codes correcteurs

Le schéma de chiffrement de McEliece fut le premier schéma de chiffrement basé sur des codes correcteurs, et l'un des premiers candidats pour les schémas de chiffrement à clé publique. Dans son article, McEliece se base sur la difficulté de décodage des codes aléatoires (A.2) pour faire son schéma :

Définition 36 (Schéma de chiffrement de McEliece [17])**.** *Le schéma est le suivant :*

— ***KeyGen :***

1. *On choisit une matrice $G \in \mathbb{F}_q^{k \times n}$ génératrice d'un code t-correcteur qu'on sait décoder ;*
2. *On choisit deux matrices : $P \in \mathbb{F}_q^{n \times n}$ une matrice de permutation, $S \in \mathbb{F}_q^{k \times k}$ inversible ;*
3. *La clé publique est $G' := S.G.P$: Le but étant de masquer la matrice G, en la faisant paraître comme aléatoire.*
4. *La clé secrète est le tripler G, P et S ;*

— ***Enc :*** *Pour chiffrer un message $\mathbf{m} \in \mathbb{F}_q^k$, on procède comme la suite :*

1. *$\mathbf{e} \leftarrow U(\mathbb{F}_q^n)$ avec $w_H(\mathbf{e}) \leq t$, et on fait $\mathbf{c} := \mathbf{m}.G' + \mathbf{e}$;*
2. *$\mathbf{c} \leftarrow Enc(G', \mathbf{m})$ est le chiffré de $\mathbf{m}$;*

— ***Dec :*** *Pour déchiffrer c on procède comme la suite :*

1. *Calculer $\mathbf{z} := \mathbf{c}.P^{-1}$;*
2. *On décode le mot $\mathbf{z}$ et on obtient $\mathbf{z}'$;*
3. *$\mathbf{m} := \mathbf{z}'.S^{-1} \leftarrow Dec((G, P, S), \mathbf{c})$ est le message déchiffré de $\mathbf{c}$;*

Il utilise ainsi un code décodable efficacement provenant d'une famille de codes qui semble arbitraire ou aléatoire. Dans son article, il fait usage des codes de Goppa, qui sont (A) assez nombreux et donc très utiles en cryptographie.

Proposition 14. *Le schéma de McEliece est complet.*

Preuve 21. *Si* $\mathbf{c} = \mathbf{m}.G' + \mathbf{e} = \mathbf{m}.S.G.P + \mathbf{e}$ *avec* $w_H(\mathbf{e}) \leq t$, *alors* $\mathbf{c}.P^{-1} = \mathbf{m}.S.G + \mathbf{e}.P^{-1}$. *Comme* P *est une permutation, alors* P^{-1} *l'est aussi et donc* $w_H(\mathbf{e}) = w_H(\mathbf{e}.P^{-1}) \leq t$.
Ainsi, si on décode $\mathbf{c}.P^{-1}$ *on trouve bien* $\mathbf{m}.S$. *Quitte à multiplier à gauche par* S^{-1}, *on récupère le message* $\mathbf{m}$.■

On peut résumer cela dans le schéma suivant :

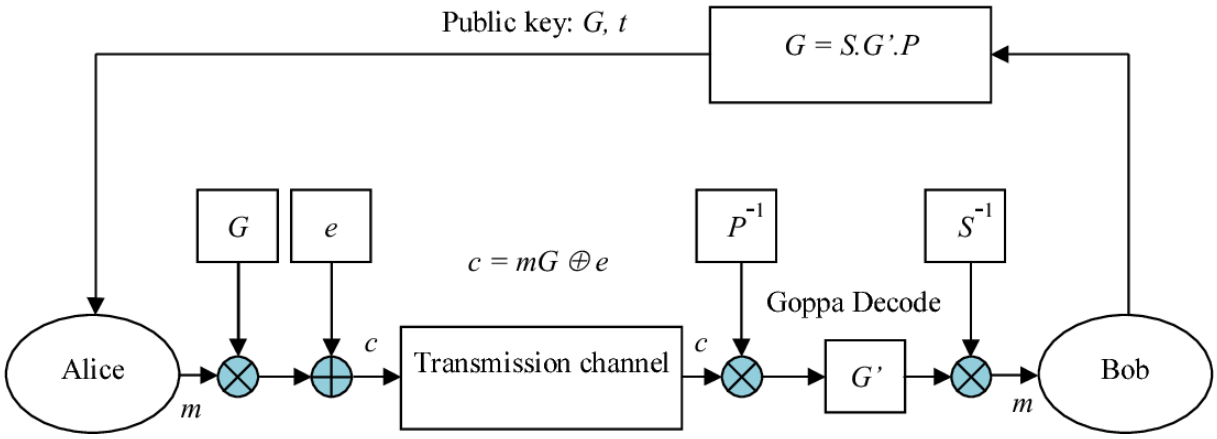


FIGURE B.2 – Cryptosystème de McEliece [10]

B.4 Schéma de signature et notion de sécurité

Définition 37 (Schéma de signature)**.** *Un schéma de signature est l'ensemble de trois algorithmes* $(KeyGen, Sign, Verify)$ *tel que :*

- $KeyGen(\lambda)$ *: est un algorithme qui prend en entrée un paramètre de sécurité* λ *et qui sort deux clés, une clé de signature* sk *et une clé de vérification* vk *;*
- $Sign(sk, m)$ *: est un algorithme qui prend en entrée un message* m *et le secret* sk *et qui sort une signature* σ *de* m *;*
- $Verify(pk, m, \sigma)$ *: est un algorithme qui prend en entrée une signature* σ, *le message signé* m *et la clé de vérification* vk *et qui sort* $v \in \{0, 1\}$.

On remarque que, à l'inverse des schémas de chiffrement symétrique, l'utilisateur signe un message avec sa clé de signature (secrète), et la signature peut être vérifiée par tout le monde avec la clé de vérification (publique) vk.

Définition 38 (Complétude des schémas de signature)**.** *Un schéma de signature* $(KeyGen, Sign, Verify)$ *est dite complet si, pour tout paramètre de sécurité* λ, *et pour tout message* m, *la probabilité :*

$$\mathbb{P}(Verify(vk, Sign(sk, m)) \neq 1 \mid (sk, vk) \leftarrow KeyGen(\lambda)) \tag{B.2}$$

est négligeable.

Comme les schémas de chiffrements symétriques, les schémas de signatures doivent aussi vérifier certaines notions de sécurité, on trouve classiquement les deux suivantes :

Dans la suite, $\mathcal{A}$ est un algorithme polynomial, qu'on nomme adversaire. $\mathcal{V}$ sera le vérifieur.

Définition 39 (Existential unforgeability under one- time chosen message attacks (EU-CMA))**.** *On commence par définir l'expérience suivante* ***EUF-CMA*** *:*

- $\mathcal{V}$ *génère* $(sk, vk) \leftarrow KeyGen(\lambda)$ *et donne* vk *à* $\mathcal{A}$ *;*

— $\mathcal{A}$ *génère* n *messages* $m_1, .., m_n$ *et les envoie à* $\mathcal{V}$ *;*
— $\mathcal{V}$ *calcul* $\sigma_1 = Sign(sk, m_1), .., \sigma_n = Sign(sk, m_n)$ *et envoie la séquence* $(m_1, \sigma_1), .., (m_n, \sigma_n)$ *à* $\mathcal{A}$ *;*
— $\mathcal{A}$ *envoie un couple* (M, Σ) *à* $\mathcal{V}$ *;*

$\mathcal{A}$ *gagne si et seulement si :* $\begin{cases} Verify(vk, M, \Sigma) = 1 \\ Et \\ \forall i \in [|1, n|],\ m_i \neq M \end{cases}$

C'est-à-dire, l'adversaire $\mathcal{A}$ *arrive à signer un autre message en analysant une série de signatures qu'il possède en avance.*

Il existe une version forte de cette experience :

Définition 40 (Strong Existential unforgeability under one- time chosen message attacks (SEU-CMA))**.** *On commence par définir l'expérience suivante* ***SEU-CMA*** *:*
— $\mathcal{V}$ *génère* $(sk, vk) \leftarrow KeyGen(\lambda)$ *et donne* vk *à* $\mathcal{A}$ *;*
— $\mathcal{A}$ *génère* n *messages* $m_1, .., m_n$ *et les envoie à* $\mathcal{V}$ *;*
— $\mathcal{V}$ *calcul* $\sigma_1 = Sign(sk, m_1), .., \sigma_n = Sign(sk, m_n)$ *et envoie la séquence* $(m_1, \sigma_1), .., (m_n, \sigma_n)$ *à* $\mathcal{A}$ *;*
— $\mathcal{A}$ *envoie un couple* (M, Σ) *à* $\mathcal{V}$ *;*

$\mathcal{A}$ *gagne si et seulement si :* $\begin{cases} Verify(vk, M, \Sigma) = 1 \\ Et \\ \forall i \in [|1, n|],\ (m_i, \sigma_i) \neq (M, \Sigma) \end{cases}$

C'est-à-dire que l'adversaire $\mathcal{A}$ *peut proposer une autre signature pour un message déjà signé.*

Définition 41. *Un schéma de signature* $(KeyGen, Sign, Verify)$ *est sûr pour la sécurité* ***EU-CMA*** *(resp.* ***SEU-CMA****) si pour tout adversaire polynomial* $\mathcal{A}$*, la probabilité*

$$\mathbb{P}(\mathcal{A} \text{ gagne dans } l'experience\ EU - CMA) \ (resp.\ \mathbb{P}(\mathcal{A} \text{ gagne dans } l'experience\ S - CMA))$$

est négligeable en la taille de λ*.*

B.4.1 État de l'art des signatures basées sur les codes

Aujourd'hui, il existe de nombreux schémas de chiffrement basés sur les codes correcteurs, par exemple le schéma de **McEliece**. En revanche, les signatures basées sur les codes sont rares. À ce jour, aucune signature basée sur les codes correcteurs n'est standardisée par le **NIST**. Il existe quelques schémas de signatures intéressants basés sur les codes, mais ceux-ci ne sont, jusqu'à présent, pas suffisamment compétitifs par rapport aux autres schémas de signatures post-quantiques, tels que ceux basés sur les réseaux (lattices).
On cite par exemple :
— **Wave** [4], qui est un schéma de signature post-quantique, reposant sur des problèmes difficiles de la théorie des codes correcteurs,
— Le schéma de chiffrement **BIKE** [5], qui peut être adapté en un schéma de signature,
Pour le moment, le seul schéma sérieux est **Wave**.

4. https ://wave-sign.org/wave_documentation.pdf
5. https ://bikesuite.org/

Annexe C

Implantation des algorithmes de décodage en Magma

Durant ce stage, j'ai implémenté plusieurs algorithmes de décodage, pour mieux les comprendre en pratique. Les implémentations reposent sur les fondations théoriques établies en Annexe A.

C.1 Algorithme de décodage et de décodage en liste des codes de Reed-Solomon ((A.3.1.3),(A.3.1.2))

D'abord, il nous faudrait construire les matrices Q^i ,$i \in [|0, \ell|]$ qui apparaissent dans $\left[Q^0|Q^1|..|Q^\ell\right]$:

```
/////////////// Algorithme pour construire les matrices Q^i ///////////////////////
Q_Matrix := function(n,k,p,tau,q,r,x)
        Q := KMatrixSpace(GF(q),n,n-tau-p*(k-1)) ! 0;

        for i in {1..n} do
                for j in {1..n-tau-p*(k-1)} do
                        Q[i][j] := (x[i]^j)*r[i]^p;
                end for;
        end for;
        return Q; // La matrice Q^p
end function;
////////////////////////////////////////////////////////////////////////////////
```

Puis, il nous reste à construire la matrice $Q^* := \left[Q^0|Q^1|..|Q^\ell\right]$ (voir A.3.1.3) :

```
//////////////////////////// Trouver le polynome Q^*
    ////////////////////////////////////////
Q_polynome := function(n,k,p,tau,q,r,x)
        Q := Q_Matrix(n,k,0,tau,q,r,x);
        for i in {1..p} do
                Q := HorizontalJoin(Q,Q_Matrix(n,k,i,tau,q,r,x)) ;
        end for;
        N := NullspaceMatrix(Transpose(Q));
        return N[1]; // Retourne une solution non nulle X au syst me Q^*.X=0

end function;
////////////////////////////////////////////////////////////////////////////////////
```

Le décodage se fait naturellement avec l'algorithme suivant :

```
//////////////////////////////// Sudan ////////////////////////////////////////////////
RS_list_decoding := function(n,k,p,tau,q,r,x)
        List := [];
        Q := Q_polynome(n,k,p,tau,q,r,x);
        Q := [Q[i] : i in {1..Ncols(Q)}];


        P<X> := PolynomialRing(GF(q));
        D<Y> := PolynomialRing(P);

        Qi := [P ! Q[1..n-tau]] cat [ P ! Q[(l+1)*(n-tau) -(k-1)*l*(l+1)/2 +1 .. (l+2)*(n
            -tau) -(k-1)*(l+1)*(l+2)/2] : l in {0..p-1}]; // La matrice Q^*

        Poly := &+[ (D ! Qi[i+1])*Y^i : i in {0..p}];

        fact := Factorisation(D ! Poly);
        for i in {1..#fact} do
                if Degree(fact[i][1]) eq 1 then
                        Append(~List,(D ! Y)- (D !fact[i][1]));
                end if;
        end for;

        return [VectorSpace(GF(q),n) ! [Evaluate(Evaluate(List[i],1),x[j]) : j in {1..n}]
             : i in {1..#List} ];
end function;
//////////////////////////////////////////////////////////////////////////////////////
```

Le décodage des codes de Reed-Solomon (voir A.3.1.2) se fait par l'algorithme suivant, en utilisant les algorithmes implémentés pour le décodage en liste :

```
//////////////////////////////// Welch_Berlekamp //////////////////////////////////////
RS_decoding := function(n,k,x,q,r)
        tau := (n-k) div 2;
        Q := Q_polynome(n,k,1,tau,q,r,x);
        Q1 := [Q[i] : i in {1..n-tau}];
        Q2 := [Q[i] : i in {n-tau+1..Ncols(Q)}];
        poly1 := CreatePolynomial(Q1, GF(q));
        poly2 := CreatePolynomial(Q2, GF(q));
        poly := - poly1 div poly2;
        return VectorSpace(GF(q),n) ! [Evaluate(poly, x[i]) : i in {1..n}];
end function;
//////////////////////////////////////////////////////////////////////////////////////
```

C.2 Algorithme de décodage des codes BCH/Reed-Solomon cycliques (A.3.3.3)

C.2.1 Décodage avec l'algorithme de Peterson

L'algorithme suivant retourne le polynôme localisateur d'erreurs défini ici A.3.3.3 en utilisant l'algorithme de Peterson :

```
////////////// BCH decodage Peterson cas binaire ///////////////
Polynome_localisateur := function(n,p,m,t,beta,r)
        P<x> := PolynomialRing(GF(p));
        R := P ! Eltseq(r);

        B := [beta^i : i in {1..2*t}];

        u  := t ;
        Su := KMatrixSpace(GF(p^m),u,u) ! [ [ Evaluate(R,beta^(i+j)) : j in {0..u-1} ] :
            i in {1..u}] ;

        while Rank(Su) ne u do
                u -:= 1;
                Su := KMatrixSpace(GF(p^m),u,u) ! [[Evaluate(R,beta^(i+j)) : j in {0..u
                    -1}] : i in {1..u}];
        end while;

        S := VectorSpace(GF(p^m),u) ! [-Evaluate(R,beta^(u+i)) : i in {1..u} ] ;

        sig := Solution(Su,S);

        return ([1] cat [sig[u-i+1] : i in {1..u}]);
end function;
////////////////////////////////////////////////////////////////
```

Puis on utilise l'algorithme suivant pour décoder un mot bruité r jusqu'à t erreurs :

```
/////// Decodage des BCH  ///////////////////
BSH_error := function(n,p,m,t,beta,r)
        P<x> := PolynomialRing(GF(p^m));

        Q := P ! Polynome_localisateur(n,p,m,t,beta,r);
        u := Degree(Q);
        R := P ! Eltseq(r);

        Racines := [];
        order := [] ;

        for i in {1..p^m-2} do
                if Evaluate(Q,beta^i) eq 0 then
                        Append(~Racines, beta^(p^m -1 -i));
                        Append(~order,p^m -1 -i);
                end if;
        end for;
        c := r;

        for i in order do
                c[i+1] +:= 1;
        end for;

        return c; // r = c + e avec w_H(e) <= t
end function;
////////////////////////////////////////////////////////////////
```

C.2.2 Décodage euclidien

L'algorithme suivant est une variante de l'algorithme d'Euclide étendu pour les polynômes :

```
//////////////////// decodage Euclidien des BCH (cas binaire) ///////////////
Euclidian := function(a,b,top,p,m)
        P<x> := PolynomialRing(GF(p^m));

        r_1 := P ! a ;
        f_1 := P ! 1 ;
        g_1 := P ! 0 ;

        r0 := P ! b ;
        f0 := P ! 0 ;
        g0 := P ! 1 ;

        while Degree(r_1) ge top do

                q_i := r_1 div r0;
                r_i := r_1 mod r0;

                t_f := f0;
                t_g := g0;

                f0 := f_1 - q_i*f0 ;
                g0 := g_1 - q_i*g0 ;

                f_1 := t_f ;
                g_1 := t_g ;

                r_1 := r0 ;
                r0  := r_i;
        end while;

        return g_1; // g_1 tel que a.g_1 + b.f_1=r_1
end function;
////////////////////////////////////////////////////////////////////////////
```

Puis comme dans le théorème A.3.3.3, on trouve le polynôme localisateur pour un mot de code corrompu avec t erreurs r :

```
//////////////////////////// ///////////////////////////////////////
euclid := function(n,p,m,t,r,beta)
        P<x> := PolynomialRing(GF(p^m));
        R := P ! Eltseq(r);

        S := &+[Evaluate(R,beta^i)*x^(i) : i in {0..2*t-1}];
        a := x^(2*t);
        sig := Euclidian(a,S,t,p,m);
        return sig/Eltseq(sig)[1];
end function;
////////////////////////////////////////////////////////////////////////
```

Après on peut utiliser un algorithme comme **BSH_error** ci-dessus pour décoder un mot.

C.3 Algorithme de décodage itératif : Bit flipping (A.3.4.2)

Comme j'ai beaucoup utilisé les codes LDPC durant ce stage, il y a plusieurs versions possibles de l'algorithme Bit flipping. Chaque version avec sa propre spécificité (pour en connaître plus, je vous renvoie vers la thèse de Julia Chaulet ici [5]).

C.3.1 Version classique : On choisit le seuil

La version classique n'est rien d'autre que celle qu'on voit ici A.3.4.2 :

```
////////////////////////// Decodage a un seuil choisi ////////////////////////////////
decode_LDPC := function(K,n,k,s,threshold,iter)
        N          := iter;
        S          := { i : i in {1..k} | s[i] ne 0 };
        e          := VectorSpace(GF(2),n) ! 0;
        _N_        := N;
        while S ne { } and _N_ gt 0 do
            for j :=  1 to n do
                if S eq {} then
                        break;
                end if;
                I  := Set(K[j])  meet S;
                if #I gt threshold/2 then
                    e[j] := e[j] + 1;
                    for i in K[j] do
                        s[i] := s[i] + 1;
                    end for;
                    S := { i : i in {1..k} | s[i] ne 0 };
                    continue;
                end if;
            end for;
            _N_  := _N_ - 1;
        end while;

        return e;
end function;
//////////////////////////////////////////////////////////////////////////
```

L'algorithme prend en entrée :

- K : Représentation par ligne de la matrice de parité avec laquelle on souhaite décoder ;
- n le nombre de colonnes de K ;
- k : Le nombre de lignes de la matrice D ;
- $\mathbf{s}$: Le syndrome dans le problème de décodage par syndrome ;
- *threshold* : Le seuil à partir duquel on flippe les bits ;
- *iter* : Le nombre d'itérations à effectuer ;

En sortie, on aura un mot $\mathbf{e}$ qui :

1. Soit $D.\mathbf{e}^T = \mathbf{s}^T$, ou ;
2. $D.\mathbf{e}^T$ est assez proche de $\mathbf{s}^T$ au sens de Hamming ;

C.3.2 Version améliorée : Décodage à seuil maximal

La version suivante nécessite de faire plus d'itérations, i.e, avec un *iter* assez grand pour espérer décoder.

```
////////////////////////////////// Decodage a seuil maximal ////////////////////////////
bitflip_max := function(D,s,k,iter)
        N := iter ;
        S := { i : i in {1..k} | s[i] ne 0 } ;
        e := VectorSpace(GF(2),#D) ! 0;
        _N_ := 0 ;
        while  S ne {} and _N_ lt N do
                b   := 0 ;
                ind := 1 ;

                for j in {1..#D} do
                        if #(S meet Set(D[j])) gt b then
                                b   := #(S meet Set(D[j]));
                                ind := j;
                        end if;
                end for;

                e[ind] +:= 1 ;
                for i in D[ind] do
                        s[i] +:= 1 ;
                end for;
                S := { i : i in {1..k} | s[i] ne 0 } ;
                _N_ +:=1;
        end while;
        return e;
end function;
```

L'algorithme prend en entrée :

- D : Représentation par ligne de la matrice de parité avec laquelle on souhaite décoder ;
- $\mathbf{s}$: Le syndrome dans le problème de décodage par syndrome ;
- k : Le nombre de lignes de la matrice D ;
- *iter* : Le nombre d'itérations à effectuer ;

En sortie, on aura un mot $\mathbf{e}$ qui :

1. Soit $D.\mathbf{e}^T = \mathbf{s}^T$;
2. Soit $D.\mathbf{e}^T$ est assez proche de $\mathbf{s}^T$ au sens de Hamming ;

C.3.3 Version BIKE : Seuil adapté [1]

C'est la version utilisée dans le schéma de chiffrement **BIKE** à base de codes correcteurs, soumise au **NIST** en 2022.

```
////////////////////////// Decodage a seuil adaptatif ////////////////////////////////////
decode_LDPC := function(K,n,k,s,iter)
        N          := iter;
        S          := { i : i in {1..k} | s[i] ne 0 };
        e          := VectorSpace(GF(2),n) ! 0;
        threshold  := [ #K[j]/2 : j in {1..#K} ];
        _N_        := N;
        while S ne { } and _N_ gt 0 do
            for j :=  1 to n do
                if S eq {} then
                        break;
                end if;
                I  := Set(K[j])  meet S;
                if #I gt threshold[j] then
                    e[j] := e[j] + 1;
                    for i in K[j] do
                        s[i] := s[i] + 1;
                    end for;
                    S := { i : i in {1..k} | s[i] ne 0 };
                    continue;
                end if;
            end for;
            _N_  := _N_ - 1;
        end while;

        return e;
end function;
////////////////////////////////////////////////////////////////////////////////////////
```

C.4 Algorithme de décodage complet

L'algorithme suivant est celui qui résout $A_i.\mathbf{E_i}^T = \mathbf{S_i}^T$ ici 2.4.3

```
////////////// Decodage par recherche exhausstive ////////////////////////////////////
generic_decoding := function(A,s)
        k := Nrows(A); // lignes de A
        n := Ncols(A); // colonnes de A
        J := [] ; // Contient les indices des colonnes de A contenant s
        if s ne 0 then
                for j in {1..n} do
                        if (VectorSpace(GF(2),k) ! [A[i][j] : i in {1..k}]) eq (
                            VectorSpace(GF(2),k) ! s) then
                                J := [j] ;
                        end if;
                end for;

                if #J eq 0 then // Si aucune collone de A ne contient s, on cherche les
                    sommes de deux colonnes de A qui donne s
                        for j1 in {1..n} do
                                for j2 in {1..n} do
                                        if (VectorSpace(GF(2),k) ! [A[i][j1] + A[i][j2] :
                                             i in {1..k}]) eq s then
                                            J := [j1,j2] ;
                                        end if;
                                end for;
                        end for;
                end if;
        end if;

        return J;
end function;
////////////////////////////////////////////////////////////////////////////////////
```

Puis pour décoder un code avec une matrice de parité de la forme :

$$H := \begin{bmatrix} A_1 & 0 & 0 & .. & 0 \\ 0 & A_2 & 0 & .. & 0 \\ 0 & 0 & A_3 & .. & 0 \\ : & : & : & .. & : \\ 0 & 0 & 0 & .. & A_p \end{bmatrix}$$

on utilise l'algorithme ci-dessus pour décoder bloc par bloc (voir 2.4.3) :

```
////////////////////////////// Decoding Full Decoding Codes ////////////////////////////
Decodagecomplet := function(A,s,L0,r0,d)
        e := VectorSpace(GF(2),L0*d) ! 0 ;

        for i in {1..d} do
                assert Nrows(A[i]) eq r0;
                J := generic_decoding(A[i],s[i]) ;
                for j in J do
                        e[j + (i-1)*L0] := 1;
                end for;
        end for;

        return e;
end function;
////////////////////////////////////////////////////////////////////////////////////
```

L'algorithme prend en entrée

- Une liste $A = [A_1, A_2, .., A_d]$ de d matrices, correspondantes aux blocs diagonales de H ;
- Un syndrome sous forme de liste de d petits syndromes $s = [\mathbf{s_1}, \mathbf{s_2}, .., \mathbf{s_d}]$;
- $L0$ le nombre de colonnes des A_i ;
- r_0 le de nombre de lignes des A_i et la taille des $\mathbf{s_i}$;
- d le nombre de blocs diagonaux de H ;

Remarque : On peut rajouter à l'algorithme une contrainte sur le poids de la sortie e pour avoir $\sum_{i=1}^{d} w_H(E_i) \leq t$.

Pour créer la liste de matrices $A = [A_1, A_2, .., A_d]$ où $A_i := (I_{r_0}|A_i^*)$ on utilise l'algorithme suivant :

```
1  /////////// Construction d'une matrice par bloc ////////////////////////////////
2  full_decoding_PMatrix := function(L,r,m)
3          return [HorizontalJoin(IdentityMatrix(GF(2), r),Random(KMatrixSpace(GF(2),r,L-r))
               ) : _ in {1..m}] ;
4  end function;
5  ////////////////////////////////////////////////////////////////////////////////
```

Annexe D

algorithmes utiles en Magma

D.1 distance de Gilbert-Varshamov

Pour calculer la distance de Gilbert-Varshamov vu ici (A.1) :
Cas binaire :

```
binary_gv := function(n,k)
    d := 0;
    s := Binomial(n,d);
    while Log(2,s) lt (n-k) do
        d +:= 1;
        s +:= Binomial(n,d);
    end while;
    return d;
end function;
```

Cas général :

```
qary_gv := function(q,n,k)
   d := 1;
   s := Log(q,q - 1) + Log(q,Binomial(n,d));
   while s lt (n - k) and d lt n do
       s := s + Log(q,q - 1)+ Log(q,n-d) - Log(q,d+1);
       d := d + 1;
   end while;

   return d;
end function;
```

La fonction entropie binaire h_2 (ou h lorsqu'il n'y a pas d'ambiguïté) :

```
h2 := function(x)
        return (-x*Log(x) - (1-x)*Log(1-x))/Log(2);
end function;
```

D.2 Construire les matrices de parité d'un QC-LDPC à deux blocs circulants

```
//////////////////////// LDPC K_E ////////////////////////////////
LDPC := function(n,w)
    W1      := RandomSubset({1..n},w); // Un sous ensemble aleatoire de {1,2,..,n} de
        cardinal w
    K_E     := [ [ ((i - 1 + j) mod n) + 1 : i in W1 ]  : j in {0..n-1} ];

    return K_E;
end function;
//////////////////////////////////////////////////////////////////////
```

L'algorithme ci-dessus prend en entrée $n \in \mathbb{N}^*$ et un poids $w \in [|0, n|]$, et sort la forme **en colonne** [1] d'une matrice aléatoire de la forme :

$$M := \begin{bmatrix} a_1 & a_2 & .. & a_n \\ a_n & a_1 & .. & a_{n-1} \\ : & : & .. & : \\ a_2 & a_3 & .. & a_1 \end{bmatrix}$$

avec $(a_1, a_2, .., a_n) \in \mathbb{F}_q^n$. avec $w_H((a_1, a_2, .., a_n)) = w$.
Puis, il suffit d'appliquer cet algorithme et faire des concaténations avec la commande *cat* pour avoir des matrices de la forme $D := (D_1|D_2|..|D_p)$ avec p un entier pair, $D_i \in \mathbb{F}_2^{n\times n}$ circulante et de poids w par ligne.

```
    LDPCC := function(k,L,w)

    W1      := RandomSubset({1..k},w);
    K_D1    := [ [ ((i - 1 + j) mod k) + 1 : i in W1 ]  : j in {0..k-1} ];

    W2      := RandomSubset({1..k},w);
    K_D2    := [ [ ((i - 1 + j) mod k) + 1 : i in W2 ]  : j in {0..k-1} ];

    K_D     := K_D1 cat K_D2 ;
    K_E     := [K_D2[i] cat [j+(L div 2) : j in K_D1[i]] : i in {1..k}] ;

    return K_D, K_E;
end function;
```

L'algorithme ci-dessus prend en entrée deux nombres entiers positifs $k < L$, et un poids $w \in [|0, k|]$ et sort :

- La représentation en colonne K_D d'une matrice $D \in \mathbb{F}_2^{k\times 2k}$ de la forme $D := (D_1|D_2)$ où les deux matrices D_1 et D_2 sont circulantes de poids w par colonne.
- La représentation en colonne K_E d'une matrice E de la forme $E := \begin{pmatrix} D_1 \\ D_2 \end{pmatrix}$

Et donc $D.E = \mathbf{0}$.

1. Voir 2.4.2