Génération d'équations booléennes pour l'AES

« Let it further be agreed, that by the combination xy shall be represented that class of things to which the names or descriptions represented by x and y are simultaneously applicable. Thus, if x alone stands for "white things", and y for "sheep", let xy stand for "white sheep" ; and in like manner, if z stand for "horned things", and x and y retain their previous interpretations, let zxy represent "horned white sheep", i.e. that collection of things to which the name "sheep", and the descriptions "white" and "horned" are together applicable. »

George Boole [27]

4.1 Les fonctions booléennes

4.1.1 Définition

Soient l'ensemble $B = \{0, 1\}$ et $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ une algèbre booléenne, alors $\mathcal{B}_2^n = (x_1, x_2, \cdots, x_n)$ tel que $x_i \in \mathcal{B}_2$ et $1 \leqq i \leqq n$, est un sous-ensemble de $\mathcal{B}_2$ contenant tous les n-tuples de 0 et 1. La variable x_i est appelée variable booléenne si elle n'accepte que des valeurs appartenant à B, c'est-à-dire, si et seulement si $x_i = 0$ ou $x_i = 1$ quel que soit $1 \leqq i \leqq n$.

Définition 4.1. Une fonction booléenne de degré n avec $n > 1$ est une fonction f définie de $\mathcal{B}_2^n \to \mathcal{B}_2$, c'est-à-dire construite à partir de variables booléennes et n'acceptant de valeurs de retour que dans l'ensemble $B = \{0, 1\}$.

Par exemple, la fonction $f(x_1, x_2) = x_1 \wedge \neg x_2$ définie de $\mathcal{B}_2^2 \to \mathcal{B}_2$ est une fonction booléenne de degré deux avec :

$$f(0,0) = 0 \tag{4.1}$$
$$f(0,1) = 0 \tag{4.2}$$
$$f(1,0) = 1 \tag{4.3}$$
$$f(1,1) = 0 \tag{4.4}$$

Définition 4.2. Soient n et m deux entiers positifs. Une fonction booléenne vectorielle est une fonction booléenne f définie de $\mathcal{B}_2^n \to \mathcal{B}_2^m$.

À titre d'exemple, la S-box de l'AES est une fonction booléenne vectorielle avec $n = m = 8$.
Enfin, nous pouvons définir une fonction booléenne aléatoire comme étant une fonction booléenne f dont les valeurs sont des variables aléatoires indépendantes et identiquement distribuées, c'est-à-dire :

$$\forall (x_1, x_2, \cdots, x_n) \in \mathcal{B}_2^n, \quad P[f(x_1, x_2, \cdots, x_n) = 0] = \frac{1}{2}$$

Le nombre de fonctions booléennes est limité et dépendant de n. Ainsi, il existe 2^{2^n} fonctions booléennes. De même, le nombre de fonctions booléennes vectorielles est limité et dépendant de n et m. Ainsi, il existe $(2^m)^{2^n}$ fonctions booléennes vectorielles.
Si nous prenons, par exemple, $n = 2$, il existe alors $\left(2^2\right)^2 = 16$ fonctions booléennes de degré deux. Ces 16 fonctions booléennes sont présentées dans le tableau de la figure 4.1 page 61. Parmi les fonctions booléennes de degré 2, les plus connues sont les fonctions OR, AND et XOR (voir fig. 4.3 p. 62), (voir fig. 4.4 p. 62) et (voir fig. 4.2 p. 61). Le programme que nous avons développé pour réaliser ces visualisations est disponible sur Internet [81, VisBool3d]. Il a pour objectif de représenter les fonctions booléennes de degré deux en trois dimensions. La hauteur des colonnes correspond à la valeur obtenue en appliquant la fonction booléenne choisie à l'abscisse et à l'ordonnée correspondante.
Le support $supp(f)$ d'une fonction booléenne est l'ensemble des éléments x tel que $f(x) \neq 0$, le poids de Hamming $wt(f)$ d'une fonction booléenne est le cardinal de son support et nous avons donc :

$$wt(f) = |\{x \in \mathcal{B}_2^n \quad | \quad f(x) = 1\}|$$

Une fonction booléenne est dite équilibrée si $wt(f) = 2^{n-1}$. De même, une fonction booléenne vectorielle $\mathcal{B}_2^n \to \mathcal{B}_2^m$ est dite équilibrée si $wt(f) = 2^{n-m}$ [38].
Par exemple, le support de la fonction $f(x_1, x_2) = x_1 \vee x_2$, correspondant au `OU` logique, est $supp(f) = \{(0,1),(1,0),(1,1)\}$ et son poids est $wt(\mathrm{f}) = 3$.

4.1.2 Représentations

Il existe de multiples représentations des fonctions booléennes [66]. Nous allons nous intéresser à la plus simple – la table de vérité – et à celle que nous utiliserons par la suite – une représentation dans $GF(2)$.

4.1.2.1 La table de vérité

Les différentes valeurs prises par une fonction booléenne peuvent être présentées sous la forme d'un tableau appelé table de vérité. La table de vérité caractérise une fonction booléenne.

f_0	0
f_1	$x_1 \wedge x_2$
f_2	$x_1 \wedge \neg x_2$
f_3	x_1
f_4	$\neg x_1 \wedge x_2$
f_5	x_2
f_6	$x_1 \veebar x_2$
f_7	$x_1 \vee x_2$
f_8	$\neg(x_1 \vee x_2)$
f_9	$\neg(x_1 \veebar x_2)$
f_{10}	$\neg x_2$
f_{11}	$x_1 \vee \neg x_2$
f_{12}	$\neg x_1$
f_{13}	$\neg x_1 \vee x_2$
f_{14}	$\neg(x_1 \wedge x_2)$
f_{15}	1

FIGURE 4.1 – Les 16 fonctions booléennes de degré 2

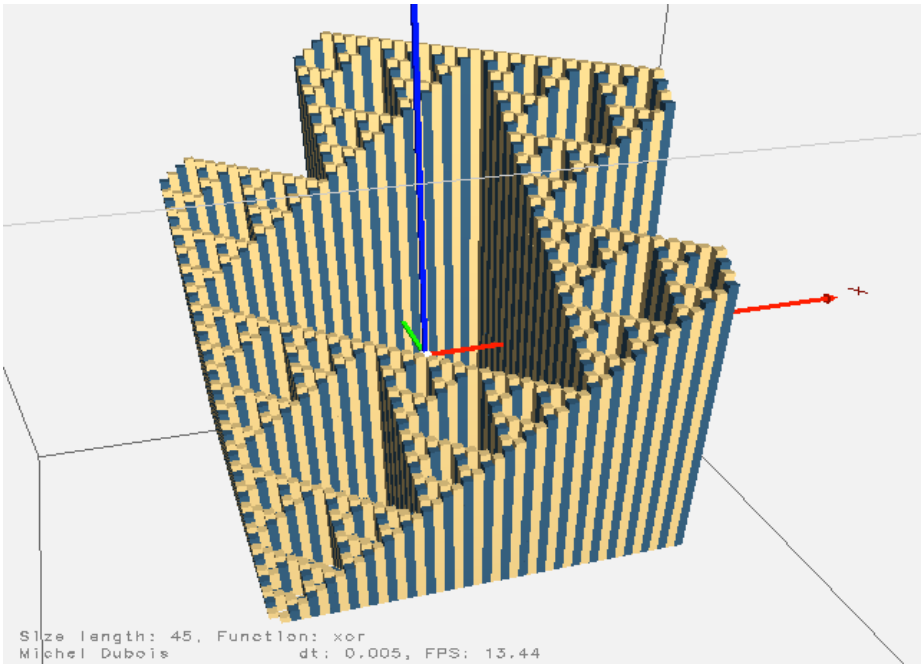

FIGURE 4.2 – La fonction XOR

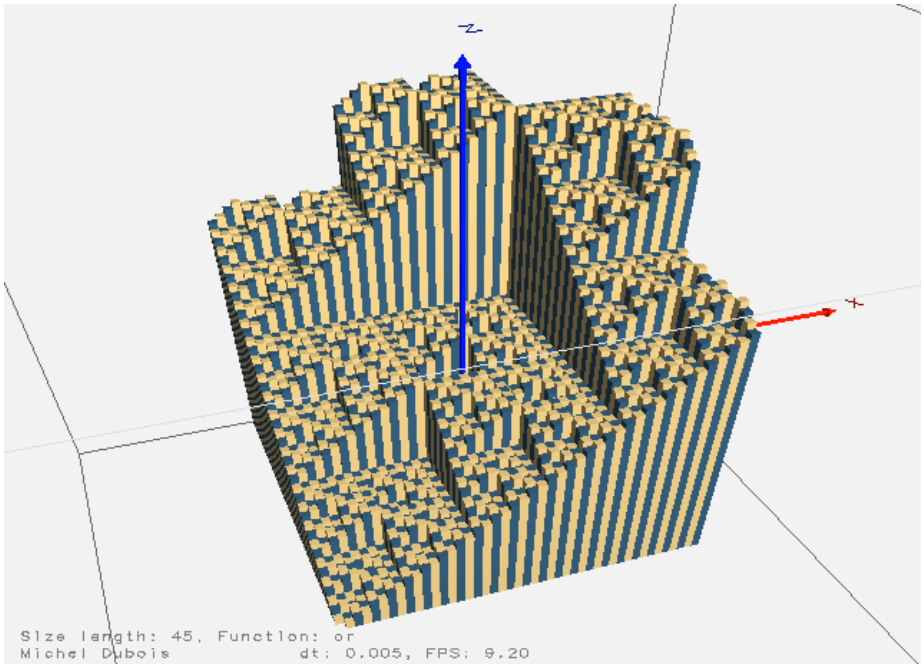

FIGURE 4.3 – La fonction `OR`

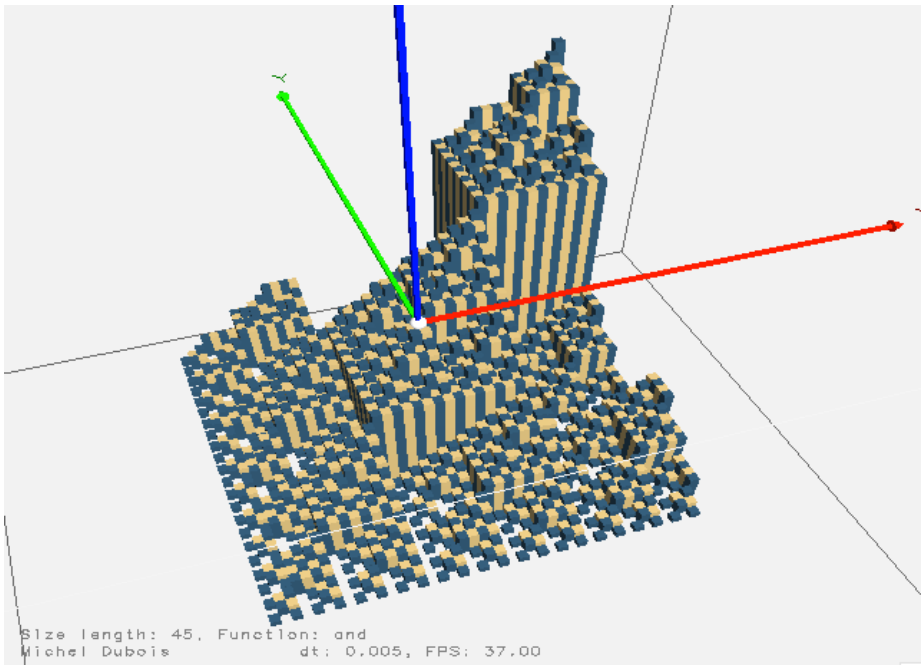

FIGURE 4.4 – La fonction `AND`

À titre d'exemple, le tableau de la figure 4.5 page 63 détaille les tables de vérité des 16 fonctions booléennes de degré deux.

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

x_1	x_2	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

FIGURE 4.5 – Les tables de vérité des 16 fonctions booléennes de degré 2

4.1.2.2 La représentation dans GF(2)

Une fonction booléenne peut également être présentée sous la forme d'une suite de conjonctions comprenant des disjonctions, des négations et/ou des variables. Il s'agit alors de la forme normale conjonctive (CNF). Ainsi, la séquence $f = (a \vee b) \wedge (\neg a \vee b)$ est la forme normale conjonctive de la fonction f. À l'inverse, une fonction booléenne peut être présentée sous la forme d'une suite de disjonctions comprenant des conjonctions, des négations et/ou des variables. Il s'agit alors de la forme normale disjonctive (DNF). Ainsi, la séquence $g = (a \wedge b) \vee (\neg a \wedge b)$ est la forme normale disjonctive de la fonction g. Les CNF et DNF existent et ne sont pas uniques [66].
Intéressons nous maintenant à la représentation des fonctions booléennes dans $GF(2)$.
L'ensemble $B = \{0, 1\}$ associé aux opérations $\wedge$, $\vee$ et $\neg$ est l'algèbre booléenne $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ avec les tables de vérité des opérations décrites dans la figure 4.6 page 63. Si nous introduisons les deux opérations binaires $\oplus$ et $\bullet$ définies par les tables de vérité de la figure 4.7 page 64, alors $\mathcal{B}_2$ et le corps de Galois $GF(2)$ sont assimilables. Plus précisément, l'algèbre de Boole $(B, \wedge, \vee, \neg)$ et le corps $(GF(2), \bullet, \oplus)$ sont liées par les formules de transformation suivantes :

$a \wedge b = a \bullet b$ $\quad$ $a \bullet b = a \wedge b$
$a \vee b = a \oplus b \oplus (a \bullet b)$ $\quad$ $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$
$\neg a = a \oplus 1$

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

a	0	1
$\neg a$	1	0

FIGURE 4.6 – Règles pour l'algèbre de Boole à deux éléments

Nous pouvons maintenant définir une fonction booléenne comme étant une fonction $f : \mathbb{F}_2^n \to \mathbb{F}_2$ avec $\mathbb{F}_2^n$ l'ensemble des vecteurs binaires de longueur $n > 1$.

$\bullet$	0	1
0	0	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

FIGURE 4.7 – Tables de vérité de $\bullet$ et $\oplus$

Le poids de Hamming $wH(x)$ du vecteur binaire $x \in \mathbb{F}_2^n$ est le nombre de ses coordonnées non nulles c'est-à-dire la taille de l'ensemble $\{i \in \mathbb{N} \quad | \quad x_i \neq 0\}$. Le poids de Hamming d'une fonction booléenne $f : \mathbb{F}_2^n \to \mathbb{F}_2$ est la taille de son support. Enfin, la distance de Hamming entre deux fonctions booléennes f et g est la taille de l'ensemble $\{x \in \mathbb{F}_2^n \quad | \quad f(x) \neq g(x)\}$.
Parmi les représentation classiques des fonctions booléennes, celle la plus fréquemment utilisée en cryptographie est la représentation polynomiale à n-variables sur $GF(2)$. Cette représentation est de la forme [37] :

$$f(x) = \bigoplus_{I \in P(N)} a_I \left(\prod_{i \in I} x_i \right) = \bigoplus_{I \in P(N)} a_i x^I$$

$P(N)$ désigne l'ensemble des puissances de $N = \{1, \cdots, n\}$. Chaque coordonnée x_i apparaît dans ce polynôme avec un exposant au moins égal à un parce que, dans $\mathbb{F}_2$, nous avons $x^2 = x$. Cette représentation est décrite dans $\mathbb{F}_2[x_1, \cdots, x_n]/(x_1^2 \oplus x_1, \cdots, x_n^2 \oplus x_n)$.
Cette représentation des fonctions booléennes dans $GF(2)$ est appelée expansion de Reed-Muller ou polynômes de Zhegalkin ([152] page 169) ou, plus couramment, *forme normale algébrique* ou *Algebraic Normal Form* (ANF) en anglais. Le degré de $ANF(f)$ correspond au plus haut degré des monômes de $ANF(f)$ à coefficients non nuls. Enfin, la forme normale algébrique d'une fonction booléenne existe et est unique.
En résumé, toute fonction booléenne peut être représentée, de façon unique, par sa forme normale algébrique sous la forme de l'équation :

$$\begin{aligned} f(x_1, \cdots, x_n) = a_0 & \\ & \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \\ & \oplus a_{1,2} x_1 x_2 \oplus \cdots \oplus a_{n-1,n} x_{n-1} x_n \\ & \oplus \cdots \oplus \\ & \oplus a_{1,2,\ldots,n} x_1 x_2 \ldots x_n \end{aligned}$$

Prenons un exemple. Soit la fonction $f(x_1, x_2, x_3)$ décrite par la table de vérité suivante :

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

La forme algébrique normale de $f(x_1, x_2, x_3)$ est :

$$\begin{aligned} f(x_1, x_2, x_3) = a_0 & \\ & \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \\ & \oplus a_{1,2} x_1 x_2 \oplus a_{1,3} x_1 x_3 \oplus a_{2,3} x_2 x_3 \\ & \oplus a_{1,2,3} x_1 x_2 x_3 \end{aligned}$$

et nous devons déterminer les valeurs des variables a.
Le poids de la fonction f est $wt(f) = 3$. Nous pouvons donc réduire f à la somme de 3 fonctions atomiques f_1, f_2 et f_3. La fonction $f_1 = 1$ si seulement si $1 \oplus x_1 = 1$, $1 \oplus x_2 = 1$ et $x_3 = 1$. De là nous pouvons déduire que l'ANF de la fonction f_1 peut être obtenue par expansion du produit $(1 \oplus x_1)(1 \oplus x_2)x_3$. En appliquant ce raisonnement aux fonctions f_2 et f_3 nous obtenons l'équation suivante :

$$\begin{aligned} ANF(f) &= (1 \oplus x_1)(1 \oplus x_2)x_3 \oplus x_1(1 \oplus x_2)x_3 \oplus x_1 x_2 x_3 \\ &= x_1 x_2 x_3 \oplus x_2 x_3 \oplus x_3 \end{aligned} \tag{4.5}$$

4.2 Mécanisme des équations

Après cette présentation des fonctions booléennes, nous disposons des outils nécessaires à l'élaboration de systèmes d'équations booléennes décrivant l'*Advanced Encryption standard*. Afin d'avancer progressivement nous allons commencer par appliquer notre méthode sur le mini-aes puis nous l'étendrons à l'AES.

4.2.1 Transformée de Möbius

Nous venons de voir comment générer simplement la forme algébrique normale (ANF) d'une fonction booléenne. La méthode présentée n'est pas facilement automatisable dans un programme informatique. Nous allons donc lui préférer l'utilisation de la transformée de Möbius.
La transformée de Möbius de la fonction booléenne f est définie par [141] :

$$\begin{aligned} TM(f) &: \mathbb{F}_2^n \to \mathbb{F}_2 \\ u &= \bigoplus_{v \leqslant u} f(v) \mathrm{mod} 2 \end{aligned}$$

avec $v \leqslant u$ si et seulement si $\forall i, v_i = 1 \Rightarrow u_i = 1$.
De là, nous pouvons définir la forme algébrique normale d'une fonction booléenne f à n variables par :

$$\bigoplus_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^n} TM(u)x_1^{u_1}\cdots x_n^{u_n}$$

Afin de mieux appréhender les mécanismes mis en œuvre, dans l'utilisation de la transformée de Möbius, prenons un exemple avec la fonction `MajParmi3`. Cette fonction de $\mathbb{F}_2^3 \to \mathbb{F}_2$ est caractérisée par la table de vérité présentée dans la figure 4.8 page 66.

x_1	x_2	x_3	`MajParmi3`
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

FIGURE 4.8 – La table de vérité de la fonction `MajParmi3`

En calculant la transformée de Möbius de la fonction nous obtenons le résultat de la figure 4.9 page 66.

x_1	x_2	x_3	`MajParmi3`	$\to$	calcul de $TM(f)$				$TM(f)$
0	0	0	0	$\to$	0	0	0	0	0
0	0	1	0	$\to$	0	0	0	0	0
0	1	0	0	$\to$	0	0	0	0	0
0	1	1	1	$\to$	1	1	1	1	1
1	0	0	0	$\to$	0	0	0	0	0
1	0	1	1	$\to$	1	1	1	1	1
1	1	0	1	$\to$	1	1	1	1	1
1	1	1	1	$\to$	1	0	1	0	0

FIGURE 4.9 – Calcul de la transformée de Möbius pour `MajParmi3`

Une fois la transformée de Möbius de la fonction obtenue, nous prenons les éléments de $\mathbb{F}_2^3$ pour lesquels $TM(\text{MajParmi3}) \neq 0$. Dans notre cas nous avons les triplets $(0,1,1), (1,0,1), (1,1,0)$ d'où nous pouvons déduire l'équation :

$$\text{MajParmi3}(x_1, x_2, x_3) = x_2x_3 \oplus x_1x_3 \oplus x_1x_2$$

Avec l'addition correspondant à un `XOR` et la multiplication à un `AND`.
La mise en œuvre de la transformée de Möbius en langage Python est réalisée par les deux fonctions décrites dans le listing 15 page 67. Pour calculer la transformée de Möbius, nous utilisons l'algorithme de Cooley-Tukey [56].

```python
def xorTab(t1, t2):
    """Takes two tabs t1 and t2 of same lengths and returns t1 XOR t2."""
    result = ''
    for i in xrange(len(t1)):
        result += str(int(t1[i]) ^ int(t2[i]))
    return result

def moebiusTransform(tab):
    """Takes a tab and return tab[0 : len(tab)/2],
    tab[0 : len(tab)/2] ^ tab[len(tab)/2 : len(tab)].
    usage: moebiusTransform(1010011101010100) --> [1100101110001010]"""
    if len(tab) == 1:
        return tab
    else:
        t1 = tab[0 : len(tab)/2]
        t2 = tab[len(tab)/2 : len(tab)]
        t2 = xorTab(t1, t2)
        t1 = moebiusTransform(t1)
        t2 = moebiusTransform(t2)
        t1 += t2
        return t1
```

Listing 15 – Calcul de la transformée de Möbius en python

4.3 Application au mini-AES

Nous allons maintenant élaborer le système d'équations booléennes décrivant l'AES. En raison de la complexité de ses fonctions internes, nous commençons par appliquer le processus précédemment décrit sur une version simplifiée de l'AES : le mini-AES.

4.3.1 Le mini-AES

C'est avec l'objectif d'aider les étudiants en cryptographie et les cryptanalystes à mieux comprendre les mécanismes internes de l'AES que Raphael Chung-Wei Phan a présenté, en 2002, sa mini version de l'AES [162]. Cette version utilise des paramètres restreints par rapport à l'AES tout en préservant sa structure interne et ses propriétés algébriques.
Le mini-AES est un algorithme de chiffrement par bloc reposant sur les mêmes primitives mathématiques que son grand frère l'AES. Les éléments atomiques avec lequel le mini-AES travaille sont des éléments du corps fini $GF(2^4)$ appelés *nibbles*. Comme l'AES, le mini-AES utilise un tableau d'états contenant quatre nibbles ce qui en fait un algorithme de chiffrement par bloc de 16 bits.
Le processus de chiffrement du mini-AES consiste en deux tours faisant intervenir les fonctions *NibbleSub* appliquant la SBOX au tableau d'états, *ShiftRow* exécutant une rotation des cases du tableau d'états et *MixColumn* multipliant chaque colonne du tableau d'états par une matrice constante. L'architecture des tours est présentée dans la figure 4.10 page 68 et l'implémentation du mini-AES en langage python que nous avons développé est disponible sur Internet [81, BooleanAES].

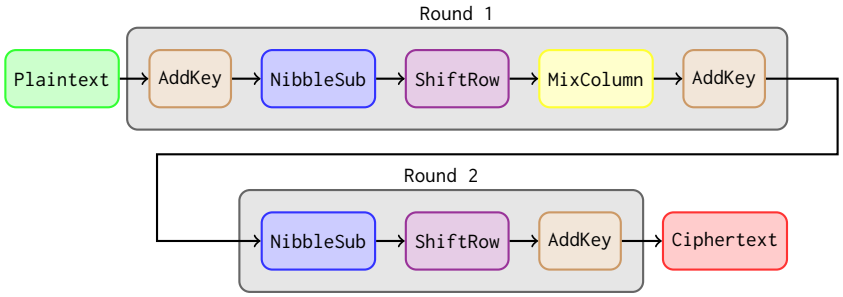

FIGURE 4.10 – Architecture des tours du mini-aes

4.3.2 Les équations pour le mini-AES

En reprenant le principe de génération d'équations énoncé plus haut, nous allons définir les équations pour le mini-AES.
Afin de simplifier le processus nous réduisons le mini-AES à cinq fonctions booléennes de $\mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$, une fonction pour chaque tour et trois fonctions pour la dérivation de la clef.
Les fonctions pour les tours X_1 et X_2 résultent de la conjonction des fonctions `NibbleSub` $NS()$, `ShiftRow` $SR()$ et `MixColumn` $MC()$ telle que, en prenant le bloc de 16 bits $B = (b_1, \cdots, b_{16})$, nous ayons

$$X_1(B) = MC \circ SR \circ NS(B)$$

et

$$X_2(B) = SR \circ NS(B)$$

Les trois fonctions K_1, K_2 et K_3 décrivent le processus de dérivation de la clef tel que, à partir du bloc de clef de 16 bits $K = (k_1, \cdots, k_{16})$, nous ayons les clefs utilisées dans les rounds $k_i = (k_{i,1}, \cdots, k_{i,16})$ avec $i \in (1, 2, 3)$.
Finalement, le mini-AES peut s'écrire sous la forme de deux équations R_1 et R_2, décrivant chacune un tour, telles que :

$$\begin{aligned} B' = R_1(B) =& X_1(B \oplus K_1(K)) \oplus K_2(K) \\ =& (x_{1,1}, \cdots, x_{1,16}) \oplus (k_{2,1}, \cdots, k_{2,16}) \\ =& (b'_1, \cdots, b'_{16}) \\ B'' = R_2(B') =& X_2(B') \oplus K_3(K) \\ =& (x_{2,1}, \cdots, x_{2,16}) \oplus (k_{3,1}, \cdots, k_{3,16}) \\ =& (b''_1, \cdots, b''_{16}) \end{aligned}$$

Avec $x_{1,i} = b_i \oplus k_{1,i} \quad \forall i \in (1, \cdots, 16)$ et B, B', B'' désignant respectivement le bloc de 16 bits en entrée, le bloc de 16 bits à la fin du premier tour et le bloc de 16 bits à la fin du deuxième tour et K le bloc de 16 bits de clef.
Nous pouvons alors calculer les tables de vérité des fonctions booléennes K_1, K_2, K_3, X_1 et X_2 (Cf. figure 4.11 page 69). Puis en utilisant la méthodologie retenue pour la fonction `MajParmi3`, nous obtenons un ensemble de 16 équations pour chaque fonction booléenne, soit une équation pour chaque bit de bloc.

Cependant, du fait que, contrairement à la fonction `MajParmi3`, nous avons deux variables différentes et que les fonctions de chaque tour sont composées, ces dernières seront présentées et traitées séparément dans chaque fichier.

$$
\begin{aligned}
X_1(1026) &= 1000101101011001\\
X_1(1027) &= 0011110001011001\\
X_1(1028) &= 0101100101011001\\
X_1(1029) &= 1100110101011001\\
&\dots\\
X_1(32000) &= 0100001000000111\\
X_1(32001) &= 0011111100000111\\
X_1(32002) &= 0010011100000111\\
X_1(32003) &= 1001000000000111\\
&\dots\\
X_1(65000) &= 1111101100011000\\
X_1(65001) &= 1110001100011000\\
X_1(65002) &= 0101010000011000\\
X_1(65003) &= 0010100100011000
\end{aligned}
$$

FIGURE 4.11 – Quelques équations extraites de la table de vérité de la fonction $X_1()$

Les équations ainsi obtenues pour les fonctions X_1 et X_2 sont détaillées dans l'annexe B page 165.

4.3.3 Mise en forme des équations

Afin d'en faciliter l'analyse et notamment d'y tenter une étude combinatoire nous allons mettre en œuvre une présentation spécifique pour les équations ainsi obtenues.
Le principe retenu consiste à générer un fichier par bit pour les fonctions R_1 et R_2, nous aurons donc finalement 32 fichiers. Le contenu de chacun de ces fichiers est constitué de lignes contenant des suites de 0 et de 1. Chaque ligne décrit un monôme de l'équation et le passage d'une ligne à l'autre signifie l'application d'un `XOR` (cf. figure 4.12 page 70).
Dans le but de faciliter la compréhension du mécanisme retenu nous détaillons la réalisation du fichier correspondant au bit b'_1 de la fonction R_1 dans la figure 4.13 page 71.

4.4 Application à l'AES

Nous allons maintenant appliquer à l'AES le mécanisme décrit ci-dessus pour le mini-AES. La difficulté de ce passage à l'échelle réside dans le fait que, avec le mini-AES, nous avons des fonctions booléennes de $F_2^{16} \to F_2^{16}$ et qu'il est facile

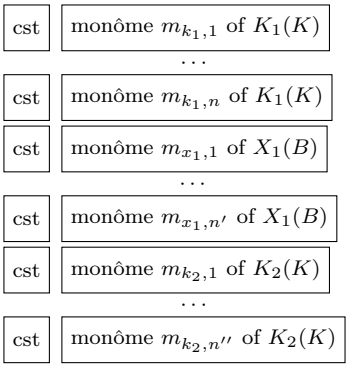

FIGURE 4.12 – Structure du fichier pour la fonction $R_1(b)$

de calculer leurs tables de vérité. Les fonctions de chiffrement et de déchiffrement de l'algorithme de l'AES prennent 128 bits en entrée et fournissent 128 bits en sortie. Nous aurons donc des fonctions booléennes de $F_2^{128} \rightarrow F_2^{128}$ et il est impossible de calculer leurs tables de vérité. En effet, dans ce cas, nous avons $2^{128} = 3,402823 \times 10^{38}$ combinaisons possibles de blocs de 128 bits et l'espace de stockage nécessaire pour archiver ces blocs est de $3,868562 \times 10^{25}$ téraoctets. Nous devons donc trouver une solution pour décrire les fonctions de chiffrement et de déchiffrement de l'AES sous la forme de fonctions booléennes et obtenir ainsi le même résultat que pour l'algorithme du mini-AES.
À l'issue, comme pour le mini-AES, nous devons obtenir 128 fichiers, chacun décrivant la transformation d'un bit de bloc. Le contenu de chacun de ces fichiers consiste en lignes contenant des séquences de 0 et de 1. Chaque ligne décrivant un monôme de la forme normale algébrique des équations booléennes et la transition d'une ligne à une autre correspondant à l'application d'un `XOR`. Comme pour le mini-AES, les variables étant de natures différentes et les fonctions étant combinées, les fonctions de tours seront présentées et traitées par blocs indépendants.

4.4.1 Les équations pour les fonctions de chiffrement

Nous allons maintenant détailler la solution mise en œuvre pour chacune des sous-fonctions de l'algorithme de chiffrement de l'AES.

4.4.1.1 Solution pour la fonction SubBytes

La fonction `SubBytes` est une substitution non-linéaire qui travaille sur chaque octet du tableau d'états en utilisant une table de substitution (S-Box).
Cette fonction est appliquée indépendamment sur chaque octet du bloc d'entrée. Hors, la S-Box de l'AES est une fonction prenant 8 bits en entrée et fournissant 8 bits en sortie. Nous pouvons donc la décrire sous la forme d'une fonction booléenne de $F_2^8 \rightarrow F_2^8$. À partir de là, nous pouvons calculer la table de vérité de la S-Box et utiliser la transformée de Möbius pour obtenir la forme algébrique normale de la S-Box. En appliquant ensuite ces résultats sur les 16 octets du bloc d'entrée, nous obtenons 128 équations, chacune décrivant un bit de bloc.

$b'_1 \rightarrow K_1(K) \rightarrow k_1,$

$$\begin{aligned} X_1(B) \rightarrow & 1 \oplus x_{1,15}x_{1,16} \oplus x_{1,14} \oplus x_{1,14}x_{1,16} \oplus x_{1,13} \oplus x_{1,13}x_{1,15} \\ & \oplus x_{1,13}x_{1,15}x_{1,16} \oplus x_{1,4} \oplus x_{1,3}x_{1,4} \oplus x_{1,2}x_{1,4} \oplus x_{1,2}x_{1,3} \\ & \oplus x_{1,2}x_{1,3}x_{1,4} \oplus x_{1,1}x_{1,3} \oplus x_{1,1}x_{1,3}x_{1,4} \oplus x_{1,1}x_{1,2} \\ & \oplus x_{1,1}x_{1,2}x_{1,3} \end{aligned}$$

$$K_2(K) \rightarrow 1 \oplus k_{16} \oplus k_{14} \oplus k_{14}k_{15} \oplus k_{14}k_{15}k_{16} \oplus k_{13} \oplus k_{13}k_{14} \oplus k_{13}k_{14}k_{15} \oplus k_1$$

k_1	0	1000000000000000
1	1	0000000000000000
$x_{1,15}x_{1,16}$	0	0000000000000011
$x_{1,14}$	0	0000000000000100
$x_{1,14}x_{1,16}$	0	0000000000000101
$x_{1,13}$	0	0000000000001000
$x_{1,13}x_{1,15}$	0	0000000000001010
$x_{1,13}x_{1,15}x_{1,16}$	0	0000000000001011
$x_{1,4}$	0	0001000000000000
$x_{1,3}x_{1,4}$	0	0011000000000000
$x_{1,2}x_{1,4}$	0	0101000000000000
$x_{1,2}x_{1,3}$	0	0110000000000000
$x_{1,2}x_{1,3}x_{1,4}$	0	0111000000000000
$x_{1,1}x_{1,3}$	0	1010000000000000
$x_{1,1}x_{1,3}x_{1,4}$	0	1011000000000000
$x_{1,1}x_{1,2}$	0	1100000000000000
$x_{1,1}x_{1,2}x_{1,3}$	0	1110000000000000
1	1	0000000000000000
k_{16}	0	0000000000000001
k_{14}	0	0000000000000100
$k_{14}k_{15}$	0	0000000000000110
$k_{14}k_{15}k_{16}$	0	0000000000000111
k_{13}	0	0000000000001000
$k_{13}k_{14}$	0	0000000000001100
$k_{13}k_{14}k_{15}$	0	0000000000001110
k_1	0	1000000000000000

FIGURE 4.13 – Fichier correspondant au bit b'_1

À titre d'exemple, l'équation du traitement du bit b_{127} par la fonction `SubByte` est donnée dans la figure 4.14 page 72.

$$\begin{aligned}\text{SubByte}(b_{127}) = {} & 1 \oplus b_{127} \oplus b_{126}b_{127} \oplus b_{125} \oplus b_{125}b_{126} \oplus b_{124} \oplus b_{124}b_{126} \oplus b_{124}b_{125} \oplus b_{124}b_{125}b_{126} \oplus \\ & b_{124}b_{125}b_{126}b_{127} \oplus b_{123} \oplus b_{123}b_{127} \oplus b_{123}b_{126} \oplus b_{123}b_{126}b_{127} \oplus b_{123}b_{125} \oplus b_{123}b_{125}b_{127} \oplus \\ & b_{123}b_{125}b_{126} \oplus b_{123}b_{124}b_{127} \oplus b_{123}b_{124}b_{126} \oplus b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{122}b_{127} \oplus b_{122}b_{125}b_{127} \oplus \\ & b_{122}b_{125}b_{126}b_{127} \oplus b_{122}b_{124}b_{127} \oplus b_{122}b_{124}b_{125} \oplus b_{122}b_{124}b_{125}b_{127} \oplus b_{122}b_{124}b_{125}b_{126} \oplus \\ & b_{122}b_{123}b_{126}b_{127} \oplus b_{122}b_{123}b_{125}b_{127} \oplus b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{127} \oplus \\ & b_{121}b_{126} \oplus b_{121}b_{126}b_{127} \oplus b_{121}b_{125} \oplus b_{121}b_{125}b_{127} \oplus b_{121}b_{125}b_{126} \oplus b_{121}b_{125}b_{126}b_{127} \oplus \\ & b_{121}b_{124}b_{127} \oplus b_{121}b_{124}b_{125}b_{126} \oplus b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{123} \oplus b_{121}b_{123}b_{127} \oplus \\ & b_{121}b_{123}b_{126} \oplus b_{121}b_{123}b_{125}b_{126} \oplus b_{121}b_{123}b_{124}b_{127} \oplus b_{121}b_{123}b_{124}b_{126} \oplus b_{121}b_{123}b_{124}b_{126}b_{127} \oplus \\ & b_{121}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{122} \oplus b_{121}b_{122}b_{126} \oplus b_{121}b_{122}b_{125} \oplus \\ & b_{121}b_{122}b_{125}b_{127} \oplus b_{121}b_{122}b_{124}b_{127} \oplus b_{121}b_{122}b_{124}b_{126}b_{127} \oplus b_{121}b_{122}b_{124}b_{125}b_{126} \oplus \\ & b_{121}b_{122}b_{123} \oplus b_{121}b_{122}b_{123}b_{127} \oplus b_{121}b_{122}b_{123}b_{126} \oplus b_{121}b_{122}b_{123}b_{126}b_{127} \oplus b_{121}b_{122}b_{123}b_{125} \oplus \\ & b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{121}b_{122}b_{123}b_{124}b_{127} \oplus b_{121}b_{122}b_{123}b_{124}b_{125} \oplus b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus \\ & b_{120}b_{126}b_{127} \oplus b_{120}b_{125} \oplus b_{120}b_{125}b_{127} \oplus b_{120}b_{125}b_{126}b_{127} \oplus b_{120}b_{124}b_{126} \oplus b_{120}b_{124}b_{125} \oplus \\ & b_{120}b_{124}b_{125}b_{127} \oplus b_{120}b_{124}b_{125}b_{126} \oplus b_{120}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{123}b_{127} \oplus b_{120}b_{123}b_{126}b_{127} \oplus \\ & b_{120}b_{123}b_{125} \oplus b_{120}b_{123}b_{125}b_{127} \oplus b_{120}b_{123}b_{125}b_{126} \oplus b_{120}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{123}b_{124} \oplus \\ & b_{120}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{122} \oplus b_{120}b_{122}b_{125} \oplus b_{120}b_{122}b_{125}b_{127} \oplus \\ & b_{120}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{124} \oplus b_{120}b_{122}b_{124}b_{126}b_{127} \oplus b_{120}b_{122}b_{124}b_{125} \oplus \\ & b_{120}b_{122}b_{124}b_{125}b_{126} \oplus b_{120}b_{122}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{127} \oplus b_{120}b_{122}b_{123}b_{126} \oplus \\ & b_{120}b_{122}b_{123}b_{125} \oplus b_{120}b_{122}b_{123}b_{125}b_{127} \oplus b_{120}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{127} \oplus \\ & b_{120}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{122}b_{123}b_{124}b_{125} \oplus b_{120}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{120}b_{121} \oplus \\ & b_{120}b_{121}b_{125} \oplus b_{120}b_{121}b_{125}b_{126} \oplus b_{120}b_{121}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{124} \oplus b_{120}b_{121}b_{124}b_{126} \oplus \\ & b_{120}b_{121}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{124}b_{125} \oplus b_{120}b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{123}b_{127} \oplus \\ & b_{120}b_{121}b_{123}b_{125}b_{127} \oplus b_{120}b_{121}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{123}b_{124}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{122} \oplus b_{120}b_{121}b_{122}b_{126} \oplus \\ & b_{120}b_{121}b_{122}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{125}b_{126} \oplus b_{120}b_{121}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{124} \oplus \\ & b_{120}b_{121}b_{122}b_{124}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{125}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{122}b_{123}b_{125} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus \\ & b_{120}b_{121}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125}b_{127}\end{aligned}$$

FIGURE 4.14 – Équation du bit b_{127} de la fonction `SubByte`

4.4.1.2 Solution pour la fonction ShiftRows

Dans la fonction `ShiftRows`, les octets de la troisième colonne du tableau d'états sont décalés cycliquement selon un offset dont la taille est dépendante du numéro de la ligne. Les octets de la première ligne ne subissent pas ce décalage.
Pour cette fonction, nous n'avons pas besoin de calculer de fonction booléenne spécifique. En effet, la seule transformation réalisée consiste à décaler des octets dans le tableau d'états. Dans nos fichiers, cette transformation peut aisément être résolue par l'utilisation d'un `XOR`.
Ainsi, à titre d'exemple, le deuxième octet du tableau d'états devient le sixième octet après l'application de `ShiftRows`. Ce qui se traduit par les lignes suivantes :

```
00000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

À titre d'exemple, les équations de la fonction `ShiftRows` pour les bits b_0 à b_{63} sont données dans la figure 4.15 page 73.

$\text{ShiftRows}(b_0) = b_0$	$\text{ShiftRows}(b_{16}) = b_{80}$	$\text{ShiftRows}(b_{32}) = b_{32}$	$\text{ShiftRows}(b_{48}) = b_{112}$
$\text{ShiftRows}(b_1) = b_1$	$\text{ShiftRows}(b_{17}) = b_{81}$	$\text{ShiftRows}(b_{33}) = b_{33}$	$\text{ShiftRows}(b_{49}) = b_{113}$
$\text{ShiftRows}(b_2) = b_2$	$\text{ShiftRows}(b_{18}) = b_{82}$	$\text{ShiftRows}(b_{34}) = b_{34}$	$\text{ShiftRows}(b_{50}) = b_{114}$
$\text{ShiftRows}(b_3) = b_3$	$\text{ShiftRows}(b_{19}) = b_{83}$	$\text{ShiftRows}(b_{35}) = b_{35}$	$\text{ShiftRows}(b_{51}) = b_{115}$
$\text{ShiftRows}(b_4) = b_4$	$\text{ShiftRows}(b_{20}) = b_{84}$	$\text{ShiftRows}(b_{36}) = b_{36}$	$\text{ShiftRows}(b_{52}) = b_{116}$
$\text{ShiftRows}(b_5) = b_5$	$\text{ShiftRows}(b_{21}) = b_{85}$	$\text{ShiftRows}(b_{37}) = b_{37}$	$\text{ShiftRows}(b_{53}) = b_{117}$
$\text{ShiftRows}(b_6) = b_6$	$\text{ShiftRows}(b_{22}) = b_{86}$	$\text{ShiftRows}(b_{38}) = b_{38}$	$\text{ShiftRows}(b_{54}) = b_{118}$
$\text{ShiftRows}(b_7) = b_7$	$\text{ShiftRows}(b_{23}) = b_{87}$	$\text{ShiftRows}(b_{39}) = b_{39}$	$\text{ShiftRows}(b_{55}) = b_{119}$
$\text{ShiftRows}(b_8) = b_{40}$	$\text{ShiftRows}(b_{24}) = b_{120}$	$\text{ShiftRows}(b_{40}) = b_{72}$	$\text{ShiftRows}(b_{56}) = b_{24}$
$\text{ShiftRows}(b_9) = b_{41}$	$\text{ShiftRows}(b_{25}) = b_{121}$	$\text{ShiftRows}(b_{41}) = b_{73}$	$\text{ShiftRows}(b_{57}) = b_{25}$
$\text{ShiftRows}(b_{10}) = b_{42}$	$\text{ShiftRows}(b_{26}) = b_{122}$	$\text{ShiftRows}(b_{42}) = b_{74}$	$\text{ShiftRows}(b_{58}) = b_{26}$
$\text{ShiftRows}(b_{11}) = b_{43}$	$\text{ShiftRows}(b_{27}) = b_{123}$	$\text{ShiftRows}(b_{43}) = b_{75}$	$\text{ShiftRows}(b_{59}) = b_{27}$
$\text{ShiftRows}(b_{12}) = b_{44}$	$\text{ShiftRows}(b_{28}) = b_{124}$	$\text{ShiftRows}(b_{44}) = b_{76}$	$\text{ShiftRows}(b_{60}) = b_{28}$
$\text{ShiftRows}(b_{13}) = b_{45}$	$\text{ShiftRows}(b_{29}) = b_{125}$	$\text{ShiftRows}(b_{45}) = b_{77}$	$\text{ShiftRows}(b_{61}) = b_{29}$
$\text{ShiftRows}(b_{14}) = b_{46}$	$\text{ShiftRows}(b_{30}) = b_{126}$	$\text{ShiftRows}(b_{46}) = b_{78}$	$\text{ShiftRows}(b_{62}) = b_{30}$
$\text{ShiftRows}(b_{15}) = b_{47}$	$\text{ShiftRows}(b_{31}) = b_{127}$	$\text{ShiftRows}(b_{47}) = b_{79}$	$\text{ShiftRows}(b_{63}) = b_{31}$

FIGURE 4.15 – Équations pour les bits b_0 à b_{63} de la fonction `ShiftRows`

4.4.1.3 Solution pour la fonction MixColumns

La fonction `MixColumns` agit sur le tableau d'états, colonne par colonne, traitant chaque colonne comme un polynôme à quatre termes. Chacune de ces colonnes est multipliée par une matrice carrée. Pour chaque colonne nous avons donc :

$$\begin{pmatrix} b'_i \\ b'_{i+1} \\ b'_{i+2} \\ b'_{i+3} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} b_i \\ b_{i+1} \\ b_{i+2} \\ b_{i+3} \end{pmatrix}$$

Ainsi, pour le premier octet de la colonne nous avons cette équation :

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus 01 \bullet b_{i+2} \oplus 01 \bullet b_{i+3}$$

Comme dans GF_2^8, 01 est l'identité pour la multiplication, cette équation devient :

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus b_{i+2} \oplus b_{i+3}$$

Nous avons la même simplification pour toutes les équations décrivant la multiplication de la colonne du tableau d'états par la matrice carrée. Par conséquent nous devons seulement calculer les tables de vérité pour la multiplication par 02 et 03 dans GF_2^8.
À titre d'exemple, les équations des bits b_{120} à b_{127} sont données dans la figure 4.16 page 74.

4.4.1.4 Solution pour la fonction d'expansion de la clef

Pour rappel, dans l'algorithme de l'AES-128, Nb = 4 mots et Nr = 10 mots, avec 1 mot = 4 octets = 32 bits.

$$\begin{aligned}
\text{MixColumns}(b_{120}) &= b_{97} \oplus b_{96} \oplus b_{104} \oplus b_{112} \oplus b_{121} \\
\text{MixColumns}(b_{121}) &= b_{98} \oplus b_{97} \oplus b_{105} \oplus b_{113} \oplus b_{122} \\
\text{MixColumns}(b_{122}) &= b_{99} \oplus b_{98} \oplus b_{106} \oplus b_{114} \oplus b_{123} \\
\text{MixColumns}(b_{123}) &= b_{100} \oplus b_{99} \oplus b_{96} \oplus b_{107} \oplus b_{115} \oplus b_{124} \oplus b_{120} \\
\text{MixColumns}(b_{124}) &= b_{101} \oplus b_{100} \oplus b_{96} \oplus b_{108} \oplus b_{116} \oplus b_{125} \oplus b_{120} \\
\text{MixColumns}(b_{125}) &= b_{102} \oplus b_{101} \oplus b_{109} \oplus b_{117} \oplus b_{126} \\
\text{MixColumns}(b_{126}) &= b_{103} \oplus b_{102} \oplus b_{96} \oplus b_{110} \oplus b_{118} \oplus b_{127} \oplus b_{120} \\
\text{MixColumns}(b_{127}) &= b_{103} \oplus b_{96} \oplus b_{111} \oplus b_{119} \oplus b_{120}
\end{aligned}$$

FIGURE 4.16 – Équations pour les bits b_{120} à b_{127} de la fonction `MixColumns`

La fonction `AddRoundKey` ajoute une clef de tour au tableau d'état par une simple opération de `XOR` bit à bit. Ces clefs de tour sont calculées par une fonction d'expansion de la clef. Cette dernière génère un ensemble de $Nb(Nr + 1) = 44$ mots de 32 bits soit 11 clefs de 128 bits dérivées de la première clef. L'algorithme utilisé pour l'expansion de la clef fait intervenir deux fonctions `SubWord` et `RotWord` ainsi qu'une constante de tour `Rcon`.
La génération d'une fonction booléenne globale d'expansion de la clef est impossible du fait que la génération de la clef du tour n fait intervenir la clef du tour $n-1$. Cette imbrication des clefs de tour ne permet pas de générer une fonction booléenne globale. Par contre il est possible de générer une fonction booléenne correspondant au calcul d'une clef de tour.
Le premier mot w_{i_0} de la clef de tour i est calculé selon l'équation suivant :

$$w_{i_0} = (SW \circ RW(w_{(i-1)_3})) \oplus Rcon_i \oplus w_{(i-1)_0}$$

avec $SW()$ et $RW()$ correspondant respectivement aux fonctions `SubWord` et `RotWord`.
Les mots suivants w_{i_1}, w_{i_2} et w_{i_3} sont calculés selon l'équation suivante :

$$w_{i_n} = w_{i_{n-1}} \oplus w_{(i-1)_n}$$

avec $1 \leq n \leq 3$.
Les fonctions `SubWord` et `RotWord` sont construites sur le même principe que les fonctions `SubBytes` et `ShiftRows`, nous pouvons donc réutiliser la méthodologie précédente.
En langage python, la fonction de génération d'un mot s'écrit selon le code suivant (voir listing 16 p. 75).
Dans ce code, plusieurs cas de figure sont pris en compte.
La fonction `generateWord` prend en paramètre le numéro du mot à générer, nous savons que ce numéro est compris entre 0 et 43. Si le numéro est inférieur à 4, la fonction retourne la fonction booléenne identité puisque la première clef utilisée par l'AES est la clef de chiffrement. Si le numéro modulo 4 est nul, la fonction retourne une fonctionne booléenne décrivant la composition des fonctions `SubWord` et `RotWord` et l'application du `XOR` avec la constante `Rcon`. Enfin si le numéro modulo 4 est non nul, la fonction retourne la fonction booléenne décrivant le `XOR` avec le mot correspondant au tour précédent.

```python
def generateWord(num):
    if (num < 4):
        w = generateGenericWord(wordSize*num, 'x')
    if (num >= 4):
        if ((num % 4) == 0):
            w = generateWord(3)
            w = rotWord(w)
            w = subWord(w, rconList[(num/4)-1])
            w = xorWords(w, generateWord(0))
        else:
            w = generateWord(num-1)
            w = xorWords(w, generateWord(num%4))
    return w
```

Listing 16 – Fonction de génération d'un mot de clef en python

Nous avons maintenant une fonction booléenne décrivant un tour d'expansion de la clef. Comme nous l'avons vu plus haut, l'algorithme d'expansion de la clef fait intervenir au tour n les clefs du tour $n-1$. Pour pouvoir intégrer notre fonction booléenne dans le processus de chiffrement de l'AES, il faut, à chaque tour, ajouter une variable temporaire correspondant à la clef du tour précédent. À titre d'exemple, l'équation booléenne du bit b_0 du quatrième mot sur les 44 mots générés par le processus d'expansion de la clef, est donnée dans la figure 4.17 page 75.

$$w_4(k_0) = k_{109} \oplus k_{109}k_{111} \oplus k_{109}k_{110} \oplus k_{108}k_{109}k_{111} \oplus k_{108}k_{109}k_{110} \oplus k_{108}k_{109}k_{110}k_{111} \oplus k_{107} \oplus k_{107}k_{110}k_{111} \oplus k_{107}k_{109} \oplus k_{107}k_{109}k_{110}k_{111} \oplus k_{107}k_{108}k_{110}k_{111} \oplus k_{107}k_{108}k_{109}k_{110} \oplus k_{107}k_{108}k_{109}k_{110}k_{111} \oplus k_{106} \oplus k_{106}k_{110}k_{111} \oplus k_{106}k_{109}k_{111} \oplus k_{106}k_{109}k_{110}k_{111} \oplus k_{106}k_{108} \oplus k_{106}k_{108}k_{111} \oplus k_{106}k_{108}k_{110} \oplus k_{106}k_{108}k_{109} \oplus k_{106}k_{108}k_{109}k_{111} \oplus k_{106}k_{108}k_{109}k_{110} \oplus k_{106}k_{107}k_{111} \oplus k_{106}k_{107}k_{109}k_{110} \oplus k_{106}k_{107}k_{108} \oplus k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{111} \oplus k_{105}k_{110}k_{111} \oplus k_{105}k_{109} \oplus k_{105}k_{109}k_{110} \oplus k_{105}k_{108}k_{111} \oplus k_{105}k_{108}k_{110} \oplus k_{105}k_{108}k_{110}k_{111} \oplus k_{105}k_{108}k_{109}k_{111} \oplus k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{105}k_{107} \oplus k_{105}k_{107}k_{109} \oplus k_{105}k_{107}k_{109}k_{111} \oplus k_{105}k_{107}k_{109}k_{110} \oplus k_{105}k_{107}k_{109}k_{110}k_{111} \oplus k_{105}k_{107}k_{108}k_{111} \oplus k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{106}k_{111} \oplus k_{105}k_{106}k_{109} \oplus k_{105}k_{106}k_{108}k_{111} \oplus k_{105}k_{106}k_{108}k_{109}k_{110} \oplus k_{105}k_{106}k_{107} \oplus k_{105}k_{106}k_{107}k_{110}k_{111} \oplus k_{105}k_{106}k_{107}k_{109}k_{110} \oplus k_{105}k_{106}k_{107}k_{108} \oplus k_{105}k_{106}k_{107}k_{108}k_{111} \oplus k_{105}k_{106}k_{107}k_{108}k_{109} \oplus k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104} \oplus k_{104}k_{111} \oplus k_{104}k_{110} \oplus k_{104}k_{109}k_{111} \oplus k_{104}k_{109}k_{110}k_{111} \oplus k_{104}k_{108}k_{111} \oplus k_{104}k_{108}k_{109}k_{111} \oplus k_{104}k_{108}k_{109}k_{110} \oplus k_{104}k_{107}k_{110} \oplus k_{104}k_{107}k_{110}k_{111} \oplus k_{104}k_{107}k_{109}k_{111} \oplus k_{104}k_{107}k_{108}k_{111} \oplus k_{104}k_{107}k_{108}k_{110} \oplus k_{104}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{107}k_{108}k_{109} \oplus k_{104}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{106} \oplus k_{104}k_{106}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{108} \oplus k_{104}k_{106}k_{108}k_{111} \oplus k_{104}k_{106}k_{107} \oplus k_{104}k_{106}k_{107}k_{110} \oplus k_{104}k_{106}k_{107}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{111} \oplus k_{104}k_{105}k_{109} \oplus k_{104}k_{105}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{108}k_{111} \oplus k_{104}k_{105}k_{108}k_{110} \oplus k_{104}k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{107} \oplus k_{104}k_{105}k_{107}k_{111} \oplus k_{104}k_{105}k_{107}k_{110} \oplus k_{104}k_{105}k_{107}k_{109} \oplus k_{104}k_{105}k_{107}k_{109}k_{110} \oplus k_{104}k_{105}k_{107}k_{108}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{110} \oplus k_{104}k_{105}k_{106}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{109} \oplus k_{104}k_{105}k_{106}k_{109}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107} \oplus k_{104}k_{105}k_{106}k_{107}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_0$$

FIGURE 4.17 – Équation du bit de clef b_0 du $4^{\text{ème}}$ mot

4.4.1.5 Solution globale

Nous avons maintenant une fonction booléenne pour chacune des fonctions `SubBytes` $SB()$, `ShiftRows` $SR()$ et `MixColumns` $MC()$. Hors, dans l'agencement d'un tour, ces fonctions sont combinées. Ainsi, pour un bloc de 128 bits $B = (b_1, \cdots, b_{128})$ en sortie de la fonction `AddRoundKey`, le bloc $B' = (b'_1, \cdots, b'_{128})$ en sortie de la combinaison de ces trois fonctions est tel que :

$$B' = MC \circ SR \circ SB(B)$$

Pour pouvoir réaliser le même type de fichiers que ceux mis en œuvre pour le mini-AES, il est nécessaire de réduire la composition de ces trois fonctions en une seule équation booléenne. Pour y parvenir, il suffit de remplacer chaque variable en entrée d'une fonction par sa valeur en sortie de la fonction précédente selon l'équation suivante :

$$b'_i = MC(SR(SB(b_i))) \quad \forall i \in (1, \cdots, 128)$$

En langage python, la fonction de génération d'un tour s'écrit selon le code suivant (voir listing 17 p. 76).

```
1  def writeRoundEnc(numRound, equaSB, equaSR, equaMC):
2      printColor('## Round%s' % numRound, GREEN)
3      resultSR = []
4      resultMC = []
5      for i in xrange(blockSize):
6          equaSR[i] = equaSR[i].split('_')
7          resultSR.append(equaSB[int(equaSR[i][1])])
8
9      for i in xrange(blockSize):
10         tmp = ''
11         for monomial in equaMC[i].split('+'):
12             tmp += resultSR[int(monomial.split('_')[1])]
13             tmp += '+'
14         resultMC.append(tmp.rstrip('+'))
15     binMon = generateBinaryMonomes(resultMC)
16     return resultMC
```

Listing 17 – Calcul de l'équation pour une fonction de tour chiffrement

L'équation booléenne d'un tour de l'AES pour le bit b_0 est donnée dans la figure 4.18 page 77.
Finalement, nous pouvons maintenant décrire le processus intégral de chiffrement de l'AES sous la forme d'équations booléennes. La fonction en langage python calculant ce processus est donnée dans le listing 18 page 78.

4.4.2 Les équations pour les fonctions de déchiffrement

Nous allons maintenant détailler la solution mise en œuvre pour chacune des sous-fonctions de l'algorithme de déchiffrement de l'AES.

$$
\begin{aligned}
b'_0 = {} & 1 \oplus b_4 \oplus b_4b_6 \oplus b_4b_6b_7 \oplus b_4b_5 \oplus b_3b_7 \oplus b_3b_6b_7 \oplus b_3b_5b_7 \oplus b_3b_5b_6b_7 \oplus b_3b_4b_6 \oplus b_3b_4b_6b_7 \oplus b_3b_4b_5 \oplus \\
& b_3b_4b_5b_6 \oplus b_2 \oplus b_2b_7 \oplus b_2b_6b_7 \oplus b_2b_5b_7 \oplus b_2b_5b_6 \oplus b_2b_5b_6b_7 \oplus b_2b_4 \oplus b_2b_4b_7 \oplus b_2b_4b_5b_7 \oplus b_2b_4b_5b_6 \oplus \\
& b_2b_4b_5b_6b_7 \oplus b_2b_3b_7 \oplus b_2b_3b_5b_7 \oplus b_2b_3b_5b_6 \oplus b_2b_3b_4b_6b_7 \oplus b_2b_3b_4b_5 \oplus b_2b_3b_4b_5b_7 \oplus b_2b_3b_4b_5b_6 \oplus b_1 \oplus \\
& b_1b_5b_7 \oplus b_1b_5b_6 \oplus b_1b_4b_7 \oplus b_1b_4b_6 \oplus b_1b_4b_6b_7 \oplus b_1b_4b_5b_7 \oplus b_1b_4b_5b_6 \oplus b_1b_4b_5b_6b_7 \oplus b_1b_3 \oplus b_1b_3b_7 \oplus \\
& b_1b_3b_6 \oplus b_1b_3b_5 \oplus b_1b_3b_5b_7 \oplus b_1b_3b_4 \oplus b_1b_3b_4b_7 \oplus b_1b_3b_4b_6 \oplus b_1b_3b_4b_5 \oplus b_1b_2b_7 \oplus b_1b_2b_6 \oplus b_1b_2b_4b_5 \oplus \\
& b_1b_2b_4b_5b_7 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_1b_2b_3 \oplus b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_7 \oplus \\
& b_0b_6 \oplus b_0b_5b_6 \oplus b_0b_4 \oplus b_0b_4b_6b_7 \oplus b_0b_4b_5 \oplus b_0b_4b_5b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_3b_7 \oplus b_0b_3b_6 \oplus b_0b_3b_5 \oplus b_0b_3b_5b_6 \oplus \\
& b_0b_3b_4b_7 \oplus b_0b_3b_4b_6 \oplus b_0b_3b_4b_5b_6 \oplus b_0b_3b_4b_5b_6b_7 \oplus b_0b_2 \oplus b_0b_2b_7 \oplus b_0b_2b_5b_7 \oplus b_0b_2b_5b_6b_7 \oplus b_0b_2b_4 \oplus b_0b_2b_4b_6 \oplus \\
& b_0b_2b_4b_5 \oplus b_0b_2b_4b_5b_6 \oplus b_0b_2b_3b_6 \oplus b_0b_2b_3b_6b_7 \oplus b_0b_2b_3b_5b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_6 \oplus b_0b_2b_3b_4b_6b_7 \oplus \\
& b_0b_1b_6 \oplus b_0b_1b_6b_7 \oplus b_0b_1b_4 \oplus b_0b_1b_4b_6b_7 \oplus b_0b_1b_3b_7 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_3b_5b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_3b_4b_7 \oplus \\
& b_0b_1b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_5 \oplus b_0b_1b_2 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_1b_2b_5b_6 \oplus b_0b_1b_2b_5b_6b_7 \oplus b_0b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_4b_5 \oplus \\
& b_0b_1b_2b_3 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus 1 \oplus \\
& b_{44} \oplus b_{44}b_{46} \oplus b_{44}b_{46}b_{47} \oplus b_{44}b_{45} \oplus b_{43}b_{47} \oplus b_{43}b_{46}b_{47} \oplus b_{43}b_{45}b_{47} \oplus b_{43}b_{45}b_{46}b_{47} \oplus b_{43}b_{44}b_{46} \oplus \\
& b_{43}b_{44}b_{46}b_{47} \oplus b_{43}b_{44}b_{45} \oplus b_{43}b_{44}b_{45}b_{46} \oplus b_{42} \oplus b_{42}b_{47} \oplus b_{42}b_{46}b_{47} \oplus b_{42}b_{45}b_{47} \oplus b_{42}b_{45}b_{46} \oplus b_{42}b_{45}b_{46}b_{47} \oplus \\
& b_{42}b_{44} \oplus b_{42}b_{44}b_{47} \oplus b_{42}b_{44}b_{45}b_{47} \oplus b_{42}b_{44}b_{45}b_{46} \oplus b_{42}b_{44}b_{45}b_{46}b_{47} \oplus b_{42}b_{43}b_{47} \oplus b_{42}b_{43}b_{45}b_{47} \oplus \\
& b_{42}b_{43}b_{45}b_{46} \oplus b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{42}b_{43}b_{44}b_{45} \oplus b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{42}b_{43}b_{44}b_{45}b_{46} \oplus b_{41} \oplus b_{41}b_{45}b_{47} \oplus \\
& b_{41}b_{45}b_{46} \oplus b_{41}b_{44}b_{47} \oplus b_{41}b_{44}b_{46} \oplus b_{41}b_{44}b_{46}b_{47} \oplus b_{41}b_{44}b_{45}b_{47} \oplus b_{41}b_{44}b_{45}b_{46} \oplus b_{41}b_{44}b_{45}b_{46}b_{47} \oplus \\
& b_{41}b_{43} \oplus b_{41}b_{43}b_{47} \oplus b_{41}b_{43}b_{46} \oplus b_{41}b_{43}b_{45} \oplus b_{41}b_{43}b_{45}b_{47} \oplus b_{41}b_{43}b_{44} \oplus b_{41}b_{43}b_{44}b_{47} \oplus b_{41}b_{43}b_{44}b_{46} \oplus \\
& b_{41}b_{43}b_{44}b_{45} \oplus b_{41}b_{42}b_{47} \oplus b_{41}b_{42}b_{46} \oplus b_{41}b_{42}b_{44}b_{45} \oplus b_{41}b_{42}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{44}b_{46}b_{47} \oplus b_{41}b_{42}b_{43} \oplus \\
& b_{41}b_{42}b_{43}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45} \oplus \\
& b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{40} \oplus b_{40}b_{47} \oplus b_{40}b_{46} \oplus b_{40}b_{46}b_{47} \oplus \\
& b_{40}b_{45} \oplus b_{40}b_{45}b_{47} \oplus b_{40}b_{45}b_{46} \oplus b_{40}b_{45}b_{46}b_{47} \oplus b_{40}b_{44} \oplus b_{40}b_{44}b_{47} \oplus b_{40}b_{44}b_{46} \oplus b_{40}b_{44}b_{46}b_{47} \oplus b_{40}b_{44}b_{45} \oplus \\
& b_{40}b_{44}b_{45}b_{47} \oplus b_{40}b_{44}b_{45}b_{46} \oplus b_{40}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{43} \oplus b_{40}b_{43}b_{47} \oplus b_{40}b_{43}b_{46} \oplus b_{40}b_{43}b_{46}b_{47} \oplus \\
& b_{40}b_{43}b_{45} \oplus b_{40}b_{43}b_{45}b_{46} \oplus b_{40}b_{43}b_{44}b_{47} \oplus b_{40}b_{43}b_{44}b_{46} \oplus b_{40}b_{43}b_{44}b_{45}b_{46} \oplus b_{40}b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{42} \oplus \\
& b_{40}b_{42}b_{47} \oplus b_{40}b_{42}b_{45}b_{47} \oplus b_{40}b_{42}b_{44} \oplus b_{40}b_{42}b_{44}b_{46} \oplus b_{40}b_{42}b_{44}b_{45} \oplus b_{40}b_{42}b_{44}b_{45}b_{46} \oplus b_{40}b_{42}b_{43}b_{46} \oplus \\
& b_{40}b_{42}b_{43}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{46} \oplus b_{40}b_{41}b_{46}b_{47} \oplus b_{40}b_{41}b_{44} \oplus b_{40}b_{41}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{47} \oplus b_{40}b_{41}b_{43}b_{46} \oplus b_{40}b_{41}b_{43}b_{45}b_{47} \oplus \\
& b_{40}b_{41}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{45} \oplus b_{40}b_{41}b_{42} \oplus \\
& b_{40}b_{41}b_{42}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{45}b_{46} \oplus b_{40}b_{41}b_{42}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{45} \oplus \\
& b_{40}b_{41}b_{42}b_{43} \oplus b_{40}b_{41}b_{42}b_{43}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{40}b_{41}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{45} \oplus b_{45}b_{47} \oplus b_{45}b_{46} \oplus b_{44}b_{45}b_{47} \oplus b_{44}b_{45}b_{46} \oplus b_{44}b_{45}b_{46}b_{47} \oplus \\
& b_{43} \oplus b_{43}b_{46}b_{47} \oplus b_{43}b_{45} \oplus b_{43}b_{45}b_{46}b_{47} \oplus b_{43}b_{44}b_{46}b_{47} \oplus b_{43}b_{44}b_{45}b_{46} \oplus b_{43}b_{44}b_{45}b_{46}b_{47} \oplus b_{42} \oplus b_{42}b_{46}b_{47} \oplus \\
& b_{42}b_{45}b_{47} \oplus b_{42}b_{45}b_{46}b_{47} \oplus b_{42}b_{44} \oplus b_{42}b_{44}b_{47} \oplus b_{42}b_{44}b_{46} \oplus b_{42}b_{44}b_{45} \oplus b_{42}b_{44}b_{45}b_{47} \oplus b_{42}b_{44}b_{45}b_{46} \oplus \\
& b_{42}b_{43}b_{47} \oplus b_{42}b_{43}b_{45}b_{46} \oplus b_{42}b_{43}b_{44} \oplus b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{41} \oplus b_{41}b_{46}b_{47} \oplus b_{41}b_{45} \oplus \\
& b_{41}b_{45}b_{46} \oplus b_{41}b_{44}b_{47} \oplus b_{41}b_{44}b_{46} \oplus b_{41}b_{44}b_{46}b_{47} \oplus b_{41}b_{44}b_{45}b_{47} \oplus b_{41}b_{44}b_{45}b_{46}b_{47} \oplus b_{41}b_{43} \oplus b_{41}b_{43}b_{45} \oplus \\
& b_{41}b_{43}b_{45}b_{47} \oplus b_{41}b_{43}b_{45}b_{46} \oplus b_{41}b_{43}b_{45}b_{46}b_{47} \oplus b_{41}b_{43}b_{44}b_{47} \oplus b_{41}b_{43}b_{44}b_{45}b_{47} \oplus b_{41}b_{42}b_{47} \oplus b_{41}b_{42}b_{45} \oplus \\
& b_{41}b_{42}b_{44}b_{47} \oplus b_{41}b_{42}b_{44}b_{45}b_{46} \oplus b_{41}b_{42}b_{43} \oplus b_{41}b_{42}b_{43}b_{46}b_{47} \oplus b_{41}b_{42}b_{43}b_{45}b_{46} \oplus b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{41}b_{42}b_{43}b_{44}b_{47} \oplus b_{41}b_{42}b_{43}b_{44}b_{45} \oplus b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{40} \oplus b_{40}b_{47} \oplus b_{40}b_{46} \oplus b_{40}b_{45}b_{47} \oplus b_{40}b_{45}b_{46}b_{47} \oplus \\
& b_{40}b_{44}b_{47} \oplus b_{40}b_{44}b_{45}b_{47} \oplus b_{40}b_{44}b_{45}b_{46} \oplus b_{40}b_{43}b_{46} \oplus b_{40}b_{43}b_{46}b_{47} \oplus b_{40}b_{43}b_{45}b_{47} \oplus b_{40}b_{43}b_{44}b_{47} \oplus \\
& b_{40}b_{43}b_{44}b_{46} \oplus b_{40}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{43}b_{44}b_{45} \oplus b_{40}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{42} \oplus b_{40}b_{42}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{44} \oplus \\
& b_{40}b_{42}b_{44}b_{47} \oplus b_{40}b_{42}b_{43} \oplus b_{40}b_{42}b_{43}b_{46} \oplus b_{40}b_{42}b_{43}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{45}b_{46}b_{47} \oplus b_{40}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{47} \oplus b_{40}b_{41}b_{45} \oplus b_{40}b_{41}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{44}b_{47} \oplus b_{40}b_{41}b_{44}b_{46} \oplus \\
& b_{40}b_{41}b_{44}b_{45}b_{46}b_{47} \oplus b_{40}b_{41}b_{43} \oplus b_{40}b_{41}b_{43}b_{47} \oplus b_{40}b_{41}b_{43}b_{46} \oplus b_{40}b_{41}b_{43}b_{45} \oplus b_{40}b_{41}b_{43}b_{45}b_{46} \oplus \\
& b_{40}b_{41}b_{43}b_{44}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{43}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{46} \oplus b_{40}b_{41}b_{42}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{42}b_{45} \oplus b_{40}b_{41}b_{42}b_{45}b_{46} \oplus b_{40}b_{41}b_{42}b_{44}b_{47} \oplus b_{40}b_{41}b_{42}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{44}b_{46}b_{47} \oplus \\
& b_{40}b_{41}b_{42}b_{44}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{43} \oplus b_{40}b_{41}b_{42}b_{43}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{45}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44} \oplus \\
& b_{40}b_{41}b_{42}b_{43}b_{44}b_{46} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{46}b_{47} \oplus b_{40}b_{41}b_{42}b_{43}b_{44}b_{45}b_{47} \oplus b_{85} \oplus b_{85}b_{87} \oplus b_{85}b_{86} \oplus b_{84}b_{85}b_{87} \oplus \\
& b_{84}b_{85}b_{86} \oplus b_{84}b_{85}b_{86}b_{87} \oplus b_{83} \oplus b_{83}b_{86}b_{87} \oplus b_{83}b_{85} \oplus b_{83}b_{85}b_{86}b_{87} \oplus b_{83}b_{84}b_{86}b_{87} \oplus b_{83}b_{84}b_{85}b_{86} \oplus \\
& b_{83}b_{84}b_{85}b_{86}b_{87} \oplus b_{82} \oplus b_{82}b_{86}b_{87} \oplus b_{82}b_{85}b_{87} \oplus b_{82}b_{85}b_{86}b_{87} \oplus b_{82}b_{84} \oplus b_{82}b_{84}b_{87} \oplus b_{82}b_{84}b_{86} \oplus b_{82}b_{84}b_{85} \oplus \\
& b_{82}b_{84}b_{85}b_{87} \oplus b_{82}b_{84}b_{85}b_{86} \oplus b_{82}b_{83}b_{87} \oplus b_{82}b_{83}b_{85}b_{86} \oplus b_{82}b_{83}b_{84} \oplus b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{82}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{81}b_{87} \oplus b_{81}b_{86}b_{87} \oplus b_{81}b_{85} \oplus b_{81}b_{85}b_{86} \oplus b_{81}b_{84}b_{87} \oplus b_{81}b_{84}b_{86} \oplus b_{81}b_{84}b_{86}b_{87} \oplus b_{81}b_{84}b_{85}b_{87} \oplus \\
& b_{81}b_{84}b_{85}b_{86}b_{87} \oplus b_{81}b_{83} \oplus b_{81}b_{83}b_{85} \oplus b_{81}b_{83}b_{85}b_{87} \oplus b_{81}b_{83}b_{85}b_{86} \oplus b_{81}b_{83}b_{85}b_{86}b_{87} \oplus b_{81}b_{83}b_{84}b_{87} \oplus \\
& b_{81}b_{83}b_{84}b_{85}b_{87} \oplus b_{81}b_{82}b_{87} \oplus b_{81}b_{82}b_{85} \oplus b_{81}b_{82}b_{84}b_{87} \oplus b_{81}b_{82}b_{84}b_{85}b_{86} \oplus b_{81}b_{82}b_{83} \oplus b_{81}b_{82}b_{83}b_{86}b_{87} \oplus \\
& b_{81}b_{82}b_{83}b_{85}b_{86} \oplus b_{81}b_{82}b_{83}b_{84} \oplus b_{81}b_{82}b_{83}b_{84}b_{87} \oplus b_{81}b_{82}b_{83}b_{84}b_{85} \oplus b_{81}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{80} \oplus b_{80}b_{87} \oplus \\
& b_{80}b_{86} \oplus b_{80}b_{85}b_{87} \oplus b_{80}b_{85}b_{86}b_{87} \oplus b_{80}b_{84}b_{87} \oplus b_{80}b_{84}b_{85}b_{87} \oplus b_{80}b_{84}b_{85}b_{86} \oplus b_{80}b_{83}b_{86} \oplus b_{80}b_{83}b_{86}b_{87} \oplus \\
& b_{80}b_{83}b_{85}b_{87} \oplus b_{80}b_{83}b_{84}b_{87} \oplus b_{80}b_{83}b_{84}b_{86} \oplus b_{80}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{83}b_{84}b_{85} \oplus b_{80}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{80}b_{82} \oplus b_{80}b_{82}b_{85}b_{86}b_{87} \oplus b_{80}b_{82}b_{84} \oplus b_{80}b_{82}b_{84}b_{87} \oplus b_{80}b_{82}b_{83} \oplus b_{80}b_{82}b_{83}b_{86} \oplus b_{80}b_{82}b_{83}b_{86}b_{87} \oplus \\
& b_{80}b_{82}b_{83}b_{85}b_{86}b_{87} \oplus b_{80}b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{80}b_{81}b_{87} \oplus b_{80}b_{81}b_{85} \oplus b_{80}b_{81}b_{85}b_{86}b_{87} \oplus \\
& b_{80}b_{81}b_{84}b_{87} \oplus b_{80}b_{81}b_{84}b_{86} \oplus b_{80}b_{81}b_{84}b_{85}b_{86}b_{87} \oplus b_{80}b_{81}b_{83} \oplus b_{80}b_{81}b_{83}b_{87} \oplus b_{80}b_{81}b_{83}b_{86} \oplus \\
& b_{80}b_{81}b_{83}b_{85} \oplus b_{80}b_{81}b_{83}b_{85}b_{86} \oplus b_{80}b_{81}b_{83}b_{84}b_{87} \oplus b_{80}b_{81}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{83}b_{84}b_{85}b_{87} \oplus \\
& b_{80}b_{81}b_{82}b_{86} \oplus b_{80}b_{81}b_{82}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{85} \oplus b_{80}b_{81}b_{82}b_{85}b_{86} \oplus b_{80}b_{81}b_{82}b_{84}b_{87} \oplus b_{80}b_{81}b_{82}b_{84}b_{86} \oplus \\
& b_{80}b_{81}b_{82}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{84}b_{85}b_{87} \oplus b_{80}b_{81}b_{82}b_{83} \oplus b_{80}b_{81}b_{82}b_{83}b_{86} \oplus b_{80}b_{81}b_{82}b_{83}b_{85}b_{87} \oplus \\
& b_{80}b_{81}b_{82}b_{83}b_{84} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{86} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{86}b_{87} \oplus b_{80}b_{81}b_{82}b_{83}b_{84}b_{85}b_{87} \oplus b_{125} \oplus \\
& b_{125}b_{127} \oplus b_{125}b_{126} \oplus b_{124}b_{125}b_{127} \oplus b_{124}b_{125}b_{126} \oplus b_{124}b_{125}b_{126}b_{127} \oplus b_{123} \oplus b_{123}b_{126}b_{127} \oplus b_{123}b_{125} \oplus \\
& b_{123}b_{125}b_{126}b_{127} \oplus b_{123}b_{124}b_{126}b_{127} \oplus b_{123}b_{124}b_{125}b_{126} \oplus b_{123}b_{124}b_{125}b_{126}b_{127} \oplus b_{122} \oplus b_{122}b_{126}b_{127} \oplus \\
& b_{122}b_{125}b_{127} \oplus b_{122}b_{125}b_{126}b_{127} \oplus b_{122}b_{124} \oplus b_{122}b_{124}b_{127} \oplus b_{122}b_{124}b_{126} \oplus b_{122}b_{124}b_{125} \oplus b_{122}b_{124}b_{125}b_{127} \oplus \\
& b_{122}b_{124}b_{125}b_{126} \oplus b_{122}b_{123}b_{127} \oplus b_{122}b_{123}b_{125}b_{126} \oplus b_{122}b_{123}b_{124} \oplus b_{122}b_{123}b_{124}b_{126}b_{127} \oplus \\
& b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{127} \oplus b_{121}b_{126}b_{127} \oplus b_{121}b_{125} \oplus b_{121}b_{125}b_{126} \oplus b_{121}b_{124}b_{127} \oplus b_{121}b_{124}b_{126} \oplus \\
& b_{121}b_{124}b_{126}b_{127} \oplus b_{121}b_{124}b_{125}b_{127} \oplus b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{121}b_{123} \oplus b_{121}b_{123}b_{125} \oplus b_{121}b_{123}b_{125}b_{127} \oplus \\
& b_{121}b_{123}b_{125}b_{126} \oplus b_{121}b_{123}b_{125}b_{126}b_{127} \oplus b_{121}b_{123}b_{124}b_{127} \oplus b_{121}b_{123}b_{124}b_{125}b_{127} \oplus b_{121}b_{122}b_{127} \oplus \\
& b_{121}b_{122}b_{125} \oplus b_{121}b_{122}b_{124}b_{127} \oplus b_{121}b_{122}b_{124}b_{125}b_{126} \oplus b_{121}b_{122}b_{123} \oplus b_{121}b_{122}b_{123}b_{126}b_{127} \oplus \\
& b_{121}b_{122}b_{123}b_{125}b_{126} \oplus b_{121}b_{122}b_{123}b_{124} \oplus b_{121}b_{122}b_{123}b_{124}b_{127} \oplus b_{121}b_{122}b_{123}b_{124}b_{125} \oplus \\
& b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus b_{120} \oplus b_{120}b_{127} \oplus b_{120}b_{126} \oplus b_{120}b_{125}b_{127} \oplus b_{120}b_{125}b_{126}b_{127} \oplus b_{120}b_{124}b_{127} \oplus \\
& b_{120}b_{124}b_{125}b_{127} \oplus b_{120}b_{124}b_{125}b_{126} \oplus b_{120}b_{123}b_{126} \oplus b_{120}b_{123}b_{126}b_{127} \oplus b_{120}b_{123}b_{125}b_{127} \oplus \\
& b_{120}b_{123}b_{124}b_{127} \oplus b_{120}b_{123}b_{124}b_{126} \oplus b_{120}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{123}b_{124}b_{125} \oplus b_{120}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{122} \oplus b_{120}b_{122}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{124} \oplus b_{120}b_{122}b_{124}b_{127} \oplus b_{120}b_{122}b_{123} \oplus b_{120}b_{122}b_{123}b_{126} \oplus \\
& b_{120}b_{122}b_{123}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{125}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{127} \oplus b_{120}b_{121}b_{125} \oplus b_{120}b_{121}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{124}b_{127} \oplus b_{120}b_{121}b_{124}b_{126} \oplus \\
& b_{120}b_{121}b_{124}b_{125}b_{126}b_{127} \oplus b_{120}b_{121}b_{123} \oplus b_{120}b_{121}b_{123}b_{127} \oplus b_{120}b_{121}b_{123}b_{126} \oplus b_{120}b_{121}b_{123}b_{125} \oplus \\
& b_{120}b_{121}b_{123}b_{125}b_{126} \oplus b_{120}b_{121}b_{123}b_{124}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{123}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{122}b_{126} \oplus b_{120}b_{121}b_{122}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{125} \oplus b_{120}b_{121}b_{122}b_{125}b_{126} \oplus \\
& b_{120}b_{121}b_{122}b_{124}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{124}b_{125}b_{127} \oplus \\
& b_{120}b_{121}b_{122}b_{123} \oplus b_{120}b_{121}b_{122}b_{123}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{125}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{124} \oplus \\
& b_{120}b_{121}b_{122}b_{123}b_{124}b_{126} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{126}b_{127} \oplus b_{120}b_{121}b_{122}b_{123}b_{124}b_{125}b_{127} \oplus k_{109} \oplus \\
& k_{109}k_{111} \oplus k_{109}k_{110} \oplus k_{108}k_{109}k_{111} \oplus k_{108}k_{109}k_{110} \oplus k_{108}k_{109}k_{110}k_{111} \oplus k_{107} \oplus k_{107}k_{110}k_{111} \oplus \\
& k_{107}k_{109} \oplus k_{107}k_{109}k_{110}k_{111} \oplus k_{107}k_{108}k_{110}k_{111} \oplus k_{107}k_{108}k_{109}k_{110} \oplus k_{107}k_{108}k_{109}k_{110}k_{111} \oplus k_{106} \oplus \\
& k_{106}k_{110}k_{111} \oplus k_{106}k_{109}k_{111} \oplus k_{106}k_{109}k_{110}k_{111} \oplus k_{106}k_{108} \oplus k_{106}k_{108}k_{111} \oplus k_{106}k_{108}k_{110} \oplus \\
& k_{106}k_{108}k_{109} \oplus k_{106}k_{108}k_{109}k_{111} \oplus k_{106}k_{108}k_{109}k_{110} \oplus k_{106}k_{107}k_{111} \oplus k_{106}k_{107}k_{109}k_{110} \oplus k_{106}k_{107}k_{108} \oplus \\
& k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{111} \oplus k_{105}k_{110}k_{111} \oplus k_{105}k_{109} \oplus k_{105}k_{109}k_{110} \oplus \\
& k_{105}k_{108}k_{111} \oplus k_{105}k_{108}k_{110} \oplus k_{105}k_{108}k_{110}k_{111} \oplus k_{105}k_{108}k_{109}k_{111} \oplus k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{105}k_{107} \oplus \\
& k_{105}k_{107}k_{109} \oplus k_{105}k_{107}k_{109}k_{111} \oplus k_{105}k_{107}k_{109}k_{110} \oplus k_{105}k_{107}k_{109}k_{110}k_{111} \oplus k_{105}k_{107}k_{108}k_{111} \oplus \\
& k_{105}k_{107}k_{108}k_{109}k_{111} \oplus k_{105}k_{106}k_{111} \oplus k_{105}k_{106}k_{109} \oplus k_{105}k_{106}k_{108}k_{111} \oplus k_{105}k_{106}k_{108}k_{109}k_{110} \oplus \\
& k_{105}k_{106}k_{107} \oplus k_{105}k_{106}k_{107}k_{110}k_{111} \oplus k_{105}k_{106}k_{107}k_{109}k_{110} \oplus k_{105}k_{106}k_{107}k_{108} \oplus \\
& k_{105}k_{106}k_{107}k_{108}k_{111} \oplus k_{105}k_{106}k_{107}k_{108}k_{109} \oplus k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104} \oplus k_{104}k_{111} \oplus \\
& k_{104}k_{110} \oplus k_{104}k_{109}k_{111} \oplus k_{104}k_{109}k_{110}k_{111} \oplus k_{104}k_{108}k_{111} \oplus k_{104}k_{108}k_{109}k_{111} \oplus k_{104}k_{108}k_{109}k_{110} \oplus \\
& k_{104}k_{107}k_{110} \oplus k_{104}k_{107}k_{110}k_{111} \oplus k_{104}k_{107}k_{109}k_{111} \oplus k_{104}k_{107}k_{108}k_{111} \oplus k_{104}k_{107}k_{108}k_{110} \oplus \\
& k_{104}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{107}k_{108}k_{109} \oplus k_{104}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{106} \oplus k_{104}k_{106}k_{109}k_{110}k_{111} \oplus \\
& k_{104}k_{106}k_{108} \oplus k_{104}k_{106}k_{108}k_{111} \oplus k_{104}k_{106}k_{107} \oplus k_{104}k_{106}k_{107}k_{110} \oplus k_{104}k_{106}k_{107}k_{110}k_{111} \oplus \\
& k_{104}k_{106}k_{107}k_{109}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_{104}k_{105}k_{111} \oplus \\
& k_{104}k_{105}k_{109} \oplus k_{104}k_{105}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{108}k_{111} \oplus k_{104}k_{105}k_{108}k_{110} \oplus \\
& k_{104}k_{105}k_{108}k_{109}k_{110}k_{111} \oplus k_{104}k_{105}k_{107} \oplus k_{104}k_{105}k_{107}k_{111} \oplus k_{104}k_{105}k_{107}k_{110} \oplus k_{104}k_{105}k_{107}k_{109} \oplus \\
& k_{104}k_{105}k_{107}k_{109}k_{110} \oplus k_{104}k_{105}k_{107}k_{108}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{107}k_{108}k_{109}k_{111} \oplus \\
& k_{104}k_{105}k_{106}k_{110} \oplus k_{104}k_{105}k_{106}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{109} \oplus k_{104}k_{105}k_{106}k_{109}k_{110} \oplus \\
& k_{104}k_{105}k_{106}k_{108}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{108}k_{109}k_{111} \oplus \\
& k_{104}k_{105}k_{106}k_{107} \oplus k_{104}k_{105}k_{106}k_{107}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{109}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108} \oplus \\
& k_{104}k_{105}k_{106}k_{107}k_{108}k_{110} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{110}k_{111} \oplus k_{104}k_{105}k_{106}k_{107}k_{108}k_{109}k_{111} \oplus k_0
\end{aligned}
$$

FIGURE 4.18 – Équation du bit b_0 pour un tour

```
def generateEncFullFiles():
    printColor('## Ciphering process', YELLOW)
    createAESFiles('enc')
    addRoundKey(0, 'enc')
    writeRoundEnc(0, subBytes(), shiftRows(), mixColumns())
    addRoundKey(1, 'enc')
    writeRoundEnc(1, subBytes(), shiftRows(), mixColumns())
    addRoundKey(2, 'enc')
    writeRoundEnc(2, subBytes(), shiftRows(), mixColumns())
    addRoundKey(3, 'enc')
    writeRoundEnc(3, subBytes(), shiftRows(), mixColumns())
    addRoundKey(4, 'enc')
    writeRoundEnc(4, subBytes(), shiftRows(), mixColumns())
    addRoundKey(5, 'enc')
    writeRoundEnc(5, subBytes(), shiftRows(), mixColumns())
    addRoundKey(6, 'enc')
    writeRoundEnc(6, subBytes(), shiftRows(), mixColumns())
    addRoundKey(7, 'enc')
    writeRoundEnc(7, subBytes(), shiftRows(), mixColumns())
    addRoundKey(8, 'enc')
    writeRoundEnc(8, subBytes(), shiftRows(), mixColumns())
    addRoundKey(9, 'enc')
    writeFinalRoundEnc(9, subBytes(), shiftRows())
    addRoundKey(10, 'enc')
    writeEndFlag('enc')
    printColor('## Files generated', YELLOW)
```

Listing 18 – Calcul des fonctions booléennes du processus de chiffrement de l'AES

4.4.2.1 Solution pour la fonction de tour

L'algorithme de déchiffrement de l'AES utilise les fonctions `InvShiftRows`, `InvSubBytes` et `InvMixColumns`. Ces fonctions sont respectivement les fonctions inverses des fonctions `ShiftRows`, `SubBytes` et `MixColumns` utilisées dans le processus de chiffrement. Le pseudo code de la fonction de déchiffrement peut s'écrire de la façon suivante (voir fig. 4.19 p. 79), `Nb` correspondant au nombre de mots de 32 bits et `Nr` au nombre de tours utilisés dans l'algorithme.
Les mécanismes internes aux trois fonctions utilisées dans le tour lors du déchiffrement sont similaires à ceux des fonctions de chiffrement. Nous utilisons donc le même raisonnement que celui mis en œuvre plus haut pour générer les équations booléennes correspondantes.
À titre d'exemple, les équations booléennes des trois transformations utilisées dans le processus de déchiffrement pour le bit b_0 sont données dans la figure 4.20 page 79.

4.4.2.2 Solution pour la fonction d'expansion de la clef

La fonction d'expansion de la clef est la même pour les processus de chiffrement et de déchiffrement. Les équations booléennes que nous avons construites précédemment sont donc réutilisables.

```
1: function InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2:     byte state[4,Nb]
3:     state ← in
4:     AddRounkey(state, w[Nr*Nb, (Nr+1)*Nb-1])
5:     for round=Nr-1 step -1 downto 1 do
6:         InvShiftRows(state)
7:         InvSubBytes(state)
8:         AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
9:         InvMixColumns(state)
10:    end for
11:    InvShiftRows(state)
12:    InvSubBytes(state)
13:    AddRounkey(state, w[0, Nb-1])
14:    return state
15: end function
```

FIGURE 4.19 – Pseudo-code pour le déchiffrement

$$\text{invSubBytes}(b_0) = b_6b_7 \oplus b_5b_6 \oplus b_4 \oplus b_4b_7 \oplus b_4b_5b_7 \oplus b_4b_5b_6 \oplus b_4b_5b_6b_7 \oplus b_3b_7 \oplus b_3b_6b_7 \oplus b_3b_5 \oplus b_3b_5b_6 \oplus b_3b_5b_6b_7 \oplus b_3b_4 \oplus b_3b_4b_7 \oplus b_3b_4b_6b_7 \oplus b_3b_4b_5b_6 \oplus b_3b_4b_5b_6b_7 \oplus b_2b_6 \oplus b_2b_5 \oplus b_2b_5b_6 \oplus b_2b_5b_6b_7 \oplus b_2b_4b_6 \oplus b_2b_4b_6b_7 \oplus b_2b_4b_5b_7 \oplus b_2b_3b_7 \oplus b_2b_3b_6b_7 \oplus b_2b_3b_5b_7 \oplus b_2b_3b_5b_6b_7 \oplus b_2b_3b_4b_6 \oplus b_2b_3b_4b_5b_6 \oplus b_1b_7 \oplus b_1b_6 \oplus b_1b_6b_7 \oplus b_1b_5 \oplus b_1b_4b_6b_7 \oplus b_1b_4b_5b_7 \oplus b_1b_3b_6 \oplus b_1b_3b_6b_7 \oplus b_1b_3b_5 \oplus b_1b_3b_5b_6b_7 \oplus b_1b_2 \oplus b_1b_2b_7 \oplus b_1b_2b_6b_7 \oplus b_1b_2b_5b_6b_7 \oplus b_1b_2b_4 \oplus b_1b_2b_4b_7 \oplus b_1b_2b_4b_6b_7 \oplus b_1b_2b_4b_5b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_1b_2b_3b_7 \oplus b_1b_2b_3b_5b_7 \oplus b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_4 \oplus b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_1b_2b_3b_4b_5 \oplus b_1b_2b_3b_4b_5b_6 \oplus b_0b_7 \oplus b_0b_5b_7 \oplus b_0b_5b_6b_7 \oplus b_0b_4b_6b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_3 \oplus b_0b_3b_6 \oplus b_0b_3b_5 \oplus b_0b_3b_5b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_0b_3b_4b_6b_7 \oplus b_0b_3b_4b_5 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_2b_6 \oplus b_0b_2b_5 \oplus b_0b_2b_5b_6 \oplus b_0b_2b_5b_6b_7 \oplus b_0b_2b_4 \oplus b_0b_2b_4b_7 \oplus b_0b_2b_4b_6 \oplus b_0b_2b_4b_5 \oplus b_0b_2b_4b_5b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_5 \oplus b_0b_2b_3b_4b_5b_6 \oplus b_0b_1b_7 \oplus b_0b_1b_5b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_1b_4b_7 \oplus b_0b_1b_4b_6b_7 \oplus b_0b_1b_4b_5b_6b_7 \oplus b_0b_1b_3 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_3b_6b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_1b_2b_5 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_1b_2b_4 \oplus b_0b_1b_2b_4b_6 \oplus b_0b_1b_2b_4b_5 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_2b_4b_5b_6 \oplus b_0b_1b_2b_3 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_2b_3b_5b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_5$$

$$\text{invShiftRows}(b_0) = b_0$$

$$\text{invMixColumns}(b_0) = b_3 \oplus b_2 \oplus b_1 \oplus b_{11} \oplus b_9 \oplus b_8 \oplus b_{19} \oplus b_{18} \oplus b_{16} \oplus b_{27} \oplus b_{24}$$

FIGURE 4.20 – Équations booléennes des trois fonctions de déchiffrement pour le bit b_0

4.4.2.3 Solution globale

Nous avons maintenant une fonction booléenne pour chacune des fonctions `InvSubBytes` $ISB()$, `InvShiftRows` $ISR()$ et `InvMixColumns` $IMC()$. Cependant, contrairement à l'agencement des tours intermédiaires du processus de chiffrement, ces trois fonctions ne sont pas combinées entres elles. En effet, la fonction `AddRoundKey` intervient non plus en fin de tour mais s'intercale entre les fonctions `InvSubBytes` et `InvMixColumns`.
Ainsi, pour un bloc $B = (b_1, \cdots, b_{128})$ et une clef $K = (k_1, \cdots, k_{128})$ en entrée du tour, le bloc $B' = (b'_1, \cdots, b'_{128})$ en sortie est tel que :

$$B' = IMC(ISB \circ ISR(B) \oplus AD(K))$$

Afin de réduire les équations booléennes, nous n'allons donc pouvoir combiner que les équations de `InvSubBytes` et `InvShiftRows`. Comme précédemment, pour

y parvenir il suffit de remplacer chaque variable en entrée d'une fonction par sa valeur en sortie de la fonction précédente selon l'équation suivante :

$$b'_i = ISB(ISR(b_i)) \quad \forall i \in (1, \cdots, 128)$$

En langage python, la fonction de génération d'un tour s'écrit selon le code suivant (voir listing 19 p. 80).

```
1 def writeRoundDec(numRound, equaSB, equaSR):
2     printColor('## Round %s' % numRound, GREEN)
3     resultSR = []
4     for i in xrange(blockSize):
5         equaSR[i] = equaSR[i].split('_')
6         resultSR.append(equaSB[int(equaSR[i][1])])
7     binMon = generateBinaryMonomes(resultSR)
8     return resultSR
```

Listing 19 – Calcul de l'équation pour une fonction de tour de déchiffrement

Comme pour le processus de chiffrement, nous pouvons maintenant décrire le processus intégral de déchiffrement de l'AES sous la forme d'équations booléennes. La fonction en langage python calculant ce processus est donnée dans le listing 20 page 81.

4.4.3 Implémentation et preuve

Nous disposons maintenant de deux systèmes d'équations booléennes correspondants aux processus de chiffrement et de déchiffrement de l'AES. Ces deux systèmes comptent chacun :

- 128 équations, une par bit de bloc ;
- 1280 variables pour le bloc en entrée ;
- 1280 variables pour la clef.

Concernant les variables de clefs, le fait que nous ayons une équation booléenne par clef de tour implique que nous ayons un jeu de 128 nouvelles variables à chaque tour soit 1280 variables pour l'AES 128. Chacune des variables de clef du tour n étant décrite en fonction des variables de clef du tour $n-1$. En conséquence et du fait de l'opération de `XOR` entre la clef de tour et les bits issus de la fonction de tour, nous sommes obligés d'insérer un nouveau jeu de 128 variables pour décrire l'évolution du bloc de données à chaque tour.

Finalement nous avons décrit les processus de chiffrement et de déchiffrement de l'AES sous la forme de deux systèmes d'équations booléennes à 128 équations et 2560 variables. À titre de comparaison nous rappelons, dans la figure 4.21 page 81, les résultats obtenus avec les systèmes d'équations présentés au début de cette thèse.

Ce mécanisme nous permet alors de décrire l'ensemble du processus de chiffrement et de déchiffrement de l'AES sous la forme de fichiers en utilisant la même représentation que pour le mini-AES. Pour chaque processus de chiffrement et de déchiffrement, nous avons donc 128 fichiers, un par bit de bloc. Dans ces fichiers, chaque ligne décrit un monôme et le passage d'une ligne à la suivante

```
def generateDecFullFiles():
    printColor('## Deciphering process', YELLOW)
    createAESFiles('dec')
    addRoundKey(10, 'dec')
    writeRoundDec(9, invSubBytes(), invShiftRows())
    addRoundKey(9, 'dec')
    writeInvMixColumns(9)
    writeRoundDec(8, invSubBytes(), invShiftRows())
    addRoundKey(8, 'dec')
    writeInvMixColumns(8)
    writeRoundDec(7, invSubBytes(), invShiftRows())
    addRoundKey(7, 'dec')
    writeInvMixColumns(7)
    writeRoundDec(6, invSubBytes(), invShiftRows())
    addRoundKey(6, 'dec')
    writeInvMixColumns(6)
    writeRoundDec(5, invSubBytes(), invShiftRows())
    addRoundKey(5, 'dec')
    writeInvMixColumns(5)
    writeRoundDec(4, invSubBytes(), invShiftRows())
    addRoundKey(4, 'dec')
    writeInvMixColumns(4)
    writeRoundDec(3, invSubBytes(), invShiftRows())
    addRoundKey(3, 'dec')
    writeInvMixColumns(3)
    writeRoundDec(2, invSubBytes(), invShiftRows())
    addRoundKey(2, 'dec')
    writeInvMixColumns(2)
    writeRoundDec(1, invSubBytes(), invShiftRows())
    addRoundKey(1, 'dec')
    writeInvMixColumns(1)
    writeRoundDec(0, invSubBytes(), invShiftRows())
    addRoundKey(0, 'dec')
    writeEndFlag('dec')
    printColor('## Files generated', YELLOW)
```

Listing 20 – Calcul des fonctions booléennes du processus de déchiffrement de l'AES

Source	Nombre d'équations	Nombre de variables	Nombre de termes
Nicolas Courtois et Joseph Pieprzyk [63]	8000	1600	2^{20}
Sean Murphy et Matthew Robshaw [147]	2688	3968	5248
Michel Dubois et Eric Filiol [87]	128	2560	7881

FIGURE 4.21 – Tableau comparatif des systèmes d'équations décrivant le processus de chiffrement de l'AES

correspond à l'opération `XOR`. Un fichier d'exemple ainsi obtenu est donné dans l'annexe C page 169.
Pour implémenter ce mécanisme de description de l'algorithme de chiffrement de l'AES et générer les 128 fichiers, nous avons développé et utilisé un script python s'appuyant sur celui décrit plus haut dans notre présentation de l'AES. Les fichiers source de ce programme sont disponibles sur Internet [81, BooleanAES]. Le programme principal, `aes_equa.py`, offre la possibilité d'une part, de générer les fichiers pour les fonctions de chiffrement et de déchiffrement de l'AES avec les fonctions `generateEncFullFiles()` et `generateDecFullFiles()` et, d'autre part, de contrôler que le chiffrement et le déchiffrement à partir des fichiers obtenus est bien conforme.
Ainsi, les fonctions `controlEncFullFiles()` et `controlDecFullFiles()` réalisent respectivement le chiffrement et le déchiffrement à partir des fichiers générés précédemment. La fonction `controlEncFullFiles()` prend en entrée un bloc de 128 bits de texte clair et un bloc de 128 bits de clef tandis que la fonction `controlDecFullFiles()` prend en entrée un bloc de 128 bits de chiffré et un bloc de 128 bits de clef. Les blocs choisis sont ceux fournis comme vecteurs de test dans l'annexe B du FIPS 197 [155]. Les résultats obtenus correspondent à ceux fournis dans le FIPS : les fichiers que nous avons générés décrivent bien l'algorithme de chiffrement et de déchiffrement de l'AES.

4.4.3.1 Résultats obtenus pour le processus de chiffrement

Le résultat obtenu par la fonction `generateEncFullFiles()` est montré dans le listing 21 page 83 et le résultat obtenu par la fonction `controlEncFullFiles()` est montré dans le listing 22 page 84. La fonction de contrôle `controlEncFullFiles()`, dont le code est donné dans le listing 23 page 85, injecte les 128 variables initiales correspondant au bloc de texte clair et les 1280 variables correspondant aux blocs de clefs de chaque tour, dans les fonctions booléennes .

4.4.3.2 Résultats obtenus pour le processus de déchiffrement

Selon le même principe que pour les fonctions booléennes du chiffrement, le résultat obtenu par la fonction `generateDecFullFiles()` est montré dans le listing 24 page 86 et le résultat obtenu par la fonction `controlDecFullFiles()` est montré dans le listing 25 page 87.
Que ce soit pour le processus de chiffrement que pour celui du déchiffrement, les résultats que nous obtenons en utilisant les fichiers pour chiffrer ou déchiffrer un bloc sont conformes à ceux donnés dans le FIPS 197. Nos systèmes d'équations booléennes sont donc justes et décrivent bien l'algorithme de l'AES.

—=oOo=—

Après avoir présenté succinctement l'algèbre de Boole, les fonctions booléennes et deux de leurs représentations, nous avons élaboré un processus nous permettant de traduire l'algorithme de chiffrement du mini-AES en fonctions booléennes. Nous avons ensuite appliqué ce processus à l'algorithme de chiffrement et de déchiffrement de l'AES. Puis nous avons défini un mode de représentation

```
./aes_equa.py
## Ciphering process
## Create directory AES_files
## AddRoundKey0
## Round0
## AddRoundKey1
## Round1
## AddRoundKey2
## Round2
## AddRoundKey3
## Round3
## AddRoundKey4
## Round4
## AddRoundKey5
## Round5
## AddRoundKey6
## Round6
## AddRoundKey7
## Round7
## AddRoundKey8
## Round8
## AddRoundKey9
## Round9
## AddRoundKey10
## Files generated
```

Listing 21 – Résultat du programme de création des fichiers pour le chiffrement

de ces fonctions booléennes sous la forme de fichiers informatiques. Enfin, nous avons mis au point un programme permettant d'implémenter ce processus et de contrôler que les résultats attendus sont conformes à ceux fournis dans le FIPS. Finalement, nous avons obtenu deux nouveaux systèmes d'équations booléennes, le premier décrivant le processus de chiffrement tandis que le deuxième décrit le processus de déchiffrement de l'*Advanced Encryption Standard* et comprenant chacun 128 équations et $(128 \times 10) + (128 \times 10) = 2560$ variables.
Ce travail a fait l'objet de plusieurs publications en France [83] et à l'étranger [82], [84], [88] et [87].

```
 ./aes_equa.py
## Clear block 00112233445566778899aabbccddeeff
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey0
00102030405060708090a0b0c0d0e0f0 32
## Round0
5f72641557f5bc92f7be3b291db9f91a 32
## addRoundKey1
89d810e8855ace682d1843d8cb128fe4 32
## Round1
ff87968431d86a51645151fa773ad009 32
## addRoundKey2
4915598f55e5d7a0daca94fa1f0a63f7 32
## Round2
4c9c1e66f771f0762c3f868e534df256 32
## addRoundKey3
fa636a2825b339c940668a3157244d17 32
## Round3
6385b79ffc538df997be478e7547d691 32
## addRoundKey4
247240236966b3fa6ed2753288425b6c 32
## Round4
f4bcd45432e554d075f1d6c51dd03b3c 32
## addRoundKey5
c81677bc9b7ac93b25027992b0261996 32
## Round5
9816ee7400f87f556b2c049c8e5ad036 32
## addRoundKey6
c62fe109f75eedc3cc79395d84f9cf5d 32
## Round6
c57e1c159a9bd286f05f4be098c63439 32
## addRoundKey7
d1876c0f79c4300ab45594add66ff41f 32
## Round7
baa03de7a1f9b56ed5512cba5f414d23 32
## addRoundKey8
fde3bad205e5d0d73547964ef1fe37f1 32
## Round8
e9f74eec023020f61bf2ccf2353c21c7 32
## addRoundKey9
bd6e7c3df2b5779e0b61216e8b10b689 32
## Round9
7ad5fda789ef4e272bca100b3d9ff59f 32
## addRoundKey10
69c4e0d86a7b0430d8cdb78070b4c55a 32
69c4e0d86a7b0430d8cdb78070b4c55a (FIPS result)
```

Listing 22 – Résultat du programme de contrôle des fichiers pour le chiffrement

```
def controlEncFullFiles():
    clearBlock = '00112233445566778899aabbccddeeff'
    key = '000102030405060708090a0b0c0d0e0f'
    cipherBlock = '69c4e0d86a7b0430d8cdb78070b4c55a'

    printColor('## Clear block %s' % (clearBlock), BLUE)
    print largeHex2Bin(clearBlock), len(largeHex2Bin(clearBlock))
    printColor('## Key block %s' % (key), BLUE)
    print largeHex2Bin(key), len(largeHex2Bin(key))

    key = largeHex2Bin(key)
    clearBlock = largeHex2Bin(clearBlock)

    block = controlBlock('enc', '## addRoundKey0', '## Round0', clearBlock, key)
    block = controlBlock('enc', '## Round0', '## addRoundKey1', block)
    block = controlBlock('enc', '## addRoundKey1', '## Round1', block, key)
    block = controlBlock('enc', '## Round1', '## addRoundKey2', block)
    block = controlBlock('enc', '## addRoundKey2', '## Round2', block,\
        largeHex2Bin('d6aa74fdd2af72fadaa678f1d6ab76fe'))
    block = controlBlock('enc', '## Round2', '## addRoundKey3', block)
    block = controlBlock('enc', '## addRoundKey3', '## Round3', block,\
        largeHex2Bin('b692cf0b643dbdf1be9bc5006830b3fe'))
    block = controlBlock('enc', '## Round3', '## addRoundKey4', block)
    block = controlBlock('enc', '## addRoundKey4', '## Round4', block,\
        largeHex2Bin('b6ff744ed2c2c9bf6c590cbf0469bf41'))
    block = controlBlock('enc', '## Round4', '## addRoundKey5', block)
    block = controlBlock('enc', '## addRoundKey5', '## Round5', block,\
        largeHex2Bin('47f7f7bc95353e03f96c32bcfd058dfd'))
    block = controlBlock('enc', '## Round5', '## addRoundKey6', block)
    block = controlBlock('enc', '## addRoundKey6', '## Round6', block,\
        largeHex2Bin('3caaa3e8a99f9deb50f3af57adf622aa'))
    block = controlBlock('enc', '## Round6', '## addRoundKey7', block)
    block = controlBlock('enc', '## addRoundKey7', '## Round7', block,\
        largeHex2Bin('5e390f7df7a69296a7553dc10aa31f6b'))
    block = controlBlock('enc', '## Round7', '## addRoundKey8', block)
    block = controlBlock('enc', '## addRoundKey8', '## Round8', block,\
        largeHex2Bin('14f9701ae35fe28c440adf4d4ea9c026'))
    block = controlBlock('enc', '## Round8', '## addRoundKey9', block)
    block = controlBlock('enc', '## addRoundKey9', '## Round9', block,\
        largeHex2Bin('47438735a41c65b9e016baf4aebf7ad2'))
    block = controlBlock('enc', '## Round9', '## addRoundKey10', block)
    block = controlBlock('enc', '## addRoundKey10', '## end', block,\
        largeHex2Bin('549932d1f08557681093ed9cbe2c974e'))
    print('%s (FIPS result)' % (cipherBlock))
```

Listing 23 – Code de la fonction `controlEncFullFiles()`

```
./aes_equa.py
## Deciphering process
## Create directory AES_files
## AddRoundKey10
## Round 9
## AddRoundKey9
## InvMixColumns 9
## Round 8
## AddRoundKey8
## InvMixColumns 8
## Round 7
## AddRoundKey7
## InvMixColumns 7
## Round 6
## AddRoundKey6
## InvMixColumns 6
## Round 5
## AddRoundKey5
## InvMixColumns 5
## Round 4
## AddRoundKey4
## InvMixColumns 4
## Round 3
## AddRoundKey3
## InvMixColumns 3
## Round 2
## AddRoundKey2
## InvMixColumns 2
## Round 1
## AddRoundKey1
## InvMixColumns 1
## Round 0
## AddRoundKey0
## Files generated
```

Listing 24 – Résultat du programme de création des fichiers pour le déchiffrement

```
./aes_equa.py
## Cipher block 69c4e0d86a7b0430d8cdb78070b4c55a
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey10
7ad5fda789ef4e272bca100b3d9ff59f 32
## Round9
bd6e7c3df2b5779e0b61216e8b10b689 32
## addRoundKey9
e9f74eec023020f61bf2ccf2353c21c7 32
## invMixColumns9
54d990a16ba09ab596bbf40ea111702f 32
## Round8
fde3bad205e5d0d73547964ef1fe37f1 32
## addRoundKey8
baa03de7a1f9b56ed5512cba5f414d23 32
## invMixColumns8
3e1c22c0b6fcbf768da85067f6170495 32
## Round7
...
## Round3
fa636a2825b339c940668a3157244d17 32
## addRoundKey3
4c9c1e66f771f0762c3f868e534df256 32
## invMixColumns3
3bd92268fc74fb735767cbe0c0590e2d 32
## Round2
4915598f55e5d7a0daca94fa1f0a63f7 32
## addRoundKey2
ff87968431d86a51645151fa773ad009 32
## invMixColumns2
a7be1a6997ad739bd8c9ca451f618b61 32
## Round1
89d810e8855ace682d1843d8cb128fe4 32
## addRoundKey1
5f72641557f5bc92f7be3b291db9f91a 32
## invMixColumns1
6353e08c0960e104cd70b751bacad0e7 32
## Round0
00102030405060708090a0b0c0d0e0f0 32
## addRoundKey0
00112233445566778899aabbccddeeff 32
00112233445566778899aabbccddeeff (FIPS result)
```

Listing 25 – Résultat du programme de contrôle des fichiers pour le déchiffrement