Index

1. Codes de Reed-Solomon

1.1 Définition

Les codes de Reed-Solomon RS(k,t) sont formés de n symboles, avec n = q - 1 au maximum et q = 2^k, chaque symbole appartenant à GF(q) qui est le corps de Galois (Galois Field) à q éléments, k représente donc le nombre de bits par symbole. Le nombre t représente le nombre de symboles d'erreurs que ce code sera capable de corriger.

1.2 Information

Le nombre de symboles de contrôle est de 2t. Par conséquent, le nombre de symboles d'information que l'on peut transmettre est de m = n - 2t.

1.3 Exemple

Soit le code RS(4,3) formé de n = 15 symboles, chaque symbole étant contenu dans GF(16), donc 4 bits par symbole. Soit sous forme polynômiale :

$$w(x) = w_{14} . x^{14} + w_{13} . x^{13} + ... + w_1 . x^1 + w_0$$

avec w_i contenu dans GF(16) pour i = 0,..,14.

Les coefficients $w_0, w_1, \dots, w_5$ sont les symboles du contrôle de parité (parity check), et les coefficients $w_6, w_7, \dots, w_{14}$ représente les symboles d'information à transmettre.

2. Corps de Galois

La notation pour la suite, sera la suivante :

- F représente un corps fini (Field).
- F_p représente le corps fini à p éléments.
- F^* représente le groupe multiplicatif du corps F.

2.1 Propriétés d'un corps fini

On rappelle ici, quelques propriétés utiles des corps finis. Soit F un corps fini de caractéristique p ayant q éléments. Alors,

a) F est un F_p-espace vectoriel de dimension k, avec $q = p^k$.
b) Le groupe additif de F est isomorphe au groupe $(Z/pZ,+)^k$.
c) F^* est cyclique d'ordre $q - 1 = p^k - 1$.
d) Tout élément x de F^* vérifie $x^{q-1} = 1$.

Commentaire : Le point c) nous affirme qu'il existe un élément α, dit primitif, qui engendre le groupe multiplicatif F^*. Le point a) nous dit que F est un espace vectoriel de dimension k sur le corps F_p, le choix d'une base sera généralement les puissance entière de l'élément primitif, soit $1, \alpha$, α^2, α^3, ..., α^{k-1}. Un élément c de F s'écrira alors de la manière suivante :

$c = c_{k-1} . \alpha^{k-1} + \ ... + c_1 . \alpha^1 + c_0 . 1$
avec c_i appartenant à F_p pour $i = 0,..,k-1$.

2.2 Construction d'un corps fini

La construction d'un corps fini est basée sur le théorème suivant :

Théorème : Soit α un nombre algébrique sur F. Soit k le degré de son polynôme irréductible sur F. L'espace vectoriel engendré sur F par $1, \alpha, \alpha^2, \ldots, \alpha^{k-1}$ est alors un corps, et la dimension de cet espace vectoriel est k.

Ce théorème nous donne une méthode pour construire un corps avec une base donnée, mais ne nous donne pas de méthode pour trouver un élément primitif, α n'est pas forcément un élément primitif.

2.2.1 Représentation des éléments

Il existe principalement deux méthodes pour représenter les éléments d'un corps fini, soit :

i) F_q est un F_p-espace vectoriel de dimension k, on choisit une certaine base $\{b_1,\ldots,b_k\}$ de cet espace et on repère chaque élément de F_q par ses composantes sur cette base.

ii) On prend un élément α de F^* qui engendre ce groupe et tout élément non nul de F_q s'écrit d'une manière, et d'une seule, sous la forme α^m, avec $m = 0,\ldots,q-1$.

Pour décrire un élément du corps, on utilisera exclusivement la première représentation, car elle permet d'associer un nombre à chaque élément du corps. Mais on prendra soin de tabuler ces deux représentations, la raison en est que la représentation i) se prête très bien pour l'addition et que la représentation ii) est bien adaptée pour la multiplication. Comme on a besoin de ces deux opérations, on travaillera avec ces deux représentations qui seront tabulées une fois pour toute.

Avertissement : On ne s'intéressera, par la suite, que de corps finis de caractéristique 2, qui sont les seuls utilisés pour les codes de Reed-Solomon.

2.2.2 Exemple

Soit $F_2 = \{0,1\}$ le corps à deux éléments. On vérifie que le polynôme $P(x) = x^2 + x + 1$ est bien irréductible sur F_2. Soit α une racine de P(x), on aura alors

$$P(\alpha) = \alpha^2 + \alpha + 1 = 0 \quad \text{donc} \quad \alpha^2 = \alpha + 1$$

ce qui veut dire que l'espace vectoriel engendré sur F_2 par $1,\alpha$ est alors un corps, et la dimension de cet espace vectoriel est de 2. Le corps à 4 éléments sera alors $F_4 = \{0,1,\alpha , \alpha + 1\}$. Si l'on choisit $\{1,\alpha\}$ comme base, la représentation i) sera la suivante :

$$\begin{array}{ll}
0 & = 0.\alpha + 0.1 = (0,0) = 0 \\
1 & = 0.\alpha + 1.1 = (0,1) = 1 \\
\alpha & = 1.\alpha + 0.1 = (1,0) = 2 \\
\alpha + 1 & = 1.\alpha + 1.1 = (1,1) = 3
\end{array}$$

Cette représentation est très pratique, car elle permet d'associer un nombre à chaque élément.
Si l'on avait choisit $\{1,\alpha + 1\}$ comme base, on aurait alors la représentation suivante :

$$\begin{array}{ll}
0 & = 0.(\alpha + 1) + 0.1 = (0,0) = 0 \\
1 & = 0.(\alpha + 1) + 1.1 = (0,1) = 1 \\
\alpha + 1 & = 1.(\alpha + 1) + 0.1 = (1,0) = 2 \\
\alpha & = 1.(\alpha + 1) + 1.1 = (1,1) = 3
\end{array}$$

On comprend facilement que le choix de la base est fondamental pour savoir de quoi l'on parle.

Pour effectuer une addition sur ce corps, il faut voir que l'on additionne en fait deux vecteurs, et donc que l'on effectue l'addition composante par composante, et ceci sur F_2. Soit par exemple,

$$
\begin{aligned}
(0,1) + (0,1) &= (0 + 0, 1 + 1) = (0,0) &\rightarrow\quad 1 + 1 = 0\\
(1,1) + (1,0) &= (1 + 1, 1 + 0) = (0,1) &\rightarrow\quad 3 + 2 = 1\\
(1,1) + (0,1) &= (1 + 0, 1 + 1) = (1,0) &\rightarrow\quad 3 + 1 = 2\\
(0,1) + (1,0) &= (0 + 1, 1 + 0) = (1,1) &\rightarrow\quad 1 + 2 = 3
\end{aligned}
$$

Sur un corps de caractéristique 2, l'addition revient à faire un **OU EXCLUSIF (notation ^)** entre deux nombres. Soit par exemple,

3 + 2 = 11 ^ 10 = 01 = 1
3 + 1 = 11 ^ 01 = 10 = 2

Cela nous donne une opération assez simple à réaliser et à implémenter dans un programme.

Pour la multiplication, il nous faut trouver un élément primitif qui génère le groupe F_4^*. Dans notre exemple, α est un élément primitif, en effet

$$
\begin{aligned}
\alpha^0 &= 1\\
\alpha^1 &= \alpha\\
\alpha^2 &= \alpha + 1
\end{aligned}
$$

ce qui nous donne les trois éléments de notre groupe.

Si l'on prend $\{1,\alpha\}$ comme base, on aura alors la table de multiplication suivante :

$1 * 1 = 1$
$1 * 2 = 1 * \alpha = \alpha = 2$
$1 * 3 = 1 * (\alpha + 1) = \alpha + 1 = 3$
$2 * 1 = 1 * 2 = 2$
$2 * 2 = \alpha * \alpha = \alpha^2 = \alpha + 1 = 3$
$2 * 3 = \alpha * (\alpha + 1) = \alpha^2 + \alpha = \alpha + 1 + \alpha = 1$
$3 * 1 = 1 * 3 = 3$
$3 * 2 = 2 * 3 = 1$
$3 * 3 = (\alpha + 1) * (\alpha + 1) = \alpha^2 + \alpha + \alpha + 1 = \alpha^2 + 1 = \alpha = 2$

ce qui termine notre exemple.

3. Technique de codage

Considérons un code de Reed-Solomon avec ses symboles dans $GF(2^k)$, où k est le nombre de bits par symbole. Soit

$$i(x) = c_{m-1} . x^{m-1} + ... + c_1 . x^1 + c_0$$

le polynôme d'information, et soit

$$p(x) = c_{2t-1} . x^{2t-1} + ... + c_1 . x^1 + c_0$$

le polynôme de contrôle, le code de Reed-Solomon sous sa forme polynômiale sera alors

$$c(x) = i(x) . x^{2t} + p(x) = c_{n-1} . x^{n-1} + ... + c_1 . x^1 + c_0$$

avec c_i appartenant à $GF(2^k)$ pour i = 0,..,n-1.

Un vecteur à n symboles, $(c_{n-1}, ..., c_1, c_0)$ est un code de Reed-Solomon si et seulement si son polynôme correspondant c(x) est un multiple du polynôme générateur g(x). La méthode courante pour construire un tel polynôme, est de diviser $i(x) . x^{2t}$ par g(x) et d'additionner le reste à c(x). En effet,

$$i(x) . x^{2t} = q(x) . g(x) + r(x)$$

où r(x) est le reste de la division de i(x) par g(x),

$$c(x) = i(x) . x^{2t} + p(x) = q(x) . g(x) + r(x) + p(x) = q(x) . g(x)$$

pour que c(x) soit un multiple de g(x), soit c(x) = q(x) . g(x), il faut que p(x) = r(x). Comme on travaille toujours sur des corps de caractéristique 2, l'opération de soustraction sera toujours égale à l'opération d'addition, soit de manière algébrique -1 = +1.

Cela nous donne une méthode pour construire le polynôme de contrôle, il suffit de prendre le reste de la division du polynôme $i(x) . x^{2t}$ par $g(x)$.

Il nous reste encore à construire le polynôme générateur g(x). Il est définit de la manière suivante :

$$g(x) = (x + \alpha)(x + \alpha^2)...(x + \alpha^{2t}) = x^{2t} + g_{2t-1} . x^{2t-1} + ... + g_1 . x^1 + g_0$$

où les coefficients g_i appartiennent à $GF(2^k)$, et α est un élément primitif de $GF(2^k)$.

3.1 Exemple de codage

Soit le code RS(4,3), avec n = 15, et soit un polynôme P(x) irréductible sur F2, avec $P(x) = x^4 + x^3 + 1$. Soit α une racine de P(x), qui est également un élément primitif, et sera utilisé pour construire GF(16). La base utilisée sera alors $\{\alpha^3,\alpha^2,\alpha,1\}$ = {8,4,2,1}, il y aura donc 16 symboles, qui vont de 0 à 15. Maintenant, on désire coder les 9 symboles d'informations suivants :

i = [9, 8, 7, 6, 5, 4, 3, 2, 1]   (matrice 1 x 9)

soit sous forme polynômiale :

$$i(x) = 9x^8 + 8x^7 + 7x^6 + 6x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1$$

$$\begin{aligned} g(x) &= (x + 2)(x + 4)(x + 8)(x + 9)(x + 11)(x + 15) \\ &= x^6 + 3x^5 + x^4 + 4x^3 + 7x^2 + 13x + 15 \end{aligned}$$

$$\begin{aligned} c(x) &= q(x) \,.\, g(x) \\ &= (9x^8 + 10x^7 + 9x^6 + x^5 + x^4 + 12x^3 + 3x^2 + 11x + 4) \,.\, g(x) \\ &= i(x) \,.\, x^{2t} + r(x) \\ &= i(x) \,.\, x^{2t} + 6x^5 + 15x^4 + 15x^3 + 15x^2 + 11x + 14 \end{aligned}$$

4. Technique de décodage

Considérons un code de Reed-Solomon c(x) correspondant au code transmis, et soit d(x) le code que l'on reçoit. Le code d'erreur est définit par

$$e(x) = d(x) - c(x) = d(x) + c(x)$$

car - et + sont équivalent dans $GF(2^k)$.

La première opération a faire, est de calculer les syndromes, qui sont définis de la manière suivante :

$$S_i = e(\alpha^i) = d(\alpha^i) + c(\alpha^i) = d(\alpha^i) \quad \text{avec} \quad i = 1, ..., 2t$$

car $c(\alpha^i) = q(\alpha^i) . g(\alpha^i) = 0$ puisque $g(\alpha^i) = 0$ pour $i = 1, ..., 2t$ par définition de g(x). Le syndrome sous forme polynômiale sera

$$s(x) = S_{2t} . x^{2t-1} + ... + S_3 . x^2 + S_2 . x + S_1$$

Soit M l'ensemble des positions d'erreurs, on définit le polynôme de localisation (locator polynomial) par

$$l(x) = (1 - \alpha^{p1}.x)(1 - \alpha^{p2}.x) ... (1 - \alpha^{pj}.x) \quad \text{avec} \quad p1, ..., pj \text{ dans } M$$

Si $b = \alpha^i$ est une racine de l(x), donc l(b) = 0, on aura alors un des monôme qui s'annule, soit $(1 - \alpha^p.b) = 0$. Ce qui nous donne la relation suivante $b = \alpha^{-p} = \alpha^{q-1}.\alpha^{-p} = \alpha^{q-p-1} = \alpha^i$ ($\alpha^{q-1} = 1$), la position de l'erreur sera alors donné par $p = q-1-i$ qui est positif et que l'on connait, puisque l'on connait $b = \alpha^i$, racine de l(x). Il nous faudra donc chercher toutes les racines de l(x). Il ne faut pas oublier que α est un élément primitif, et que tout élément de $GF(2^k)^*$ s'exprime comme une puissance de l'élément primitif, qui est un générateur du groupe $GF(2^k)^*$.

4.1 Equation fondamentale

Il nous reste encore à calculer l(x). Pour cela, on fait usage de la relation fondamentale suivante :

$$\mathbf{l(x) \;.\; s(x) \;=\; w(x) \;+\; u(x) \;.\; x^{2t}}$$

Comme l'on travaille dans GF(2^k), on peut l'écrire sous la forme

$$\mathbf{l(x) \;.\; s(x) \;+\; u(x) \;.\; x^{2t} \;=\; w(x)}$$

On peut voir cette équation comme la relation de Bezout avec des nombres, soit

$$v \,.\, b + u \,.\, a = w$$

où a et b sont des nombres donnés, et w est le plus grand diviseur commun des nombres a et b. u et v sont deux nombres qui satisfont cette relation. Notre équation ci-dessus est identique, sauf qu'elle s'applique à des polynôme.
On pose b = s(x), a = x^{2t} et on applique l'algorithme d'Euclide pour trouver u(x),l(x),w(x), et les racines de l(x) sont utilisées pour trouver la position de l'erreur. Soit b_i une racine de l(x), la valeur de l'erreur à la position p_i se calcule de la manière suivante :

$e_i = w(b_i) \,/\, l'(b_i)$ où l'(x) est la dérivée du polynôme l(x).

En fait, l'algorithme d'Euclide ne donne pas l(x),v(x),w(x) mais K.l(x), K.v(x), K.w(x), c'est-à-dire à une constante K près. Mais pour trouver les racines de l(x), qu'on aie l(x) ou K.l(x) cela nous donne le même résultat, et comme la valeur de l'erreur est calculée par le rapport $K.w(b_i) \,/\, K.l'(b_i)$, on obtient le bon résultat.

4.2 Algorithme d'Euclide

On rappelle ici, le principe de l'algorithme d'Euclide appliqué à l'équation v . b + u . a = w.

I) Initialisation

$q_0 = -$	$r_0 = a$	$u_0 = 1$	$v_0 = 0$
$q_1 = -$	$r_1 = b$	$u_1 = 0$	$v_1 = 1$

II) Calcul de q

On divise r_{i-1} par r_i, ce qui nous donne un quotient q et un reste r, soit $r_{i-1} = q \cdot r_i + r$ et on pose $q_{i+1} = q$.

III) Calculs

$$r_{i+1} = r_{i-1} - q_{i+1} \cdot r_i$$
$$u_{i+1} = u_{i-1} - q_{i+1} \cdot u_i$$
$$v_{i+1} = v_{i-1} - q_{i+1} \cdot v_i$$

IV) Test

Si r_{i+1} est différent de 0, on incrémente i et on retourne au point II, et si $r_{i+1} = 0$ l'agorithme est terminé.

On a alors,

a) r_i est le plus grand diviseur commun entre a et b.
b) $r_i = v_i \cdot b + u_i \cdot a$.

En fait, la relation b) est valable pour tout i.

4.3 Reed-Solomon algorithme

L'algorithme de décodage des codes de Reed-Solomon sera le suivant :

1) Calcul du polynôme syndrome $s(x)$.

Si $s(x) = 0$, il n'y a pas d'erreurs : **STOP**.

2) On applique l'algorithme d'Euclide avec

$a(x) = x^{2t}$ et $b(x) = s(x)$.

L'algorithme se termine dès que le degré de $r_i(x)$ est $< t$.
Si $r_i(x) = 0$, il y a plus que t erreurs : **STOP**.

3) On pose $L(x) = K \cdot l(x) = v_i(x)$.
On cherche toutes les racines de $L(x)$: $b_1, ..., b_r$.

4) Pour chaque racine $b_i = \alpha^j$, la position p_i de l'erreur est

$p_i = q - j - 1 = 2^k - j - 1$.

5) On pose $W(x) = K \cdot w(x) = r_i(x)$.
La valeur de l'erreur à la position p_i est donnée par

$e_i = W(b_i) / L'(b_i) = K \cdot w(b_i) / K \cdot l'(b_i) = w(b_i) / l'(b_i)$.

La nouvelle valeur à la position p_i sera alors $c_i = d_i + e_i$.

On évalue toujours un polynôme ainsi que sa dérivée à l'aide d'un schéma d'Horner, et ceci afin de minimiser le nombre de calculs.

4.4 Schéma d'Horner

On donne ici, le schéma d'Horner pour évaluer un polynôme et sa dérivée à une valeur donnée.

Soit $p(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0$, on désire évaluer notre polynôme et sa dérivée à $x = \beta$, soit $p(\beta)$ et $p'(\beta)$.

I) Initialisation

$b_n = a_n$
$a_n = b_n$
$k = n$

II) Itération

$b_k = a_k + \beta \cdot b_{k+1}$
si $k > 0$ alors $c_k = b_k + \beta \cdot c_{k+1}$

III) Contrôle

Si k est différent de 0, on décrémente k et on retourne au point II, sinon l'agorithme est terminé.

On aura alors $p(\beta) = b_0$ et $p'(\beta) = c_1$.

4.5 Exemple de décodage

Reprenons notre exemple 3.1 ci-dessus, soit notre code de Reed-Solomon :

$$c(x) = 9x^{14} + 8x^{13} + 7x^{12} + 6x^{11} + 5x^{10} + 4x^{9} + 3x^{8} + 2x^{7} + 1x^{6} + 6x^{5} + 15x^{4} + 15x^{3} + 15x^{2} + 11x + 14$$

Choisissons un polynôme d'erreur, soit

$$e(x) = 7x^{11} + 10x^{2}$$

Notre code reçu sera alors,

$$d(x) = c(x) + e(x)$$
$$d(x) = 9x^{14} + 8x^{13} + 7x^{12} + 1x^{11} + 5x^{10} + 4x^{9} + 3x^{8} + 2x^{7} + 1x^{6} + 6x^{5} + 15x^{4} + 15x^{3} + 5x^{2} + 11x + 14$$

On calcule les syndromes, soit $S_i = d(\alpha^i) = d(2^i)$ avec $i = 1, ..., 6$, ce qui nous donne le syndrome sous forme polynômiale :

$$s(x) = 1x^{5} + 15x^{4} + 7x^{3} + 8x^{2} + 11$$

On pose $a(x) = x^6$ et $b(x) = s(x)$, et on applique l'algorithme d'Euclide, cela nous donne :

$$L(x) = 6x^{2} + 9x + 1$$
$$W(x) = 5x + 11$$
$$U(x) = 6x$$

On cherche les 2 racines pour $l(x)$, on trouve :

$b_1 = 9 = \alpha^4$ et $b_2 = 6 = \alpha^{13}$ d'où l'on tire :

$p_1 = 15 - 4 = 11$ et $p_2 = 15 - 13 = 2$.

Il nous reste à trouver la valeur de l'erreur à ces positions. Pour cela, on calcule

$e_1 = W(9) / L'(9) = 13 / 9 = 7$
$e_2 = W(6) / L'(6) = 12 / 9 = 10$

Ce qui nous donne un polynôme de correction :

$v(x) = 7x^{11} + 10x^2$

On remarquera que l'on a $v(x) = e(x)$, et que si l'on effectue la correction, cela nous donne

$c'(x) = d(x) + v(x) = c(x) + e(x) + v(x) = c(x)$

et que l'on retrouve notre code de départ, ce qui bien le but recherché.

4.6 Détection d’erreurs > t

Voici les principaux contrôles qu’ils faut effectuer afin de détecter un nombres d’erreurs > t.

Après avoir terminer l’algorithme d’Euclide, deux cas peuvent se présenter :

A1) $x^t \mid s(x)$ => $W(x) = 0$.
A2) $s(x) = w(x)$ => $l(x) = 1$ et donc pas de racine.

Lors de la recherche des racines de $l(x)$, trois cas peuvent se présenter :

B1) 0 est racine de $l(x)$.
B2) $l(x)$ a des racines multiples.
B3) $l(x)$ ne se décompose pas en produit de facteur linéaire.
On trouve donc moins de racine qu’il ne devrait y en avoir.

Lors de la recherche de la position de l’erreur, un cas peu se présenter :

C1) $p < 0$ ou $p >= n$.

Il est clair que lorsque l’on a un nombre d’erreurs > t, tous ces cas peuvent se présenter. Il faut donc absolument faire ces contrôles afin de détecter cette situation qui peut dès lors se présenter.

5. Présentation du programme

5.1 Classes

Voici les différentes classes utilisées pour le programme :

```
              RSApplet
             |        |
             |        |
             |        |
        RSFrame      fieldFrame
         |   |            |
         |   |            |
         |   |            |
         |  RSAlgorithme  |
         |      |      |  |
         |      |      |  |
         |      |      |  |
      polynomial      field
```

RSApplet	:	Applet du programme reed-solomon simulator.
RSFrame	:	Frame de la partie codage/décodage RS.
RSAlgorithme	:	Algorithmes de codage/décodage RS.
polynomial	:	Définition de la classe polynôme.
fieldFrame	:	Frame de la partie corps de Galois.
field	:	Définition et algorithmes de la classe field.

5.2 Liste des fichiers

Voici la liste de tous les fichiers qui accompagnent ce programme.

RSApplet.html	: pour lancer l'applet.
RSApplet.java	: applet java.
RSApplet.class	: version compilée.
RSFrame.java	: pour les codes de Reed-Solomon.
RSFrame.class	: version compilée.
RSAlgorithm.java	: algorithmes pour les codes RS.
RSAlgorithm.class	: version compilée.
fieldFrame.java	: pour les corps de Galois GF(2^k).
fieldFrame.class	: version compilée.
field.java	: algorithmes pour GF(2^k).
field.class	: version compilée.
polynomial.java	: définition de l'objet polynomial.
polynomial.class	: version compilée.
AppletViewer.bat	: pour lancer l'applet sous Win95.
Javac.bat	: pour compiler un prog. sous Win95.
Java.bat	: pour lancer une appli. sous Win95.
fond.jpg	: background pour l'applet.
rs.html	: documentation en version html.
rs.cwk	: documentation en version ClarisWorks.
rs.ps	: documentation en version PostScript.

6. Conclusions

Me voilà arrivé au bout de mon effort, je dois dire que j'ai eu beaucoup de plaisir à faire ce travail et que je l'ai trouvé très intéressant. Cela m'a permis de travailler à fond sur les corps de Galois et de faire connaissance avec les codes de Reed-Solomon. Lorsque j'ai commencé ce travaille, je n'avais aucune connaissance du langage Java, et grâce à ce projet, j'ai acquis de bonnes connaissances dans ce langage. Je souhaiterais rajouter une partie à ce programme, c'est la visualisation d'une image graphique que l'on code et décode avec adjonction de perturbations avant décodage, ou pourrait voir graphiquement la très grande efficacité de ces codes pour la transmission d'informations.

A.1 Tables

On donne ici, les tables d'addition et de multiplication du corps de Galois GF(2^4) avec $P(x) = x^4 + x^3 + 1$.

Table d'addition

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table de multiplication

*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	9	11	13	15	1	3	5	7
3	0	3	6	5	12	15	10	9	1	2	7	4	13	14	11	8
4	0	4	8	12	9	13	1	5	11	15	3	7	2	6	10	14
5	0	5	10	15	13	8	7	2	3	6	9	12	14	11	4	1
6	0	6	12	10	1	7	13	11	2	4	14	8	3	5	15	9
7	0	7	14	9	5	2	11	12	10	13	4	3	15	8	1	6
8	0	8	9	1	11	3	2	10	15	7	6	14	4	12	13	5
9	0	9	11	2	15	6	4	13	7	14	12	5	8	1	3	10
10	0	10	13	7	3	9	14	4	6	12	11	1	5	15	8	2
11	0	11	15	4	7	12	8	3	14	5	1	10	9	2	6	13
12	0	12	1	13	2	14	3	15	4	8	5	9	6	10	7	11
13	0	13	3	14	6	11	5	8	12	1	15	2	10	7	9	4
14	0	14	5	11	10	4	15	1	13	3	8	6	7	9	2	12
15	0	15	7	8	14	1	9	6	5	10	2	13	11	4	12	3

A.2 Références

[OP] **Error-Correcting Codes and Finite Fields.**
Oliver Pretzel. Clarendon Press, Oxford.

[LC] **Error control coding.**
S.Lin & D.Costello. Prentice-Hall.