RECHERCHE EFFICACE DES RACINES DE POLYNÔMES DANS LES CORPS DE CARACTÉRISTIQUE 2

Vincent Herbert (Travail en commun avec Bhaskar Biswas)

C2 2009

INRIA Paris Rocquencourt

Plan

Pourquoi chercher des racines de polynômes ?

Nous rencontrons ce problème en cryptographie basée sur les codes.

En effet, les cryptosystèmes de type McEliece sont souvent basés sur les codes de Goppa binaires.

La recherche de racines est l'étape la plus consommatrice en temps, dans l'implémentation du décodage algébrique des codes de Goppa binaires.

R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. JPL DSN Progress Report, pages 114 - 116, 1978.

Qu'est-ce que le cryptosystème à clef publique de McEliece ?

Donnons un aperçu de la version originale de McEliece.

Clef Publique : Un code linéaire binaire [n,k], c.-à-d. un $\mathbb{F}_2$-sous-espace k-dimensionnel de $\mathbb{F}_2^n$, décrit par une matrice génératrice G.
Clef Privée : Un algorithme de décodage efficace jusqu'à la capacité de correction t du code.

Chiffrement : Transformez le texte clair x de k bits en un mot de code $x.G$, ajoutez e, une erreur uniformément aléatoire de longueur n et de poids t.
Déchiffrement : Corrigez les t erreurs, inversez la transformation pour obtenir le message original. Ce procédé est aussi appelé décodage.

Que sont les codes de Goppa binaires ?

Soient $m > 0$, $n \leq 2^m$ et $a = (a_1, ..., a_n) \in \mathbb{F}_2^n$.

Le code binaire de Goppa $\Gamma(L, g)$, de longeur n, est défini par :

- Support $L = (\alpha_1, ..., \alpha_n)$ n-uplet d'éléments distincts de $\mathbb{F}_{2^m}$;
- Polynôme de Goppa $g(z) \in \mathbb{F}_{2^m}[z]$, sans facteur carré, unitaire de degré $t > 0$ avec aucune racine dans L.

$\Gamma(L, g)$ est un sous-code sur le sous-corps $\mathbb{F}_2$ d'un code de Goppa particulier sur $\mathbb{F}_{2^m}$.

Nous avons $a \in \Gamma(L, g)$ si et seulement si :

$$R_a(z) := \sum_{i=1}^{n} \frac{a_i}{z - \alpha_i} = 0 \text{ sur } \mathbb{F}_{2^m}[z]/(g(z)).$$

Comment décoder les codes de Goppa binaires ?

Soient e, x, y des vecteurs binaires de longueur n. Nous devons trouver x le mot de code envoyé sachant que $y = x + e$ où y est le mot reçu et e le mot erreur. On est capable de corriger jusqu'à t erreurs.

Le décodage algébrique s'effectue en trois temps :

1. Calcul du syndrôme R_y du mot reçu.

$$R_y(z) = R_e(z) = \sum_{i=1}^{n} \frac{e_i}{z - \alpha_i} \text{ sur } \mathbb{F}_{2^m}[z]/(g(z)).$$

2. Résolution de l'équation clef pour obtenir le polynôme localisateur d'erreurs σ_e.

$$R_e(z) \cdot \sigma_e(z) = \sigma'_e(z) \text{ sur } \mathbb{F}_{2^m}[z]/(g(z)).$$

3. Trouver les racines du polynôme localisateur d'erreurs

$$\sigma_e(z) := \prod_{i=1}^{n} (z - \alpha_i)^{e_i} \; ; \; \sigma_e(\alpha_i) = 0 \Leftrightarrow e_i \neq 0.$$

Comment trouver les racines efficacement ?

Plusieurs approches sont possibles, leur efficacité dépend de la taille des paramètres m et t.

- La recherche de Chien calcule les racines en évaluant astucieusement le polynôme en tous les points de L. Cette méthode est recommandée pour des implémentations hardwares et des applications de théorie des codes dans lequel m est petit.
- BTA est un algorithme récursif utilisant les propriétés de la fonction trace. C'est une méthode plus rapide avec les paramètres recommandés dans les cryptosystèmes de type McEliece.

Quel est le coût du déchiffrement ?

Rappelons que, en pratique, $n = 2^m$ et $mt \leq n$.

Complexité théorique = nombre d'opérations binaires requises pour déchiffrer dans le pire cas.

- Calcul du syndrôme $\mathcal{O}(mnt)$
- Résolution de l'équation clef $\mathcal{O}(mt^2)$
- Trouver les racines du polynôme localisateur d'erreurs
  - recherche de Chien $\mathcal{O}(mnt)$
  - Berlekamp Trace Algorithm (abr. BTA) $\mathcal{O}(m^2t^2)$

Complexité expérimentale = temps d'exécution moyen pour le déchiffrement.

Pour les paramètres recommandés (c.-à-d. $m = 11$, $t = 32$), trouver les racines avec BTA (resp. la recherche de Chien) prend 72% (resp. 86%) du temps total de déchiffrement.

Comment BTA fonctionne ?

Fonction Trace $\mathrm{Tr}(\cdot) : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2$

$$\mathrm{Tr}(z) := z + z^2 + z^{2^2} + \ldots + z^{2^{m-1}}.$$

La fonction $\mathrm{Tr}(\cdot)$ est une fonction $\mathbb{F}_2$-linéaire et surjective.

On sait que :

$$\forall i \in \mathbb{F}_2 \ , \ \mathrm{Tr}(z) - i = \prod_{\gamma \ t.q. \ \mathrm{Tr}(\gamma)=i} (z - \gamma).$$

De plus, on a l'égalité : $z^{2^m} - z = \mathrm{Tr}(z) \cdot (\mathrm{Tr}(z) - 1)$.

Comment BTA fonctionne ? (suite)

Soit $B = (\beta_1, \ldots, \beta_m)$ une base de $\mathbb{F}_{2^m}$ sur $\mathbb{F}_2$.

Tout $\alpha \in \mathbb{F}_{2^m}$ est représenté de façon unique par le m-uplet :

$$(\mathrm{Tr}(\beta_1 \cdot \alpha), \ldots, \mathrm{Tr}(\beta_m \cdot \alpha)).$$

BTA scinde tout $f \in \mathbb{F}_{2^m}[z]$ t.q. $f(z) | (z^{2^m} - z)$ en facteurs linéaires en calculant itérativement sur $\beta \in B$ et récursivement sur f :

$$g(z) := \mathrm{pgcd}(f(z), \mathrm{Tr}(\beta \cdot z)) \text{ et } h(z) := \frac{f(z)}{g(z)}.$$

BTA retourne toujours avec succès les facteurs linéaires de f.

Premier appel : $f = \sigma_e$ et $\beta = \beta_1$.

Comment réduire la complexité temporelle ?

L'inconvénient de BTA est le grand nombre d'appels récursifs quand les paramètres du système augmentent.

Nous le réduisons en mélangeant BTA et les algorithmes de Zinoviev qui sont des méthodes ad-hoc pour trouver les racines de polynômes de degré ≤ 10 sur $\mathbb{F}_{2^m}$.

Nous appellerons ce procédé BTZ par la suite.

BTZ dépend d'un paramètre d_{max} qui est le degré maximum jusqu'auquel nous utilisons les méthodes de Zinoviev.

V.A. Zinoviev, On the solution of equations of degree ≤ 10 over finite fields GF(2^m), Rapport de Recherche INRIA n° 2829, 1996

Pseudocode d'une version simplifiée de BTZ

Algorithme 1 - BTZ(f, d, i)

Premier appel : $f \leftarrow \sigma_e$; $d \leftarrow d_{max} \in \{2, \ldots, 10\}$; $i \leftarrow 1$.
si degré$(f) \leq d$ **alors**
  retourner ZINOVIEV(f, d);
sinon
  $g \leftarrow \text{pgcd}(f, \text{Tr}(\beta_i \cdot z))$;
  $h \leftarrow f/g$;
  retourner BTZ$(g, d, i+1) \cup$ BTZ$(h, d, i+1)$;
fin si

Que sont les algorithmes de Zinoviev ?

Les méthodes de Zinoviev trouvent un multiple affine de n'importe quel polynôme de degré ≤ 10 sur $\mathbb{F}_{2^m}$. Les méthodes diffèrent selon le degré.

Polynôme affine

$$A(z) = L(z) + c \text{ où } L \text{ est un polynôme linéarisé, } c \in \mathbb{F}_{q^m}.$$

Polynôme linéarisé

$$L(z) = \sum_{i=0}^{n} l_i \cdot z^{q^i}$$

avec q une puissance d'un nombre premier, $l_i \in \mathbb{F}_{q^m}$ et $l_n = 1$.
Dans notre cas, on prend $q = 2$.

Après quoi, trouver les racines du polynôme affine est plus facile que dans le cas général.

Obtenir un multiple affine d'un polynôme de degré 2 ou 3

Considérons l'équation suivante : $z^2 + \alpha z + \beta = 0, \qquad \alpha, \beta \in \mathbb{F}_{2^m}$.

Remarquez que $z^2 + \alpha z$ est déjà un polynôme linéarisé. Rien à faire ici.

Maintenant considérons l'équation : $z^3 + az^2 + bz + c = 0, \quad a, b, c \in \mathbb{F}_{2^m}$

Nous devons éliminer les termes non-linéaires.

Pour cela, nous ajoutons une racine particulière en multipliant le membre de droite par $(z + a)$.

Nous obtenons $z^4 + dz^2 + ez + f = 0$ avec $d = a^2 + b$, $e = ab + c$, $f = ac$.

Nous obtenons ce que nous voulons, un multiple affine d'un polynôme de degré 3.

Quels résultats obtenons-nous ?

Nous déterminons une formule de récurrence de complexité pour BTZ.

Nous utilisons alors de la programmation dynamique pour estimer sa complexité théorique dans le pire cas.

Nous déterminons ainsi le meilleur d_{max} à utiliser pour avoir l'efficacité optimale sur la gamme suivante de paramètres :

$m = 8, 11, 12, 13, 14, 15, 16, 20, 30, 40$; $t = 10..300$; $d_{max} = 2..10$.

Soit K la fonction coût de n'importe quelle opération arithmétique sur $\mathbb{F}_{2^m}$. Nous prenons $K(+) = 1$; $K(\times) = 1$ ou $K(\times) = m$.

Quels résultats obtenons-nous ? (suite)

Pour $m = 11$, $t = 32$, la théorie recommande $d_{max} = 5$.

Le gain théorique, en terme de nombre d'opérations sur $\mathbb{F}_{2^m}$, de BTZ avec $d_{max} = 5$ par rapport à BTA est 46%, celle vis-à-vis de la méthode de Chien est 93%.

Plus grand est t, plus grand est le d_{max} optimal, selon la théorie.

En pratique, avec $m = 11$, $t = 32$ et $d_{max} = 4$, BTZ prend 60% du temps total de déchiffrement contre 72% pour BTA et 86% pour Chien.

		Chien	BTA	BTZ_4
# cycles CPU requis pour	trouver les racines	3200	1300	800
	déchiffrer un octet	3700	1800	1300

Merci à C2 2009 !

Des questions ou commentaires ?

Vous pouvez me contacter sur :

Vincent.Herbert@inria.fr

Les transparents seront disponibles dans peu de temps sur :

`http://www-roc.inria.fr/secret/Vincent.Herbert/`

Pourquoi est-ce plus facile de trouver les racines d'un polynôme affine ?

Considérons un polynôme affine $A(z) = L(z) + c = \sum_{i=0}^{m-1} l_i \cdot z^{2^i} + c$.

Considérons $(\alpha_1, \cdots, \alpha_m)$, une $\mathbb{F}_2$-base de $\mathbb{F}_{2^m}$, $(l_i)_{1 \leq i \leq m}$, c et x, des élements de $\mathbb{F}_{2^m}$.

Supposons que x est une racine de A.

$$
\begin{aligned}
A(x) = 0 \quad &\Leftrightarrow \quad L(x) = c \\
&\Leftrightarrow \quad \sum_{i=1}^{m} x_i \cdot L(\alpha_i) = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(en utilisant la linéarité de } L\text{)} \\
&\Leftrightarrow \quad \sum_{i=1}^{m} \sum_{j=1}^{m} x_i \; l_{i,j} \cdot \alpha_i = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(système linéaire en } x_i\text{)}
\end{aligned}
$$

Comment fonctionne la recherche de Chien ?

La recherche de Chien est un algorithme récursif. Nous pouvons dire que c'est une recherche exhaustive intelligente.

Soit $f(x) = a_0 + a_1 \cdot x + \cdots + a_t \cdot x^t$ un polynôme sur $\mathbb{F}_{2^m}$ et soit α un générateur du groupe multiplicatif $\mathbb{F}_{2^m}^*$.

$$\begin{aligned} f(\alpha^i) &= a_0 + a_1 \cdot \alpha^i + \cdots + a_t \cdot (\alpha^i)^t \\ f(\alpha^{i+1}) &= a_0 + a_1 \cdot \alpha^{i+1} + \cdots + a_t \cdot (\alpha^{i+1})^t \\ &= a_0 + a_1 \cdot \alpha^i \cdot \alpha + \cdots + a_t \cdot (\alpha^i)^t \cdot \alpha^t \end{aligned}$$

On pose $a_{i,j} = a_j(\alpha^i)^j$. Il est facile d'obtenir $f(\alpha^{i+1})$ à partir de $f(\alpha^i)$ puisque nous avons $a_{i+1,j} = a_{i,j} \cdot \alpha^j$.

De plus, si $\sum_{j=0}^{t} a_{i,j} = 0$, alors α^i est une racine de f.

Seconde description des codes de Goppa binaires

Soient $m > 0$ et $n \leq 2^m$.

Le code binaire de Goppa $\Gamma(L, g)$, de longueur n, est défini par :

- Support $L = (\alpha_1, ..., \alpha_n)$ n-uplet d'éléments distincts de $\mathbb{F}_{2^m}$;
- Polynôme de Goppa $g(z) \in \mathbb{F}_{2^m}[z]$, sans facteur carré, unitaire de degré $t > 0$ sans racine dans L ;

$\Gamma(L, g)$ est un sous-code sur le sous-corps $\mathbb{F}_2$ d'un code de Goppa particulier sur $\mathbb{F}_{2^m}$, corps de caractéristique 2 qui a pour matrice de contrôle de parité H.

$$H := \begin{pmatrix} \frac{1}{g(\alpha_1)} & \frac{\alpha_1}{g(\alpha_1)} & \frac{\alpha_1^{t-1}}{g(\alpha_1)} \\ \frac{1}{g(\alpha_2)} & \frac{\alpha_2}{g(\alpha_2)} & \frac{\alpha_2^{t-1}}{g(\alpha_2)} \\ \vdots & \vdots & \vdots \\ \frac{1}{g(\alpha_n)} & \frac{\alpha_n}{g(\alpha_n)} & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{pmatrix} \in \mathscr{M}_{n,t}(\mathbb{F}_{2^m}).$$

Ainsi, nous avons $a \in \Gamma(L, g)$ si et seulement si $a.H = 0$ et $a \in \mathbb{F}_2^n$.