Codes géométriques, implémentations vectorielles & matrices de diffusion

Pierre Karpman

Inria, France
Nanyang Technological University, Singapour

Séminaire Pampers, Rennes
2014–06–25

Motivations : chiffres par bloc

Chiffre par bloc (définition)

Un chiffre par bloc est une application $\mathrm{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ t.q. $\forall k \in \{0,1\}^\kappa$, $\mathrm{E}(k,\cdot)$ est une permutation de $\{0,1\}^n$

Chiffre par bloc (construction par SPN)

Un chiffre par bloc E est un SPN si sa fonction de tour ρ se définit comme $M \circ \mathrm{S}$, avec S une fonction de *substitution* non-linéaire et M une matrice

Exemples de SPN : AES, PRESENT

Motivations : critères pour la matrice M

On cherche des matrices de « bonne diffusion »
$\Rightarrow$ Notion de *branch number* (Daemen & Rijmen, 2002)

Branch number différentiel et linéaire

Soit M une matrice sur $\mathbf{K}$ et $\mathrm{w}(\mathbf{x})$ le nombre de positions non nulles du vecteur $\mathbf{x}$
Le *branch number différentiel* de M est $\min_{\mathbf{x}\neq 0}(\mathrm{w}(x)+\mathrm{w}(M(x)))$
Le *branch number linéaire* de M est $\min_{\mathbf{x}\neq 0}(\mathrm{w}(x)+\mathrm{w}(M^t(x)))$

$\Rightarrow$ Construction *wide trail* (Ibid.)

Motivations : utilisation de codes linéaires

Distance minimale & branch number

Soit C un code $[2k, k, d]_{\mathbf{K}}$, et $(I_k \ M)$ une matrice de C sous forme systématique. Alors M a un branch number différentiel de d

$\Rightarrow$ Le branch number est maximal pour des matrices issues de codes MDS

Exemple : matrice *MixColumn* de l'AES (sur $\mathbf{F}_{2^8}$) : $\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$

Objectif : utiliser de longs codes pour améliorer la diffusion

Idée : multiplier l'état entier (plutôt que chaque colonne) par une matrice chaque tour
⇒ Diffusion complète chaque tour
⇒ Assure un grand nombre de S-boîtes actives en moyenne

Exemple : SHARK (Rijmen & al., 1996)

Enjeux :

1. Trouver des codes appropriés
   Idée de paramètres : $[32, 16, d]$, avec d proche de 17
2. Trouver des encodeurs efficaces pour ces codes

Objectif : ne pas dépendre d'implémentations par tables

Implémentations par table

Le calcul de $\mathbf{y} = M \cdot \mathbf{x} \Leftrightarrow \mathbf{y}^t = \mathbf{x}^t \cdot M^t$ peut se faire en tabulant les valeurs de $\lambda \cdot (\mathbf{m}^t)^i, \lambda \in \mathbf{K}$, où $(\mathbf{m}^t)^i$ est la ième ligne de M^t

Exemple : matrice de dimension 32 sur $\mathbf{F}_2^{32}$

```
unsigned xt;
unsigned yt;
unsigned mt[32][2];
yt = 0;
for (int i = 0; i < 32; i++){
   yt ⊕= mt[i][xt & 1];
   xt ≫= 1;
}
```

Objectif : ne pas dépendre de tables (cont.)

Problèmes :

- L'accès aux tables se fait en fonction de valeurs secrètes (c-à-d $\mathbf{x}$). $\Rightarrow$ Vulnérabilité faces aux canaux cachés temporels
- Les tables peuvent être relativement grandes, par exemple $16 \times 256 \times 128 = 64$ ko pour une matrice de $\mathscr{M}_{16}(\mathbf{F}_{2^8})$ (plus grand que le premier niveau de cache sur la plupart des processeurs)

Objectif : utiliser des implémentations *vectorielles*

Instructions vectorielles

- Opèrent sur des registres « `xmm` » particuliers, vus comme 1×128, 2×64, 4×32, 8×16 ou 16×8 bits
- Fournissent entre autres l'instruction `pshufb`, qui peut effectuer 16 accès parallèles à une table $4 \to 8$ bits *en temps constant*, ou une permutation quelconque des coefficients de son vecteur d'entrée

$\Rightarrow$ La taille naturelle des données pour `pshufb` est 4 bits

$\Rightarrow$ On voudrait travailler avec des codes sur $\mathbf{F}_{2^4}$

Objectifs : longs codes sur $\mathbf{F}_{2^4}$

- On cherche des codes $[32, 16, d]_{\mathbf{F}_{2^4}}$ avec d maximal
- De part la conjecture MDS, on ne peut pas obtenir de code de longueur $> 2^4 + 1 = 17$
- $\Rightarrow$ Les codes ne seront pas MDS
- On peut cependant chercher des codes *géométriques*, à la fois suffisamment longs et de très bonne distance minimale

Exemple : on utilisera plus loin un code hyperelliptique de paramètres $[32, 16, 15]_{\mathbf{F}_{2^4}}$

Algorithmes vectoriels pour la multiplication matrice-vecteur

On propose deux algorithmes :

1. Un algorithme *générique* indépendant de la structure de la matrice (non décrit en détail ici)
2. Un algorithme plus efficace pour certaines matrices

Les deux algorithmes s'implémentent efficacement avec des instructions vectorielles en assembleur.

Algorithme 2 : Exemple

Soit :

$$\begin{pmatrix} 1 & 0 & 2 & 2 \\ 3 & 1 & 2 & 3 \\ 2 & 3 & 3 & 2 \\ 0 & 2 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Ce produit est égal à :

$$\begin{pmatrix} x_0 \\ x_1 \\ 0 \\ x_3 \end{pmatrix} + 2 \cdot \begin{pmatrix} x_2 \\ x_2 \\ x_0 \\ x_1 \end{pmatrix} + 2 \cdot \begin{pmatrix} x_3 \\ 0 \\ x_3 \\ 0 \end{pmatrix} + 3 \cdot \begin{pmatrix} 0 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix} + 3 \cdot \begin{pmatrix} 0 \\ x_3 \\ x_2 \\ 0 \end{pmatrix}$$

Les multiplications constante-vecteur et les permutations d'éléments se calculent en temps constant avec `pshufb`

Algorithme 2 : principe en plus petit

On écrit le produit :

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

Comme :

$$a \cdot \begin{pmatrix} x_0 \\ 0 \\ 0 \end{pmatrix} + b \cdot \begin{pmatrix} x_1 \\ 0 \\ 0 \end{pmatrix} + c \cdot \begin{pmatrix} x_2 \\ 0 \\ 0 \end{pmatrix} + d \cdot \begin{pmatrix} 0 \\ x_0 \\ 0 \end{pmatrix} + e \cdot \begin{pmatrix} 0 \\ x_1 \\ 0 \end{pmatrix} + f \cdot \begin{pmatrix} 0 \\ x_2 \\ 0 \end{pmatrix} +$$

$$g \cdot \begin{pmatrix} 0 \\ 0 \\ x_0 \end{pmatrix} + h \cdot \begin{pmatrix} 0 \\ 0 \\ x_1 \end{pmatrix} + i \cdot \begin{pmatrix} 0 \\ 0 \\ x_2 \end{pmatrix}$$

Algorithme 2 : principe en plus petit

On regroupe les constantes identiques entre elles :

$$a\cdot\begin{pmatrix}x_0\\0\\0\end{pmatrix}+\ b\cdot\begin{pmatrix}x_1\\0\\0\end{pmatrix}+\ c\cdot\begin{pmatrix}x_2\\0\\0\end{pmatrix}+\ d\cdot\begin{pmatrix}0\\x_0\\0\end{pmatrix}+\ e\cdot\begin{pmatrix}0\\x_1\\0\end{pmatrix}+\ f\cdot\begin{pmatrix}0\\x_2\\0\end{pmatrix}+$$

$$g\cdot\begin{pmatrix}0\\0\\x_0\end{pmatrix}+\ h\cdot\begin{pmatrix}0\\0\\x_1\end{pmatrix}+\ i\cdot\begin{pmatrix}0\\0\\x_2\end{pmatrix}$$

Algorithme 2 : principe en plus petit

On regroupe les constantes identiques entre elles :

$$a\cdot\begin{pmatrix}x_0\\0\\0\end{pmatrix}+\ e\cdot\begin{pmatrix}0\\x_1\\0\end{pmatrix}+\ i\cdot\begin{pmatrix}0\\0\\x_2\end{pmatrix}+\ b\cdot\begin{pmatrix}x_1\\0\\0\end{pmatrix}+\ d\cdot\begin{pmatrix}0\\x_0\\0\end{pmatrix}+\ g\cdot\begin{pmatrix}0\\0\\x_0\end{pmatrix}+$$

$$h\cdot\begin{pmatrix}0\\0\\x_1\end{pmatrix}+\ f\cdot\begin{pmatrix}0\\x_2\\0\end{pmatrix}+\ c\cdot\begin{pmatrix}x_2\\0\\0\end{pmatrix}$$

Algorithme 2 : principe en plus petit

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

est égal à :

$$a \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} + b \cdot \begin{pmatrix} x_1 \\ x_0 \\ x_0 \end{pmatrix} + b \cdot \begin{pmatrix} 0 \\ 0 \\ x_1 \end{pmatrix} + f \cdot \begin{pmatrix} 0 \\ x_2 \\ 0 \end{pmatrix} + c \cdot \begin{pmatrix} x_2 \\ 0 \\ 0 \end{pmatrix}$$

est égal à :

$$a \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} + b \cdot \begin{pmatrix} x_1 \\ x_0 \\ x_1 \end{pmatrix} + b \cdot \begin{pmatrix} 0 \\ 0 \\ x_0 \end{pmatrix} + f \cdot \begin{pmatrix} 0 \\ x_2 \\ 0 \end{pmatrix} + c \cdot \begin{pmatrix} x_2 \\ 0 \\ 0 \end{pmatrix}$$

Algorithme 2 : fonction de coût

On compte le coût d'un calcul par le nombre d'instructions `pshufb`
Celui-ci dépend :

1. Du nombre de constantes > 1 différentes
2. Du nombre total de « permutations »

Les deux critères s'évaluent facilement pour une matrice donnée

1. $\Rightarrow$ Par inspection des coefficients
2. $\Rightarrow$ Pour une constante donnée, c'est égal au max. d'occurrence de cette constante par ligne

Algorithme 2 : matrices de faible coût

Observation

Si toutes les lignes d'une matrice se déduisent par permutation d'une unique ligne, le second critère est minimal parmi des matrices de même densité

- Un cas particulier est celui des matrices circulantes
  - Coût d'environ 30 pour une matrice dense de dim. 16
- De façon générale, on peut espérer un coût faible si les lignes se déduisent d'un petit sous-ensemble

Algorithme 2 : encodeurs systématiques efficaces : roadmap

- On souhaite trouver si possible des matrices de code de la forme $M = (I \ D)$ avec D circulante
- M est *doublement circulante*
- On peut espérer construire de telles matrices en exploitant les automorphismes des codes

Automorphismes & matrices doublement circulante

Groupe d'automorphisme d'un code

Le groupe d'automorphisme $\mathrm{Aut}(\mathscr{C})$ d'un code $\mathscr{C}$ de longueur n est un sous-groupe de $\mathfrak{S}_n$ tel que $\pi \in \mathrm{Aut}(\mathscr{C}) \Rightarrow (c \in \mathscr{C} \Rightarrow \pi(c) \in \mathscr{C})$

Si un automorphisme du code a deux orbites de même taille, on peut espérer construire une matrice doublement circulante en ordonnant les colonnes de la matrice d'après ces orbites

Exemple : codes de Reed-Solomon

Codes de Reed-Solomon

Soit $\mathscr{C}$ un code de Reed-Solomon de paramètres $[n,k,d]_{\mathbf{K}}$, le mot de code $\mathscr{C}(\mathbf{m})$ associé au message $\mathbf{m}$ est défini par le vecteur $(m(p_i), i = 0 \ldots n-1)$ où m est le polynôme $\sum_{i=0\ldots k-1} \mathbf{m}_i \cdot x^i$ de $\mathbf{K}[x]$ de degré au plus $k-1$, et $P = \langle p_0, \ldots, p_{n-1} \rangle$ est un ensemble ordonné de n éléments de $\mathbf{K}$

$\Rightarrow$ On utilise les orbites d'un automorphisme pour définir l'ordre des points P

Exemple : codes de Reed-Solomon (bis)

- $\pi : \mathbf{F}_{2^4} \to \mathbf{F}_{2^4},\ x \mapsto 8x$
- $O_0 = \langle 1, 8, 12, 10, 15 \rangle$ & $O_1 = \langle 2, 3, 11, 7, 13 \rangle$ sont deux orbites
- On construit un code de Reed-Solomon $[10,\ 5,\ 6]_{\mathbf{F}_{2^4}}$ avec $P = \langle 1, 8, 12, 10, 15, 2, 3, 11, 7, 13 \rangle$
- $\Rightarrow$ La matrice systématique correspondante est doublement circulante :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 12 & 10 & 2 & 6 & 3 \\ 0 & 1 & 0 & 0 & 0 & 3 & 12 & 10 & 2 & 6 \\ 0 & 0 & 1 & 0 & 0 & 6 & 3 & 12 & 10 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 6 & 3 & 12 & 10 \\ 0 & 0 & 0 & 0 & 1 & 10 & 2 & 6 & 3 & 12 \end{pmatrix}$$

Une implémentation d'un encodeur systématique par l'algorithme 2 a un coût de 10

Exemple : codes de Reed-Solomon (ter)

- $\pi : \mathbf{F}_{2^4} \to \mathbf{F}_{2^4},\ x \mapsto 7x$
- $O_0 = \langle 1, 7, 6 \rangle,\ O_1 = \langle 2, 14, 12 \rangle,\ O_2 = \langle 4, 15, 11 \rangle$, et $O_3 = \langle 8, 13, 5 \rangle$
- On construit un code de Reed-Solomon $[12,\ 6,\ 7]_{\mathbf{F}_{2^4}}$ avec $P = \langle 1, 7, 6, 2, 14, 12, 4, 15, 11, 8, 13, 5 \rangle$
- $\Rightarrow$ La matrice systématique correspondante se déduit de deux lignes :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 5 & 6 & 11 & 7 & 8 & 11 \\ 0 & 1 & 0 & 0 & 0 & 0 & 11 & 5 & 6 & 11 & 7 & 8 \\ 0 & 0 & 1 & 0 & 0 & 0 & 6 & 11 & 5 & 8 & 11 & 7 \\ 0 & 0 & 0 & 1 & 0 & 0 & 10 & 2 & 1 & 13 & 7 & 15 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 10 & 2 & 15 & 13 & 7 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 10 & 7 & 15 & 13 \end{pmatrix}$$

Une implémentation par l'algorithme 2 a un coût de 20

Application à des codes géométriques

- $\Rightarrow$ Déterminer un code avec de bons paramètres
- $\Rightarrow$ Trouver son groupe d'automorphisme
- $\Rightarrow$ Déduire de bons encodeurs

Espace de Riemann-Roch : définition

Espace de Riemann-Roch (simplifié)

Soit $\mathscr{X}$ une courbe projective lisse de $\mathbf{P}^2(\mathbf{F}_{2^m})$ définie par $p(x,y,z)$, et soit $p'(x,y)$ le déhomogénéisé de p. Soit $\mathbf{F}_{2^m}[\mathscr{X}] = \mathbf{F}_{2^m}[x,y]/p'$ l'*anneau des coordonnées* de $\mathscr{X}$, et $\mathbf{F}_{2^m}(\mathscr{X})$ le *corps de fonction* de $\mathscr{X}$. Supp. que Q est l'unique point à l'infini de $\mathscr{X}$. L'espace de Riemann-Roch $\mathscr{L}(rQ)$ est l'ensemble des fonctions de $\mathbf{F}_{2^m}(\mathscr{X})$ avec des pôles uniquement en Q d'ordre $\leq r$

Théorème de Riemann-Roch

Soit $\mathscr{L}(rQ)$ un espace de Riemann-Roch défini sur une courbe $\mathscr{X}$ de genre g. Alors $\dim(\mathscr{L}(rQ)) \geq r+1-g$, avec égalité quand $r > 2g-2$

Codes géométriques : définition

Code géométrique (simplifié)

Soit $\mathscr{X}$ une courbe projective lisse de $\mathbf{P}^2(\mathbf{F}_{2^m})$ avec un unique point Q à l'infini et $\#\mathscr{X}$ points affine. Supposons que $\#\mathscr{X} \geq n$ et que r est tel que $\dim(\mathscr{L}(rQ)) = k$, et soit $(f_0, \ldots, f_{k-1})$ une base de cet espace. On définit le mot de code associé au message $\mathbf{m}$ du code géométrique $\mathscr{C}$ de paramètres $[n, k, d]_{\mathbf{F}_{2^m}}$ comme le vecteur $\sum_{i=0\ldots k-1}(\mathbf{m}_i \cdot f_i(p_j), j = 0 \ldots n-1)$, où $P = \langle p_0, \ldots, p_{n-1} \rangle$ est un ensemble ordonné de points de $\mathscr{X}/\{Q\}$

Codes géométriques : propriétés

- La longueur maximale d'un code construit sur $\mathscr{X}$ est $\#\mathscr{X}$
- À paramètre n & k fixés, il y a $\binom{\#\mathscr{X}}{n} \cdot n!$ codes *équivalents*
- Soit $\mathscr{C}$ un code de paramètres n & k avec r t.q. $\dim(\mathscr{L}(rQ) = k)$, alors la distance minimale du code est $n - r$
  $\Rightarrow$ Si la courbe est de genre g et que $r > 2g - 2$, c'est égal à $n - k - g + 1$

Codes géométriques : exemple hyperelliptique

- La courbe de genre 2 $\mathscr{X}_H$ définie sur $\mathbf{P}^2(\mathbf{F}_{2^4})$ par $x^5 = y^2z + yz^4$ a 33 points, dont un point à l'infini : $Q = [0:1:0]$
- L'espace $\mathscr{L}(17Q)$ est de dimension $17 + 1 - g = 16$
- $\Rightarrow$ On peut définir un code géométrique de paramètres $[32, 16, 15]_{\mathbf{F}_{2^4}}$
- $\Rightarrow$ D'une matrice systématique $(I_{16}\ D)$, on peut déduire une matrice de diffusion D de branch number 15
- Le code est *auto-dual* et $D \cdot D^t = I_{16}$

Code hyperelliptique : matrice de diffusion

$$\begin{pmatrix}
11 & 6 & 1 & 6 & 10 & 14 & 10 & 9 & 13 & 3 & 3 & 12 & 9 & 15 & 2 & 9 \\
6 & 12 & 0 & 4 & 2 & 8 & 9 & 2 & 5 & 11 & 9 & 5 & 4 & 1 & 15 & 6 \\
9 & 11 & 2 & 2 & 1 & 11 & 13 & 15 & 13 & 3 & 2 & 1 & 14 & 1 & 3 & 10 \\
0 & 0 & 9 & 8 & 11 & 6 & 2 & 1 & 11 & 10 & 15 & 10 & 10 & 15 & 1 & 14 \\
13 & 13 & 3 & 15 & 3 & 1 & 11 & 2 & 9 & 2 & 10 & 14 & 1 & 11 & 1 & 2 \\
1 & 9 & 8 & 4 & 14 & 10 & 2 & 5 & 15 & 2 & 12 & 12 & 9 & 10 & 1 & 9 \\
5 & 9 & 11 & 2 & 15 & 1 & 12 & 4 & 6 & 0 & 6 & 4 & 5 & 8 & 2 & 9 \\
1 & 4 & 14 & 9 & 13 & 2 & 10 & 12 & 0 & 6 & 6 & 9 & 2 & 0 & 11 & 10 \\
13 & 10 & 3 & 9 & 2 & 15 & 6 & 6 & 11 & 1 & 9 & 9 & 12 & 14 & 10 & 3 \\
0 & 10 & 6 & 12 & 11 & 0 & 4 & 9 & 1 & 14 & 10 & 2 & 9 & 2 & 13 & 6 \\
2 & 0 & 5 & 6 & 9 & 0 & 1 & 5 & 15 & 12 & 13 & 15 & 1 & 11 & 13 & 11 \\
11 & 2 & 10 & 1 & 1 & 15 & 0 & 8 & 0 & 9 & 14 & 10 & 10 & 6 & 11 & 15 \\
12 & 14 & 10 & 11 & 3 & 10 & 6 & 0 & 5 & 11 & 1 & 8 & 2 & 9 & 2 & 3 \\
15 & 2 & 2 & 5 & 1 & 10 & 9 & 4 & 1 & 8 & 9 & 9 & 12 & 10 & 14 & 12 \\
15 & 1 & 12 & 5 & 13 & 11 & 0 & 6 & 2 & 5 & 11 & 1 & 15 & 0 & 9 & 13 \\
5 & 6 & 11 & 0 & 2 & 9 & 14 & 11 & 12 & 10 & 3 & 2 & 8 & 10 & 3 & 1
\end{pmatrix}$$

Code hyperelliptique : automorphismes

- On peut construire des automorphismes d'un code géométrique à partir de ceux de la courbe
- $\Rightarrow$ On utilise les automorphismes de $\mathscr{X}_H$ décrits par Duursma (1999) :
  - $\pi_0 : \mathbf{F}_{2^4}^2 \to \mathbf{F}_{2^4}^2,\ (x,y) \mapsto (\zeta x, y)$ avec $\zeta^5 = 1$
  - $\pi_{1_{(a,b)}} : \mathbf{F}_{2^4}^2 \to \mathbf{F}_{2^4}^2,\ (x,y) \mapsto (x+a, y+a^8x^2+a^4x+b^4)$ avec (a,b) un point affine de $\mathscr{X}_H$
  - $\Rightarrow$ groupe d'ordre 160

Code hyperelliptique : orbites des automorphismes

Table : Combinaison possible des tailles d'orbites générées par π_0 and π_1.

Tailles d'orbite	1	2	4	5	10
Type 1	32	0	0	0	0
Type 2	0	16	0	0	0
Type 3	0	0	8	0	0
Type 4	2	0	0	6	0
Type 5	0	1	0	0	3

$\Rightarrow$ On peut espérer utiliser des orbites de taille 8 pour former des matrices de forme $\begin{pmatrix} I_4 & 0_4 & 0_4 & 0_4 & A & B & C & D \\ 0_4 & I_4 & 0_4 & 0_4 & E & F & G & H \\ 0_4 & 0_4 & I_4 & 0_4 & J & K & L & M \\ 0_4 & 0_4 & 0_4 & I_4 & N & O & P & Q \end{pmatrix}$

Mais aucun ordre de points suivant ces orbites ne permet d'obtenir une matrice sous forme systématique ?

Code hyperelliptique : extension des automorphismes

- On étend les automorphismes précédents avec le Frobenius $\theta : \mathbf{F}_{2^4}^2 \to \mathbf{F}_{2^4}^2,\ (x,y) \mapsto (x^2, y^2)$
- $\Rightarrow$ Ajoute 160 nouveaux automorphismes de la courbe
- Ne constituent plus des automorphismes du *code*
- $\Rightarrow$ Mais permettent d'obtenir des orbites de taille plus grande, comme 4 de taille 8
- Exemple : $\sigma = \theta \circ \sigma_2 \circ \sigma_1$ avec :
  $\sigma_1 : (x,y) \mapsto (x+1, y+x^2+x+7)$, $\sigma_2 : (x,y) \mapsto (12x, y)$
  - Seul σ^4 est un automorphisme du code

Code hyperelliptique : résultat

On utilise σ^4 pour construire une matrice $(I_{16}\ D)$ avec D de la forme :

$$(\mathbf{a}^0,\mathbf{a}^1,\mathbf{a}^2,\mathbf{a}^3,\sigma^4(\mathbf{a}^0),\sigma^4(\mathbf{a}^1),\sigma^4(\mathbf{a}^2),\sigma^4(\mathbf{a}^3),\mathbf{a}^8,\mathbf{a}^9,\mathbf{a}^{10},\mathbf{a}^{11},\sigma^4(\mathbf{a}^8),\sigma^4(\mathbf{a}^9),\sigma^4(\mathbf{a}^{10}),\sigma^4(\mathbf{a}^{11}))^t$$

Par exemple :

$$\mathbf{a}^0 = (5,2,1,3,8,5,1,5,12,10,14,6,7,11,4,11)$$
$$\mathbf{a}^4 = \sigma^4(\mathbf{a}^0) = (8,5,1,5,5,2,1,3,7,11,4,11,12,10,14,6).$$

$\Rightarrow$ La matrice se compresse en 8 lignes
$\Rightarrow$ Coût de 52

Matrice compressée : matrice complète

$$\begin{pmatrix}
5 & 2 & 1 & 3 & 8 & 5 & 1 & 5 & 12 & 10 & 14 & 6 & 7 & 11 & 4 & 11 \\
2 & 2 & 4 & 1 & 5 & 12 & 2 & 1 & 9 & 15 & 8 & 11 & 7 & 6 & 9 & 3 \\
1 & 4 & 4 & 3 & 1 & 2 & 15 & 4 & 5 & 13 & 10 & 12 & 9 & 6 & 7 & 13 \\
3 & 1 & 3 & 3 & 5 & 1 & 4 & 10 & 14 & 2 & 14 & 8 & 15 & 13 & 7 & 6 \\
8 & 5 & 1 & 5 & 5 & 2 & 1 & 3 & 7 & 11 & 4 & 11 & 12 & 10 & 14 & 6 \\
5 & 12 & 2 & 1 & 2 & 2 & 4 & 1 & 7 & 6 & 9 & 3 & 9 & 15 & 8 & 11 \\
1 & 2 & 15 & 4 & 1 & 4 & 4 & 3 & 9 & 6 & 7 & 13 & 5 & 13 & 10 & 12 \\
5 & 1 & 4 & 10 & 3 & 1 & 3 & 3 & 15 & 13 & 7 & 6 & 14 & 2 & 14 & 8 \\
12 & 9 & 5 & 14 & 7 & 7 & 9 & 15 & 7 & 6 & 11 & 3 & 15 & 5 & 13 & 7 \\
10 & 15 & 13 & 2 & 11 & 6 & 6 & 13 & 6 & 6 & 7 & 9 & 5 & 10 & 2 & 14 \\
14 & 8 & 10 & 14 & 4 & 9 & 7 & 7 & 11 & 7 & 7 & 6 & 13 & 2 & 8 & 4 \\
6 & 11 & 12 & 8 & 11 & 3 & 13 & 6 & 3 & 9 & 6 & 6 & 7 & 14 & 4 & 12 \\
7 & 7 & 9 & 15 & 12 & 9 & 5 & 14 & 15 & 5 & 13 & 7 & 7 & 6 & 11 & 3 \\
11 & 6 & 6 & 13 & 10 & 15 & 13 & 2 & 5 & 10 & 2 & 14 & 6 & 6 & 7 & 9 \\
4 & 9 & 7 & 7 & 14 & 8 & 10 & 14 & 13 & 2 & 8 & 4 & 11 & 7 & 7 & 6 \\
11 & 3 & 13 & 6 & 6 & 11 & 12 & 8 & 7 & 14 & 4 & 12 & 3 & 9 & 6 & 6
\end{pmatrix}$$

Code hyperelliptique : encodeurs aléatoires

- On peut aussi chercher aléatoirement de « bons » ordres de points
- $\Rightarrow$ Espace de taille $32! \approx 2^{117,7}$
- On trouve rapidement des encodeurs de taille 43 $\Rightarrow$

coût	#mat.	#mat. (cumul)	proportion (cumul)
43	146 482	146 482	0.00000053
44	73 220	219 702	0.00000080
45	218 542	438 244	0.0000016
46	879 557	1 317 801	0.0000048
47	1 978 159	3 295 960	0.000012
48	5 559 814	8 855 774	0.000032
49	21 512 707	30 368 481	0.00011
50	93 289 020	123 657 501	0.00045
51	356 848 829	480 506 330	0.0017
52	1 282 233 658	1 762 739 988	0.0064
53	3 534 412 567	5 297 152 555	0.019
54	8 141 274 412	13 438 426 967	0.049
55	15 433 896 914	28 872 323 881	0.11
56	24 837 735 898	53 710 059 779	0.20
57	33 794 051 687	87 504 111 466	0.32
58	38 971 338 149	126 475 449 615	0.46
59	38 629 339 524	165 104 789 139	0.60

Applications : chiffres par blocs hypothétiques : 64 bit

- On sait définir des matrices de diffusion de $\mathscr{M}_{16}(\mathbf{F}_{2^4})$ de branch number linéaire et différentiel de 15
- Combien de tours nécessaires pour un chiffre avec une structure à la SHARK ?
- 15 S-boîtes de 4-bit actives tous les 2 tours
- Les meilleures S-boîtes de 4 bit ont pdm & bm $\leq 2^{-2}$
- $\Rightarrow$ Les meilleurs *chemins* sur $2n$ tours ont probabilité $2^{-2 \cdot 15 n}$
- $\Rightarrow$ 6 ou 8 tours semblent suffisant

Applications : chiffres par blocs hypothétiques : 128 bit

- On peut utiliser la même matrice dans une extension de $\mathbf{F}_{2^4}$, par ex. $\mathbf{F}_{2^8} \cong \mathbf{F}_{2^4}[t]/p(t)$
- $\Rightarrow$ Même branch number
- Le calcul reste aussi efficace que sur $\mathbf{F}_{2^4}$: $\alpha \cdot (at + b) = (\alpha at + \alpha b)$ $\Rightarrow$ deux multiplications dans $\mathbf{F}_{2^4}$ pour une valeur de taille double
- Les meilleures S-boîtes de 8 bit connues ont pdm & bm $\leq 2^{-6}$
- $\Rightarrow$ Les meilleurs *chemins* sur $2n$ tours ont probabilité $2^{-6 \cdot 15n}$
- $\Rightarrow$ 6 ou 8 tours semblent toujours suffisant

Applications : performances

Table : Performance d'implémentations vectorielles de chiffres hypothétiques de 64 & 128 bits en cycles par octet

		64 bit		128 bit	
Processeur	#tours	Alg. 1	Alg. 2	Alg. 1	Alg. 2
Xeon E5-2650	6	50 (45.5)	33 (24.2)	58 (52.3)	32.7 (26.5)
	8	66.5 (60.2)	44.5 (31.9)	76.8 (69.6)	43.8 (35.7)
Xeon E5-2609	6	72.3 (63.7)	45.3 (33.2)	79.8 (75.6)	47.1 (36.8)
	8	95.3 (84.7)	63.3 (45.6)	106.6 (97.1)	62.1 (50.3)
Xeon E5649	6	84.7	46	84.5	47
	8	111.3	59.8	111	61.9

Conclusion

- On a présenté une application de codes géométriques à la conception de matrices de diffusion
- On a introduit un algorithme vectoriel efficace pour l'implémentation de ces matrices
- On a exploité (un peu) la structure algébrique des codes et (surtout) une recherche aléatoire pour trouver de bons encodeurs
- $\Rightarrow$ On peut obtenir des chiffres avec de bonnes performance et facile à analyser
- $\Rightarrow$ On peut espérer trouver d'encore meilleurs encodeurs en utilisant l'ensemble du groupe d'automorphisme ? ? ?