

Codes correcteurs et cryptosystème de Mc Eliece

Auclair Pierre

3 mai 2014

Table des matières

1	Codes correcteurs	3
1.1	Définition d'un code correcteur	3
1.2	Utilisation des codes correcteurs	4
2	Codes de Goppa	5
2.1	Définition d'un code de Goppa	5
2.2	Construction de la matrice de parité	6
2.3	Équation clef	7
2.4	Résolution de l'équation clef	7
3	Cryptosystème de Mc Eliece	9
3.1	Les clefs	9
3.2	Le chiffrement	10
3.3	Le déchiffrement	10
3.4	Le principe de sécurité	10
3.4.1	Problème de reconnaissance du code de Goppa	10
3.4.2	Problème du décodage par syndrome	11
4	Principaux algorithmes	12
4.1	Algorithme d'Euclide étendu	12
4.2	Pivot de Gauss et applications	12
4.2.1	Pivot classique	12
4.2.2	Pivot de Gauss augmenté	13
4.2.3	Construire la matrice génératrice à partir de la matrice de parité	13
4.2.4	Inverse à gauche d'une matrice non carrée	14
4.3	Algorithme de Berlekamp	14
5	Implémentation des structures du code et du cryptosystème	15
5.1	Structures algébriques	15
5.1.1	Les matrices	15
5.1.2	Les polynômes	15
5.1.3	Les éléments dans un corps de Galois \mathbb{F}_{2^m}	16

Introduction

De nos jours, l'un des rôles principaux de l'informatique est la communication, aussi bien entre particuliers sur le réseau internet qu'au niveau de l'ingénierie militaire et spatiale. Néanmoins aucun des canaux utilisés pour transmettre l'information n'est sûr à 100% et des erreurs se produisent avec une probabilité non négligeable. C'est pourquoi une nouvelle branche de l'informatique : la théorie des codes, a émergée dans les années 1950 concevant des codes permettant de détecter et même de corriger les erreurs induites par les canaux de transmission.

Un exemple plus familier d'un tel code est le langage humain, celui-ci utilise deux procédés réutilisés par les codes plus informatiques. Tout d'abord, la répétition : lorsqu'on discute, tous les mots de la phrase ne sont pas nécessaires pour rendre compte de son sens. Il y a donc des informations redondantes dans la phrase, et on pourrait transposer cela en informatique en décidant d'envoyer plusieurs fois le même message. Ce n'est cependant pas la méthode utilisée à cause de son coût. Un autre procédé est celui de la distinguabilité des mots entre eux. Deux mots possèdent en général assez de syllabes différentes pour que même mal prononcés on puisse les distinguer. C'est plutôt ce procédé de séparer les mots les uns des autres qui est utilisé en informatique et que nous formaliserons.

Dans ce rapport nous nous intéresserons principalement aux codes de Goppa et à une de leurs applications en cryptographie dans le cryptosystème de Mc Eliece.

Chapitre 1

Codes correcteurs

1.1 Définition d'un code correcteur

Dans cette section nous allons formaliser l'idée de codes correcteurs telle qu'ébauchée dans l'introduction. L'idée est bien sur d'enrichir via le codage un message de taille k avec de la redondance pour obtenir un nouveau message de taille n . Il faut donc $k < n$.

Définition 1. [[Pan04](#)] Un code en bloc \mathbf{C} est l'image d'une application injective \mathbf{f} de \mathbb{F}_q^k dans \mathbb{F}_q^n . Ce code est dit linéaire si \mathbf{f} est une application linéaire.

$$\mathbf{f} : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$$

Nous avons aussi soulevé l'idée que pour mieux reconnaître le message d'origine, il fallait séparer les mots de code entre eux. Pour cela nous définissons une notion de distance.

Définition 2. La distance de Hamming est l'application \mathbf{d} de $\mathbb{F}_q^n \times \mathbb{F}_q^n$ dans \mathbb{N} définie par :

$$\mathbf{d} : (a, b) \in \mathbb{F}_q^n \times \mathbb{F}_q^n \mapsto \text{card}(i \in \mathbb{N} / a_i \neq b_i)$$

Maintenant que nous avons l'outil pour constater la séparation des mots de code, nous introduisons la notion de distance minimale entre deux mots. Cette distance donne un ensemble de boules chacune centrée sur un mot de code qui ne s'intersectent pas deux à deux.

Définition 3. On note t la capacité de correction d'un code correcteur, par définition :

$$t = \max_{r \in \mathbb{N}} \bigcap_{x \in \mathbf{C}} B(x, r) = \emptyset$$

De manière évidente on a l'inégalité : $2t + 1 \leq d$

Nous définissons enfin une application permettant de retrouver le message d'origine.

Définition 4. On appelle un décodage l'application $\mathbf{D} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^k$ telle que $\mathbf{D} \circ \mathbf{f} = \text{Id}_{\mathbb{F}_q^k}$. \mathbf{D} est de vraisemblance maximale si :

$$\forall x \in \mathbf{C}, \mathbf{D}(B(x, t)) = x$$

1.2 Utilisation des codes correcteurs

Nos codes correcteurs sont pour l'instant des ensembles de mots d'un espace de dimension plus grande que l'espace des messages. L'intérêt d'utiliser des applications linéaires est de pouvoir définir l'ensemble \mathbf{C} avec un minimum d'informations. La seule donnée de l'image des vecteurs de base définit entièrement \mathbf{f} et \mathbf{C} , donc le codage.

Définition 5. On appelle G matrice génératrice du code \mathbf{C} la matrice de \mathbf{f} dans la base canonique.

$$G \in M_{n,k}(\mathbb{F}_q), G = \text{Mat}(\mathbf{f})$$

Définition 6. On appelle H matrice de parité du code \mathbf{C} une matrice telle que :

$$H \in M_{n-k,n}(\mathbb{F}_q), \text{Ker}(H) = \mathbf{C}$$

L'application $x \in \mathbb{F}^n \rightarrow Hx$ s'appelle le syndrome de x .

Nous avons défini le codage de manière linéaire, il est impossible de faire de même pour le décodage. Nous allons donc utiliser la matrice de parité pour introduire la notion de syndrome qui va permettre le décodage.

Propriété 1. A la réception d'un message de \mathbb{F}^n , en considérant que le nombre d'erreurs est inférieur à t , donc que le vecteur d'erreur ϵ a un poids inférieur ou égal à t , on a : $\forall m \in \mathbb{F}^n, \exists!(x, \epsilon) \in \mathbf{C} \times \mathbb{F}^n$ tels que :

$$\begin{cases} m &= x + \epsilon \\ \mathbf{d}(\epsilon) &\leq t \\ Hm &= H\epsilon \end{cases}$$

Le principe est de déterminer ϵ . On retrouve ainsi le mot de code d'origine $x = m - \epsilon$

Propriété 2. $\forall m \in \mathbb{F}^n$, son syndrome est caractéristique de l'erreur, dans l'hypothèse d'un maximum de t erreurs. Supposons deux vecteurs d'erreurs avec le même syndrome :

$$H\epsilon = H\eta \Rightarrow H(\epsilon - \eta) = 0 \Rightarrow \epsilon - \eta \in \mathbf{C}$$

D'après l'inégalité triangulaire, $d(\epsilon - \eta) \leq 2t < d$ donc $\epsilon - \eta = 0$

Néanmoins cette méthode est relativement limitée en l'état. Pour un code linéaire quelconque, la reconnaissance du syndrome est un problème jugé difficile. L'enjeu de la théorie des codes va être de construire des structures de codes rendant le problème plus simple. Dans la suite de ce TIPE nous nous intéresserons aux codes de Goppa dont l'usage principal sera le cryptosystème de Mc Eliece étudié dans la partie 3.

Chapitre 2

Codes de Goppa

2.1 Définition d'un code de Goppa

Les codes de Goppa classiques sont des codes linéaires dont on connaît une borne inférieure de la distance minimale. Ce sont les codes utilisés dans le cryptosystème de McEliece qui va nous intéresser dans une troisième partie.

Définition 7. On définit un code de Goppa $\mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$ par son support L et un polynôme g définis par :

$$\begin{cases} g & \in \mathbb{F}_{q^m}[X] \text{ unitaire irréductible sur } \mathbb{F}_{q^m} \text{ de degré } t \\ L & = (\alpha_1, \dots, \alpha_n) \text{ d'éléments de } \mathbb{F}_{q^m} \end{cases}$$

Dans la pratique on prend $n = q^m$ et $q = 2$ pour manipuler des codes binaires. A partir de ces éléments, on définit le code de Goppa \mathbf{C} par son syndrome \mathbf{S} comme :

$$\mathbf{C} = \{y = (y_1, \dots, y_n) \in \mathbb{F}_q^n / \mathbf{S}_y(x) = \sum_{i=1}^n \frac{y_i}{x - \alpha_i} = 0 \text{ mod } g(x)\}$$

Propriété 3. Le code de Goppa ainsi défini est de distance minimale $\mathbf{d} \geq t + 1$. En effet en supposant qu'il existe un message $y = (y_1, \dots, y_n) \in \mathbf{C}$ avec t ou moins coordonnées non nulles :

$$\sum_{i=1}^n \frac{y_i}{x - \alpha_i} = \frac{a(x)}{b(x)} = 0 \text{ mod } g(x) \text{ fraction irréductible}$$

Donc $g \mid a$, ainsi a de degré supérieur ou égal à t . Or a de degré = nombre de coordonnées non nulles de y - 1 $\leq t - 1$

Par l'absurde le code de Goppa a une distance minimale $\mathbf{d} \geq t + 1$.

2.2 Construction de la matrice de parité

Nous souhaiterions donner à notre code de Goppa défini par son syndrome une structure plus visiblement linéaire. Nous allons construire sa matrice de parité.

Définition 8. [[Pan04](#)] Nous allons définir une famille de polynômes \mathbf{f}_i inverses des $(x - \alpha_i)$. Cela est possible car g étant irréductible, $\mathbb{F}[X]/g$ est un corps :

$$\mathbf{f}_i(x)(x - \alpha_i) = 1 \text{ mod } g(x)$$

On a donc que :

$$\forall y \in \mathbb{F}^n \quad \mathbf{S}_y(x) = \sum_{i=1}^n \frac{y_i}{x - \alpha_i} = \sum_{i=1}^n \mathbf{f}_i(x) y_i$$

Propriété 4. Évaluons les \mathbf{f}_i sous forme de fraction rationnelle :

$$\mathbf{f}_i(x) = \frac{1}{g(\alpha_i)} \frac{g(x) - g(\alpha_i)}{x - \alpha_i}$$

Maintenant tentons de les rendre polynomiaux, pour cela utilisons une formule de factorisation bien connue :

$$x^j - \alpha_i^j = (x - \alpha_i) \sum_{k=0}^{j-1} x^k \alpha_i^{j-1-k}$$

On en déduit donc que :

$$\mathbf{f}_i(x) = \frac{1}{g(\alpha_i)} \sum_{j=1}^t g_j \sum_{k=0}^{j-1} x^k \alpha_i^{j-1-k} = \frac{1}{g(\alpha_i)} \sum_{k=0}^{t-1} x^k \sum_{j=k+1}^t g_j \alpha_i^{j-1-k}$$

Maintenant qu'on a montré leur existence et qu'on les a calculés, nous utilisons les \mathbf{f}_i pour construire la matrice de parité. Pour cela nous allons identifier un polynôme comme un vecteur de degré t .

Définition 9. On a donc des polynômes $(f_i)_{1 \leq i \leq n}$ de degrés $t-1$ que l'on va identifier comme des vecteurs colonnes de dimension $t-1$.

$$\forall y \in \mathbb{F}^n \quad \mathbf{S}_y(x) = \sum_{i=1}^n \frac{y_i}{x - \alpha_i} = \sum_{i=1}^n \mathbf{f}_i(x) y_i$$

$$\forall y \in \mathbb{F}^n \quad \mathbf{S}_y(x) = (f_1(x), \dots, f_n(x))y$$

On se retrouve donc avec une matrice de calcul de syndrome :

$$\begin{pmatrix} \frac{1}{g(\alpha_1)} g_t & \frac{1}{g(\alpha_2)} g_t & \cdots & \frac{1}{g(\alpha_n)} g_t \\ \frac{1}{g(\alpha_1)} (g_{t-1} + g_t \alpha_1) & \frac{1}{g(\alpha_2)} (g_{t-1} + g_t \alpha_2) & \cdots & \frac{1}{g(\alpha_n)} (g_{t-1} + g_t \alpha_n) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{g(\alpha_1)} (g_1 + \dots + g_t \alpha_1^{t-1}) & \frac{1}{g(\alpha_2)} (g_1 + \dots + g_t \alpha_2^{t-1}) & \cdots & \frac{1}{g(\alpha_n)} (g_1 + \dots + g_t \alpha_n^{t-1}) \end{pmatrix}$$

Pour construire la matrice de parité, comme seul le noyau nous intéresse, nous allons simplifier cette matrice et poser H matrice de parité égale à :

$$\begin{pmatrix} \frac{1}{g(\alpha_1)} & \frac{1}{g(\alpha_2)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{1}{g(\alpha_1)} g_t \alpha_1 & \frac{1}{g(\alpha_2)} g_t \alpha_2 & \cdots & \frac{1}{g(\alpha_n)} g_t \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{g(\alpha_1)} g_t \alpha_1^{t-1} & \frac{1}{g(\alpha_2)} g_t \alpha_2^{t-1} & \cdots & \frac{1}{g(\alpha_n)} g_t \alpha_n^{t-1} \end{pmatrix}$$

Maintenant que nous possédons sa matrice de parité, le codage est possible. La construction de G matrice génératrice de notre code de Goppa s'effectue simplement. N'étant pas spécifique aux codes de Goppa, l'algorithme utilisé est décrit plus tard.

2.3 Équation clef

Nous sommes dorénavant dotés des outils algébriques nécessaires au codage de nos messages. Nous allons maintenant étudier comment les décoder.

Définition 10. [Fin04] Nous allons introduire le polynôme localisateur d'erreurs σ , qui par sa connaissance permet de déterminer l'emplacement des erreurs dans le message. Dans l'hypothèse d'un nombre d'erreurs permettant la correction $\deg(\sigma) \leq \frac{t}{2}$.

$$\forall y \in \mathbf{C}, \sigma_{y+\epsilon}(x) = \prod_{\epsilon_i \neq 0} (x - \alpha_i) \mod g(x)$$

L'enjeu du décodage va être de déterminer le polynôme localisateur d'erreurs pour un message donné et d'en déterminer les racines.

Définition 11. Nous allons définir le polynôme évaluateur d'erreurs ω défini par :

$$\omega(x) = \sigma(x) \mathbf{S}_y(x) \mod g(x)$$

Par construction $\deg(\omega) < \frac{t}{2}$. L'intérêt de ce second polynôme est de mettre en évidence l'équation clef du décodage :

$$\omega(x) = \sigma(x) \mathbf{S}_y(x) + k(x)g(x)$$

2.4 Résolution de l'équation clef

La résolution de cette équation clef est la détermination de σ . Pour cela il existe plusieurs techniques, nous étudierons celle utilisant l'algorithme d'Euclide. L'algorithme d'Euclide étendu tel qu'utilisé est redonné plus tard.

Propriété 5. [CL] On a unicité de la résolution de l'équation clef pour les polynômes σ et ω de degrés $\deg(\sigma) \leq \frac{t}{2}$ et $\deg(\omega) < \frac{t}{2}$ à un scalaire près. On

suppose de plus que les deux polynômes sont de degrés minimum, c'est à dire premiers entre eux. En effet, soient (σ_1, ω_1) et (σ_2, ω_2) deux solutions de degrés inférieurs à $\frac{t}{2}$. On a :

$$\omega_1(x) = \sigma_1(x)\mathbf{S}_y(x) \bmod g(x) , \quad \omega_2(x) = \sigma_2(x)\mathbf{S}_y(x) \bmod g(x)$$

$$\omega_1(x)\sigma_2(x) = \sigma_1(x)\mathbf{S}_y(x)\sigma_2(x) \bmod g(x) , \quad \omega_2(x)\sigma_1(x) = \sigma_2(x)\mathbf{S}_y(x)\sigma_1(x) \bmod g(x)$$

Ainsi :

$$\omega_1(x)\sigma_2(x) - \omega_2(x)\sigma_1(x) = 0 \bmod g(x)$$

Or le degré du polynôme du membre de gauche est de degré $\leq t-1$. Ce polynôme est donc nul et $\omega_1(x)\sigma_2(x) = \omega_2(x)\sigma_1(x)$. Avec l'hypothèse de degrés minimums, on a égalité des couples à un scalaire près.

Il s'agit désormais d'explicitier un tel couple solution pour accéder au polynôme localisateur d'erreurs. Pour cela nous utilisons l'algorithme d'Euclide étendu.

Propriété 6. On définit des suites (r_n, s_n, t_n) telles que $r_n(x) = s_n(x)\mathbf{S}_y(x) + t_n(x)g(x)$. La relation de récurrence est donnée par l'algorithme d'Euclide étendu. Comme les solutions de l'équation clef existent par construction, on sait que le PGCD de $g(x)$ et $\mathbf{S}_y(x)$ a un degré strictement inférieur à $\frac{t}{2}$.

Ainsi on sait que la suite des $r_n(x)$ a des degrés décroissants et $\exists s / \deg(r_s(x)) < \frac{t}{2}$, avec $\deg(r_{s-1}(x)) \geq \frac{t}{2}$. Montrons que $\deg(s_s(x)) \leq \frac{t}{2}$.

$$s_{i+1}(x) = s_{i-1} - (r_{i-1}(x) \operatorname{div} r_i(x))s_i(x)$$

Donc :

$$\deg(s_s) = \sum_{i=2}^{s-1} \deg(r_{i-1}(x) \operatorname{div} r_i(x)) = \deg(r_1(x)) - \deg(r_{s-1}(x)) \leq \frac{t}{2}$$

L'algorithme d'Euclide étendu, stoppé au bon moment concernant les contraintes sur les degrés donne une solution particulière. Grâce à l'unicité, nous avons déterminé le polynôme localisateur d'erreur dont la détermination des racines caractérise le vecteur d'erreur du message. Pour cela nous utiliserons au choix, l'algorithme de Berlekamp-Hensel ou une recherche exhaustive.

Chapitre 3

Cryptosystème de Mc Eliece

Le cryptosystème de Mc Eliece mis au point en 1978 est l'un des premiers systèmes de cryptage à clef publique. Son principe repose sur l'utilisation de codes correcteurs d'erreurs, plus particulièrement des codes de Goppa. Ce cryptosystème est fréquemment comparé au système RSA, puisqu'il a été inventé presque au même moment. La différence de taille des clefs entre ces deux systèmes pour un même niveau de sécurité a jusqu'ici grandement pénalisé le développement du cryptosystème de Mc Eliece. Néanmoins, des projets comme celui de l'OTAN¹ remettent au jour ce système pouvant résister à l'avènement d'ordinateurs quantiques. C'est ce cryptosystème que nous avons étudié.

3.1 Les clefs

Un cryptage asymétrique repose sur un principe simple : tout le monde a accès à une clef publique permettant le chiffrement et seul le créateur de cette clef possède la clef privée permettant le déchiffrement.

Pour ce faire, on commence par créer G la matrice génératrice d'un code de Goppa de paramètres :

- k dimension du bloc de message à coder
- n dimension du bloc de message à envoyer
- t la capacité de correction

La clef privée représente l'ensemble :

- G matrice génératrice (n,k)
- P une matrice (n,n) de permutation
- Q une matrice (k,k) inversible

La clef publique partagée est l'ensemble :

- $G' = PGQ$

1. Projet OTAN : SPS 984520 Secure implementation of post-quantum cryptography

- La capacité de correction t

3.2 Le chiffrement

Soit $m \in \mathbf{F}_2^k$ le message à chiffrer, l'envoyeur calcule l'image du message par le code G puis ajoute un vecteur $e \in \mathbf{F}_2^n$ de poids inférieur à t .

$$x = G'm + e = PGQm + e$$

C'est ce vecteur $x \in \mathbf{F}_2^n$ qui correspond au message chiffré.

3.3 Le déchiffrement

Soit $x \in \mathbf{F}_2^n$ le message chiffré reçu, on calcule :

$$P^{-1}x = GQm + P^{-1}e$$

Le vecteur $GQm \in \mathbf{F}_2^n$ est un mot du code de Goppa et donc $P^{-1}e \in \mathbf{F}_2^n$ est un vecteur d'erreur de poids inférieur à t car P matrice de permutation. On utilise notre algorithme efficace de décodage du code de Goppa, on obtient donc Qm , connaissant Q inversible on a :

$$m = Q^{-1}Qm$$

D'où le déchiffrement.

3.4 Le principe de sécurité

Le cryptosystème de Mc Eliece repose sur 2 problèmes difficiles mathématiques. Premièrement, la difficulté à partir d'une matrice $G' = PGQ$ de retrouver la matrice G en temps polynômial. Secondement, de décoder un code correcteur qui paraît aléatoire, c'est le problème de reconnaissance du syndrome que nous avons évité en utilisant un code de Goppa dont on connaît la structure et donc un algorithme efficace de décodage.

3.4.1 Problème de reconnaissance du code de Goppa

[Fin04] Quand on parle de l'indistinguabilité d'un code de Goppa permuté, il s'agit de reconnaître un code de Goppa sans la donnée de son support ni de g . Dans notre cas, il s'agit de retrouver la structure du code à partir de la matrice génératrice ou de parité permutée. On peut formaliser le problème ainsi :

Problème 1. Soit H_g , la matrice de parité d'un code de Goppa de polynôme g ordonnée. Un distinguéur de code de Goppa est un algorithme prenant en entrée

une matrice H' . Ce distingueur doit vérifier avec une bonne probabilité s'il existe un triplet (g, P, Q) tel que :

$$\begin{aligned} P &\in S_n(\mathbb{F}) \\ Q &\in GL_k(\mathbb{F}) \\ H' &= PH_gQ \end{aligned}$$

Aucun distingueur sous exponentiel n'est connu à ce jour. Les meilleurs distingueurs consistent en une recherche exhaustive de codes de Goppa ayant des paramètres communs et de tester l'équivalence.

3.4.2 Problème du décodage par syndrome

[Vé95] L'objectif est de montrer que le problème de caractérisation du syndrome suivant est NP-complet.

Problème 2. Soit H une matrice (k, n) de parité d'un code correcteur linéaire. Soit y un vecteur de \mathbb{F}^n . On cherche l'unique ϵ tel que :

$$\begin{aligned} Hy &= H\epsilon \\ \mathbf{d}(\epsilon) &< t \end{aligned}$$

Dans la théorie de la NP-complétude, au problème dit d'optimisation énoncé ci-dessus on associe le problème de décision suivant.

Problème 3. Soit H une matrice (k, n) de parité d'un code correcteur linéaire. Soit y un vecteur de \mathbb{F}^n . On cherche s'il existe ϵ tel que :

$$\begin{aligned} Hy &= H\epsilon \\ \mathbf{d}(\epsilon) &< t \end{aligned}$$

Ce problème à été réduit polynomialement au problème dit des trois mariages suivant par Mc Eliece, Berlekamp et Van Tilborg.

Problème 4. Soit T un ensemble fini. Soit $U \subset T \times T \times T$. On cherche s'il existe W tel que :

$$\begin{aligned} W &\subset U \\ \text{card}(W) &= \text{card}(T) \\ \forall (x, y) \in W^2, \forall i, x_i &\neq y_i \end{aligned}$$

Ce problème étant NP-complet, le problème de décision associé au problème du décodage par syndrome l'est aussi. Donc à fortiori le problème de décodage par syndrome est NP-complet.

Chapitre 4

Principaux algorithmes

4.1 Algorithme d'Euclide étendu

Algorithme 1. *On veut obtenir des coefficients de Bézout pour les deux polynômes A et B . On définit trois suites (r_n, s_n, t_n) telles que $\forall n \ r_n = s_n A + t_n B$. On a les relations de récurrence à deux éléments suivantes :*

$$\begin{aligned}r_{n+1} &= r_{n-1} - (r_{n-1} \operatorname{div} r_n) r_n \\s_{n+1} &= s_{n-1} - (r_{n-1} \operatorname{div} r_n) s_n \\t_{n+1} &= t_{n-1} - (r_{n-1} \operatorname{div} r_n) t_n\end{aligned}$$

Ces relations de récurrence conservent bien la propriété, avec $(\deg(r_n))_n$ suite strictement décroissante. Comme vu durant la démonstration du décodage, le calcul du PGCD ne nous intéresse même pas. On coupe l'exécution de l'algorithme dès que $\deg(r_n) < \frac{t}{2}$. Pour l'initialisation on pose :

$$\begin{aligned}(r_0, s_0, t_0) &= (A, 1, 0) \\(r_1, s_1, t_1) &= (B, 0, 1)\end{aligned}$$

4.2 Pivot de Gauss et applications

4.2.1 Pivot classique

L'algorithme du pivot de Gauss fait partie du programme de Sup. Nous ne redonnons ni une preuve du résultat ni un algorithme complet sinon un résumé pratique de manière récursive.

Algorithme 2. *Soit M une matrice de taille $n+1$.*

- 1. On cherche une ligne i dont le premier coefficient est non nul*
- 2. S'il n'existe pas de tel i on arrête l'algorithme car M n'est pas inversible*
- 3. On échange la première ligne et la i ième ligne*

4. On multiplie la première ligne pour avoir un premier coefficient 1
5. $\forall i \geq 2, l_i \leftarrow l_i - M_{i,1} \times l_1$
6. On itère pour le bloc inférieur droit de taille n

Ceci trigonalise la matrice, et il suffit de faire la même opération en remontant. En effectuant les mêmes opérations sur la matrice identité, on obtient la matrice M^{-1}

4.2.2 Pivote de Gauss augmenté

Toutes nos sources montrent comment générer une matrice de parité pour un code de Goppa. Cependant le cryptosystème classique de Mc Eliece utilise des matrices génératrices. Pour passer de l'un à l'autre nous avons utilisé un pivote de Gauss augmenté.

Les deux principales augmentations sont de permettre d'effectuer les calculs même si la matrice n'est pas inversible et de stocker les permutations effectuées dans une pile. Cela donne un algorithme tel que le suivant.

Algorithme 3. Soit M une matrice de taille $n+1$.

1. On cherche une ligne i dont le premier coefficient est non nul
2. Si un tel i existe, on échange la première ligne et la i ième ligne et on stocke cette permutation
3. S'il n'existe pas de tel i , on cherche un (i,j) tel que $M_{i,j} \neq 0$, on échange la i ième ligne avec la première et la j ième colonne avec la première en stockant ces permutations
4. S'il n'existe pas de tel (i,j) on arrête l'algorithme, cette étape se produit toujours en pratique.
5. On multiplie la première ligne pour avoir un premier coefficient 1
6. $\forall i \geq 2, l_i \leftarrow l_i - M_{i,1} \times l_1$
7. On itère pour le bloc inférieur droit de taille n

On n'utilise jamais cette version du pivote de Gauss sur une matrice inversible. Le résultat intéressant est la pile des permutations qui permet de réorganiser, par les permutations, la matrice M comme suit :

$$M = \begin{pmatrix} A & X \\ Y & Z \end{pmatrix}$$

Où $A \in GL_r(\mathbb{F})$ avec r rang de M .

4.2.3 Construire la matrice génératrice à partir de la matrice de parité

C'est une des deux principales applications du pivote de Gauss augmenté. Soit H matrice de parité, $H \in M_{n,k}(\mathbb{F})$. On cherche $G \in M_{k,n}(\mathbb{F})$ de rang maximal telle que $HG = 0$.

Pour cela on utilise l'algorithme pour se ramener au cas de $H' = LHC$. $L \in GL_n(\mathbb{F})$ représente les permutations de lignes et $C \in GL_k(\mathbb{F})$ de colonnes. On a donc :

$$H' = \begin{pmatrix} U & V \\ W & Z \end{pmatrix}, G' = \begin{pmatrix} X \\ I_{n-r} \end{pmatrix}$$

Où U est inversible de taille $r = \text{rang}(H')$. On veut $\text{rang}(G) = \dim(\text{Ker}(H)) = n - r$ d'où le découpage de G.

$$H'G' = \begin{pmatrix} UX + V \\ WX + Z \end{pmatrix} = 0$$

On se retrouve avec un système de rang r, il suffit de poser $X = -U^{-1}V$.

Maintenant qu'on a construit G' tel que $LHC'G' = 0$, avec L et C inversibles. La matrice génératrice est la matrice :

$$G = CG'$$

4.2.4 Inverse à gauche d'une matrice non carrée

C'est ici la deuxième application. Après avoir corrigé les erreurs, on se retrouve avec un mot $y = Gx \in \mathbb{F}^n$. On aimerait pouvoir calculer un inverse à gauche de G nommé D. Normalement G est de rang n-r égal à son nombre de colonnes.

On procède de même que ci-dessus avec :

$$D' = (U^{-1} \quad 0_r), G' = LGC = \begin{pmatrix} U \\ V \end{pmatrix}$$

On a donc $D'LGC = I_{n-r}$. Pour avoir la matrice de décodage on pose :

$$D = C^{-1}D'L$$

4.3 Algorithme de Berlekamp

[Ram08] Pour générer un code de Goppa aléatoire, il faut un polynôme de $\mathbb{F}_{2^m}[X]$ de degré t irréductible. Nous avons utilisé l'algorithme de Berlekamp pour tester l'irréductibilité d'un polynôme dans un corps fini. Nous en donnons ici le principe et quelques morceaux de démonstration.

Chapitre 5

Implémentation des structures du code et du cryptosystème

5.1 Structures algébriques

5.1.1 Les matrices

```
1  def __init__(self, nbligne, nbcolonne, tableau):
2      """Definition de la matrice avec gestion du cas erreur de taille"""
3      try:
4          if len(tableau) != nbligne * nbcolonne :
5              raise IndexError("Erreur de taille de la matrice durant l'initialisation")
6          self.nbligne=nbligne
7          self.nbcolonne=nbcolonne
8          self.tableau=tableau[::]
9      except IndexError as ex:
10         print(ex)
11         print [nbligne, nbcolonne, tableau]
12     except AttributeError as ex:
13         print(ex)
14         print [nbligne, nbcolonne, tableau]
```

Les matrices sont représentées par un tableau linéaire dont le comportement est caractérisé par le nombre de lignes et le nombre de colonnes. La flexibilité de Python nous permet d'utiliser cette classe aussi bien pour des réels que pour des éléments de corps fini.

5.1.2 Les polynômes

```
1  def __init__(self, liste):
```



```

2      """Initialisation de l'objet"""
3      try:
4          self.liste=liste[:]
5      except TypeError:
6          print("Un polynome recoit une liste en facteur")

```

Les polynômes à coefficients aussi bien dans \mathbb{R} que dans des corps finis.

5.1.3 Les éléments dans un corps de Galois \mathbb{F}_{2^m}

```

1      def __init__(self,n,p=2):
2          """Initialisation"""
3          diff = d(n,p)
4          while diff>= 0:
5              n ^= p<<diff
6              diff = d(n,p)
7          self.valeur = n
8          self.modulo = p

```

Un élément d'un corps de Galois sur \mathbb{F}_{2^m} est un polynôme de $\mathbb{Z}/2\mathbb{Z}[X]/P$ avec P un polynôme irréductible de $\mathbb{Z}/2\mathbb{Z}[X]$. Notre implémentation repose sur la compréhension du binaire par l'interpréteur Python : un entier est stocké sous sa forme binaire sur laquelle on peut effectuer des opérations binaires (xor, and, or et décalages de bits). Grâce à cette caractéristique de Python on peut utiliser des entiers comme des polynômes de $\mathbb{Z}/2\mathbb{Z}$. On obtient une structure d'éléments de \mathbb{F}_{2^m} très rapide.

Bibliographie

- [CL] Antoine Chambert-Loir. *Codes de Goppa - Préparation à l'agrégation - option Calcul formel*.
- [Fin04] Matthieu Finiasz. *Nouvelles constructions utilisant des codes correcteurs d'erreurs en cryptographie à clef publique*. PhD thesis, Polytechnique, 2004.
- [Pan04] A.A Pantchichkine. *Mathématiques des codes correcteurs d'erreurs*. Institut Fourier, 2004.
- [Ram08] Sanjay Ramassamy. *Quelques aspects de la factorisation des polynômes sur \mathbb{Z} et sur les corps finis*, 2008.
- [Vé95] Pascal Véron. *Problème SD, Opérateur Trace, Schémas d'Identification et Codes de Goppa*. PhD thesis, Université de Toulon, 1995.