Codes correcteurs et cryptosystème de Mc Eliece

Auclair Pierre

15 mai 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

1 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 et la surcharge des opérateurs permise par Python. Nous avions ainsi 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 un polynôme dans $\mathbb{F}_2[X]/P$ peut être stocké comme un nombre en base binaire. Les opérations sur ces éléments deviennent des opérations binaires beaucoup plus rapides que l'implémentation plus 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 carré. 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.

2 Codes de Goppa

Définition

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

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

Par un calcul donné en annexe B, on construit une matrice de parité H de ce code de Goppa. Grâce au pivot de Gauss, nous avons déterminé 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 avons défini les polynômes localisateur σ et évaluateur ω d'erreurs pour un mot $y + \epsilon$ de $\mathbb{F}^n$ avec y mot de code.

$$\sigma(x) = \prod \epsilon_i(x - \alpha_i) \quad (1)$$

$$\omega(x) = \sigma(x)\mathbf{S}_\epsilon(x) \ mod \ g(x) \quad (2)$$

La connaissance du polynôme localisateur d'erreur σ et de ses racines permet la correction de l'erreur. Pour cela, nous montrons dans l'annexe C l'unicité d'un couple (σ, ω) 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

3 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

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 sont 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.

4 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 qu'à (n,k,t) = (256,100,5). Ces limitations peuvent être dus à la nature interprétée de Python ou 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 mutliplication ne suffit pas à atteindre les exigences de rapidité.

A Algorithme de Berlekamp-Hensel

B Construction de la matrice de parité

C Démonstration du décodage

Références

[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.