Codes correcteurs et cryptosystème de Mc Eliece

Auclair Pierre

4 mai 2014

Table des matières

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 sur 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 possèdes 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 f est une application linéaire.*

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

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 :*

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

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* $\mathbf{D} : \mathbb{F}_q^n \to \mathbb{F}_q^k$ *telle que* $\mathbf{D} \circ \mathbf{f} = Id_{F_q^k}$ *.* $\mathbf{D}$ *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.*

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

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

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

L'application $x \in \mathbb{F}^n \to Hx$ *s'appelle le syndrôme 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 a t, donc que le vecteur d'erreur* ϵ *a un poids inferieur ou egal a 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 determiner ϵ . On retrouve ainsi le mot de code d'origine $x = m - \epsilon$

Propriété 2. $\forall m \in \mathbb{F}^n$ *, son syndrôme est caratéristique 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 syndrôme 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* $\mathbb{F}_q^k \to \mathbb{F}_q^n$ *par son support* L *et un polynôme* g *définis par :*

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

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 :*

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

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* $y = (y_1, .., y_n) \in \mathbf{C}$ *avec* t *ou moins coordonnées non nulles :*

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

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 :*

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

On a donc que :

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

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 fi 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* $(fi)_{1 \leq i \leq n}$ *de degrés t-1 que l'on va identifier comme des vecteurs colonnes de dimension t-1.*

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

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

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

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

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 :*

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

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

Ainsi :

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

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'expliciter 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* $deg(s_s(x)) \leq \frac{t}{2}$*.*

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

Donc :

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

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 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'inventé presqu'au même moment. La différence de taille des clefs entre ces deux systèmes pour un même niveau de sécurité à jusqu'ici grandement pénalisé le développement du cryptosystème de Mc Eliece. Néanmoins, des projets comme celui de l'OTAN [1] remettent au jour ce système pouvant resister à l'avènement d'ordinateurs quantiques. C'est ce cryptosytème 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 partagé 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 parait aléatoire, c'est le problème de reconnaissance du syndrome que nous avons évité en utilisant un code de Goppa dont on connait la strucure 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 distingueur de code de Goppa est un algorithme prenant en entrée*

une matrice H'. Ce distingueur doit vérifier avec une bonne probabilté 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 enoncé 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 :*

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

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 :*

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

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'execution de l'algorithme dès que* $deg(r_n) < \frac{t}{2}$. *Pour l'initialisation on pose :*

$$\begin{aligned} (r_0, s0, t0) &= (A, 1, 0) \\ (r_1, s1, t1) &= (B, 0, 1) \end{aligned}$$

4.2 Pivot de Gauss et applications

L'algorithme du pivot de Gauss fait partie du programme de Sup. Nous ne redonnons ni une preuve du résultat ni un algorithme complet. Nous l'avons utilisé pour calculer des inverses mais nous l'avons aussi modifié pour d'autres utilisations.

4.2.1 Pivot de Gauss modifié

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 pivot de Gauss modifié.

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 2. *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 ieme 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 eme ligne avec la première et la j ieme 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 pivot 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.

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

C'est une des deux principales applications du pivot de Gauss modifié. 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 $r = rang(H')$. On veut $rang(G) = dim(Ker(H)) = n - r$ 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 $LHCG' = 0$, avec L et C inversibles. La matrice génératice est la matrice :

$$G = CG'$$

4.2.3 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 :

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

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-Hensel

[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. Il est plus simple à démontrer que sa version utilisée pour factoriser ces polynômes. Nous en donnons ici le principe et quelques morceaux de démonstration.

Propriété 7. *Soit* $H \in \mathbb{F}_{2^m}[X]$,

$$H^{2^m} - H = \prod_{c \in \mathbb{F}_{2^m - 1}} (H - c)$$

En effet,

$$\forall c \in \mathbb{F}_{2^m}, H^{2^m} - H = H(H^{2^m - 1} - 1) = H(H^{2^m - 1} - c^{2^m - 1})$$

Une formule que l'on a déjà utilisée montre que H-c divise le polynôme de droite. Comme les H-c sont premiers entre eux deux à deux et divisent $H^{2^m} - H$, leur produit le divise. L'égalité vient de l'égalité du degré et du premier coefficient.

Maintenant on veut tester l'irréductibilité d'un polynôme P unitaire sans facteur multiple.

Propriété 8. *Si* $H \in \mathbb{F}_{2^m}[X]$ *unitaire satisfait* $H^{2^m} = H \ mod(P)$, *alors*

$$P = \prod_{c \in \mathbb{F}_{2^m}} pgcd(P, H - c)$$

Pour le montrer on utilise la propriété précedente. P divise $H^{2^m} - H$ donc $P = pgcd(P, \prod(H - c))$. Les H-c étant premiers entre eux deux à deux, P divise le produit des PGCD. De plus chacun des $pgcd(P, H - c)$ divise P et ils sont premiers deux à deux donc leur produit divise P. Les deux polynômes étant unitaires l'égalité est satisfaite.

On remarque que si $0 < deg(H) < t$, pgcd(P,H-c) ne vaut jamais P, donc met en évidence que P n'est pas irréductible. On en conclut donc que l'existence de H de degré $\in 2..t-1$ implique que P est réductible. Il reste à montrer l'équivalence.

Propriété 9. *Soit $P = P_1 \times ... \times P_r$. D'après le théorème des restes chinois, $H \to (H\ mod(P_1), ...H\ mod(P_r))$ de $\mathbb{F}[X]/P \to (\mathbb{F}[X]/P_1, ..., \mathbb{F}[X]/P_r)$ est un isomorphisme d'anneaux.*

Comme les P_i sont irréductibles, $\{x \in \mathbb{F}[X]/P_1/x = x^{2m}\}$ est de cardinal 2^m. Donc il existe 2^{mr} polynômes de degrés inférieurs à t-1 vérifiant la propriété désirée. Par suite si P est réductible il existe au moins un polynôme H avec $1 < deg(H) < t$ et $H^{2^m} = H\ mod(P)$. On a donc l'équivalence voulue.

Maintenant qu'on a une équivalence sympatique entre l'irréductibilité de P et la non existence du polynôme H, on utilise le morphisme de Frobenius. C'est à dire que l'application suivante est linéaire.

$$\begin{array}{rcl} \mathbb{F}_{2^m}/P & \to & \mathbb{F}_{2^m}/P \\ H & \to & H^{2^m} \end{array}$$

On en déduit l'algorithme suivant :

Algorithme 3. *Soit P de degré t dont on veut tester l'irréductibilité :*

1. *Si $pgcd(P, P') \neq 0$ alors P possède des facteurs multiples, à fortiori P est réductible, on arrête l'éxecution.*
2. *On calcule A la matrice du morphisme de Frobenius dans la base canonique*
3. *On détermine le noyau de (A - Id) dans la base canonique*
4. *Si $dim(Ker(A - Id)) = 1$ alors les seuls H existant sont de degré 1 et P est irréductible*
5. *Sinon P est réductible*

Chapitre 5

Implémentation des structures du code et du cryptosystème

En plus du volet théorique développé ci-dessus, nous avons implémenté une version du cryptosystème de Mc Eliece. Notre code développé en Python ne nous permet pas d'atteindre les standards de sécurité mais fonctionne avec des paramètres plus modestes. Nous livrons dans cette partie quelques morceaux de codes intéressant. Si cela vous intéresse, la totalité des sources est disponible sur Github (https://github.com/kalaspa/mc-eliece).

5.1 Structures algébriques

Dans un but pédagogique nous avons redéfini des classes de polynômes et de matrices. L'intérêt supplémentaire est que ces classes fonctionnent aussi bien avec des nombres réels qu'avec nos classes d'éléments de Galois. Ceci est permis par la surcharge des opérateurs dans le langage python. Les classes de matrices et de polynômes n'ont en soit pas grand chose d'intéressant.

Un élement 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.

```
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
```

Ici d est une fonction qui calcule la différence de degré entre deux polynômes. Elle est équivalente avec nos considérations à $log_2(n) - log_2(p)$. p représente le polynôme qui forme l'idéal.

5.2 Structures de clef

Nous avons défini des classes de clef. Celles-ci sont des ensembles contenant les informations nécessaires.

5.2.1 Clef privée

La clef privée est l'objet qui permet à la fois le décryptage et la création de la clef publique.

Elle est déclarée comme suit :

```
1       """Methode d'initialisation de la clef privee
2       G de taille n,k
3       P permutation de taille n,n
4       Q inversible de taille k,k"""
5       self.G=G
6       self.P = P
7       self.D = D
8       self.Q = Q
9       self.Qinv=Qinv
10      self.g = g
11      self.support = support
12      self.L_fi = L_fi
13      self.mod = mod
14      self.correction = correction
```

Pour des considérations de rapidité et pour ne pas se répéter, on stocke les matrices inverses de Q. L_fi est la liste des polynômes défini en définition 8, ils permettent de calculer le syndrôme directement sous forme de polynôme.

La création de la clef privée se fait de manière aléatoire avec cette méthode :

```
1   def new(self,mod,correction):
2       """Methode pour generer une clef privee aleatoire"""
3
4       n = 2**(d(mod,0))
5
6       #Generation de g
7       g = P_irreductible(2*correction+1,mod)
```

```
8       print '-Polynome genere'
9
10      #Generation du support aleatoire
11      liste = range(n)
12      support = []
13      for i in range(n)[::-1]:
14        support.append(elt(liste.pop(randint(0,i)),mod))
15      L_fi = fi(g,support,mod)
16      print '-Support genere'
17
18      #Generation de la matrice de parite et de ses dependances
19      H = parite(g,support,mod)
20      print "-H generee"
21      G = generatrice(H)
22      print "-G generee"
23      D = decodage(G)
24      print '-D generee'
25
26      #Generation des matrices de melange
27      P = permutation(n)
28      print "-P generee"
29      Q = inversible(G.nbcolonne)
30      print '-Q generee'
31      Qinv = Q.inverse()
32      print "-Q inverse calculee"
33
34      return clef_privee(G,P,D,Q,Qinv,g,support,L_fi,mod,correction)
```

Les algorithmes sous jacents ont été explicités de manière théorique précedemment.

5.2.2 Clef publique

La clef publique est l'objet diffusé. Elle contient uniquement la matrice G' et la capacité de correction. Elle est crée à partir de la clef privée avec cette méthode :

```
1   def new(self,privkey):
2     """Methode pour creer une clef publique a partir d'une clef publique"""
3
4     return clef_publique(privkey.P * (privkey.G* privkey.Q),privkey.correction)
```

On notera le gain de temps en effectuant d'abord le produit matriciel des matrices de plus petite taille.

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 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.