Codes correcteurs et cryptosystème de Mc Eliece

Auclair Pierre

12 juin 2014

Le cryptosystème de Mc Eliece est un système de cryptage qui s'appuie sur la théorie des codes correcteurs. Il se situe à intersection des soucis de fiabilité et de sécurité de l'information. C'est pourquoi nous avons implémenté ce système en Python dans le cadre du sujet transfert et échange. La totalité des ressources est disponible ici : https://github.com/kalaspa/mc-eliece

Préliminaires

Codes correcteurs

[Pan04] Tout d'abord, définissons quelques notions des codes correcteurs. Un code correcteur est l'image C d'une application linéaire f de $\mathbb{F}^k$ vers $\mathbb{F}^n$. On définit, avec les conventions du cours de mathématiques, les matrices génératrices G et de parité H :

$$\begin{aligned} G &= Mat(f) \\ Ker(H) &= Im(f) \end{aligned}$$

La distance minimale d du code est le poids de Hamming le plus faible d'un mot non nul du code. La capacité de correction t d'un code correcteur vérifie cette relation :

$$t \leq \frac{d-1}{2}$$

Dans la limite de cette capacité de correction, le syndrome Hx d'un mot $x \in \mathbb{F}^n$ est caractéristique de l'erreur, ce qui permet la correction.

Outils informatiques

Pour démarrer le projet, il a fallu développer des modules pour utiliser des matrices et des polynômes sur des corps finis. Pour cela, nous avons utilisé la programmation orientée objet, le typage dynamique et la surcharge des opérateurs permise par Python. Nous avons défini des classes de polynômes et de matrices à coefficients aussi bien réels que de $\mathbb{F}_{p^m}$.

Une optimisation que nous avons trouvé consiste à considérer uniquement le cas $p = 2$. Ainsi, un élément de $\mathbb{F}_{2^m}$ équivalent à une classe d'équivalence de $\mathbb{F}_2[X]/P$ peut être stocké comme un nombre en base binaire. Les opérations sur ces éléments deviennent des opérations binaires beaucoup plus rapides que l'implémentation mathématique.

Algorithmes supplémentaires

Pour compléter le projet, nous avons implémenté les algorithmes de Gauss et de Berlekamp-Hensel. En plus du calcul de l'inverse, le pivot de Gauss nous a servi à déterminer le noyau d'une matrice ou un inverse à gauche d'une matrice non carrée. L'algorithme de Berlekamp-Hensel [Ram08] permet de vérifier l'irréductibilité d'un polynôme dans un corps fini. Étant donné qu'on doit en générer aléatoirement pour chaque clef, son usage était indispensable. Vous le trouverez développé dans l'annexe A.

Codes de Goppa

Définition

[Fin04] Nous avons choisi une définition du code de Goppa à partir de son syndrome. On identifie représentant d'une classe d'équivalence, polynôme et vecteur. Soit g un polynôme dans $\mathbb{F}_{2^m}[X]$ irréductible de degré t. Soit $L = (\alpha_1, .., \alpha_n) \in \mathbb{F}_{2^m}^n$ le support, en pratique nous prendrons $n = 2^m$. Le syndrome d'un mot $z \in \mathbb{F}_2^n$ est défini par :

$$\mathbf{S}_z(x) = \sum_{i=1}^{n} \frac{y_i}{x - \alpha_i} = 0 \ mod \ g(x)$$

Par un calcul donné en annexe B, on construit explicitement une matrice de parité H de ce code de Goppa. Grâce au pivot de Gauss, nous déterminons le noyau de H pour construire une matrice génératrice d'un code de Goppa. La capacité de correction est au moins égale à $\frac{t-1}{2}$.

Décodage

Nous définissons les polynômes localisateur σ_z et évaluateur ω_z d'erreurs pour un mot $z = y + \epsilon$ de $\mathbb{F}^n$ avec y mot de code.

$$\sigma_z(x) = \prod \epsilon_i(x - \alpha_i) \quad (1)$$
$$\omega_z(x) = \sigma_z(x)\mathbf{S}_\epsilon(x) \ mod \ g(x) \quad (2)$$

La connaissance du polynôme localisateur d'erreur σ_z et de ses racines permet la correction de l'erreur. Pour déterminer ces polynômes, nous montrons dans l'annexe C l'unicité d'un couple (σ_z, ω_z) solution de l'équation 2 appelée équation clef, avec des contraintes sur les degrés des polynômes.

Nous exhibons un tel couple solution avec l'algorithme d'Euclide étendu appliqué au couple $(S_{y+\epsilon}, g)$. Pour déterminer les racines du polynôme localisateur d'erreurs, étant donné quelles sont toutes simples, nous avons préféré évaluer ce polynôme de manière exhaustive plutôt que d'utiliser l'algorithme de Berlekamp-Hensel plus lourd.

Implémentation

Bien qu'avec cette définition, la construction de la matrice de parité soit explicite, aucune source ne montre comment former la matrice génératrice. Nous avons donc utilisé le pivot de Gauss pour déterminer Ker(H) et construire une matrice G telle que Im(G) = Ker(H).

Pour ce faire, nous avons utilisé le pivot de Gauss pour transformer avec des échanges de lignes et de colonnes H en H' telle que :

$$H' = \begin{pmatrix} U & V \\ W & Z \end{pmatrix}$$

Avec U carrée inversible de même rang r que H'. Nous avons construit G' la matrice génératrice associée comme :

$$\begin{pmatrix} -U^{-1}V \\ I_{n-r} \end{pmatrix}$$

On obtient bien $H'G' = 0$ avec $rang(G) = dim(Ker(H))$. En faisant cette transformation de H en H', on constate que $H' = LHC$, avec L les changements de lignes et C les changements de colonnes. L et C sont inversibles. On en déduit :

$$H'G' = LHCG' = 0$$
$$G = CG'$$

Mc Eliece

Principe

Le cryptosystème de Mc Eliece créé en 1978 est un des premiers cryptosystèmes asymétriques. Le principe est similaire au RSA au sens où il y a une différence entre la clef publique permettant le cryptage et la clef privée permettant le décryptage. Il est cependant le premier à utiliser des problèmes de la théorie des codes pour assurer la sécurité de l'information.

Pour mettre en place le cryptosystème, on commence par créer un code de Goppa avec les paramètres (g,L) aléatoires. On en fabrique une matrice G génératrice de taille $n \times k$. A cela on rajoute deux matrices aléatoires P permutation $n \times n$ et Q inversible $k \times k$.

Nous fabriquons la clef privée comme l'ensemble (g,L,P,Q,G). Nous publions la clef publique (G' = PGQ, t= capacité de correction).

Soit $x \in \mathbb{F}^k$ le message à transmettre. On crypte ce message avec la clef publique en calculant :

$$y = G'x + \epsilon$$

Avec ϵ erreur dans la limite de correction.

Pour décrypter ce message y, on calcule avec la clef privée :

$$P^{-1}y = GQx + P^{-1}\epsilon$$

Comme P est une matrice de permutation, l'erreur est toujours corrigeable. Nous utilisons notre méthode efficace de décodage pour trouver Qx. Connaissant Q inversible, on accède au message initial.

Sécurité

[Vé95] L'intérêt de ce système de cryptage est qu'il repose sur des problèmes de théorie des codes. A l'heure actuelle, il existe des algorithmes qui sur des ordinateurs quantiques peuvent résoudre efficacement les problèmes de théorie des nombres sur lesquels s'appuie le RSA par exemple. Il n'existe pas encore de tels algorithmes pour la théorie des codes ce qui relance son intérêt malgré la grande taille de ses clefs.

Le premier problème utilisé est celui de Syndrome Decoding. Il consiste en la détermination de l'unique vecteur d'erreur associé à un syndrome donné d'un code linéaire aléatoire. Mc Eliece, Berlekamp et Van Tilborg ont prouvé l'équivalence du problème de décision associé au problème dit des trois mariages qui est NP-complet. Ce problème est donc bien NP-complet.

Le second problème utilisé est celui de l'indistinguabilité d'un code de Goppa aléatoire. C'est à dire qu'il est difficile de distinguer un code de Goppa 'mélangé' PGQ d'un code code linéaire aléatoire. Il n'existe que des algorithmes exponentiels pour effectuer cette tache à ce jour.

Mise en œuvre

Dans notre programme, nous avons défini différentes classes clef_privée et clef_publique. Ces classes contiennent les éléments et les méthodes nécessaires au cryptage ou au décryptage. Nous les avons dotées de méthodes de sauvegarde et de chargement pour faciliter leur diffusion. Notre premier choix a été d'utiliser le module Pickle de Python pour sauvegarder ces objets. Cependant, convertir manuellement des matrices de nombres binaires en liste puis en caractères ASCII s'est avéré plus léger d'un facteur 1000.

Notre programme consiste en un script Python gérant indifféremment le cryptage, le décryptage des messages en prenant en argument les clefs utilisées. Il permet aussi la génération des paires de clef. Cependant, la totalité de ces opérations est contenue dans les classes de clef ce qui permet la réutilisation pratique des ces objets sans notre script d'interfaçage.

Conclusion

A la fin de l'année, nous disposons d'un programme fonctionnel mais trop lent pour répondre aux exigences de sécurité actuelles. Les paramètres conseillés sont (n,k,t) = (1024,524,50), nous n'arrivons en temps raisonnable (300s) qu'à (n,k,t) = (256,100,5). Ces limitations peuvent être dues à la nature interprétée de Python ou au fait que nous n'avons pas utilisé les bibliothèques écrites en C et incluses dans Python. Les opérations de multiplications et d'inversions d'aussi grandes matrices sont rédhibitoires et même l'implémentation de l'algorithme de Strassen pour la multiplication ne suffit pas à atteindre les exigences de rapidité.

A la recherche d'optimisation, nous avons cherché les modules Python adéquats Mais les matrices booléennes pré-implémentées en Python (Numpy) considèrent le 'ou' logique comme loi additive. Néanmoins nous avons mené des tests comparatifs avec Sage comme logiciel externe. La rapidité de cette bibliothèque permet bien de se ramener en temps raisonnable à un niveau de sécurité acceptable. Le code grossièrement modifié est d'ailleurs disponible dans la branche 'Sage' du dépôt.

Le cryptosystème de Mc Eliece est donc un système de chiffrage utilisable en pratique. L'obstacle de la taille des clefs semble s'amoindrir grâce aux débits disponibles de plus en plus importants. De plus, ce système s'appuie sur des problèmes de théorie des codes encore résistants face à un éventuel ordinateur quantique. Ce qui n'est pas le cas du RSA pour lequel on dispose déjà d'algorithmes quantiques permettant de le casser.

Annexes

A Algorithme de Berlekamp-Hensel

[Ram08] Pour générer un code de Goppa aléatoire, il faut un polynôme g de $\mathbb{F}_{2^m}[X]$ de degré t irréductible. Nous avons utilisé l'algorithme de Berlekamp pour tester l'irréductibilité d'un tel 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 1. *Soit $H \in \mathbb{F}_{2^m}[X]$,*

$$H^{2^m} - H = \prod_{c \in \mathbb{F}_{2^m} - \{0\}} (H - c)$$

En effet,

$$\forall c \in \mathbb{F}_{2^m} - \{0\}, H^{2^m} - H = H(H^{2^m-1} - 1) = H(H^{2^m-1} - c^{2^m-1})$$

Une formule connue 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 g unitaire sans facteur multiple.

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

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

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

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

Propriété 3. *Soit $g = g_1 \times ... \times g_r$. D'après le théorème des restes chinois, $H \to (H \ mod(g_1), ...H \ mod(g_r))$ de $\mathbb{F}[X]/g \to (\mathbb{F}[X]/g_1, ..., \mathbb{F}[X]/g_r)$ est un isomorphisme d'anneaux.*

Comme les g_i sont irréductibles, $\{x \in \mathbb{F}[X]/g_1, x = x^{2^m}\}$ est de cardinal 2^m. Donc il existe 2^{mr} polynômes de degrés inférieurs à t-1 vérifiant la propriété désirée. Par suite si g est réductible il existe au moins un polynôme H avec $1 < deg(H) < t$ et $H^{2^m} = H \ mod(g)$. On a donc l'équivalence voulue.

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

$$\begin{array}{rcl} \mathbb{F}_{2^m}[X]/g & \to & \mathbb{F}_{2^m}[X]/g \\ H & \to & H^{2^m} \end{array}$$

On en déduit l'algorithme suivant :

Algorithme 1. *Soit g de degré t dont on veut tester l'irréductibilité :*

1. *Si $pgcd(g, g') \neq 0$ alors g possède des facteurs multiples, à fortiori g est réductible, on arrête l'exécution.*
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 g est irréductible*
5. *Sinon g est réductible*

B Construction de la matrice de parité

[Pan04] 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 1. *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. On notera l'abus de langage consistant à confondre représentant de la classe d'équivalence de* $\mathbf{f}_i$ *et polynôme.*

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

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 $\mathbf{f}_i$ pour construire la matrice de parité. Pour cela nous allons identifier un polynôme comme un vecteur de dimension t.

Définition 2. *On a donc des polynômes* $(\mathbf{f}_i)_{1\leq i\leq n}$ *de degrés t-1 que l'on va identifier comme des vecteurs colonnes de dimension t.*

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

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

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 comme un calcul de noyau.

C Démonstration du décodage

La clef du décodage est la résolution de l'équation clef, c'est à dire la détermination de σ_z. 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 premiers entre eux de degrés $deg(\sigma) \leq \frac{t}{2}$ et $deg(\omega) < \frac{t}{2}$ à un scalaire près. 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.

Références

[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.