DES CODES CORRECTEURS POUR SÉCURISER L'INFORMATION NUMÉRIQUE

Vincent Herbert

Inria Paris Rocquencourt

05-12-11

Quel est le sujet de cette thèse ?

Mes travaux portent sur les codes correcteurs d'erreurs.

Ce sont des objets mathématiques ayant des applications informatiques.

Il s'agit d'assurer avec efficacité la sécurité des données numériques.

Sécuriser signifie ici :

- Reconstituer des données altérées (TNT, disque blu-ray, ...)
- Chiffrer des données confidentielles (https, carte bancaire, ...)
- Prouver l'identité d'un individu (carte de métro, ...)

Plan

La correction d'erreurs et les codes cycliques

- Les codes cycliques 3-correcteurs
- L'équivalence de codes
- La distance minimale et l'immunité spectrale

V.H., Sumanta Sarkar - IMACC 2011

Que sont les codes cycliques ?

Soient $m > 0$, q une puissance première et $n \mid q^m - 1$.

Considérons α une racine primitive n-ième de l'unité dans $\mathbb{F}_{q^m}$ et notons $M^{(i)}(X)$, le polynôme minimal de α^i vis-à-vis de $\mathbb{F}_q$.

Un code cyclique de longueur n sur $\mathbb{F}_q$ est défini par :

- Ensemble de définition $Z \subseteq [\![1, n]\!]$.
- Polynôme générateur $g \in \mathbb{F}_q[X]$, $g(X) = \text{ppcm}(\{M^{(z)}(X)\}_{z \in Z})$.

Il s'agit de l'idéal de l'anneau $\mathbb{F}_q[X]/(X^n - 1)$ engendré par g.

Un tel code est dit primitif si $n = q^m - 1$.

Dans notre cas, on prend $n = 2^m - 1$.

Un premier exemple

$\{1, 3, 5\}$ est l'ensemble de définition du code binaire BCH 3-correcteur.

La classe q-cyclotomique de i modulo n est l'ensemble :

$$C_i = \{(iq^j \mod n) \in \mathbb{Z}_n : j \in \mathbb{N}\}.$$

Posons $q = 2$ et $n = 2^4 - 1$.

$$C_1 = \{1, 2, 4, 8\}, \quad C_3 = \{3, 6, 12, 9\}, \quad C_5 = \{5, 10\}.$$

Combien d'erreurs corrige un code cyclique ?

Un code est t-correcteur si sa distance minimale est $2t + 1$.

Considérons des codes cycliques, binaires et primitifs.

Cinq classes de codes 3-correcteurs ont été identifiées en 40 ans.

On ne sait pas calculer efficacement la distance minimale d'un code cyclique.

Les classes connues de codes cycliques 3-correcteurs

Ensemble de définition	Conditions	Année
$\{1, 2^\ell + 1, 2^{3\ell} + 1\}$	$\text{pgcd}(\ell, m) = 1$ m impair	1971
$\{2^\ell + 1, 2^{3\ell} + 1, 2^{5\ell} + 1\}$	$\text{pgcd}(\ell, m) = 1$ m impair	1971
$\{1, 2^{\ell+1} + 1, 2^{\ell+2} + 3\}$	$m = 2\ell + 1$ m impair	2000
$\{1, 2^\ell + 1, 2^{2\ell} + 1\}$	$\text{pgcd}(\ell, m) = 1$ m quelconque	2009
$\{1, 3, 13\}$	m impair	2010

Condition suffisante pour être 3-correcteur

Pour tout m, un code appartenant à la classe

$$\left\{1, 2^\ell + 1, 2^{p\ell} + 1\right\} \quad \text{où} \quad \text{pgcd}(\ell, m) = 1$$

est 3-correcteur si pour tout $\beta \in \mathbb{F}_{2^m}^\times, \gamma \in \mathbb{F}_{2^m}$, l'équation :

$$x^{2^{p\ell}+1} \sum_{i=0}^{p-1} \left(\beta x^{-(2^\ell+1)}\right)^{2^{i\ell}} = \gamma$$

possède au plus 5 solutions en x dans $\mathbb{F}_{2^m}^\times$.

Étude pratique d'une classe de codes cycliques

Il s'agit de la classe des codes cycliques avec l'ensemble de définition :

$$\left\{1, 2^i + 1, 2^j + 1\right\} \quad \text{où} \quad \text{pgcd}(i, m) = 1.$$

Nous savons que leur distance minimale d vérifie :

$$d \in \{5, 7\}$$

et qu'il existe un mot de code de poids $d + 1$.

Nous utilisons l'algorithme de Chose-Joux-Mitton pour chercher un mot de poids 6 dans ces codes.

Pas de nouveau code 3-correcteur de cette forme pour $m < 20$.

Pouvons-nous calculer leur distribution des poids ?

Nous pouvons la calculer directement si le code a une petite dimension.

Le temps de calcul est exponentiel en la dimension.

Les codes cycliques à trois zéros possèdent une dimension $\geq n - 3m$.

Les identités de MacWilliams et Pless relient la distribution des poids d'un code linéaire et de son dual.

La dimension des duaux des codes à trois zéros est inférieure à $3m$.

Comment calculons-nous leur distribution des poids ?

Considérons un code cyclique binaire avec $Z = \{1, a, b\}$.

Nous construisons sa matrice de parité de taille $(3m \times n)$.

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{(n-1)} \\ 1 & \alpha^a & \alpha^{2a} & \dots & \alpha^{(n-1)a} \\ 1 & \alpha^b & \alpha^{2b} & \dots & \alpha^{(n-1)b} \end{pmatrix}$$

Nous générons les mots du dual en utilisant le codage de Gray.

Nous calculons leur poids de Hamming avec une instruction SSE4.

Implantation en Sage et en langage C avec l'API OpenMP.

Quels résultats numériques obtenons-nous ?

Tous les codes cycliques 3-correcteurs avec l'ensemble de définition :

$$\left\{1, 2^i + 1, 2^j + 1\right\} \quad \text{où } i \neq j$$

possèdent la même distribution des poids que le code BCH pour $m < 14$.

Qu'est-ce que l'équivalence de codes ?

Deux codes linéaires binaires sont équivalents s'ils sont égaux à une permutation près de leurs coordonnées.

Comment déterminer l'équivalence de codes ?

Deux codes équivalents ont en commun :

- la longueur
- la dimension
- la distance minimale
- la distribution des poids du code
- la distribution des poids du hull
- etc.

Ces invariants fournissent des conditions nécessaires et non suffisantes pour déterminer l'équivalence entre deux codes.

Les codes étudiés sont auto-orthogonaux.

Quels résultats numériques obtenons-nous ?

Aucun code cyclique 3-correcteur avec l'ensemble de définition :

$$\left\{1, 2^i + 1, 2^j + 1\right\} \quad \text{où } i \neq j$$

n'est équivalent au code BCH pour $m = 7$, $m = 8$ et $m = 10$.

Pour $m = 7$ et $m = 8$, nous utilisons Magma (algorithme de Leon).

Pour $m = 10$, nous nous servons de l'algorithme de séparation du support.

L'invariant utilisé pour déterminer la non-équivalence est le multi-ensemble des distributions des poids des codes poinçonnés.

Un exemple pour mieux comprendre

Soit le code cyclique C avec $Z = \{1, 2^3 + 1, 2^4 + 1\}$.

Le dual d'un code cyclique est un code cyclique.

Nous poinçonnons $C^{\perp}$ et le dual du code BCH en une position quelconque.

Leurs distributions des poids sont identiques pour $m = 9$ et $m = 10$.

Nous poinçonnons les codes une seconde fois en chaque position.

Pour $m = 9$, nous ne pouvons pas conclure. Les multi-ensembles possèdent un unique et même élément.

250 000 distributions des poids à calculer pour aller plus en avant.

Pour $m = 10$, les multi-ensembles possèdent 8 et 10 éléments.

Comment minorer la distance minimale d'un code cyclique ?

En théorie, de nombreux minorants sont connus.

Nombre d'entre eux reposent sur la distribution régulière de certains motifs contenus dans l'ensemble de définition.

- borne BCH (1960)
- borne de Hartmann-Tzeng (1972)
- borne de Roos (1982)
- bornes de van Lint-Wilson (1986)
- etc.

En pratique, les bornes de van Lint-Wilson sont difficilement calculables.

Nous utilisons l'algorithme de Schaub qui adopte une approche différente pour la minoration.

Comment l'algorithme de Schaub fonctionne-t-il ?

Il repose sur le théorème de Blahut.

Soit q une puissance première et α une racine primitive de $\mathbb{F}_{q^m}$.

Le poids d'un mot c d'un code cyclique q-aire de longueur n est égal au rang de la matrice circulante d'ordre n,

$$\begin{pmatrix} A_0 & A_1 & \dots & A_{n-2} & A_{n-1} \\ A_1 & A_2 & \dots & A_{n-1} & A_0 \\ \vdots & \vdots & & \vdots & \vdots \\ A_{n-1} & A_0 & \dots & A_{n-3} & A_{n-2} \end{pmatrix},$$

où $A_i := c(\alpha^i)$.

Comment l'algorithme de Schaub fonctionne-t-il ? (suite)

Un sous-code de C est dit à zéros constants si tous ses mots possèdent exactement les mêmes zéros.

On associe à chacun des sous-codes à zéros constants de C, une matrice circulante définie sur un semi-anneau $\{0, 1, X\}$,

$$\begin{pmatrix} B_0 & B_1 & \dots & B_{n-2} & B_{n-1} \\ B_1 & B_2 & \dots & B_{n-1} & B_0 \\ \vdots & \vdots & & \vdots & \vdots \\ B_{n-1} & B_0 & \dots & B_{n-3} & B_{n-2} \end{pmatrix},$$

où $B_i = 0$ si i est un zéro du sous-code et $B_i = 1$ sinon.

Comment l'algorithme de Schaub fonctionne-t-il ? (fin)

Les codes à zéros constants forment une partition du code C.

On borne inférieurement leur poids minimal en utilisant les lois :

$+$	0	1	X
0	0	1	X
1	1	X	X
X	X	X	X

$*$	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

La valeur minimale obtenue est la borne de Schaub.

Soit κ, le nombre de classes cyclotomiques n'appartenant pas à Z.

- $\#$ sous-codes de C à zéros constants $= 2^{\kappa}$
- Calcul de la borne inférieure sur le rang $\mathcal{O}(n^3)$

Comment optimiser l'algorithme de Schaub ?

Nous représentons les sous-codes à zéros constants de C par un arbre.

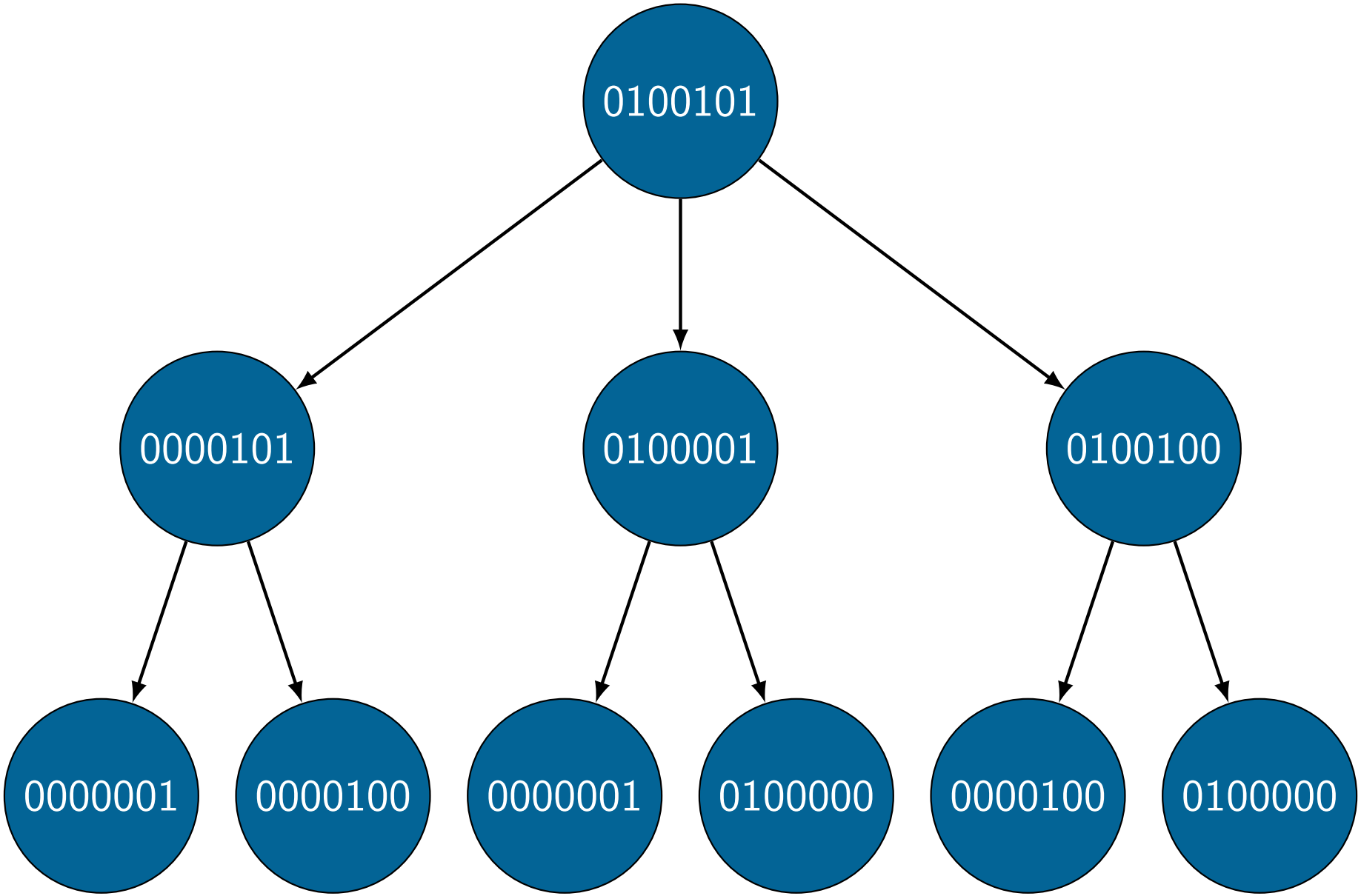
Nous diminuons le nombre de sous-codes traités en identifiant les matrices équivalentes ainsi que la taille des matrices considérées.

Nous élaguons les sous-arbres dont la racine est un nœud où la borne BCH est supérieure à celle de la borne de Schaub calculée.

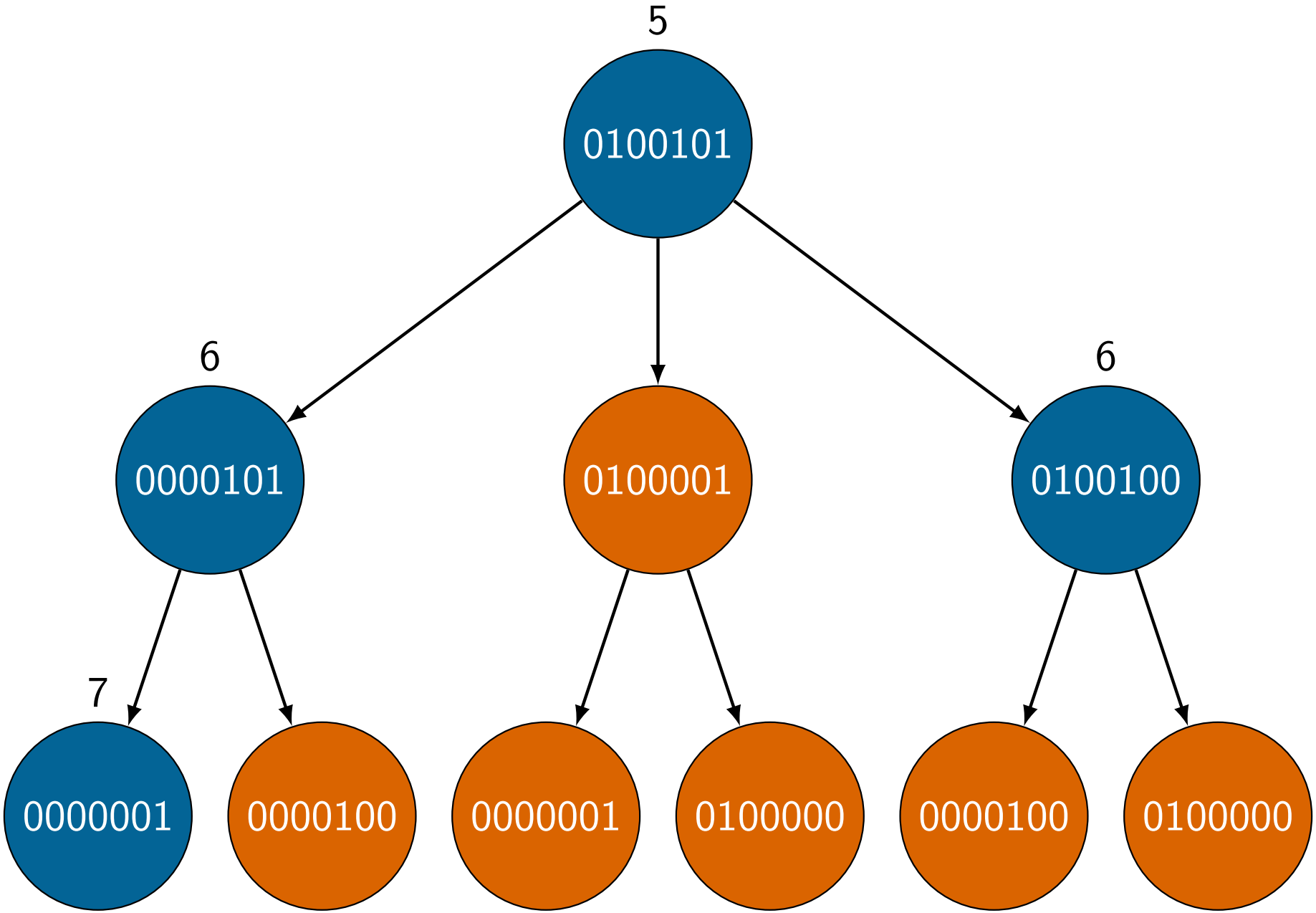
Le temps de calcul est plus long en utilisant la borne de Hartmann-Tzeng.

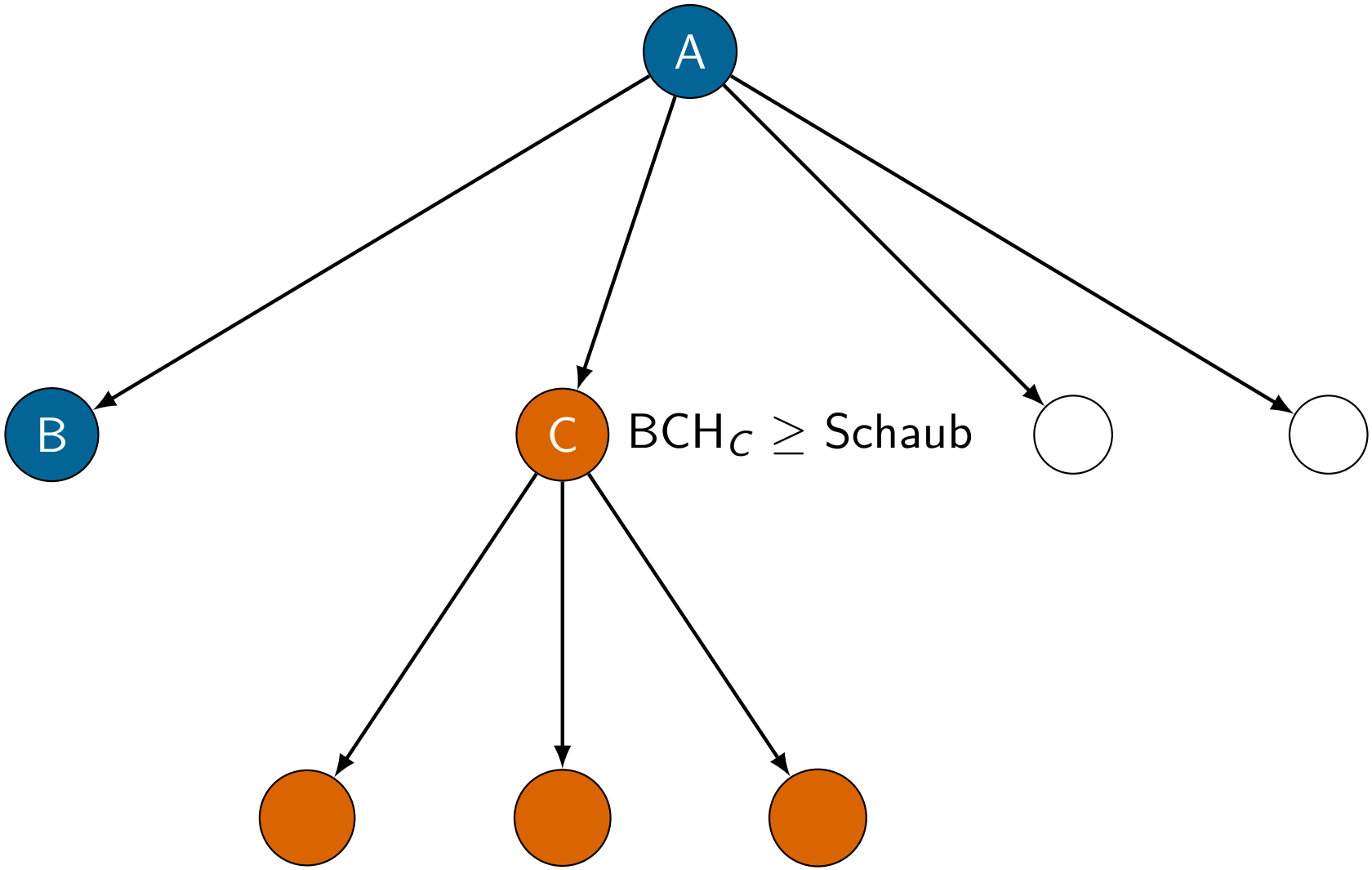
Implanté en langage C++.

$q = 8,\ n = 7,\ Z = \{1, 3, 4, 6\}.$


0100101
0000101
0100001
0100100
0000001
0000100
0000001
0100000
0000100
0100000

$q = 8,\ n = 7,\ Z = \{1, 3, 4, 6\}.$


5
0100101
6
0000101
0100001
6
0100100
7
0000001
0000100
0000001
0100000
0000100
0100000


A
B
C
$BCH_C \geq$ Schaub

L'immunité spectrale et les codes cycliques

Le concept d'immunité spectrale est apparu récemment.

Nous pouvons calculer cette quantité en déterminant la distance minimale de codes cycliques primitifs sur $\mathbb{F}_{2^m}$.

Nous utilisons notre version de l'algorithme de Schaub pour minorer l'immunité spectrale de fonctions booléennes.

Une fonction booléenne qui possède une immunité spectrale faible rend des chiffrements à flot vulnérables à des attaques algébriques.

G. Gong, S. Rønjom, T. Helleseth, and H. Hu. Fast discrete Fourier spectra attacks on stream ciphers. IEEE Transactions on Information Theory, 2011.

L'immunité spectrale et les codes cycliques (suite)

L'immunité spectrale d'une fonction booléenne f sur $\mathbb{F}_{2^m}$ est le poids minimal dans les codes cycliques de longueur $n = 2^m - 1$ sur $\mathbb{F}_{2^m}$ avec les polynômes générateurs :

$$G(z) = \text{pgcd}(f(z), z^n + 1)$$

$$H(z) = \frac{z^n + 1}{G(z)}$$

Tor Helleseth and Sondre Rønjom.
Simplifying algebraic attacks with univariate analysis. ITA 2011

Minoration de l'immunité spectrale

Longueur de code	Ensemble de définition	Borne inférieure immunité spectrale $\mathrm{Tr}(g(z))$
127	$\{1,3,5\}$	11
	$\{1,3,9\}$	13
	$\{1,5,9\}$	12
255	$\{1,3,5\}$	14
	$\{1,5,9\}$	14

- g générateur d'un code cyclique 3-correcteur $Z = \{1, 2^i + 1, 2^j + 1\}$.
- $z \mapsto \mathrm{Tr}(g(z))$ fonction booléenne sur $\mathbb{F}_{2^m}$.
- $G(z) = \mathrm{pgcd}(\mathrm{Tr}(g(z)), z^n + 1), \quad H(z) = \dfrac{z^n + 1}{G(z)}$.
- G et H possèdent des coefficients binaires.

Un exemple et des chiffres

Soit g le polynôme générateur du BCH 3-correcteur.

Longueur de code	Borne inférieure immunité spectrale $\mathrm{Tr}(g(z))$	$\deg(G)$	$\deg(H)$
127	11	56	71
255	14	139	116

Pour $m = 8$.

- Nous calculons la borne de Schaub en 13 heures.
- $2^{20} \simeq$ un million de sous-codes à zéros constants traités.
- Calcul d'une borne sur le rang en $\mathcal{O}(2^{24})$.
- Recherche exhaustive en $\mathcal{O}(2^{116})$.
- Borne de Hartmann-Tzeng $= 9$ vs. Borne de Schaub $= 14$.

Le chiffrement et les codes de Goppa

- La recherche efficace de racines en caractéristique 2

Bhaskar Biswas, V.H. - WEWoRC 2009

Pourquoi cherchons-nous 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 de dimension k 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 : Transformer le texte clair x de k bits en un mot de code $x.G$, ajouter e, une erreur uniformément aléatoire de longueur n et de poids t.
Déchiffrement : Corriger les t erreurs, inverser la transformation pour obtenir le message original.

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 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 $t > 0$ avec aucune racine dans L.

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 syndrome 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 syndrome $\mathcal{O}(mnt)$
- 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-t-il ?

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

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

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

On a l'égalité :

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

Comment BTA fonctionne-t-il ? (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 de degré ≤ 1 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 de f de degré ≤ 1.

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

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

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

Après quoi, trouver les racines du polynôme affine est plus facile que dans le cas général. Il suffit de résoudre un système linéaire.

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

On doit éliminer les termes non-linéaires.

Pour cela, on ajoute une racine particulière en multipliant l'équation par $(z + a)$.

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

On obtient ce que l'on veut, 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 $d_{max} = 5$.

En termes de nombre d'opérations sur $\mathbb{F}_{2^m}$, nous gagnons un facteur 2 vis-à-vis de BTA et un facteur 14 vis-à-vis de la méthode de Chien.

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 par octet requis pour	trouver les racines	3200	1300	800
	déchiffrer	3700	1800	1300

Bilan

Nous avons abordé les thèmes suivants :

- Les codes cycliques 3-correcteurs avec $Z = \{1, 2^i + 1, 2^j + 1\}$
- L'équivalence de codes avec le BCH 3-correcteur
- L'algorithme de Schaub et l'immunité spectrale
- La recherche efficace de racines en caractéristique 2