La cryptographie Moderne (suite)

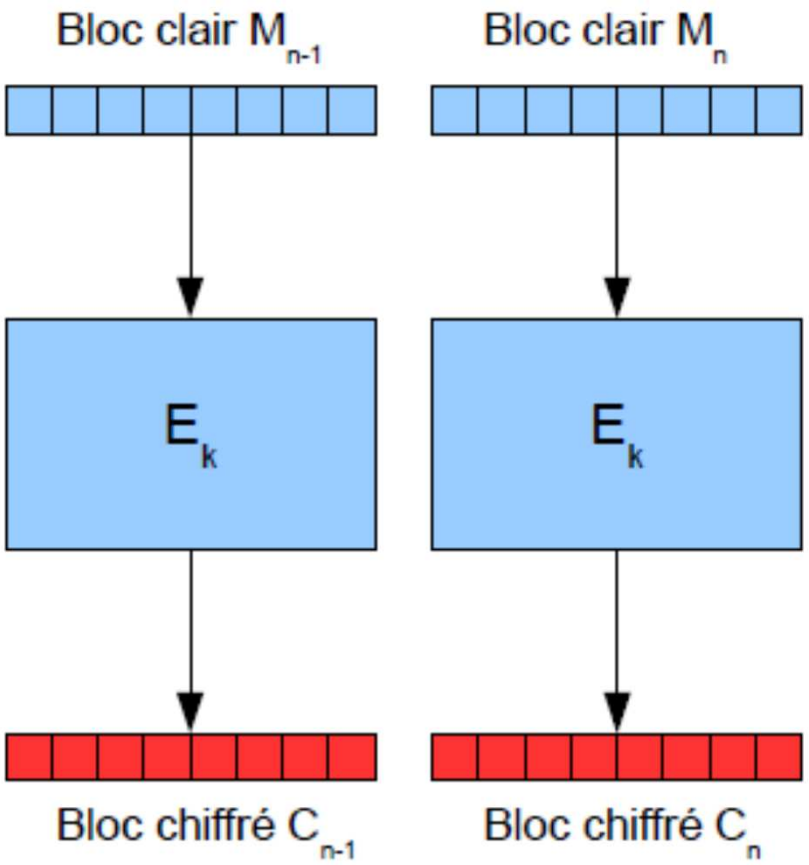
Les modes de chiffrement

- Comment traiter les informations d'un fichier par exemple ?
- Chiffrement par blocs
  - Carnet de codage électronique (ECB)
  - Chaînage de blocs (CBC)
- Chiffrement en continu
  - Chiffrement à rétroaction (CFB)
  - Chiffrement à rétroaction de sortie (OFB)

Carnet de codage électronique

- ElectronicCodeBook (ECB)
  - Chaque bloc est chiffré indépendamment des autres blocs
  - Deux blocs clairs identiques donnent deux blocs chiffrés identiques

$$M_i = M_j \Rightarrow C_i = C_j$$

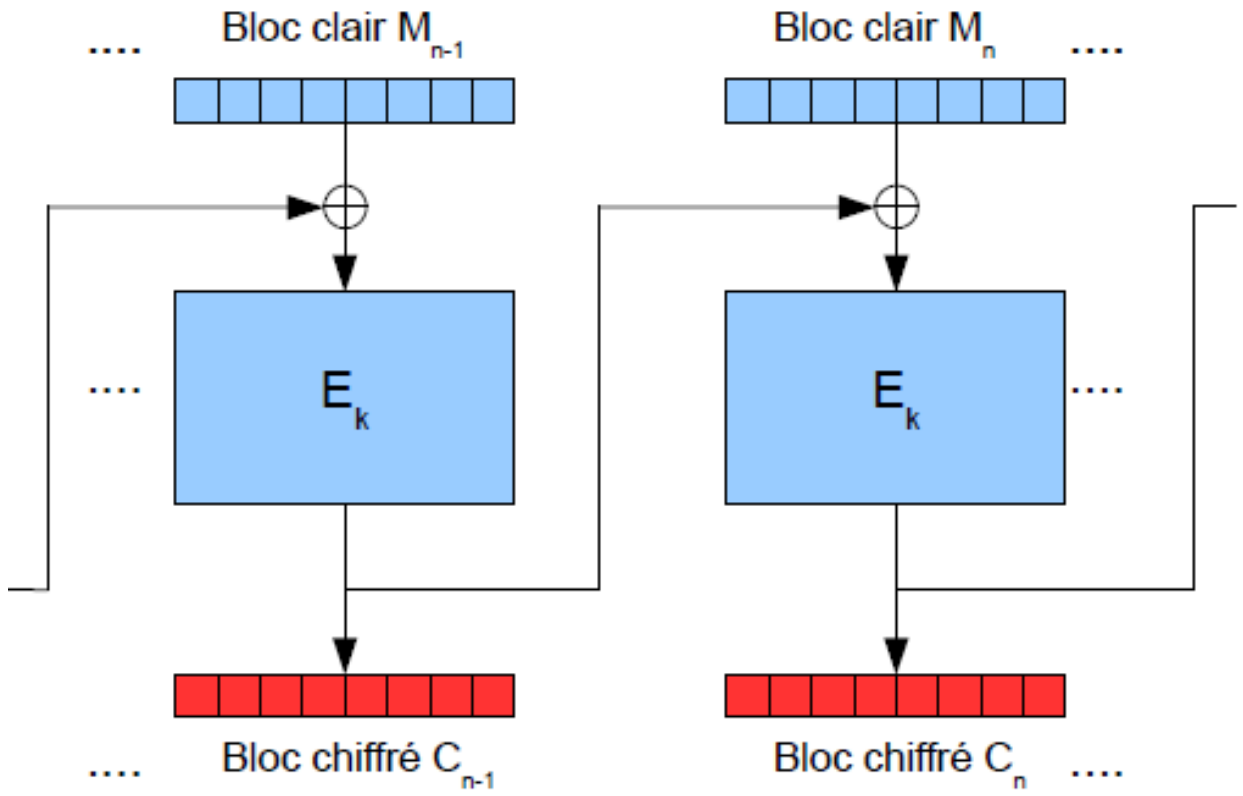


suite

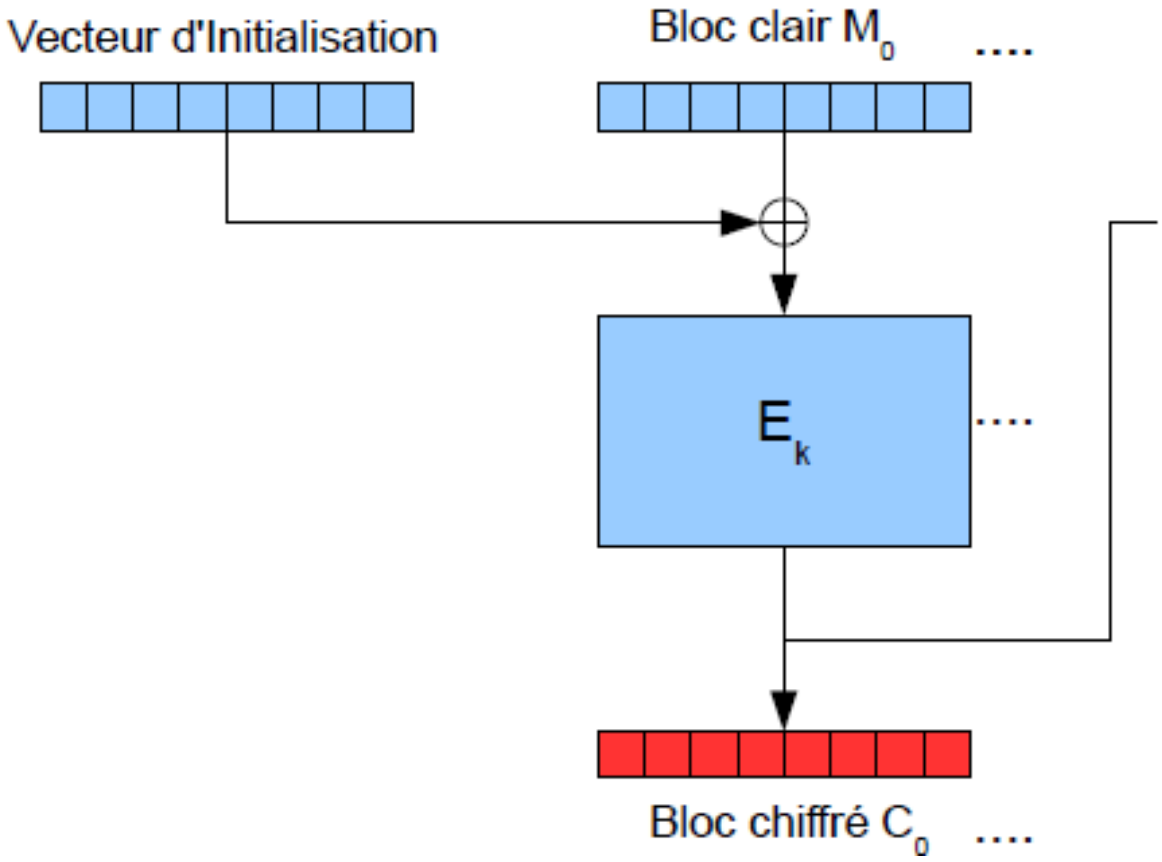
- Inconvénients :
  - la répétition dans le message clair entraine la répétition dans le message chiffré
  - Elaboration de carnets de codes avec les correspondances clair $\Leftrightarrow$ chiffré
  - Un carnet de code bien rempli permet de se dispenser de la clé

Chaînage de blocs

- Cipher-Block Chaining (CBC)
  - Le dernier bloc chiffré est combiné (XOR) avec le bloc clair suivant



- Vecteur d'initialisation (IV)
  - Résout le problème du premier bloc



- La valeur est quelconque
  - C'est celle que l'on obtiendrait à la fin de déchiffrement, donc inutile
- Peut utiliser l'heure système
  - Deux messages clairs identiques produiront des messages chiffrés différents
- Il peut être public !
  - Si l'attaquant connaît IV, il lui reste à décrypter C_0 pour décrypter le reste, c'est-à-dire trouver la clé secrète (le problème reste entier)

- Formalisation du chiffrement

$$C_i = E_k(M_i \oplus C_{i-1})$$
$$C_0 = VI$$

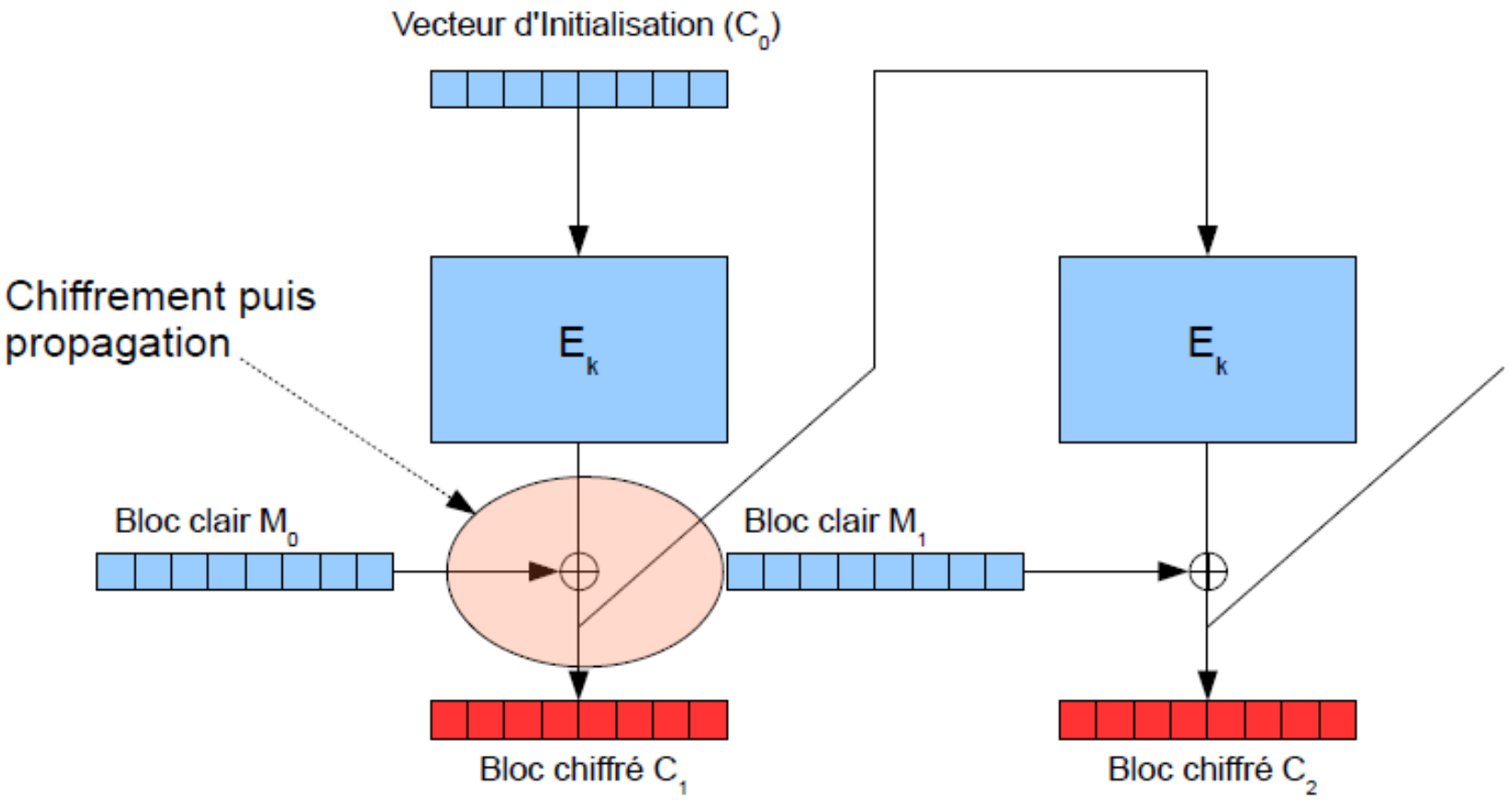
- Formalisation du déchiffrement

$$M_i = D_k(C_i) \oplus C_{i-1}$$
$$C_0 = VI$$

- Sécurité accrue par rapport à ECB
  - Recommandé pour le chiffrement de fichiers

Chiffrement à rétroaction

- Cipher FeedBack (CFB)
  - Le texte clair peut être vu comme un flux de bits (1 à la fois ou n à la fois)
  - L'algorithme de chiffrement est utilisé pour générer les clés utilisées pour chiffrer le texte clair (par XOR)


Vecteur d'Initialisation (C_0)
Chiffrement puis propagation
E_k
E_k
Bloc clair M_0
Bloc clair M_1
Bloc chiffré C_1
Bloc chiffré C_2

- Formalisation du chiffrement (où C_i et P_i sont des blocs de n bits)

$$C_i = E_k(C_{i-1}) \oplus M_i$$
$$C_0 = VI$$

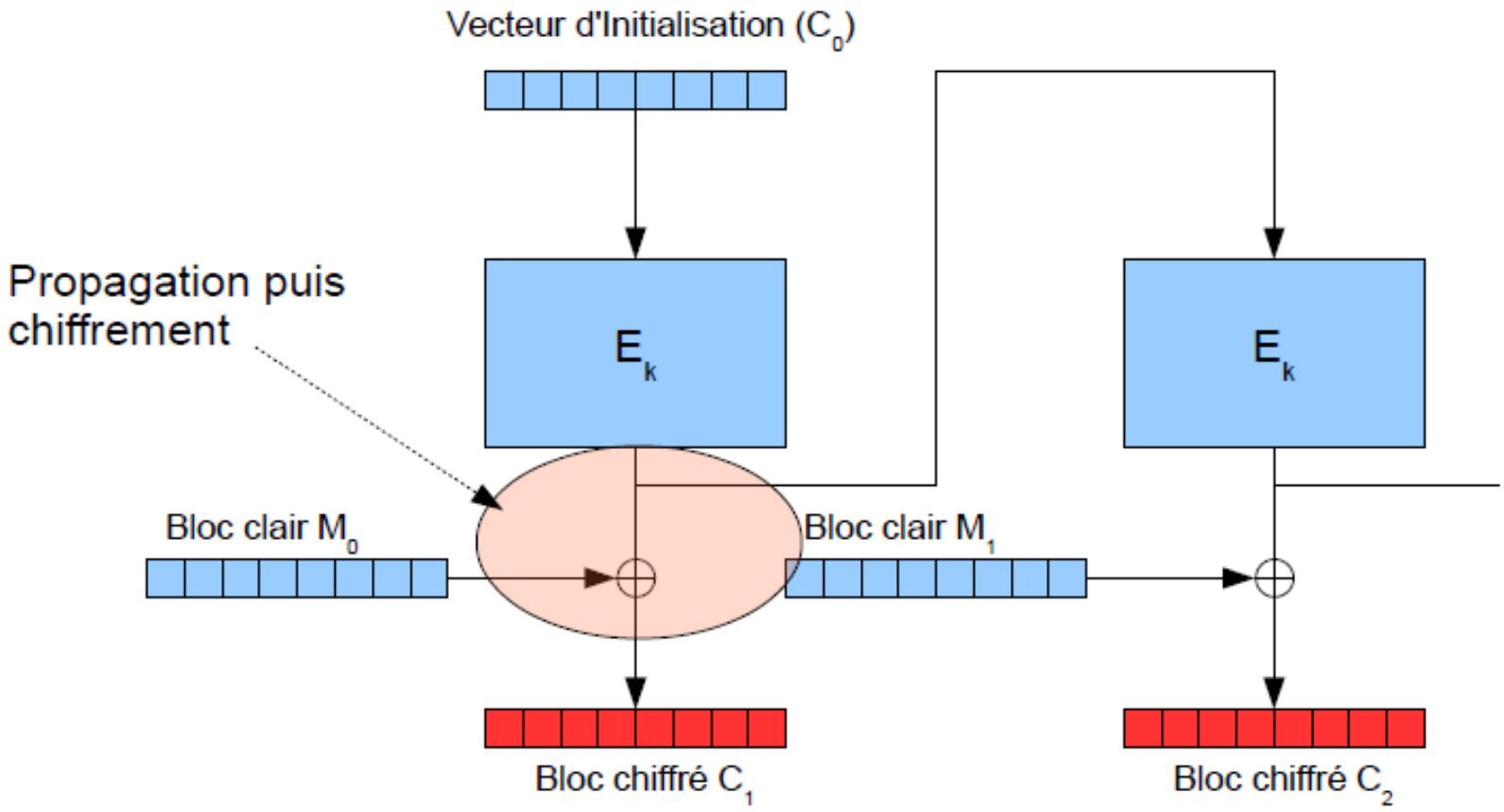
- Formalisation du déchiffrement

$$M_i = E_k(C_{i-1}) \oplus C_i$$
$$C_0 = VI$$

- Le déchiffrement est identique au chiffrement ?
  - Normal, l'algorithme de chiffrement est le XOR !

Chiffrement à rétroaction de sortie

- Output-FeedBack (OFB)
  - Mode similaire à CFB
  - La sortie de E_k est directement utilisée comme entrée pour l'étape suivante
  - E_k est utilisée uniquement comme générateur de nombres aléatoires
  - Le germe est la clé secrète !
  - Le sécurité ne repose plus que sur le caractère aléatoire de l'algorithme


Vecteur d'Initialisation (C_0)
Propagation puis chiffrement
E_k
E_k
Bloc clair M_0
Bloc clair M_1
Bloc chiffré C_1
Bloc chiffré C_2

Les modes de chiffrement

- Quel mode choisir ?
  - Trois critères essentiels :
    - La sécurité
    - L'efficacité (logiciel ou matériel)
    - La tolérance aux fautes
  - ECB est le plus facile, mais le moins sûr (à éviter)
  - CBC, CFB ou OFB peuvent convenir pour pratiquement toutes les applications
  - Il existe d'autres modes, mais plus exotiques

Advanced Encryption Standard (AES)

Plan

- Historique
- Caractéristiques d'AES
- Opérations dans $\mathbb{F}_2^8$
- L'algorithme

Historique

- Concours du NIST pour remplacer DES en 1997
- Caractéristiques exigées:
  - Chiffrement de blocs de 128 bits
  - Clés de 128, 192 ou 256 bits
  - Algorithme public et libre de droits

- 15 réponses
- Sélection achevée en 2000, après trois ans de soumissions et de discussions auprès de la communauté
- Le vainqueur : Rijndael (Rijmen et Daemen, Université catholique de Louvain, Belgique)
- Le NIST publie FIPS 197 en 2001

Caractéristiques d'AES

- Un message est traité par blocs de 128 bits
  - Soit 4 mots de 32 bits
  - Nb=4
- La clé est également traitée par mots de 32 bits
  - AES-128 : Nk=4
  - AES-192 : Nk=6
  - AES-256 : Nk=8

Suite

- Algorithme par rondes
- Le nombre de rondes dépend de la taille de la clé
  - AES-128 : Nr=10
  - AES-192 : Nr=12
  - AES-256 : Nr=14

- La variable « state »:
  - Matrice 4 lignes x ***Nb*** octets
  - Initialisée avec le bloc d'entrée
  - Toutes les opérations de l'algorithme agissent sur l'état
- Utilise une représentation polynomiale des nombres binaires : Le corps fini GF(2^8)

GF(2^8) : Définition

- GF(2) = F_2 = unique corps fini à 2 éléments
  = {0,1} avec les opérations booléennes usuelles
- $F_2[X]$ = ensemble des polynômes à coefficients dans F_2
- Soit P(X) un polynôme irréductible de degré 8 appartenant à $F_2[X]$
- Par définition, GF(2^8) = $F_2[X]$ / P

GF(2^8) : Exemple

- Prenons $P(X) = X^8 + X^4 + X^3 + X + 1$
- Éléments dans GF(2^8)

  = polynômes réduits « modulo P »

  = polynômes de degré < 8
- Exemple

$$a = X^6 + X^4 + X^2$$

- Soient a et b dans $GF(2^8)$

  $a = X^6 + X^4 + X^2$

  $b = X^2$

- Addition :

  $a+b = (X^6 + X^4 + X^2) + (X^2) = X^6 + X^4$

- Multiplication :

  $a \times b = (X^8 + X^6 + X^4) \text{ modulo } (X^8 + X^4 + X^3 + X + 1)$

  $a \times b = (X^6 + X^4) + X^4 + X^3 + X + 1$

GF(2^8) : représentation

- Chaque élément de GF(2^8) est représenté

$$b_7X^7 + b_6X^6 + \ldots + b_1X + b_0$$

- On le stocke sur l'octet représenté en binaire par $(b_7,\ldots,b_0)$
- Représentation entre 0 et 255

GF(2^8)

- Un octet peut être vu comme un polynôme :

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0 x^0$$

- Exemple :

$$01100011_2 = x^6 + x^5 + x + 1$$

- L'addition et la multiplication sont définies mais sont différentes des opérateurs binaires habituels

Suite

- L'addition notée +
  - C'est un XOR sur les coefficients binaires
  - Exemple :

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$$

$$01010111_2 \oplus 10000011_2 = 11010100_2$$

- La multiplication notée ●
  - C'est la multiplication des polynômes suivie d'une réduction par un polynôme irréductible pour ramener le polynôme résultat à un degré ≤ 7
  - Un polynôme irréductible est l'équivalent d'un nombre premier pour les entiers
  - Dans AES, le polynôme irréductible choisi m(x) est :

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

- Réduire un polynôme dans GF(2^8) correspond, pour les entiers, à calculer le reste d'une division entière

- Exemple:

$$(x^6+x^4+x^2+x+1)\bullet(x^7+x+1)$$
$$=(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)$$

- La réduction est une opération itérative consistant à éliminer successivement les coefficients > 7
  - Si le polynôme P(x) à réduire est de degré n
  - Multiplier m(x) par x^{n-8} (le résultat sera un polynôme de degré n)
  - Additionner P(x) et m(x)$\bullet x^{n-8}$ (le xor annule x^n)

- Exemple : éliminer x^{13}

$$P(x)=x^{13}+x^{11}+x^{9}+x^{8}+x^{6}+x^{5}+x^{4}+x^{3}+1$$
$$m(x)\bullet x^{5}=x^{13}+x^{9}+x^{8}+x^{6}+x^{5}$$
$$P(x)+m(x)\bullet x^{5}=x^{11}+x^{4}+x^{3}+1$$

- On recommence ces opérations pour éliminer x^{11}, etc. jusqu'à ce que l'on obtienne un polynôme de degré ≤ 7

- Exercice : réduire (Voir TD)

$$P(x)=x^{12}+x^{9}+x^{5}+1$$

- Cas particulier de la multiplication par x (=2_{10})

$$Soit\ b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0x^0$$
$$b(x)\bullet x=b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x^1$$

- En binaire, c'est un décalage à gauche
- On ne réduit le résultat que si $b_7=1$
- En binaire

$$b(x)\bullet x=1\ \ b_6b_5b_4b_3b_2b_1b_00$$
$$m(x)=1\ \ 00011011$$
$$\text{d'où}\ (b(x)\bullet x)\oplus m(x)\Leftrightarrow b_6b_5b_4b_3b_2b_1b_00\oplus 00011011$$

Toutes les opérations se font au niveau de l'octet

- Cette fonction s'appelle xtime(v)
  - Effectue un décalage de v puis un XOR avec $1b_{hex}$

- Multiplier deux nombres dans GF(2^8) revient à des appels successifs de xtime()
- Exemple : v●3
  - Dans GF(2^8), 3 est représenté par x+1
  - Donc v●3=xtime(v)⊕v
- Exemple : v●4
  - Dans GF(2^8), 4 est représenté par x^2 ou encore x.x
  - Donc v●4=xtime(xtime(v))

Exemple

- ’57’ · ’13’ = ? (1)

  On utilise une représentation en hexadécimal pour alléger les calculs:

  ‘13’=’01’+’02’+’10’

  Donc (1) devient

  (`57‘ · `01') ⊕ (`57‘ · `02') ⊕ (`57‘ · `10') = `57‘ ⊕ `*AE*‘ ⊕ `07‘ = `*FE*‘

  Vu que

  `57‘ • `02‘ = xtime(57) = `*AE*‘

  `57‘ • `04‘ = xtime(*AE*) = `47‘

  `57‘ • `08‘ = xtime(47) = `8*E*‘

  `57‘ • `10‘ = xtime(8*E*) = `07'

- Un mot de 32 bits (4 octets) peut être vu comme un polynôme de degré 3 où les coefficients appartiennent à GF(2^8)

$$a(x)=a_3x^3+a_2x^2+a_1x^1+a_0x^0 \text{ où } a_0,a_1,a_2,a_3\in GF(2^8)$$

a_3, a_2, a_1, a_0 sont les octets du mot de 32 bits

- L'addition revient à additionner (donc XOR) les coefficients

$$a(x)+b(x)=(a_3\bullet b_3)x^3+(a_2\bullet b_2)x^2+(a_1\bullet b_1)x^1+(a_0\bullet b_0)x^0$$

- La multiplication revient à développer le produit des deux polynômes puis de réduire le résultat par un polynôme irréductible
- Dans AES, le polynôme (pas irréductible!) choisi pour cette opération est x^4+1
  - La réduction de x^n par x^4+1 vaut $x^{(n \bmod 4)}$

- Dans AES, la multiplication suivie de la réduction par x^4+1 est notée $\otimes$

$$a(x) \otimes b(x) = d(x) = d_3 x^3 + d_2 x^2 + d_1 x^1 + d_0 x^0 \text{ où}$$

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$

$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$$

$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$$

$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$$

- Cette multiplication peut être exprimée sous forme d'un produit matriciel

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- Dans AES, il faut pouvoir inverser cette matrice
- x^4+1 n'est pas irréductible, donc AES utilise

$$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x^1+\{02\}x^0 \text{ d'où}$$
$$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x^1+\{0e\}x^0$$

AES

- Comme dans DES, il y a deux parties
  - La génération des clés intermédiaires
  - Le chiffrement proprement dit

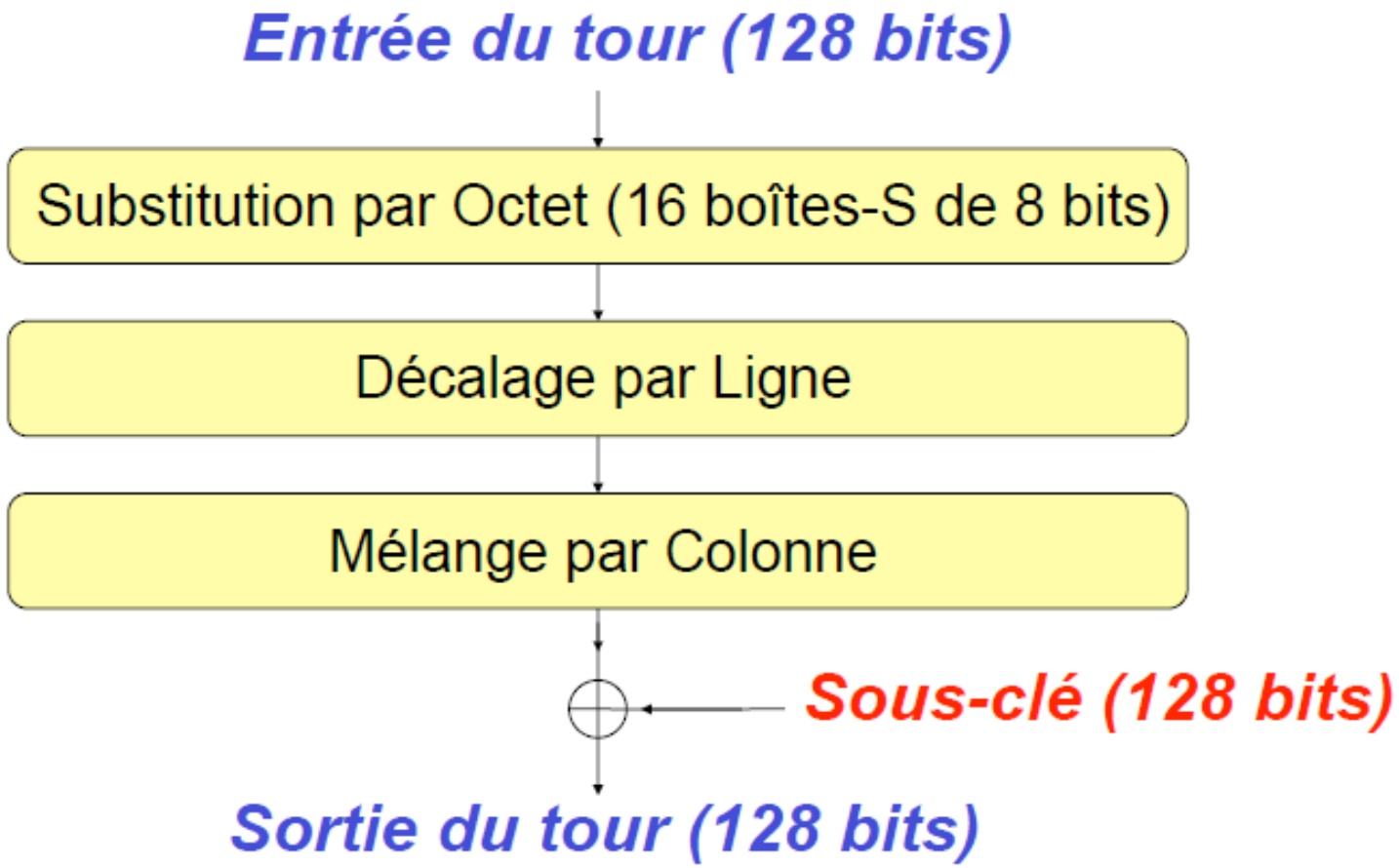
Structure générale

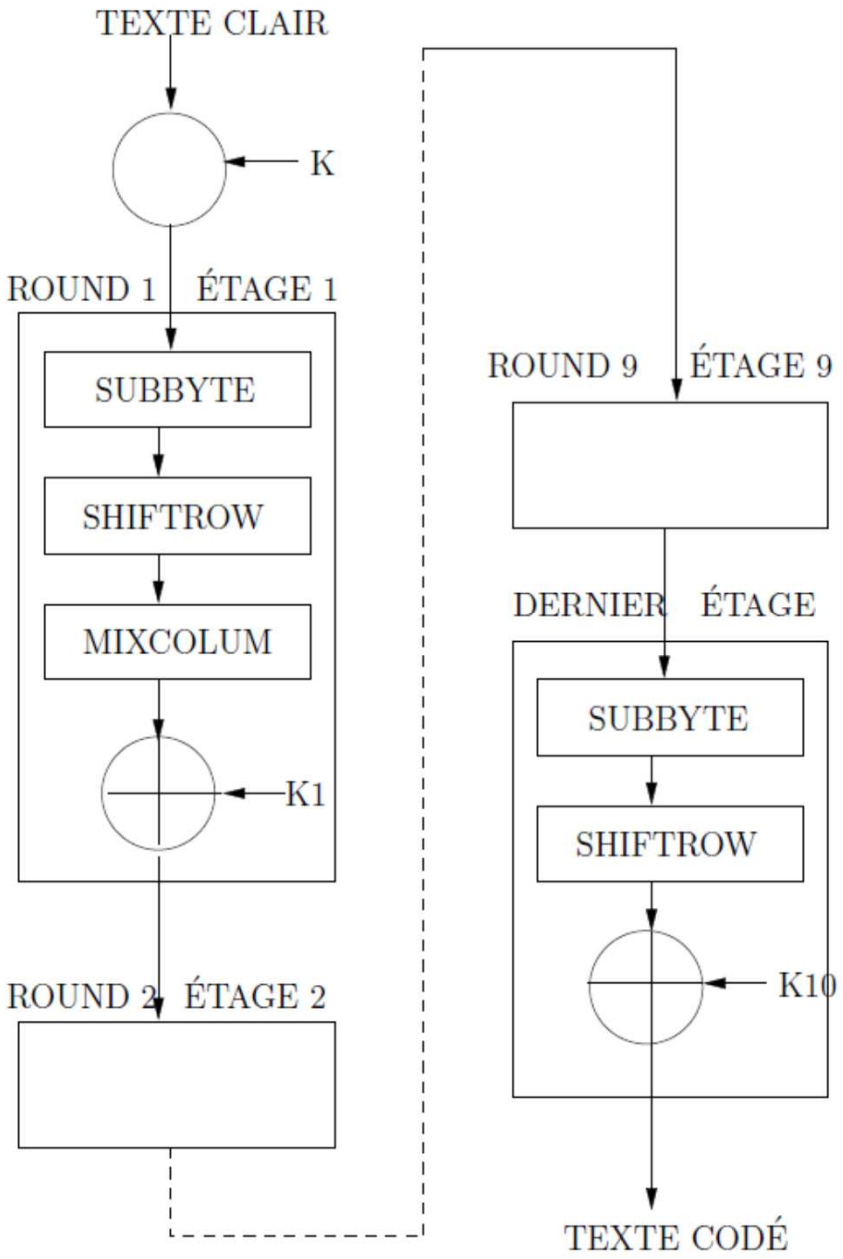
Les données sont stockées dans un « carré » de 4 x 4 = 16 cases

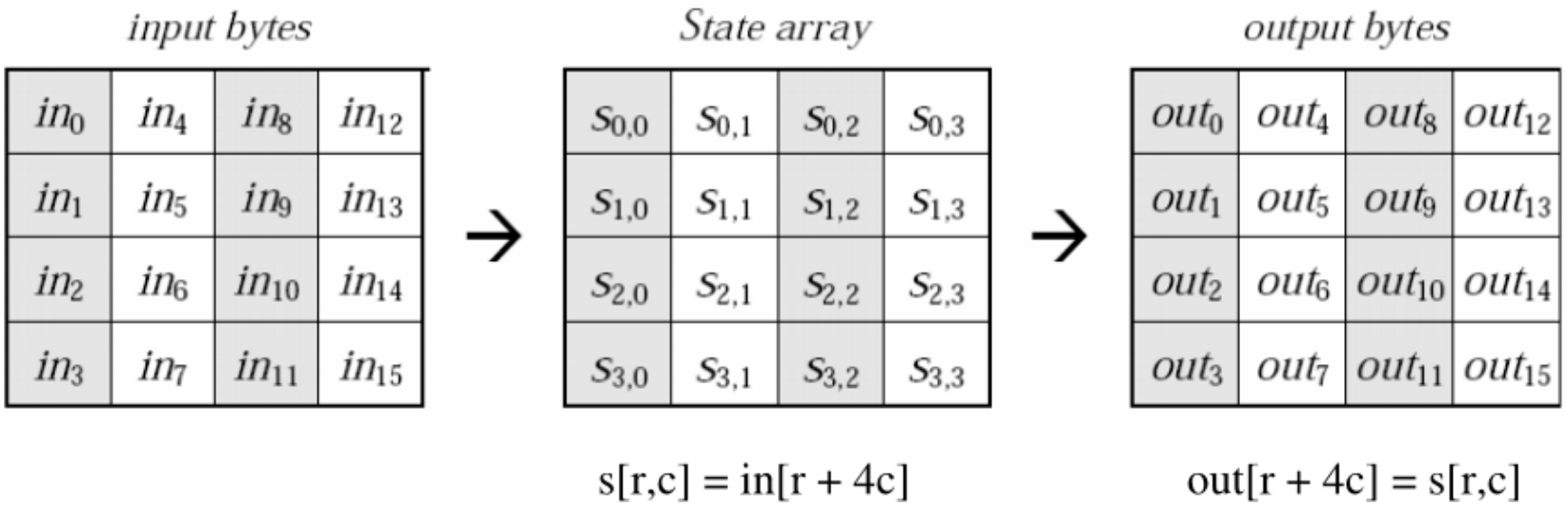
X_1	X_2	X_3	X_4
X_5	X_6	X_7	X_8
X_9	X_{10}	X_{11}	X_{12}
X_{13}	X_{14}	X_{15}	X_{16}

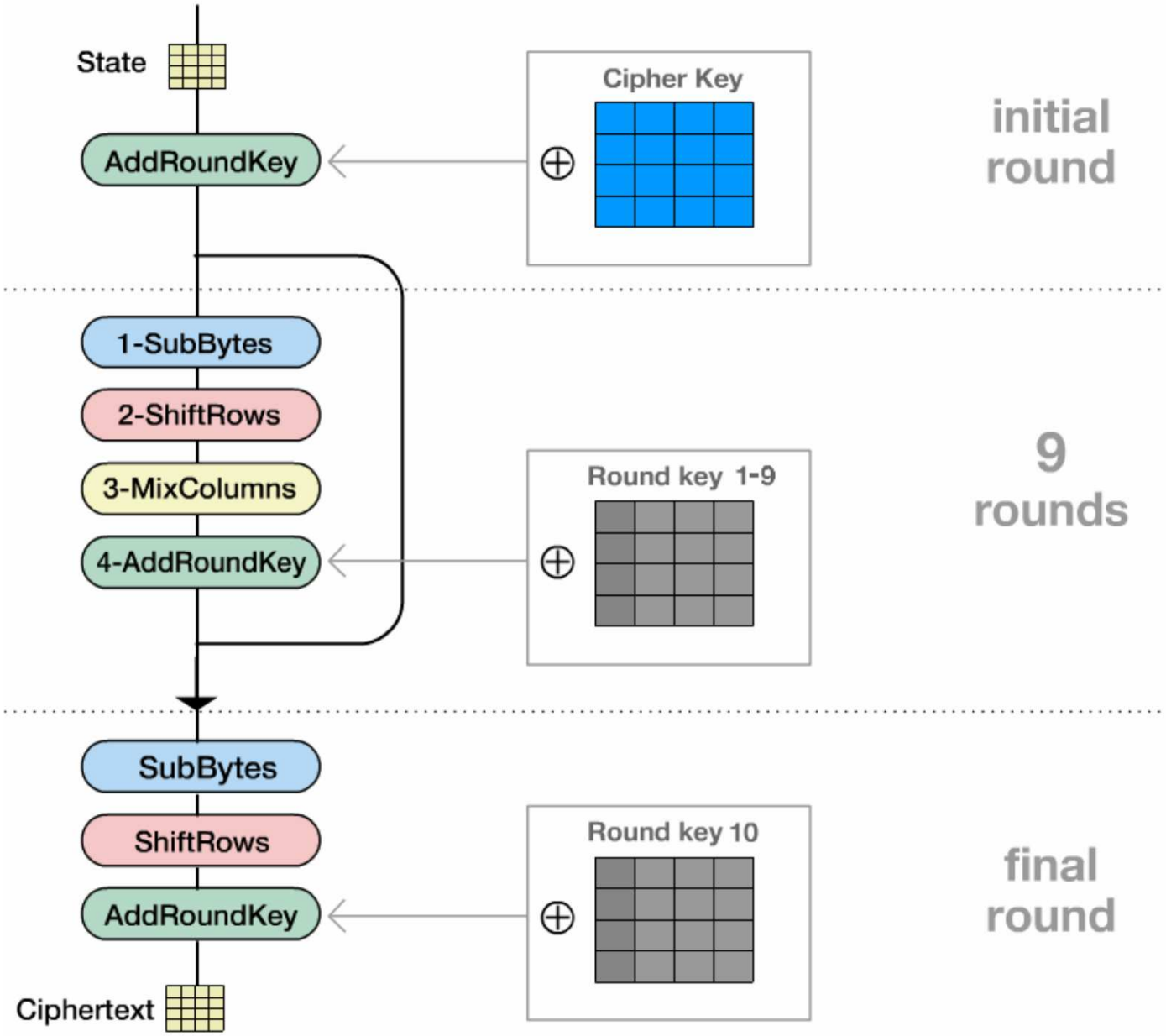
Chaque case contient 1 octet (8 x 16 = 128 bits d'état interne)

Fonction de tour

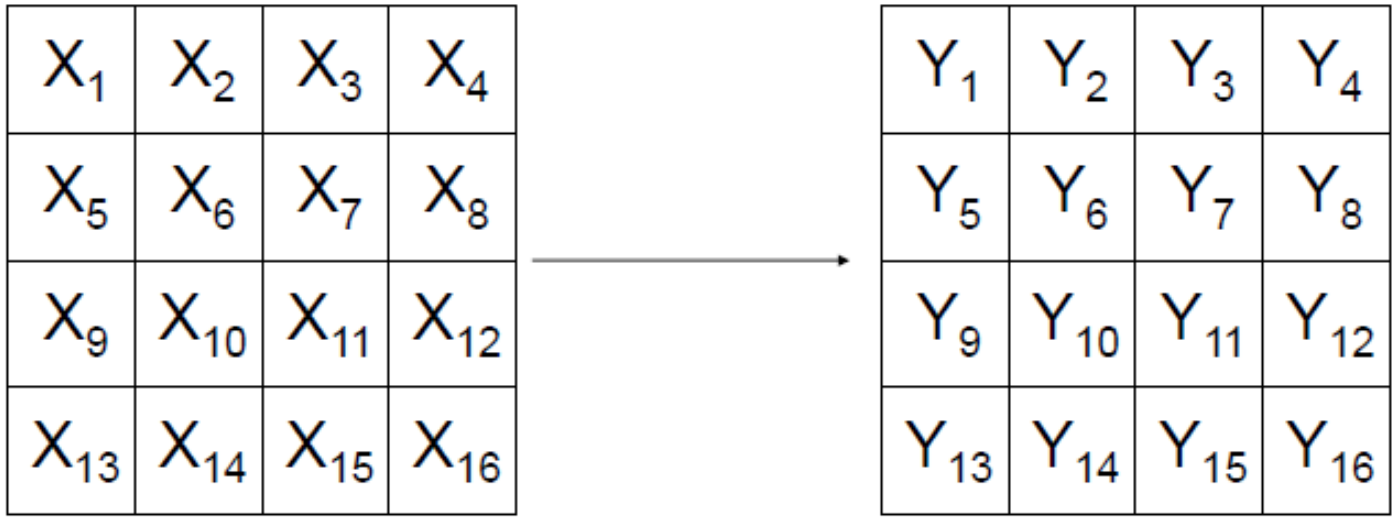



TEXTE CLAIR
K
ROUND 1 ÉTAGE 1
SUBBYTE
SHIFTROW
MIXCOLUM
K1
ROUND 2 ÉTAGE 2
ROUND 9 ÉTAGE 9
DERNIER ÉTAGE
SUBBYTE
SHIFTROW
K10
TEXTE CODÉ


input bytes
State array
output bytes
in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}
s[r,c] = in[r + 4c]
out[r + 4c] = s[r,c]


State
AddRoundKey
Cipher Key
initial round
1-SubBytes
2-ShiftRows
3-MixColumns
4-AddRoundKey
Round key 1-9
9 rounds
SubBytes
ShiftRows
AddRoundKey
Round key 10
final round
Ciphertext

Substitution par Octet



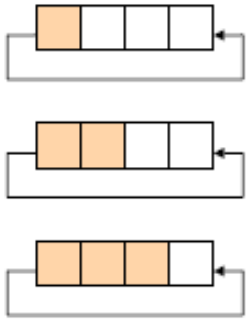
Pour tout $1 \leq i \leq 16$, $Y_i = S(X_i)$

La boîte S

$$Sbox = \begin{bmatrix}
99 & 124 & 119 & 123 & 242 & 107 & 111 & 197 & 48 & 1 & 103 & 43 & 254 & 215 & 171 & 118 \\
202 & 130 & 201 & 125 & 250 & 89 & 71 & 240 & 173 & 212 & 162 & 175 & 156 & 164 & 114 & 192 \\
183 & 253 & 147 & 38 & 54 & 63 & 247 & 204 & 52 & 165 & 229 & 241 & 113 & 216 & 49 & 21 \\
4 & 199 & 35 & 195 & 24 & 150 & 5 & 154 & 7 & 18 & 128 & 226 & 235 & 39 & 178 & 117 \\
9 & 131 & 44 & 26 & 27 & 110 & 90 & 160 & 82 & 59 & 214 & 179 & 41 & 227 & 47 & 132 \\
83 & 209 & 0 & 237 & 32 & 252 & 177 & 91 & 106 & 203 & 190 & 57 & 74 & 76 & 88 & 207 \\
208 & 239 & 170 & 251 & 67 & 77 & 51 & 133 & 69 & 249 & 2 & 127 & 80 & 60 & 159 & 168 \\
81 & 163 & 64 & 143 & 146 & 157 & 56 & 245 & 188 & 182 & 218 & 33 & 16 & 255 & 243 & 210 \\
205 & 12 & 19 & 236 & 95 & 151 & 68 & 23 & 196 & 167 & 126 & 61 & 100 & 93 & 25 & 115 \\
96 & 129 & 79 & 220 & 34 & 42 & 144 & 136 & 70 & 238 & 184 & 20 & 222 & 94 & 11 & 219 \\
224 & 50 & 58 & 10 & 73 & 6 & 36 & 92 & 194 & 211 & 172 & 98 & 145 & 149 & 228 & 121 \\
231 & 200 & 55 & 109 & 141 & 213 & 78 & 169 & 108 & 86 & 244 & 234 & 101 & 122 & 174 & 8 \\
186 & 120 & 37 & 46 & 28 & 166 & 180 & 198 & 232 & 221 & 116 & 31 & 75 & 189 & 139 & 138 \\
112 & 62 & 181 & 102 & 72 & 3 & 246 & 14 & 97 & 53 & 87 & 185 & 134 & 193 & 29 & 158 \\
225 & 248 & 152 & 17 & 105 & 217 & 142 & 148 & 155 & 30 & 135 & 233 & 206 & 85 & 40 & 223 \\
140 & 161 & 137 & 13 & 191 & 230 & 66 & 104 & 65 & 153 & 45 & 15 & 176 & 84 & 187 & 22
\end{bmatrix}$$

Décalage par ligne

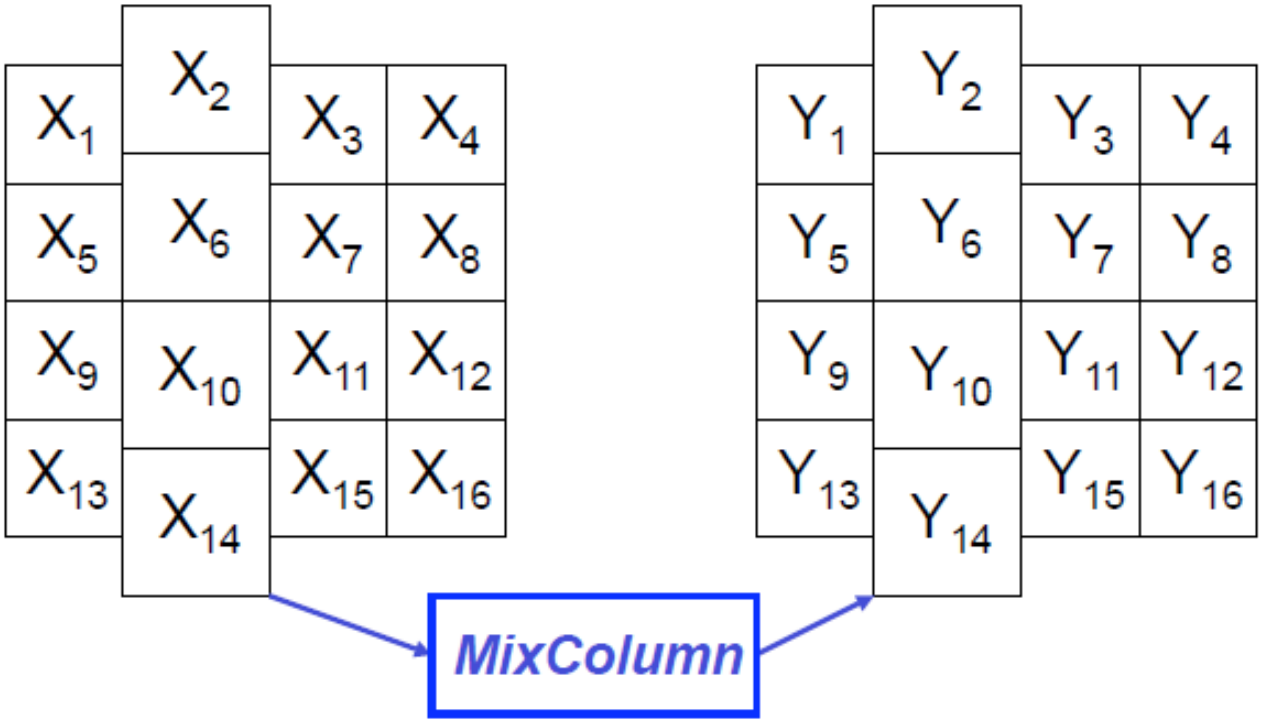
X_1	X_2	X_3	X_4
X_5	X_6	X_7	X_8
X_9	X_{10}	X_{11}	X_{12}
X_{13}	X_{14}	X_{15}	X_{16}



X_1	X_2	X_3	X_4
X_6	X_7	X_8	X_5
X_{11}	X_{12}	X_9	X_{10}
X_{16}	X_{13}	X_{14}	X_{15}

Décalage circulaire (vers la gauche) de i cases pour la ligne numéro i, $0 \leq i \leq 3$

Mélange par colonne



MixColumn() est appliquée à chaque colonne

Mélange par colonne

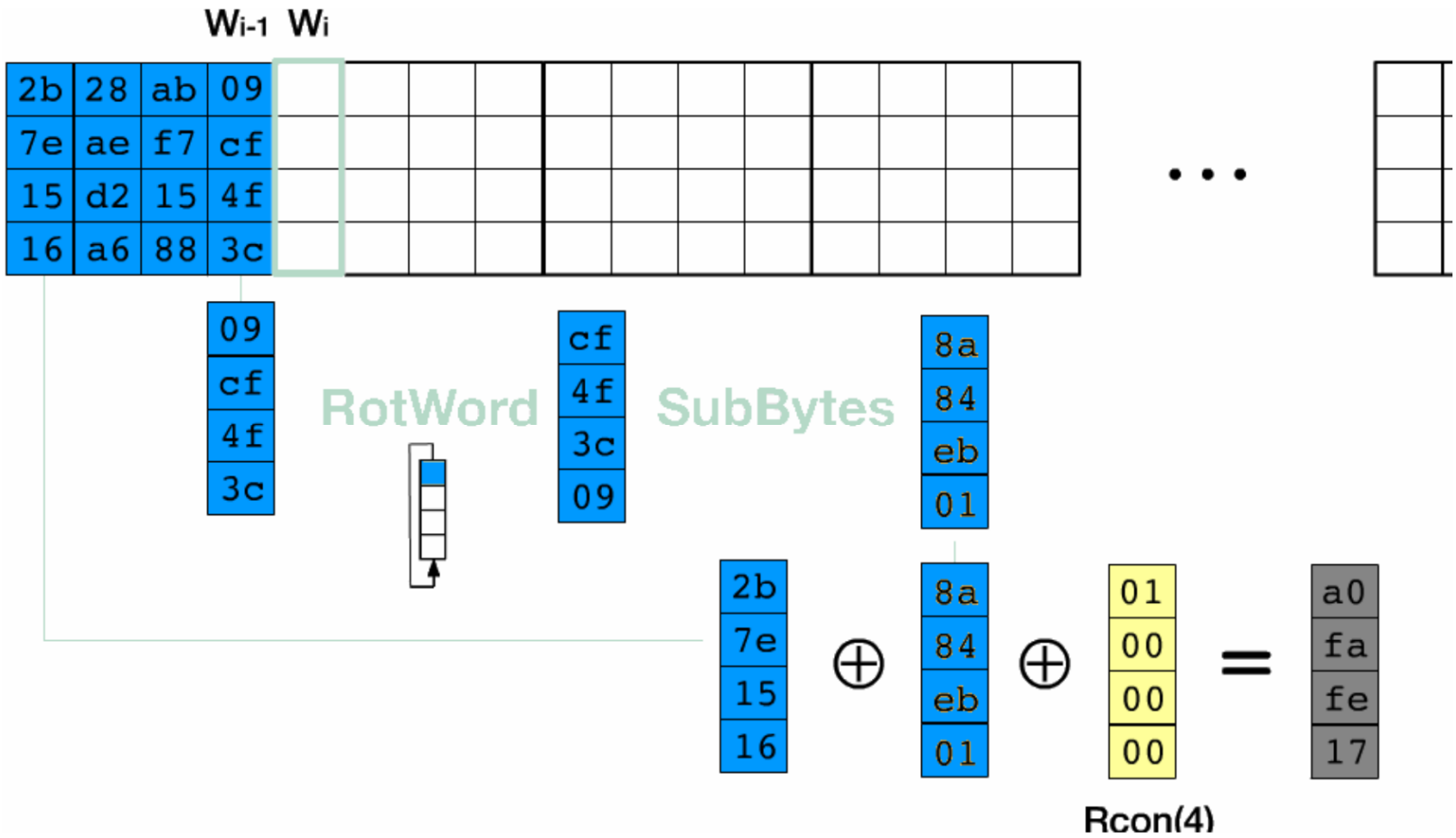
$$MixColumn\begin{pmatrix} X_1 \\ X_5 \\ X_9 \\ X_{13} \end{pmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} x \begin{matrix} X_1 \\ X_5 \\ X_9 \\ X_{13} \end{matrix}$$

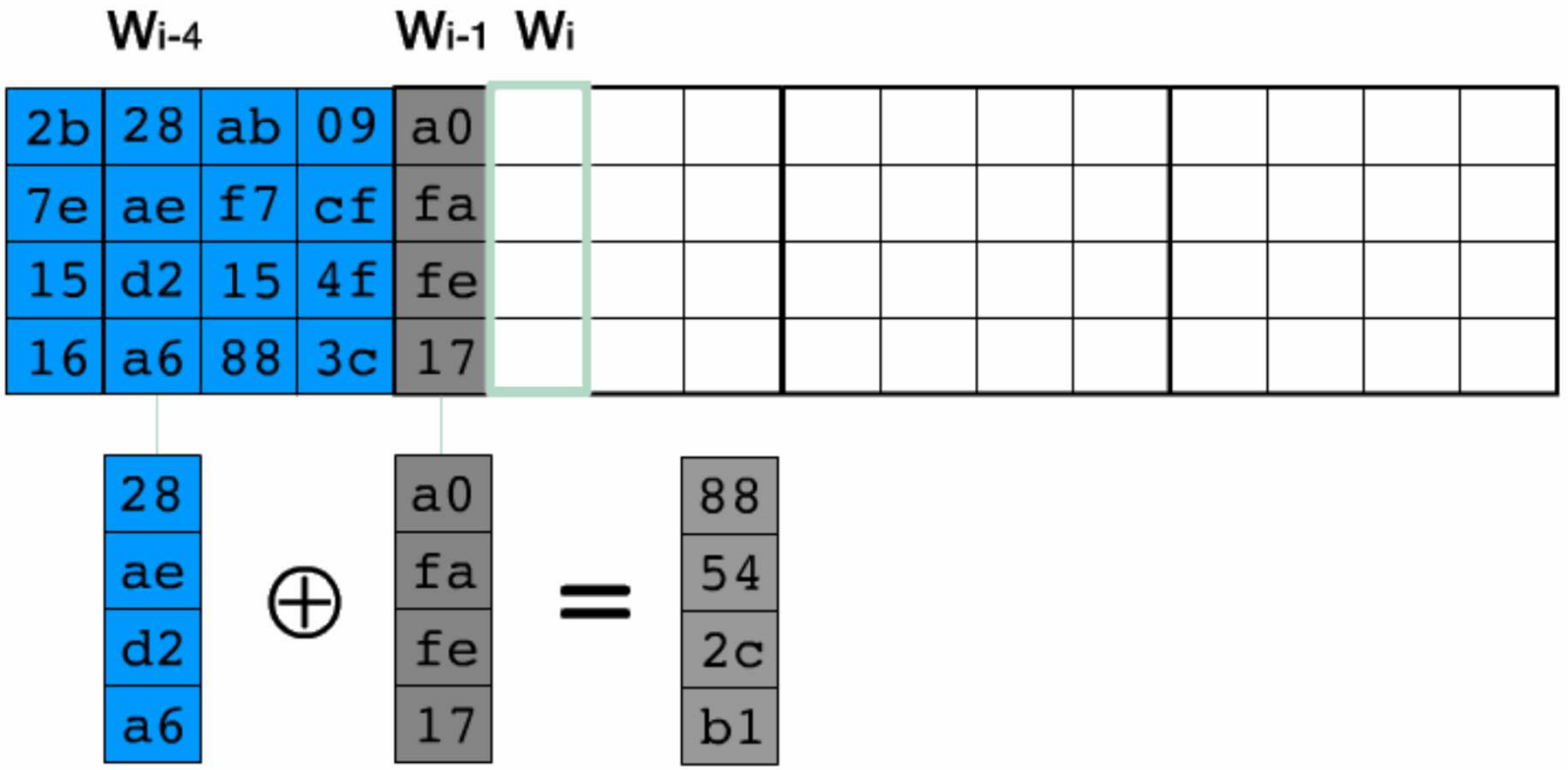
Opérations linéaires dans GF(2^8)

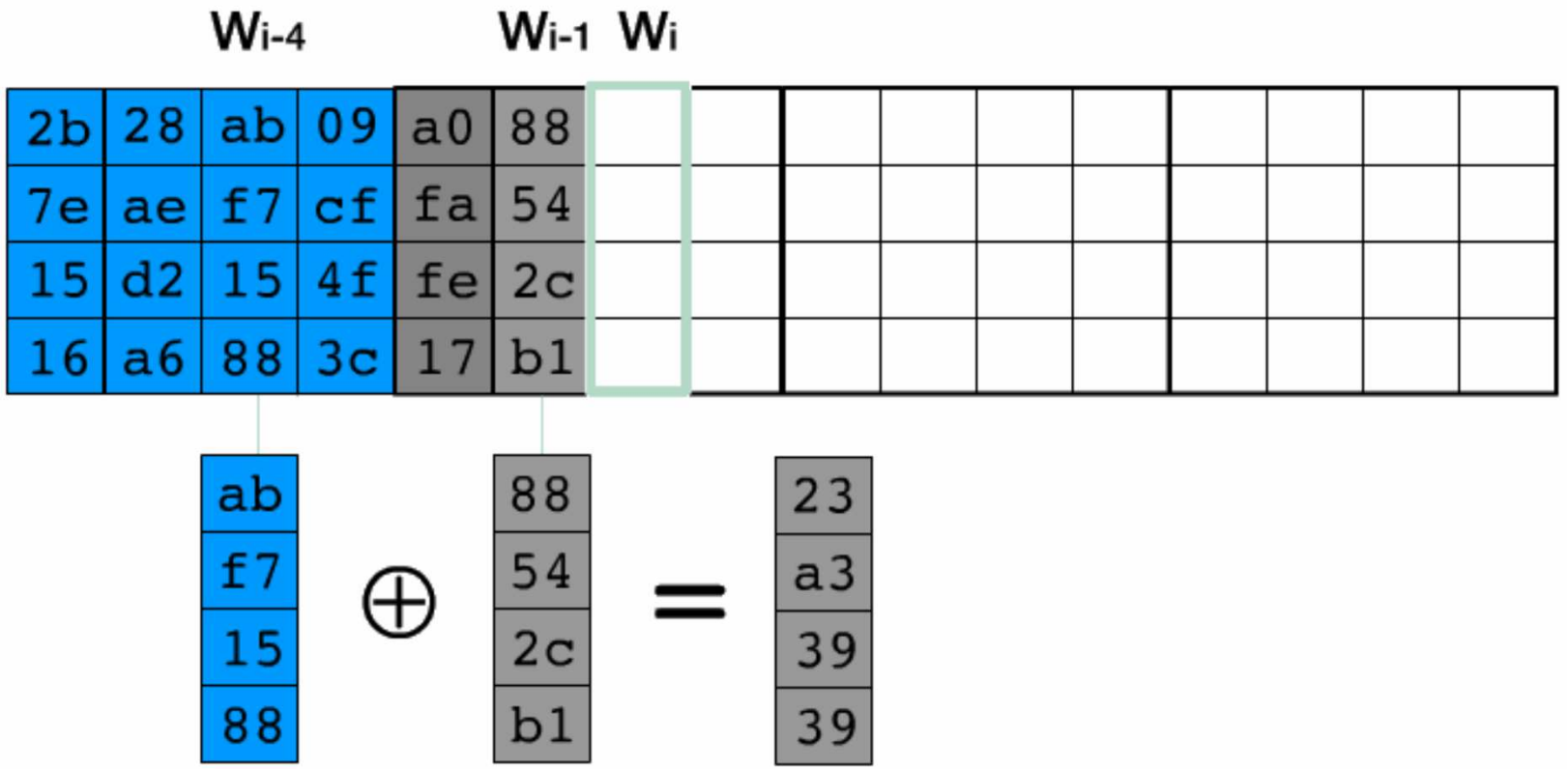
Génération des clés

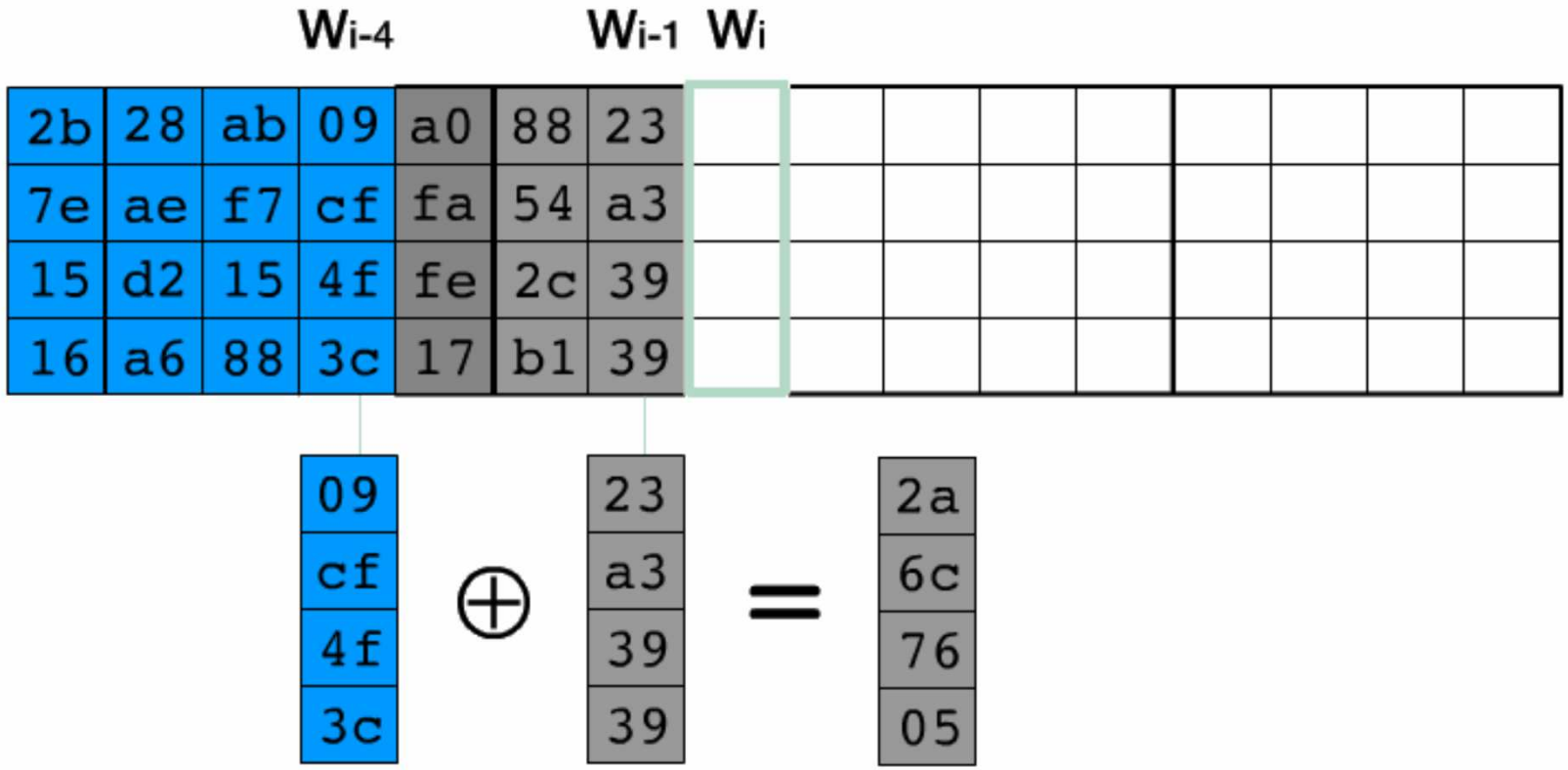
- Les clés de ronde
  - Sont générées une fois pour toute
  - Sont stockées dans le tableau d'octets de taille Nb*(Nr+1)
  - KeyExpansion() : procédure de génération des clés

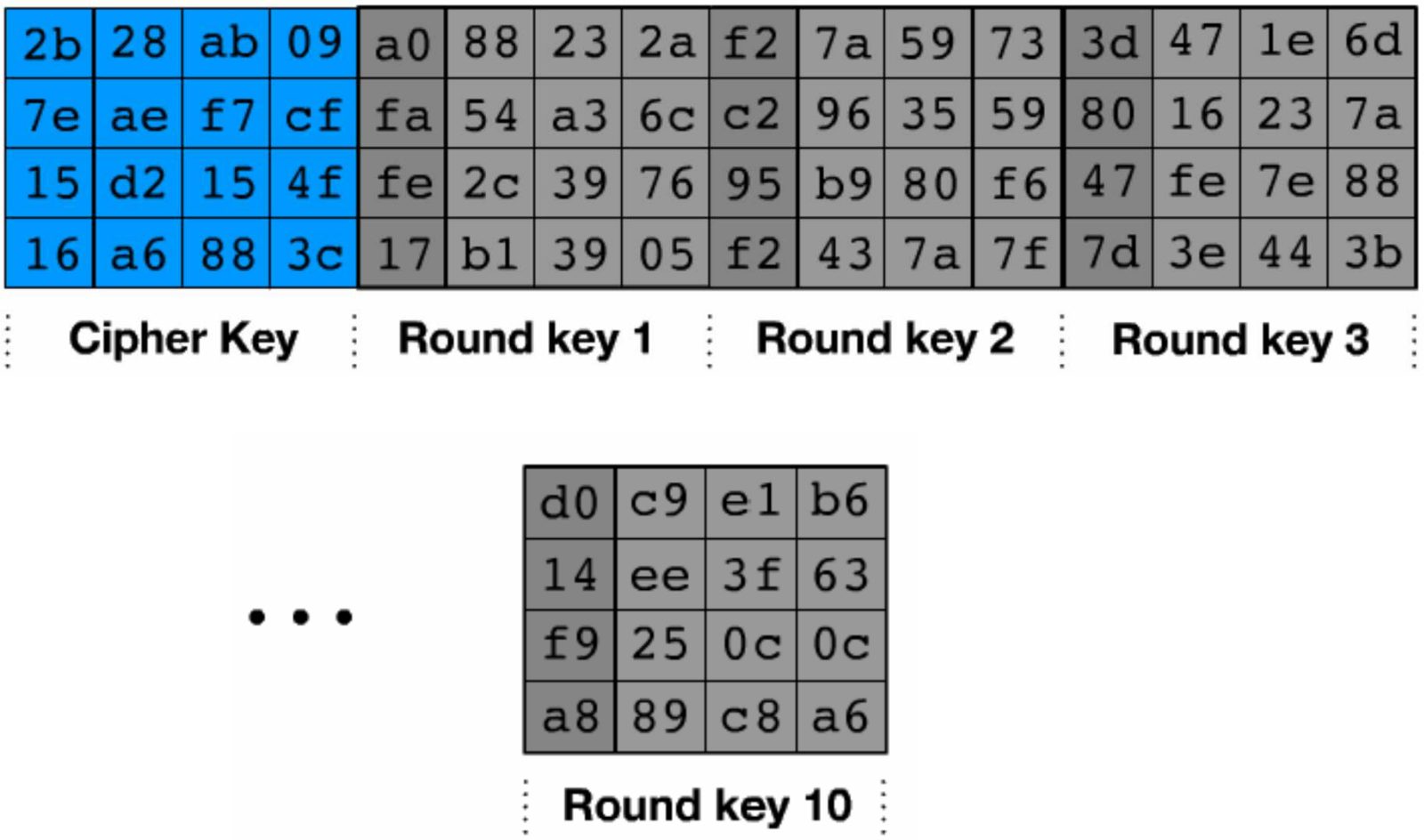
```
KeyExpansion (byte key[4*Nk], word w[Nb * (Nr+1)], Nk)
début
   word temp

   i = 0;

   tant que (i < Nk) faire
       w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
       i = i+1
   fin tant que

       i = Nk

   tant que (i < Nb * (Nr+1)) faire
       tmp = w[i-1]
       si (i mod Nk = 0) alors
          tmp = SubWord(RotWord(tmp)) Xor Rcon[i/Nk]
       sinon si (Nk > 6 et i mod Nk = 4) alors
          tmp = SubWord(tmp)
       fin si
       w[i] = w[i-Nk] Xor tmp
       i = i + 1
   fin tant que
fin
```


Wi-1 Wi
2b 28 ab 09
7e ae f7 cf
15 d2 15 4f
16 a6 88 3c
09 cf 4f 3c
RotWord
cf 4f 3c 09
SubBytes
8a 84 eb 01
2b 7e 15 16
8a 84 eb 01
01 00 00 00
a0 fa fe 17
Rcon(4)


Wi-4
Wi-1
Wi
2b 28 ab 09 a0
7e ae f7 cf fa
15 d2 15 4f fe
16 a6 88 3c 17
28 ae d2 a6
⊕
a0 fa fe 17
=
88 54 2c b1

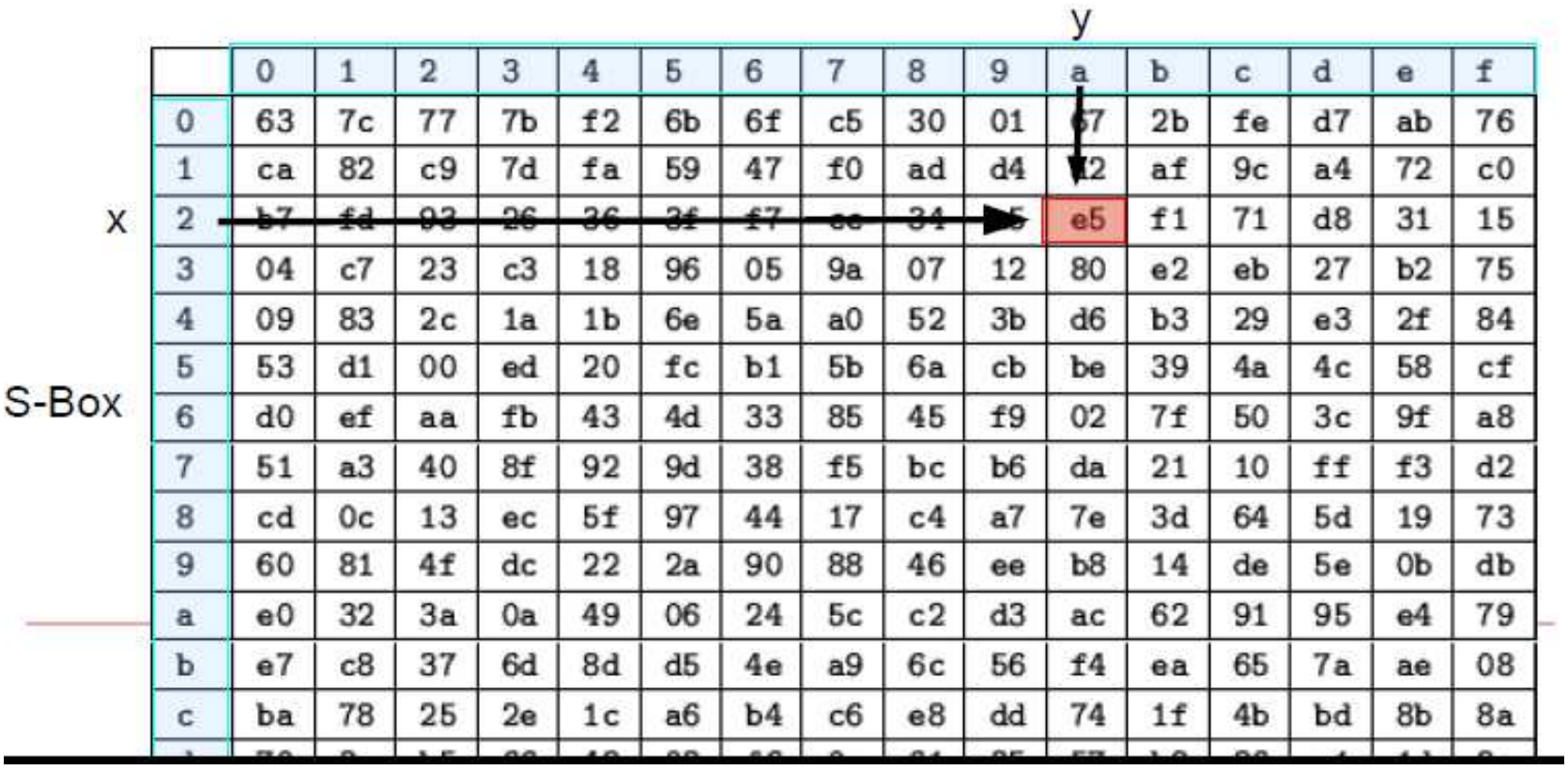

Wi-4
Wi-1
Wi
2b 28 ab 09 a0 88
7e ae f7 cf fa 54
15 d2 15 4f fe 2c
16 a6 88 3c 17 b1
ab
f7
15
88
88
54
2c
b1
23
a3
39
39


Wi-4
Wi-1
Wi
2b 28 ab 09 a0 88 23
7e ae f7 cf fa 54 a3
15 d2 15 4f fe 2c 39
16 a6 88 3c 17 b1 39
09
cf
4f
3c
23
a3
39
39
2a
6c
76
05


2b 28 ab 09 a0 88 23 2a f2 7a 59 73 3d 47 1e 6d
7e ae f7 cf fa 54 a3 6c c2 96 35 59 80 16 23 7a
15 d2 15 4f fe 2c 39 76 95 b9 80 f6 47 fe 7e 88
16 a6 88 3c 17 b1 39 05 f2 43 7a 7f 7d 3e 44 3b
Cipher Key
Round key 1
Round key 2
Round key 3
d0 c9 e1 b6
14 ee 3f 63
f9 25 0c 0c
a8 89 c8 a6
Round key 10

- SubByte()
  - Reçoit un octet en entrée et renvoie une valeur de substitution
  - Un octet est vu comme un couple de 2 nombres de 4 bits : x et y
  - La valeur de substitution est le contenu de la table « S-Box » de coordonnées (x,y)
  - L'essentiel de la sécurité repose sur la non linéarité de la S-Box : chaque valeur est le résultat d'une transformation affine dans GF(2^8)

- Exemple (en hexadécimal) :
  - En entrée : 2a
  - En sortie : e5



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a

- SubWord(mot)
  - Reçoit un mot de 4 octets
  - Renvoie un mot de 4 octets
  - Chaque octet en entrée est substitué par SubByte()
- RotWord(mot)
  - Reçoit un mot de 4 octets (a_0,a_1,a_2,a_3)
  - Renvoie le mot (a_1,a_2,a_3,a_0)

- Rcon : tableau prédéfini

$$Rcon[i] = [x^{i-1}, \{00\}, \{00\}, \{00\}]$$

- En d'autres termes :

```
static const unsigned int   Rcon[Nr+1] = {
        0x00000000, 0x01000000, 0x02000000, 0x04000000, 0x08000000, ...
};
```

Le chiffrement: Chiffrement d'un bloc

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
 début
  1        byte state[4,Nb]
  2
  3        state = in       // l'entrée est copiée dans le bloc
  4
  5        AddRoundKey(state, w[0, Nb-1])
  6
  7        pour round = 1 à Nr-1 par pas de 1 faire
  8             SubBytes(state)
  9             ShiftRows(state)
 10             MixColumns(state)
 11             AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
 12        finpour
 13
 14        SubBytes(state)
 15        ShiftRows(state)
 16        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
 17
 18        out = state    // copie du bloc dans la sortie
 fin
```

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

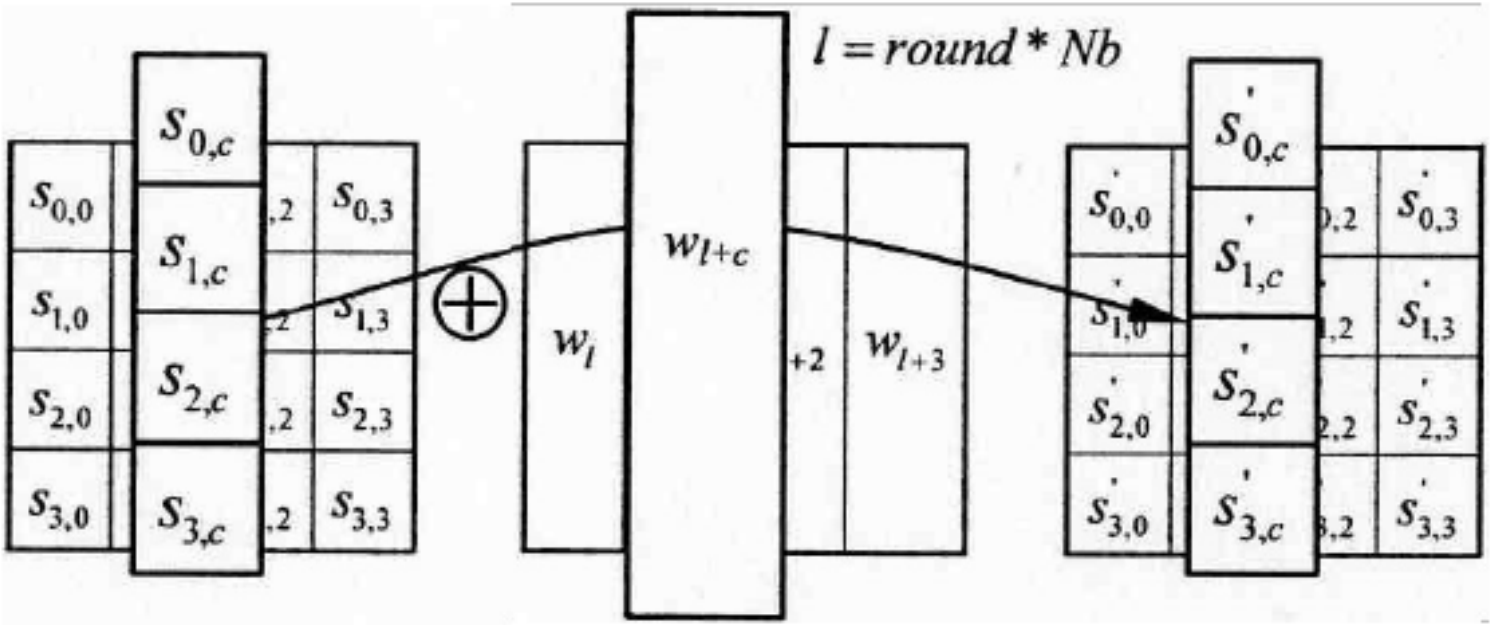
a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

Round key

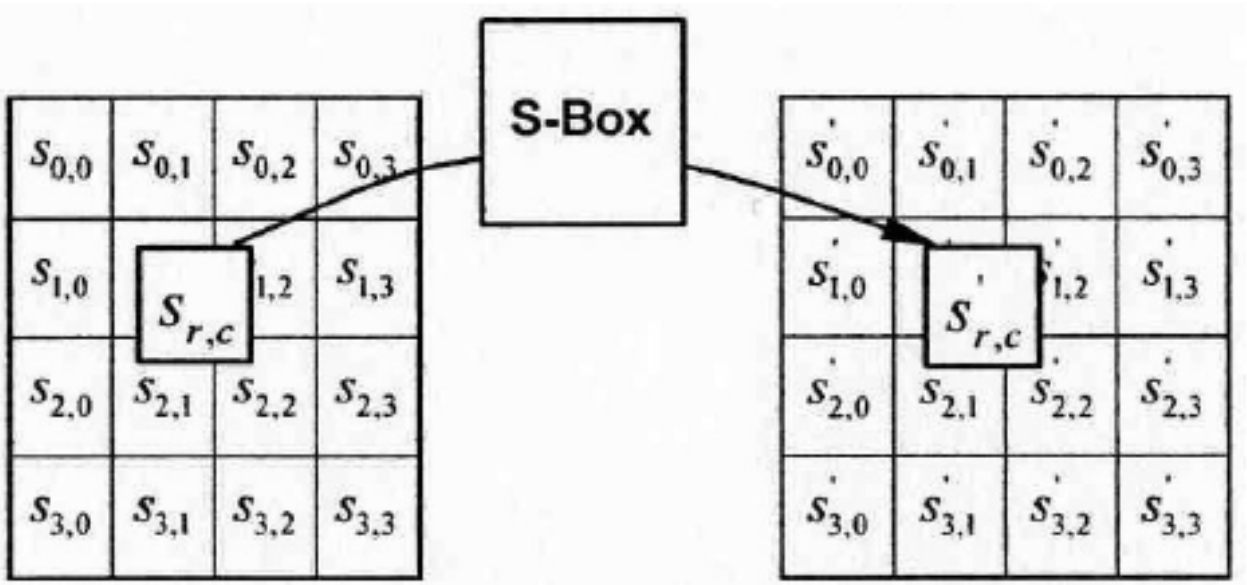
04		a0	=	a4
66	⊕	fa	=	9c
81		fe	=	7f
e5		17	=	f2

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

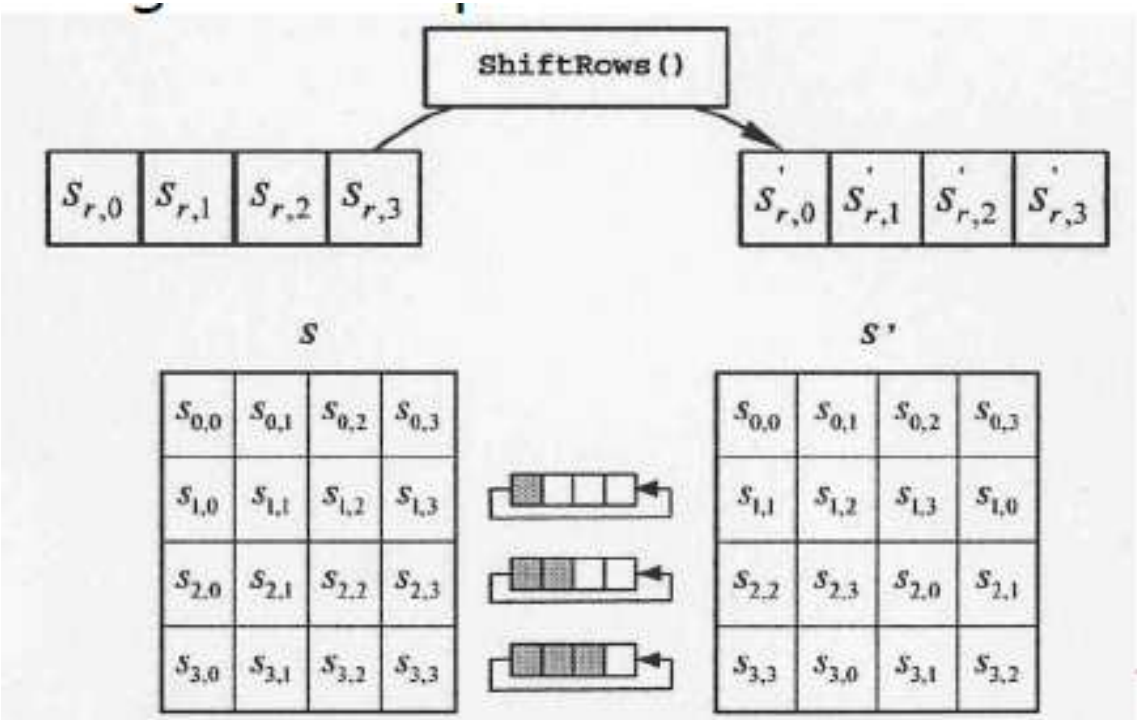
- AddRoundKey(state,clé)
  - Ajoute la clé de ronde à l'état
  - Principe :



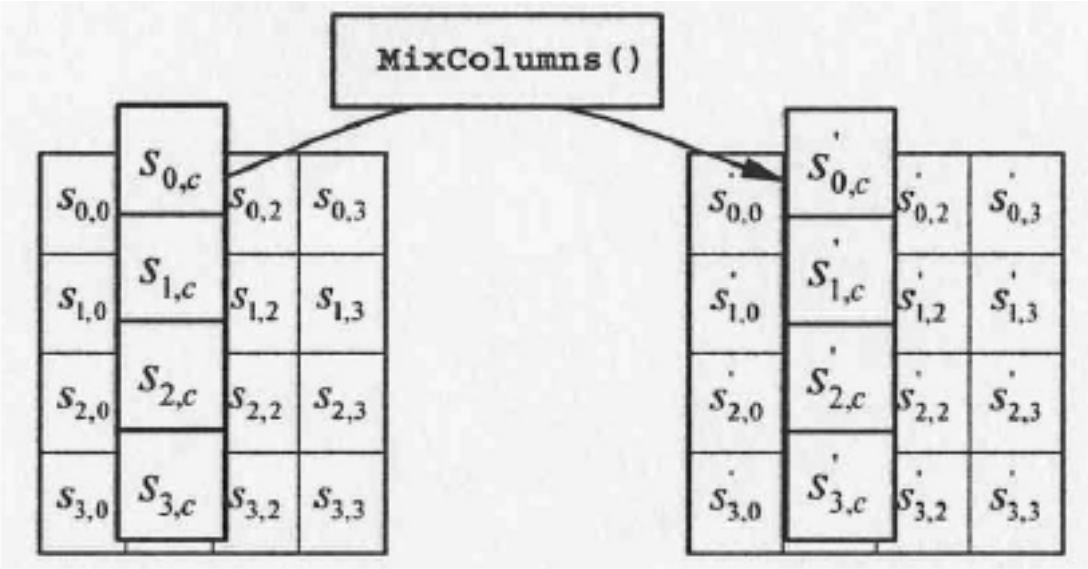
- SubBytes(state)
  - Réalise une substitution de chaque octet de l'état (voir SubByte)
  - Principe :



- ShiftRows(state)
  - Effectue une rotation vers la gauche des octets de chaque ligne
  - Le numéro de ligne indique le nombre de rotations à effectuer
  - Principe :



- MixColumns(state)
  - Principe :



- Chaque colonne est considérée comme un polynôme de GF(2^8)

- On applique le produit matriciel (cf. 18) avec la matrice suivante :

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

- En d'autres termes, pour $S'_{0,c}$:

$$S'_{0,c} = [02] \bullet S_{0,c} + [03] \bullet S_{1,c} + [01] \bullet S_{2,c} + [01] \bullet S_{3,c}$$

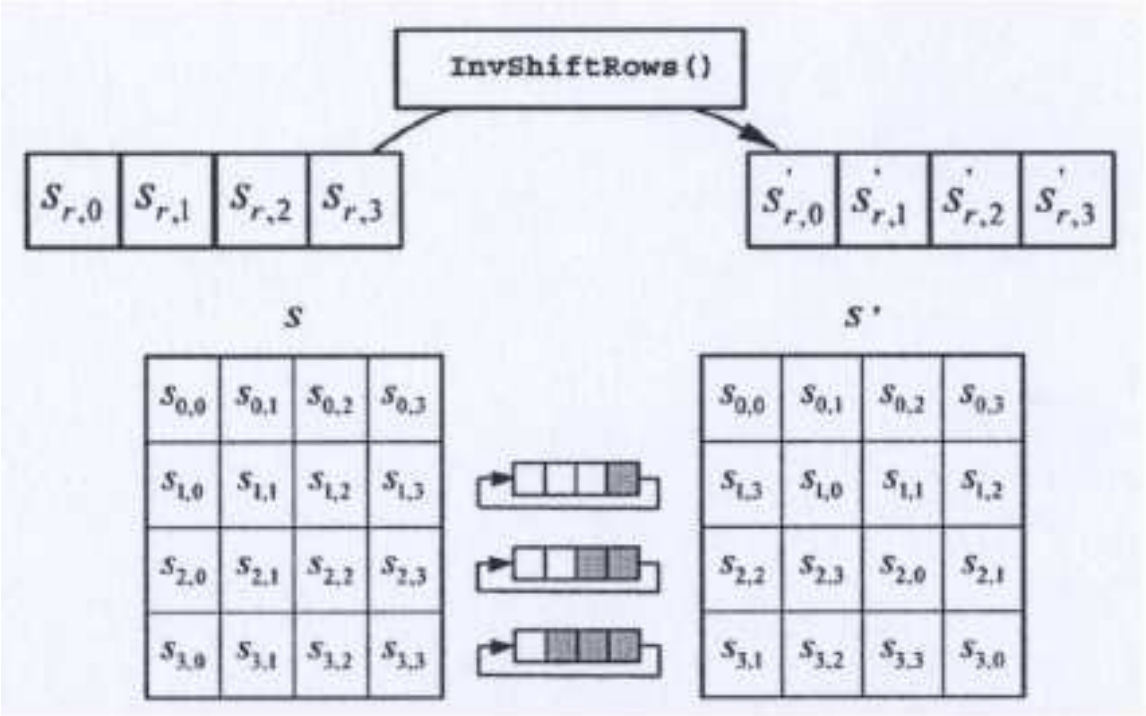
$$S'_{0,c} = xtime(S_{0,c}) \oplus (xtime(S_{1,c}) \oplus S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

Le déchiffrement

- Algorithme identique mais avec des transformations inversées
  - Les rondes sont réalisées à rebours
  - L'algorithme :

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
début
 1       byte state[4,Nb]
 2
 3       state = in      // l'entrée est copiée dans le bloc
 4
 5       AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
 6
 7       pour round = Nr-1 à 1 par pas de -1 faire
 8            InvShiftRows(state)
 9            InvSubBytes(state)
10            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
11            InvMixColumns(state)
12       finpour
13
14       InvShiftRows(state)
15       InvSubBytes(state)
16       AddRoundKey(state, w[0, Nb-1])
17
18       out = state   // copie du bloc dans la sortie
fin
```

- InvShiftRows(state)
  - Rotation circulaire vers la droite
  - Principe :



- InvSubBytes(state)
  - Applique à chaque octet de l'état, une substitution (cf. SubByte()) au moyen d'une S-Box obtenue par la transformation affine inverse

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61

- InvMixColumns(state)
  - Principe identique à MixColumns
  - La matrice est composée des coefficients de $a^{-1}(x)$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$