Projet en langage C++

Implémentation de l'algorithme A.E.S

Sebastien.Varrette@imag.fr

1 Objectifs

L'objectif de ce projet est la réalisation d'une implémentation du cryptosystème à clé secrète AES en C++.
Un cryptosystème permet à deux protagonistes, appelés traditionnellement Alice et Bob, de communiquer ensemble sur un canal peu sûr lorsqu'un opposant, Oscar, souhaite espionner cette conversation. Evidemment, Oscar ne doit pas comprendre les informations qui sont échangées.

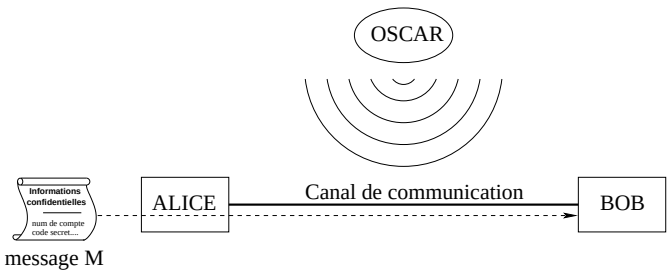

Figure 1 – Les protagonistes d'un cryptosystème

Pour un cryptosystème, on définit les expressions suivantes :
- **Texte clair** : information qu'Alice souhaite transmettre à Bob (Ex : texte en français, donnée numérique etc...)
- **Chiffrement** : processus de transformation d'un message M de telle manière à le rendre incompréhensible. Ce processus est basé sur une *fonction de chiffrement* E et permet de génèrer ainsi un **message chiffré** $C = E(M)$
- **Déchiffrement** : processus de reconstruction du message clair à partir du message chiffré, basé sur une fonction de déchiffrement D.
  On a donc $D(C) = D(E(M)) = M$ (D et E sont injectives).

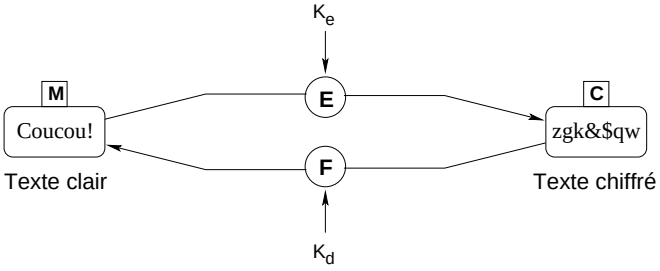

Figure 2 – Illustration du chiffrement d'un texte clair

En pratique : E et D sont paramétrées par des clefs K_e et K_d (comme illustré

dans la figure 2) et sont liées par l'équation 1 :

$$\begin{cases} E_{K_e}(M) = C \\ D_{K_d}(C) = M \end{cases} \tag{1}$$

Le lien qui unit K_e et K_d définit deux grandes catégories de systèmes cryptographiques :

1. les systemes à clef secrète (ou symétriques) ($K_e = K_d = K$). C'est le cas du système A.E.S.
2. les systèmes à clef publique (ou asymétriques) ($K_e \neq K_d$) comme par exemple le système R.S.A.

Rijndael est le chiffrement à clef privée qui a été retenu par le NIST (National Institute of Standards and Technology) comme le nouveau standard américain de chiffrement AES : Advanced Encryption Standard [DR01] visant à remplacer DES (Data Encryption Standard).
C'est un code par blocs encodant 128 bits avec des clefs de 128, 192 ou 256 bits.

2 Conventions

2.1 Entrées et Sorties

Les entrées et les sorties d'AES consistent en des séquences de 128 bits. La clé secrète de chiffrement est une suite de 128, 192 ou 256 bits.

2.2 Interprétation des octets et représentation matricielle

Dans AES, les octets correspondent à des séquences de 8 bits interprétées comme des éléments du corps fini à 256 éléments $\mathbb{F}_{256}$ (voir §3). Ensuite, tout flux d'octets est organisé sous forme matricielle, selon un modèle illustré dans la figure 3. Cette matrice aura nécessairement 4 lignes et un nombre de colonnes

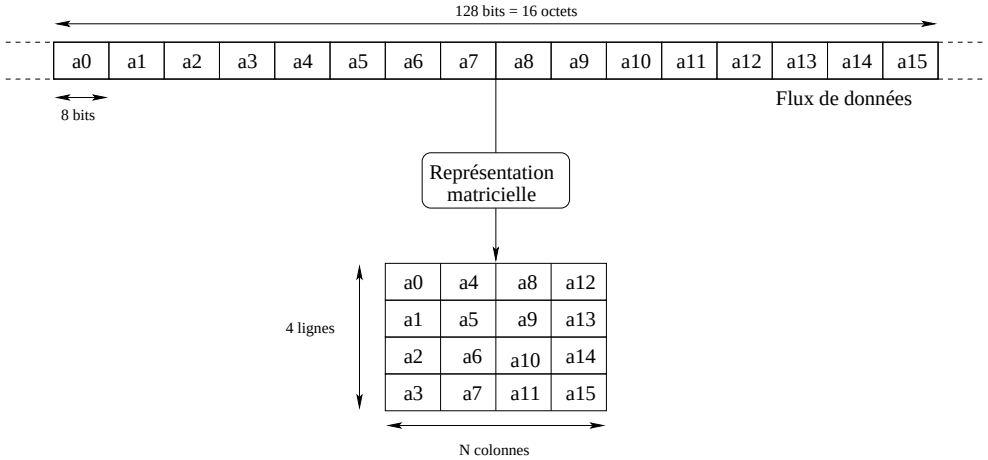

FIGURE 3 – Représentation matricielle d'un flux de 16 octets

fonction de la taille du flux, définissant ainsi une taille de bloc N. Par exemple,

pour les flux d'entrée/sortie qui, dans AES, correspondent à des séquences de 16 octets, on obtiendra des matrices de 4 lignes et $Nb = 4$ colonnes. De même,
– la matrice associée à une clé de 128 bits aura 4 lignes et $Nk = 4$ colonnes ;
– avec une clé de 192 bits, la matrice aura 4 lignes et $Nk = 6$ colonnes ;
– pour une clé de 256 bits, la matrice aura 4 lignes et $Nk = 8$ colonnes ;
Dans la suite, on notera `State` l'écriture matricielle du flux d'entrée/sortie sur lequel opère chaque étape de l'algorithme.

3 Préliminaires mathématiques : le corps $\mathbb{F}_{256}$

3.1 Rappels : construction d'un corps fini à p^m éléments

On considère d'abord le corps fini à p éléments $\mathbb{F}_p$ (ou p est premier).
Comme p est premier, $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. Soit m un entier positif. On souhaite construire le corps $\mathbb{F}_{p^m}$ possédant p^m éléments. Pour cela, on considère g un polynôme unitaire irréductible de $\mathbb{F}_p[X]$ de degré m. Alors on peut montrer que $\mathbb{F}_{p^m}$ est isomorphe à $\mathbb{F}_p[X]/g(X)$. Autrement dit, tout élément de $\mathbb{F}_{p^m}$ peut être vu comme un polynôme de $\mathbb{F}_p[X]$ (i.e dont les coefficients sont dans $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$) de degré inférieur à m :

$$\mathbb{F}_{p^m} \simeq \left\{ \sum_{i=0}^{m-1} a_i X^i, \quad a_i \in \mathbb{F}_p \right\} = \left\{ \sum_{i \geq 0} a_i X^i \mod g(X), \quad a_i \in \mathbb{F}_p \right\}$$

3.2 Applications à la constructions de $\mathbb{F}_{256}$

Dans ce cas, $p = 2$ et $m = 8$. Il convient donc de choisir un polynôme unitaire irréductible de degré 8. Plusieurs choix sont possibles.
Ainsi, les encodeurs/décodeurs CIRC utilisés pour les CD audio utilisent le polynôme $g(X) = X^8 + X^7 + X^2 + X + 1$.
Pour AES, le polynôme utilisé est : $g(X) = X^8 + X^4 + X^3 + X + 1$.
On aura donc :

$$\mathbb{F}_{256} \simeq \left\{ \sum_{i=0}^{7} a_i X^i, \quad a_i \in \{0,1\} \right\} = \left\{ \sum_{i \geq 0} a_i X^i \mod g(X), \quad a_i \in \{0,1\} \right\}$$

En pratique, on représentera le polynome $a_7X^7 + \ldots + a_1X + a_0$ par l'octet $a_7a_6 \ldots a_1a_0$. On dira qu'on utilise une notation polynomiale des éléments de $\mathbb{F}_{256}$.

3.3 Addition dans $\mathbb{F}_{256}$

L'addition de deux éléments de $\mathbb{F}_2 = \mathbb{Z}/_2\mathbb{Z}$ correspond à l'opération XOR $\oplus$. Ainsi, l'addition de deux éléments a et b de $\mathbb{F}_{256}$ correspond à l'addition de deux polynômes à coefficients dans $\mathbb{F}_2 = \mathbb{Z}/_2\mathbb{Z}$:

$$\begin{aligned} a(x) &= a_7x^7 + \ldots + a_1x + a_0 \\ b(x) &= b_7x^7 + \ldots + b_1x + a_0 \\ \Longrightarrow a(x) + b(x) &= (a_7 \oplus b_7)x^7 + \ldots + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \end{aligned}$$

Exemple :

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$	(notation polynomiale)
$\{01010111\} + \{10000011\} = \{11010100\}$	(notation binaire)
$\{57\} + \{83\} = \{d4\}$	(notation hexadécimale)

Remarque En notation hexadécimale, le polynôme $g(X)$ s'écrit `0x11B`.

3.4 Multiplication dans $\mathbb{F}_{256}$

En notation polynômiale, la multiplication dans $\mathbb{F}_{256}$ correspond à la multiplication de deux polynômes suivie d'une réduction modulo $g(x)$. Exemple :

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1) \times (x^7 + x + 1) &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ &\quad x^7 + x^5 + x^3 + x^2 + x + \\ &\quad x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

Et
$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1) \mod (x^8+x^4+x^3+x+1) = x^7+x^6+1$.
La réduction modulo $g(x)$ assure que le résultat reste un polynôme binaire de degré inférieur à 8 qui peut donc être représenté par un octet. Dans l'exemple précédent, on a obtenu : $\{57\} \times \{83\} = \{C1\}$.
En pratique, pour réaliser cette multiplication, on dispose de deux méthodes, expliquées dans les sections suivantes.

3.4.1 Méthode "lente" : multiplication par x^i.

Cette méthode consiste à réaliser effectivement la multiplication polynomiale. Soit $a \in \mathbb{F}_{256}$. En notation polynomiale, $a(x) = a_7x^7 + \ldots + a_1x + a_0$
Ainsi, $x.a(x) = a_7x^8 + \ldots + a_1x^2 + a_0x$. Il reste à effectuer la réduction modulo $g(x)$. Deux cas sont possibles :

- Soit $a_7 = 0$, on obtient directement une expression réduite et donc $x.a(x) = a_6x^7 + \ldots + a_1x^2 + a_0x$.
- Soit $a_7 = 1$ et dans ce cas : $x.a(x) = x^8 + \ldots + a_1x^2 + a_0x$. En outre, $g(x) = x^8+x^4+x^3+x+1 \equiv 0 \mod g(x) \Longrightarrow x^8 \equiv x^4+x^3+x+1 \mod g(x)$ donc

  $$x^8 + a_6x^7 + \ldots + a_1x^2 + a_0x = (a_6x^7 + \ldots + a_1x^2 + a_0x) \oplus (x^4 + x^3 + x + 1)$$

  En notation polynomiale, cette opération consiste donc à un décalage à gauche suivi éventuellement d'un XOR bits-à-bits avec `{1B}`.

Soit `xtime` la fonction qui réalise l'opération $x \times a(x) \mod g(x)$. `xtime` peut être utilisée pour définir la fonction `xi_time` qui réalise l'opération $x^i \times a(x) \mod g(x)$. Enfin, à partir de cette dernière fonction, il est possible de réaliser la multiplication complète de deux éléments de $\mathbb{F}_{256}$.

3.4.2 Méthode "rapide" à partir d'une représentation exponentielle.

Soit $w(x)$ un générateur de $\mathbb{F}_p[X]/g(X)$: dans ce cas, tout élément non nul de $\mathbb{F}_{256}$ se représente de manière unique par $w(x)^i \mod g(x)$, $0 \leq i < 255$.
On obtient ainsi une autre écriture pour $\mathbb{F}_{256}$:

$$\mathbb{F}_{256} \simeq \{0\} \cup \left\{w(x)^i \mod g(x)\right\}_{0 \leq i < 255}$$

Avec cette représentation (dite cyclique ou exponentielle), la multiplication de deux éléments a et b non nuls est aisée :

$$\begin{aligned} a(x) &= w(x)^i && (2) \\ b(x) &= w(x)^j && (3) \\ a(x) \times b(x) &= w(x)^{i+j} \mod 255 && (4) \end{aligned}$$

L'idée consiste alors a passer par cette représentation pour effectuer la multiplication. On utilise pour cela des tables de correspondance entre les éléments a(x) de $\mathbb{F}_{256}$ et l'exposant i correspondant à sa représentation exponentielle.
A noter que la valeur de w(x) **dépend** de g(x).
Le tableau suivant fournit un générateur simple pour quelques polynômes irréductibles g(x).

Polynome irréductible	Rep. hexa	générateur	Rep. hexa	Application
$x^8 + x^7 + x^2 + x + 1$	`0x187`	$w(x) = x$	`0x02`	Codes CIRC
$x^8 + x^4 + x^3 + x + 1$	`0x11B`	$w(x) = x + 1$	`0x03`	AES

TABLE 1 – Quelques exemples de générateurs pour $\mathbb{F}_p[X]/g(X)$

Ainsi, à partir de la fonction de multiplication lente, on génère une table `ExpoToPoly` de 256 entrées telle que `ExpoToPoly[k]` donne la representation polynomiale de $w(x)^k \mod g(x)$. (Par convention, on représentera l'élément 0 par $w(x)^{255}$ bien que mathematiquement, $w^{255} = 1$. La table `PolyToExpo` correspond à la table inverse. Ces tables sont fournie en annexe B.
On utilise ces tables pour effectuer efficacement la multiplication de deux éléments a et b à partir de la relation (4).

4 Description de l'algorithme de chiffrement A.E.S

Pour être tout à fait exact, l'algorithme AES n'est pas exactement celui de Rijndael dans la mesure ou ce dernier supporte des tailles de blocs plus nombreux qu'AES. AES **fixe la taille de blocs à 128 bits** - représenté par $\mathbf{N_b = 4}$. N_b reflète le nombre de mots de 32 bits dans un bloc (c'est aussi le nombre de colonnes nécessaire pour une représentation matricielle) .
AES utilise des **clés de 128, 192 ou 256 bits**. La longueur de clé est caractérisée de façon similaire par $\mathbf{N_k = 4, 6}$ ou $\mathbf{8}$.
Comme DES, AES exécute une séquence de rondes qui seront détaillés dans la suite. On note N_r le nombre de rondes qui doivent être effectuées. Ce nombre

	N_k	N_b	N_r
`AES-128`	4	4	10
`AES-192`	6	4	12
`AES-256`	8	4	14

TABLE 2 – Détail des configurations possibles

dépend des valeurs de N_b et de N_k. Les différentes configurations possibles sont détaillés dans le tableau 2.
Comme on l'a vu en §2.2, AES opère sur une matrice $4 \times N_b$ d'éléments de $\mathbb{F}_{256}$, notée `State`. Le chiffrement AES consiste en une addition initiale de clé, notée `AddRoundKey`, suivie par $N_r - 1$ rondes, chacune constitué de quatre étapes :

1. `SubBytes` : il s'agit d'une substitution non-linéaire lors de laquelle chaque octet est remplacé par un autre octet choisi dans une table particulière une *Boite-S*).
2. `ShiftRows` est une étape de transposition ou chaque éléments de la matrice est décalée cycliquement à gauche d'un certain nombre de colonnes.
3. `MixColumns` effectue une produit matriciel en opérant sur chaque colonnes (vu alors comme un vecteur) de la matrice.
4. `AddRoundKey` qui combine par addition chaque octet avec l'octet correspondant dans une clé de ronde obtenue par diversification de la clé de chiffrement.

Enfin, une ronde finale `FinalRound` est appliquée (elle correspond à une ronde dans laquelle l'étape `MixColumns` est omise).
La clé de ronde pour l'étape i sera notée `RoundKeys[i]`, et `RoundKeys[0]` référencera un des paramètres de l'addition initiale de clé. La dérivation de la clé de chiffrement `K` dans le tableau `RoundKeys[]` est notée `KeyExpansion` et sera détaillée au §4.5.
Le chiffrement AES peut se décrire de façon formelle de la façon suivante :

```
AES_Encrypt(State, K) {
    KeyExpansion(K, RoundKeys);
    /* Addition initiale */
    AddRoundKey(State, RoundKeys[0]);
    /* Les Nr-1 rondes */
    for (r=1; i<Nr; r++) {
        SubBytes(State);
        ShiftRows(State);
        MixColumns(State);
        AddRoundKey(State, RoundKeys[r]);
    }
    /* FinalRound */
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State, RoundKeys[Nr]);
}
```

Les sections suivantes détaillent chaque étape.

4.1 Etape `SubBytes`

L'étape `SubBytes` correspond à la seule transformation non-linéaire de l'algorithme. Dans cette étape, chaque élément de la matrice `State` est permuté selon une table de substitution inversible notée `SBox`. Cette table est fournie en annexe C. La figure 4 illustre par exemple la transformation de l'élément $a_{2,2}$ en l'élément $b_{2,2} = S[a_{2,2}]$.

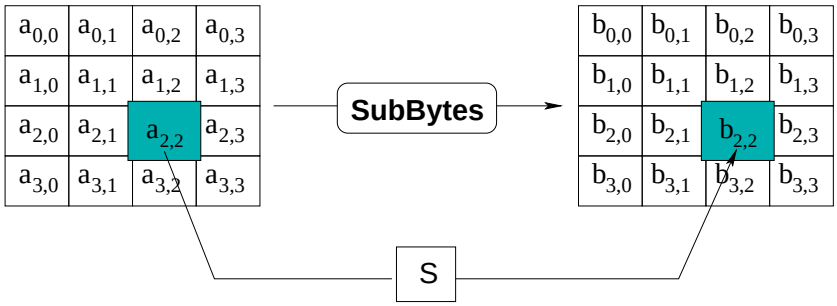

FIGURE 4 – Etape `SubBytes`

Remarque : La table `SBox` dérive de la fontion inverse $t : a \longrightarrow a^{-1}$ sur $\mathbb{F}_{256}$. Cette fonction est connue pour ses bonnes propriétés de non-linéarité. Afin d'éviter des attaques basées sur de simples propriétés algébriques, la boite-S est construite en combinant cette fonction inverse avec une transformation affine inversible f. On a donc :

$$\texttt{SBox}[a] = f\left(t(a)\right), \forall a \in \mathbb{F}_{256}$$

Les concepteurs ont également fait en sorte que cette Boite-S n'admette pas de point fixe ni de point fixe opposé :

$$\texttt{SBox}[a] + a \neq \texttt{00}, \quad \forall a \in \mathbb{F}_{256}$$
$$\texttt{SBox}[a] + a \neq \texttt{FF}, \quad \forall a \in \mathbb{F}_{256}$$

Enfin, la fonction affine f est définie par :

$$b = f(a) \Longleftrightarrow \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Opération inverse L'opération inverse de `SubBytes` est notée `InvSubBytes` et consiste à effectuer la même manipulation mais à partir de la Boite-S inverse S^{-1} notée `InvSBox`. Cette table est également fournie en annexe C.

Remarque : mathématiquement, comme la fonction t est son propre inverse, on a :

$$\texttt{SBox}^{-1}[a] = t^{-1}\left(f^{-1}(a)\right) = t\left(f^{-1}(a)\right), \forall a \in \mathbb{F}_{256}$$

La fonction affine inverse f^{-1} est définie par :

$$b = f^{-1}(a) \Longleftrightarrow \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

4.2 L'étape `Shiftrows`

L'étape `Shiftrows` opère sur les lignes de la matrice `State` et effectue pour chaque élément d'une ligne un décalage cyclique de n éléments vers la gauche. L'offset n de décalage dépend de la ligne considérée. La ligne i est décalé de C_i éléments, si bien que l'élément en position j de la ligne i est déplacé en position $(j - C_i) \mod N_b$. Les valeurs de C_i dépendent de la valeur de N_b et sont détaillée dans la table 3.

N_b	C_0	C_1	C_2	C_3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

TABLE 3 – `Shiftrows` : décalage des lignes en fonction de N_b dans Rijndael

Evidemment, cette table n'est fournit qu'a titre indicatif dans la mesure où AES fixe la taille de bloc à $N_b = 4$. L'étape `Shiftrows` est illustrée dans la figure 5.

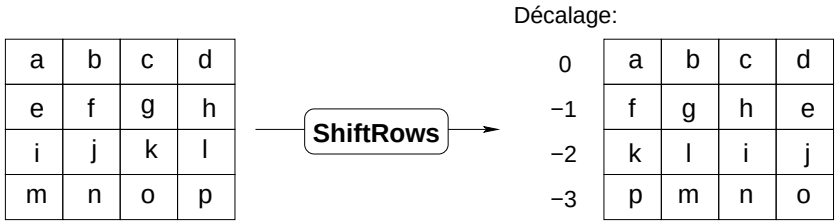

FIGURE 5 – Opération `Shiftrows` dans AES

Opération inverse L'opération inverse de `Shiftrows` est notée `InvShiftrows` et consiste à effectuer au niveau le la ligne i un décalage cyclique à droite de C_i éléments.

4.3 `MixColumns`

La transformation `MixColumns` opère sur les colonnes c de la matrice `State` en les traitant comme un polynôme $a(x)$ de degré 3 à coefficients dans $\mathbb{F}_{256}$.
L'étape `MixColumns` consiste alors a effectuer pour chaque colonne une multiplication par un polynôme $c(x)$ fixé suivi d'une réduction modulo le polynôme $x^4 + 1$. Dans `MixColumns`, on réalise donc l'opération :

$$(\texttt{03}x^3 + x^2 + x + \texttt{02}) \times a(x) \mod (x^4 + 1)$$

Matriciellement, cette opération s'ecrit :

$$b(x) = c(x) \times a(x) \mod (x^4+1) \Longleftrightarrow \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

La figure 6 illustre cette étape.

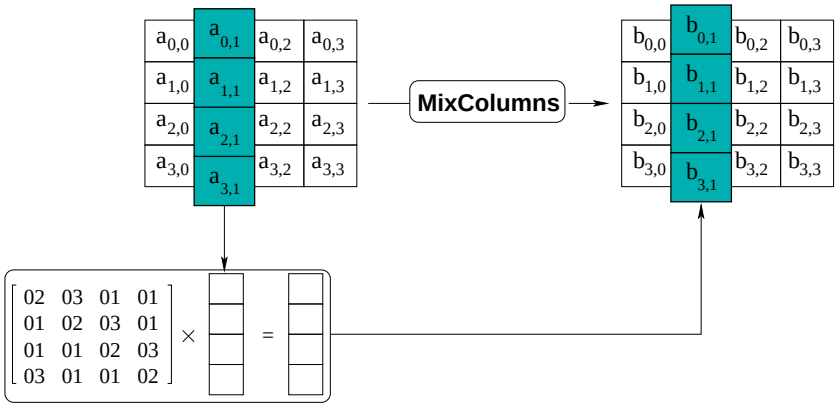

FIGURE 6 – Opération `MixColumns`

Opération inverse L'opération inverse de `MixColumns` est notée `InvMixColumns` et consiste à effectuer la même opération mais à partir d'une multiplication par le polynôme $d(x) = c^{-1}(x)$ donné par la relation :

$$(\texttt{03}x^3 + x^2 + x + \texttt{02}) \times d(x) \equiv \texttt{01} \mod (x^4 + 1)$$

On obtient ainsi :

$$d(x) = \texttt{0B}x^3 + \texttt{0D}x^2 + \texttt{09}x + \texttt{0E}$$

Matriciellement, cette étape revient a effectuer le calcul suivant :

$$b(x) = d(x) \times a(x) \mod (x^4+1) \Longleftrightarrow \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \texttt{0E} & \texttt{0B} & \texttt{0D} & \texttt{09} \\ \texttt{09} & \texttt{0E} & \texttt{0B} & \texttt{0D} \\ \texttt{0D} & \texttt{09} & \texttt{0E} & \texttt{0B} \\ \texttt{0B} & \texttt{0D} & \texttt{09} & \texttt{0E} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

4.4 `AddRoundKey`

Lors de l'étape `AddRoundKey`, la matrice `State` est modifiée en l'additionnant (au sens de l'addition termes à termes dans $\mathbb{F}_{256}$) avec une clé de ronde. Cette étape est illustrée dans la figure 7.

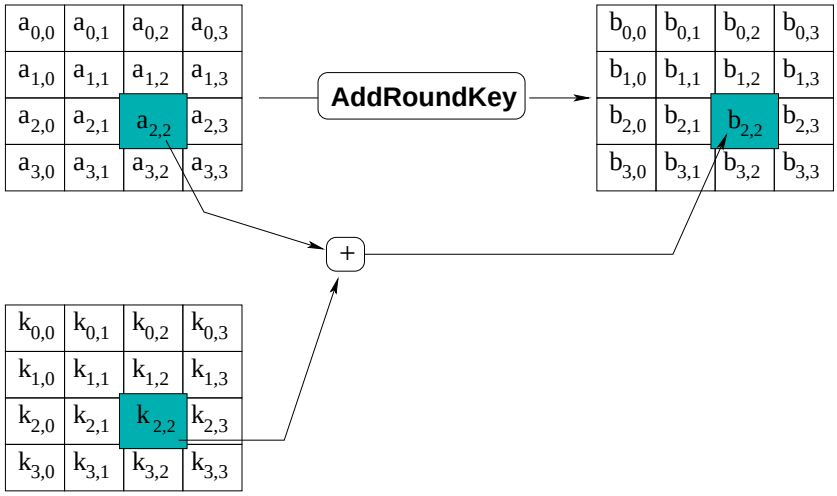

FIGURE 7 – Opération `AddRoundKey`

4.5 Détails de la diversification de la clef dans AES

Cette étape, noté `KeyExpansion`, permet de diversifier la clé de chiffrement `K` (de $4N_k$ octets) dans une clé étendue `W` de $4N_b(N_r + 1)$ octets. On disposera ainsi de $N_r + 1$ clés de rondes (chacune de $4Nb$ octets - voir figure 8).

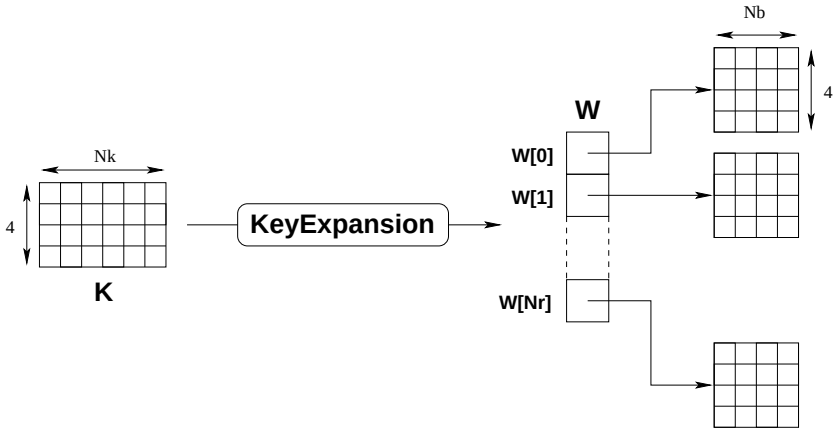

FIGURE 8 – Opération `KeyExpansion`

Les tableaux `K` et `W` peuvent être vus comme une succession de colonnes chacunes constituées de 4 octets, comme l'illustre la figure 9. Dans la suite, on notera `c[i]` (resp. `k[i]`) la $(i+1)^{\text{ème}}$ colonne de `W` (resp. de `K`).

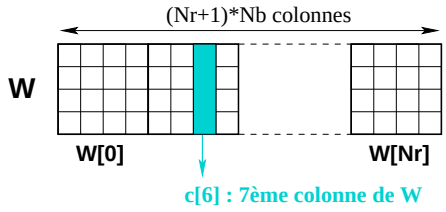

FIGURE 9 – Vision de `W` comme une succession de colonnes.

L'algorithme utilisé pour la diversification de clé diffère légèrement selon que $N_k \leq 6$ ou $N_k > 6$. Dans tous les cas, les N_k premières colonnes de `K` sont recopiées sans modifications aux N_k premières colonnes de `W`. Les colonnes suivantes sont définies récursivement à partir des colonnes précédentes. `KeyExpansion` y utilise notamment les éléments suivants :

- `SubWord` qui est une fonction prenant en entrée un mot de 4 octets et applique la boite-S `Sbox` sur chacun des octets.
- La fonction `RotWord` qui prend en entrée un mot de 4 octets $a = [a_0, a_1, a_2, a_3]$ et effectue une permutation circulaire de façon à renvoyer le mot $[a_1, a_2, a_3, a_0]$.
- Le tableau de constantes de rondes `Rcon[i]`, indépendant de N_k, qui est défini récursivement par :

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

  On pourra utiliser la fonction `xi_time` (définie au §3.4.1) pour calculer la valeur de `Rcon[i]`.

La définition formelle en pseudo-code C de l'étape `KeyExpansion` est alors la suivante :

```
KeyExpansion(K, W) {
    /* Recopie directe des Nk premiere colonnes */
    for (i=0; i<Nk; i++) W[i] = K[i];
    for (i=Nk; i<Nb*(Nr+1); i++) {
        tmp = W[i-1];
        if (i mod Nk == 0)
            tmp = SubWord(RotWord(tmp)) + Rcon[i/Nk];
        else if ((Nk > 6) && (i mod Nk == 4)) // Cas Nk > 6
            tmp = SubWord(tmp);
        W[i] = W[i-Nk] + tmp;
    }
}
```

L'annexe A.1 fournit des exemples de diversification de clés.

4.6 Déchiffrement dans AES

La routine de chiffrement peut être inversée et réordonnée pour produire un algorithme de déchiffrement utilisant les transformations `InvSubBytes`, `InvShiftRows`, `InvMixColumns`, et `AddRoundKey`. Une modélisation formelle de l'algorithme de déchiffrement en pseudo-C pourrait être :

```
AES_Decrypt(State, K) {
    KeyExpansion(K, RoundKeys);
    /* Addition initiale */
    AddRoundKey(State, RoundKeys[Nr]);
    /* Les Nr-1 rondes */
    for (r=Nr-1; i>0; r--) {
        InvShiftRows(State);
        InvSubBytes(State);
        AddRoundKey(State, RoundKeys[r]);
        InvMixColumns(State);

    }
    /* FinalRound */
    InvShiftRows(Out);
    InvSubBytes(Out);
    AddRoundKey(Out,RoundKeys[0]);
}
```

Dans cette version du déchiffrement, la séquence des transformations diffère de celle du chiffrement (voir §4), le traitement de la clé restant inchangé.
Certaines propriétés du Rijndael permettent d'implémenter une routine de déchiffrement équivalente qui respecte la séquence de transformations de la routine `AES_Encrypt`, la structure de celle-ci étant la plus efficace. Cette version équivalente n'est pas demandée ici.

5 Cahier des charges de l'exécutable à produire

La compilation des sources produites devra fournir un exécutable `aes`.
On se limitera au chiffrement/déchiffrement d'un bloc de 128 bits. Les groupes qui le souhaitent pourront étendre leur système au chiffrement/déchiffrement de chaînes de longueur quelconque, voir même d'un fichier [1].
Cet exécutable devra supporter au minimum les options suivantes :

- `-h` : affichage de l'aide.
- `-k <key>` : utilisation de la cle <key> donnee sous forme hexadecimale sur 128 bits. *Valeur par defaut* : `2b7e151628aed2a6abf7158809cf4f3c`
- `-t <text>` : le texte a chiffrer sous forme d'un bloc de 128 bits ecrit en hexadécimal. *Valeur par defaut* : `00112233445566778899aabbccddeeff`
- `-d` : déchiffrement du texte `<text>`.
- `-b` : calcule le débit (bit rate) de chiffrement (en Ko/s) .
- `-v` : verbose, affiche des informations supplementaires.

Libre à vous également d'implémenter d'autres options (pour permettre par exemple de supporter tous les modes de chiffrements (`AES-128`, `AES-192`, `AES-256`)).
Voici par exemple le résultat de l'appel de cet exécutable sur quelques options :

1. Il se posera alors un problème de convention de padding

```
[13:10:36] seb@falkor ENSEIGNEMENTS/TD/AES_C++>./aes -h
NAME
      aes - Advanced Encryption Standard (Mode AES-128)

SYNOPSIS
      aes [-h] [-k key] [-t text] [-v] [-d] [-x]

DESCRIPTION
      Encrypt a clear Text using A.E.S
      -h : print help and exit
      -k : value of the private key (hexadecimal 128 bits format)
           Default value : 0x2b7e151628aed2a6abf7158809cf4f3c
      -t : text to (de/en)crypt (hexadecimal 128 bits format)
           Default value : 0x00112233445566778899aabbccddeeff
      -v : verbose mode
      -d : decryption mode
      -b : compute the bit rate when encrypting

AUTHOR
      Sebastien Varrette <Sebastien.Varrette@imag.fr>

REPORTING BUGS
      Report bugs to <Sebastien.Varrette@imag.fr>

COPYRIGHT
      This  is free software; see the source for copying conditions.
      There is NO warranty; not even for MERCHANTABILITY or
      FITNESS FOR A PARTICULAR PURPOSE.
SEE ALSO
       Federal Information Processing Standards Publication 197
       "Advanced Encryption Standard (AES)" - nov. 2001
        http://csrc.nist.gov/encryption/aes/rijndael/
[13:15:17] seb@falkor ENSEIGNEMENTS/TD/AES_C++>./aes
00112233445566778899aabbccddeeff --> 8df4e9aac5c7573a27d8d055d6e4d64b
[13:16:46] seb@falkor ENSEIGNEMENTS/TD/AES_C++>./aes -b

Debit : 80.000 Ko/s
[13:16:51] seb@falkor ENSEIGNEMENTS/TD/AES_C++>./aes -d \
          -t 8df4e9aac5c7573a27d8d055d6e4d64b
8df4e9aac5c7573a27d8d055d6e4d64b --> 00112233445566778899aabbccddeeff
```

Pour vous aider, une archive initiale est disponible à l'adresse `http://www-id.imag.fr/~svarrett/enseignements/2005-2006/Init_src_projetAES.tgz`

Cette archive contient notamment :

- un fichier de configuration `Makefile` générique qui devrait compiler votre projet directement par la commande `make`.
- un répertoire `Include/` qui contiendra vos fichiers de header (.h).
- un répertoire `Ressources/` qui contient un binaire qui devrait vous aider à tester votre propre exécutable. Attention, ce binaire est fourni sans aucune garantie !
- Enfin, si vous prenez le temps de commenter vos sources selon les conventions

de Doxygen [2], la commande `"make doc"` devrait générer automatiquement la documentation de vos sources (dans le répertoire `Doc/`). Cela pourra vous aider lors de la rédaction du rapport.

6 Consignes d'envoi

Un rapport avec les sources de votre projet devra m'être remis par mail (`Sebastien.Varrette@imag.fr`) ainsi qu'à Roland Gillard (`Roland.Gillard@ujf-grenoble.fr`) sous forme d'un fichier unique compressé (format `AES_src_<Nom1>_<Nom2>.tgz`) **au plus tard le lundi 4 décembre**.

2. `http://www.doxugen.org`

A Aide au débuggage

A.1 Diversification de la clé

Voici par exemple les clés de rondes dérivées de la clé
K=00000000000000000000000000000000

```
RoundKeys[00] = 00000000000000000000000000000000
RoundKeys[01] = 62636363626363636263636362636363
RoundKeys[02] = 9b9898c9f9fbfbaa9b9898c9f9fbfbaa
RoundKeys[03] = 90973450696ccffaf2f457330b0fac99
RoundKeys[04] = ee06da7b876a1581759e42b27e91ee2b
RoundKeys[05] = 7f2e2b88f8443e098dda7cbbf34b9290
RoundKeys[06] = ec614b851425758c99ff09376ab49ba7
RoundKeys[07] = 217517873550620bacaf6b3cc61bf09b
RoundKeys[08] = 0ef903333ba9613897060a04511dfa9f
RoundKeys[09] = b1d4d8e28a7db9da1d7bb3de4c664941
RoundKeys[10] = b4ef5bcb3e92e21123e951cf6f8f188e
```

On pourra vérifier également cette diversification sur la clé par défaut (K=2b7e151628aed2a6abf7158809cf4f3c) :

```
RoundKeys[00] = 2b7e151628aed2a6abf7158809cf4f3c
RoundKeys[01] = a0fafe1788542cb123a339392a6c7605
RoundKeys[02] = f2c295f27a96b9435935807a7359f67f
RoundKeys[03] = 3d80477d4716fe3e1e237e446d7a883b
RoundKeys[04] = ef44a541a8525b7fb671253bdb0bad00
RoundKeys[05] = d4d1c6f87c839d87caf2b8bc11f915bc
RoundKeys[06] = 6d88a37a110b3efddbf98641ca0093fd
RoundKeys[07] = 4e54f70e5f5fc9f384a64fb24ea6dc4f
RoundKeys[08] = ead27321b58dbad2312bf5607f8d292f
RoundKeys[09] = ac7766f319fadc2128d12941575c006e
RoundKeys[10] = d014f9a8c9ee2589e13f0cc8b6630ca6
```

A.2 Chiffrement AES-128

Voici le détail des différentes étapes qui composent le chiffrement par AES du texte T = `00112233445566778899aabbccddeeff`
avec la clé K = `2b7e151628aed2a6abf7158809cf4f3c`.

```
ENCRYPT - Mode AES-128, rounds r from 0 à 10
-------
Legend :
input  : cipher input
start  : state at start of round r
s_box  : state after SBox substitution (SubBytes)
s_row  : state after ShiftRows
mixcol : state after MixColumn
k_sch  : key schedule value for round r
output : cipher output

PLAINTEXT :     00112233445566778899aabbccddeeff
KEY :           2b7e151628aed2a6abf7158809cf4f3c
--------------

R[00].input      00112233445566778899aabbccddeeff
R[00].k_sch      2b7e151628aed2a6abf7158809cf4f3c
** Start rounds **
R[01].start      2b6f37256cfbb4d1236ebf33c512a1c3
R[01].s_box      f1a89a3f500f8d3e269f08c3a6c9322e
R[01].s_row      f10f082e509f323f26c99a3ea6a88dc3
R[01].mixcol     ced99c53171cea23a8248245faa25149
R[01].k_sch      a0fafe1788542cb123a339392a6c7605
R[02].start      6e2362449f48c6928b87bb7cd0ce274c
R[02].s_box      9f26aa1bdb52b44f3d17ea10708bcc29
R[02].s_row      9f52ea29db17cc1b3d8baa4f7026b410
R[02].mixcol     1037795043a1629b199a28f82eeb1522
R[02].k_sch      f2c295f27a96b9435935807a7359f67f
R[03].start      e2f5eca23937dbd840afa8825db2e35d
R[03].s_box      98e6ce3a129ab9610979c2134c37114c
R[03].s_row      989ac24c1279113a0937ce614ce6b913
R[03].mixcol     10a6497384e9072ae44f1a200358f6ad
R[03].k_sch      3d80477d4716fe3e1e237e446d7a883b
R[04].start      2d260e0ec3fff914fa6c64646e227e96
R[04].s_box      d8f7abab2e1699fa2d5043439f93f390
R[04].s_row      d81643902e50f3ab2d93abfa9ff79943
R[04].mixcol     42a1e31df42b659ca50ce6a0fd998452
R[04].k_sch      ef44a541a8525b7fb671253bdb0bad00
R[05].start      ade5465c5c793ee3137dc39b26922952
R[05].s_box      95d95a4a4ab6b2117dff2e14f74fa500
R[05].s_row      95b62e004affa54a7d4f5a11f7d9b214
R[05].mixcol     de907f3c61113a10601cb5b023876d41
R[05].k_sch      d4d1c6f87c839d87caf2b8bc11f915bc
R[06].start      0a41b9c41d92a797aaee0d0c327e78fd
R[06].s_box      6783561ca44f5c88ac28d7fe23f3bc54
R[06].s_row      674fd754a428bc1cacf3568823835cfe
R[06].mixcol     9ccf61998b37cb5b932370417a24015d
```

```
R[06].k_sch      6d88a37a110b3efddbf98641ca0093fd
R[07].start      f147c2e39a3cf5a648daf600b02492a0
R[07].s_box      a1a02511b8ebe62452574263e7364fe0
R[07].s_row      a1eb42e0b8574f1152362524e7a0e663
R[07].mixcol     dd4af58accd642e9ff7542adabee35b2
R[07].k_sch      4e54f70e5f5fc9f384a64fb24ea6dc4f
R[08].start      931e028493898b1a7bd30d1fe548e9fd
R[08].s_box      dc72775fdca73da22166d7c0d9521e54
R[08].s_row      dca7d754dc661e5f215277a2d9723dc0
R[08].mixcol     d2bf32a7486d67b961be6019c2ba8aa4
R[08].k_sch      ead27321b58dbad2312bf5607f8d292f
R[09].start      386d4186fde0dd6b50959579bd37a38b
R[09].s_box      073c834454e1c17f532a2ab67a9a0a3d
R[09].s_row      07e12a3d542a0a44539a837f7a3cc1b6
R[09].mixcol     219df5b8985aa654ef9d5512c7ec1e04
R[09].k_sch      ac7766f319fadc2128d12941575c006e
R[10].start      8dea934b81a07a75c74c7c5390b01e6a
** FinalRound **
R[10].s_box      5d87dcb30ce0da9dc62910ed60e77202
R[10].s_row      5de010020c2972b3c6e7dc9d6087daed
R[10].k_sch      d014f9a8c9ee2589e13f0cc8b6630ca6
R[10].output     8df4e9aac5c7573a27d8d055d6e4d64b
00112233445566778899aabbccddeeff --> 8df4e9aac5c7573a27d8d055d6e4d64b
```

A.3 Autres exemples de chiffrement AES-128

– Avec la clé par défaut :
  `00000000000000000000000000000000 --> 7df76b0c1ab899b33e42f047b91b546f`
  `3243f6a8885a308d313198a2e0370734 --> 3925841d02dc09fbdc118597196a0b32`
– avec la clé `K=00000000000000000000000000000000` :
  `3243f6a8885a308d313198a2e0370734 --> e527936d049f88872a4903305b975bd1`
– Avec la clé `K=000102030405060708090a0B0C0D0E0F`
  `00112233445566778899aabbccddeeff --> 69c4e0d86a7b0430d8cdb78070b4c55a`

A.4 Chiffrement AES-192

On utilise ici la clé
`K=000102030405060708090a0B0C0D0E0F0111213141516171`
avec le texte clair :
`T = 00112233445566778899aabbccddeeff`

```
ENCRYPT - Mode AES-192, rounds r from 0 à 12
-------
Legend :
input  : cipher input
start  : state at start of round r
s_box  : state after SBox substitution (SubBytes)
s_row  : state after ShiftRows
mixcol : state after MixColumn
k_sch  : key schedule value for round r
output : cipher output
```

```
PLAINTEXT :     00112233445566778899aabbccddeeff
KEY :           000102030405060708090a0b0c0d0e0f0111213141516171
--------------

R[00].input      00112233445566778899aabbccddeeff
R[00].k_sch      000102030405060708090a0b0c0d0e0f
** Start rounds **
R[01].start      00102030405060708090a0b0c0d0e0f0
R[01].s_box      63cab7040953d051cd60e0e7ba70e18c
R[01].s_row      6353e08c0960e104cd70b751bacad0e7
R[01].mixcol     5f72641557f5bc92f7be3b291db9f91a
R[01].k_sch      0111213141516171d0eea180d4eba787
R[02].start      5e63452416a4dde327509aa9c9525e9d
R[02].s_box      58fb6e364749c111cc53b8d3dd00585e
R[02].s_row      5849b85e47535836cc006e11ddfbc1d3
R[02].mixcol     8d4798a5153ffeaefc6f2303a5bbd1fb
R[02].k_sch      dce2ad8cd0efa383d1fe82b290afe3c3
R[03].start      51a53529c5d05d2d2d91a1b135143238
R[03].s_box      d10696a5a6704cd8d88132c896fa2307
R[03].s_row      d1703207a68123a5d8fa96d896064cc8
R[03].mixcol     1c60cc24497f9502f04e66b4b9864b60
R[03].k_sch      abff8fe07f142867a3f685eb73192668
R[04].start      b79f43c4366bbd6553b8e35fca9f6d08
R[04].s_box      a9db1a1c057f7a4ded6c11cf74db3c30
R[04].s_row      a97f1130056c3c1ceddb1a4d74db7acf
R[04].mixcol     e954a4ee9e853567e023d5772b9811b8
R[04].k_sch      a2e7a4da32484719fd5f5bc3824b73a4
R[05].start      4bb30034accd727e1d7c8eb4a9d3621c
R[05].s_box      b36d631891bd40f3a410198dd366aa9c
R[05].s_row      b3bd199c9110aa18a46663f3d36d408d
R[05].mixcol     24658349bb4ce622693e0a0fc744b242
R[05].k_sch      21bdf64f52a4d027f04374fdc20b33e4
R[06].start      05d87506e9e83605997d7ef2054f81a6
R[06].s_box      6b619d6f1e9b056beefff3896b840c24
R[06].s_row      6b9bf3241eff0c6fee849d6b6b610589
R[06].mixcol     b76c619d4580480fa62af6e6f92f80d0
R[06].k_sch      de9c32e65cd741427d6ab70d2fce672a
R[07].start      69f0537b1957094ddb4041ebd6e1e7fa
R[07].s_box      f98ced21d45b01e3b90983e9f6f8942d
R[07].s_row      f95b832dd4099421b9f8ede3f68c01e9
R[07].mixcol     aafcc8921d408db8749dbe18901f5845
R[07].k_sch      df8d13d71d8620338a2bf142d6fcb000
R[08].start      7571db4500c6ad8bfeb64f5a46e3e845
R[08].s_box      9da3b96e63b4953dbb4e84be5a119b6e
R[08].s_row      9db4846e634e9b6ebb11b93d5aa395be
R[08].mixcol     0c178850e127b2acda748404611d11bf
R[08].k_sch      ab96070d845860275bd573f0465353c3
R[09].start      a7818f5d657fd28b81a1f7f4274e427c
R[09].s_box      5c0c734c4dd2b53d0c3268bfcc2f2c10
R[09].s_row      5cd268104d322c4c0c2f733dcc0cb5bf
R[09].mixcol     ad4b6e7eac11f35127fa82329daf6b93
R[09].k_sch      47c6df18913a6f183aac6815bef40832
```

```
R[10].start      ea8db1663d2b9c491d56ea27235b63a1
R[10].s_box      875dc83327f1de3ba4b187cc2639fb32
R[10].s_row      87f1873227b1fb33a439c83b265ddecc
R[10].mixcol     a8de35804e7b2e45ebae5b70b929936a
R[10].k_sch      e5217bc2a372280147f2a312d6c8cc0a
R[11].start      4dff4e42ed090644ac5cf8626fe15f60
R[11].s_box      e3162f2c55016f1b914a41aaa8f8cfd0
R[11].s_row      e30141d0554acf2c91f82f1ba8166faa
R[11].mixcol     4ff20bc597a7ee221e101a49b49f85d5
R[11].k_sch      ec64a41f5290ac2db7b1d7ef14c3ffee
R[12].start      a396afdac537420fa9a1cda6a05c7a3b
** FinalRound **
R[12].s_box      0a907957a69a2c76d332bd24e04adae2
R[12].s_row      0a9abde2a632da57d34a7976e0902c24
R[12].k_sch      e9e48be83f2c47e2d348e3fd81d84fd0
R[12].output     e37e360a991e9db500029a8b614863f4
00112233445566778899aabbccddeeff --> e37e360a991e9db500029a8b614863f4
```

A.5 Chiffrement AES-256

On utilise ici la clé
K=000102030405060708090a0B0C0D0E0F011121314151617181 91A1B1C1D1E1F1
avec le texte clair : T = 00112233445566778899aabbccddeeff

```
ENCRYPT - Mode AES-256, rounds r from 0 à 14
-------
Legend :
input  : cipher input
start  : state at start of round r
s_box  : state after SBox substitution (SubBytes)
s_row  : state after ShiftRows
mixcol : state after MixColumn
k_sch  : key schedule value for round r
output : cipher output

PLAINTEXT :     00112233445566778899aabbccddeeff
KEY :           000102030405060708090a0b0c0d0e0f011121314151617181 91a1b1c1d1e1f1
--------------

R[00].input      00112233445566778899aabbccddeeff
R[00].k_sch      000102030405060708090a0b0c0d0e0f
** Start rounds **
R[01].start      00102030405060708090a0b0c0d0e0f0
R[01].s_box      63cab7040953d051cd60e0e7ba70e18c
R[01].s_row      6353e08c0960e104cd70b751bacad0e7
R[01].mixcol     5f72641557f5bc92f7be3b291db9f91a
R[01].k_sch      011121314151617181 91a1b1c1d1e1f1
R[02].start      5e63452416a4dde3762f9a98dc6818eb
R[02].s_box      58fb6e364749c1113815b8468645ade9
R[02].s_row      5849b8e94715ad3638456e1186fbc146
R[02].mixcol     3af05ad02ab7491dc011924186752e27
R[02].k_sch      3ff9a37b3bfca57c33f5af773ff8a178
R[03].start      0509f9ab114bec61f3e43d36b98d8f5f
```

```
R[03].s_box      6b01996282b3ceef0d692705565d73cf
R[03].s_row      6bb327cf826973620d5d99ef5601ce05
R[03].mixcol     f0b0dcacb5a7ab438be853166418df3f
R[03].k_sch      7450138d350172fcb490d34d754132bc
R[04].start      84e0cf2180a6d9bf3f78805b1159ed83
R[04].s_box      5fe18afdcd24350875bccd3982cb55ec
R[04].s_row      5f24cdeccdbc55fd75cb8a0882e13539
R[04].mixcol     f3b7d5cbf6acc7442e75a9ce2b3d423b
R[04].k_sch      bedac6e68526639ab6d3cced892b6d95
R[05].start      4d6d132d738aa4de98a66523a2162fae
R[05].s_box      e33c7dd88f7e491d46244d263a4715e4
R[05].s_row      e37e4de48f2415d846477d1d3a3c4926
R[05].mixcol     f62c30dea420f2102552dcca5fbffe77
R[05].k_sch      d3a12fa7e6a05d5b52308e162771bcaa
R[06].start      258d1f794280af4b776252dc78ce42dd
R[06].s_box      3f5dc0b62ccd79b3f5aa0086bc8b2cc1
R[06].s_row      3fcd00c12caa2cb6f58bc0b3bc5d7986
R[06].mixcol     f37faa1527a11f8504102b327b0b82ec
R[06].k_sch      19bf6a2a9c9909b02a4ac55da361a8c8
R[07].start      eac0c03fbb3816352e5aee6fd86a2a24
R[07].s_box      87baba75ea07479631be28a86102e536
R[07].s_row      87072836eabee5753102ba9661ba47a8
R[07].mixcol     02c78ad186cc1a944876fddcf86fb615
R[07].k_sch      d94eed4f3feeb0146dde3e024aaf82a8
R[08].start      db89679eb922aa8025a8c3deb2c034bd
R[08].s_box      b9a7850b5693accd3fc22e1d37ba187a
R[08].s_row      b9932e7a56c2180b3fba85cd37a7ac1d
R[08].mixcol     938cf899e2eab936e309d8ff2d90f468
R[08].k_sch      68aca8fcf435a14cde7f64117d1eccd9
R[09].start      fb20506516df187a3d76bcee508e38b1
R[09].s_box      0fb7534d479eadda27386528531907c8
R[09].s_row      0f9e65c84738074d271953da53b7ad28
R[09].mixcol     0a4f18618c73a66cec3aed8ce1e2ddbf
R[09].k_sch      263ca67a19d2166e740c286c3ea3aac4
R[10].start      2c73be1b95a1b0029836c5e0df41777b
R[10].s_box      718faeaf2a32e7774605a6e19e83f521
R[10].s_row      7132a6212a05f5af4683ae779e8fe7e1
R[10].mixcol     33c57745018b34cbcbc51b09ab48fc08
R[10].k_sch      7200b44e86351502584a71132554bdca
R[11].start      41c5c30b87be21c9938f6a1a8e1c41c2
R[11].s_box      83a62e2b17aefddddc7302a2199c8325
R[11].s_row      83ae02251773832bdc9c2edd19a6fda2
R[11].mixcol     d3e746781344049fef50606c9cf0a32f
R[11].k_sch      191cdc0e00ceca6074c2e20c4a6148c8
R[12].start      cafb9a76138aceff9b928260d691ebe7
R[12].s_box      740fb8387d7e8b16144f13d0f681e994
R[12].s_row      747e13947d4fe9381481b816f60f8bd0
R[12].mixcol     ed298bc2fafbb3511ec8c429bdbe9f3e
R[12].k_sch      bd525c983b67499a632d388946798543
R[13].start      507bd75ac19cfacb7de5fca0fbc71a7d
R[13].s_box      53210ebe78de2d1fffd9b0e00fc6a2ff
R[13].s_row      53deb0ff78d9a2beffc60e1f0f212de0
R[13].mixcol     90c0ec7e9c922794a56504ecb0da4fc6
```

```
R[13].k_sch      43aa4b144364817437a663787dc72bb0
R[14].start      d36aa76adff6a6e092c36794cd1d6476
** FinalRound **
R[14].s_box      66025c029e4224e14f2e8522bda44338
R[14].s_row      664285389e2e43024fa45ce1bd022422
R[14].k_sch      3ba3bb6700c4f2fd63e9ca7425904f37
R[14].output     5de13e5f9eeab1ff2c4d969598926b15
00112233445566778899aabbccddeeff --> 5de13e5f9eeab1ff2c4d969598926b15
```

B Tables de correspondances entre ecritures polynomiales et exponentielles

```
// ExpoToPoly[k] donne la representation polynomiale de w(x)^k
// 0 est representer par w(x)^255 bien que mathematiquement, w^255=1
const int ExpoToPoly[256] = {
    0x01, 0x03, 0x05, 0x0f, 0x11, 0x33, 0x55, 0xff, 0x1a, 0x2e, 0x72, 0x96, 0xa1,
    0xf8, 0x13, 0x35, 0x5f, 0xe1, 0x38, 0x48, 0xd8, 0x73, 0x95, 0xa4, 0xf7, 0x02,
    0x06, 0x0a, 0x1e, 0x22, 0x66, 0xaa, 0xe5, 0x34, 0x5c, 0xe4, 0x37, 0x59, 0xeb,
    0x26, 0x6a, 0xbe, 0xd9, 0x70, 0x90, 0xab, 0xe6, 0x31, 0x53, 0xf5, 0x04, 0x0c,
    0x14, 0x3c, 0x44, 0xcc, 0x4f, 0xd1, 0x68, 0xb8, 0xd3, 0x6e, 0xb2, 0xcd, 0x4c,
    0xd4, 0x67, 0xa9, 0xe0, 0x3b, 0x4d, 0xd7, 0x62, 0xa6, 0xf1, 0x08, 0x18, 0x28,
    0x78, 0x88, 0x83, 0x9e, 0xb9, 0xd0, 0x6b, 0xbd, 0xdc, 0x7f, 0x81, 0x98, 0xb3,
    0xce, 0x49, 0xdb, 0x76, 0x9a, 0xb5, 0xc4, 0x57, 0xf9, 0x10, 0x30, 0x50, 0xf0,
    0x0b, 0x1d, 0x27, 0x69, 0xbb, 0xd6, 0x61, 0xa3, 0xfe, 0x19, 0x2b, 0x7d, 0x87,
    0x92, 0xad, 0xec, 0x2f, 0x71, 0x93, 0xae, 0xe9, 0x20, 0x60, 0xa0, 0xfb, 0x16,
    0x3a, 0x4e, 0xd2, 0x6d, 0xb7, 0xc2, 0x5d, 0xe7, 0x32, 0x56, 0xfa, 0x15, 0x3f,
    0x41, 0xc3, 0x5e, 0xe2, 0x3d, 0x47, 0xc9, 0x40, 0xc0, 0x5b, 0xed, 0x2c, 0x74,
    0x9c, 0xbf, 0xda, 0x75, 0x9f, 0xba, 0xd5, 0x64, 0xac, 0xef, 0x2a, 0x7e, 0x82,
    0x9d, 0xbc, 0xdf, 0x7a, 0x8e, 0x89, 0x80, 0x9b, 0xb6, 0xc1, 0x58, 0xe8, 0x23,
    0x65, 0xaf, 0xea, 0x25, 0x6f, 0xb1, 0xc8, 0x43, 0xc5, 0x54, 0xfc, 0x1f, 0x21,
    0x63, 0xa5, 0xf4, 0x07, 0x09, 0x1b, 0x2d, 0x77, 0x99, 0xb0, 0xcb, 0x46, 0xca,
    0x45, 0xcf, 0x4a, 0xde, 0x79, 0x8b, 0x86, 0x91, 0xa8, 0xe3, 0x3e, 0x42, 0xc6,
    0x51, 0xf3, 0x0e, 0x12, 0x36, 0x5a, 0xee, 0x29, 0x7b, 0x8d, 0x8c, 0x8f, 0x8a,
    0x85, 0x94, 0xa7, 0xf2, 0x0d, 0x17, 0x39, 0x4b, 0xdd, 0x7c, 0x84, 0x97, 0xa2,
    0xfd, 0x1c, 0x24, 0x6c, 0xb4, 0xc7, 0x52, 0xf6, 0x01
};
// PolyToExpo[x] donne la puissance k de w telle que x=w^k,
// ou x est donne sous forme polynomiale
const int PolyToExpo[256] = {
    0xff, 0x00, 0x19, 0x01, 0x32, 0x02, 0x1a, 0xc6, 0x4b, 0xc7, 0x1b, 0x68, 0x33,
    0xee, 0xdf, 0x03, 0x64, 0x04, 0xe0, 0x0e, 0x34, 0x8d, 0x81, 0xef, 0x4c, 0x71,
    0x08, 0xc8, 0xf8, 0x69, 0x1c, 0xc1, 0x7d, 0xc2, 0x1d, 0xb5, 0xf9, 0xb9, 0x27,
    0x6a, 0x4d, 0xe4, 0xa6, 0x72, 0x9a, 0xc9, 0x09, 0x78, 0x65, 0x2f, 0x8a, 0x05,
    0x21, 0x0f, 0xe1, 0x24, 0x12, 0xf0, 0x82, 0x45, 0x35, 0x93, 0xda, 0x8e, 0x96,
    0x8f, 0xdb, 0xbd, 0x36, 0xd0, 0xce, 0x94, 0x13, 0x5c, 0xd2, 0xf1, 0x40, 0x46,
    0x83, 0x38, 0x66, 0xdd, 0xfd, 0x30, 0xbf, 0x06, 0x8b, 0x62, 0xb3, 0x25, 0xe2,
    0x98, 0x22, 0x88, 0x91, 0x10, 0x7e, 0x6e, 0x48, 0xc3, 0xa3, 0xb6, 0x1e, 0x42,
    0x3a, 0x6b, 0x28, 0x54, 0xfa, 0x85, 0x3d, 0xba, 0x2b, 0x79, 0x0a, 0x15, 0x9b,
    0x9f, 0x5e, 0xca, 0x4e, 0xd4, 0xac, 0xe5, 0xf3, 0x73, 0xa7, 0x57, 0xaf, 0x58,
    0xa8, 0x50, 0xf4, 0xea, 0xd6, 0x74, 0x4f, 0xae, 0xe9, 0xd5, 0xe7, 0xe6, 0xad,
    0xe8, 0x2c, 0xd7, 0x75, 0x7a, 0xeb, 0x16, 0x0b, 0xf5, 0x59, 0xcb, 0x5f, 0xb0,
    0x9c, 0xa9, 0x51, 0xa0, 0x7f, 0x0c, 0xf6, 0x6f, 0x17, 0xc4, 0x49, 0xec, 0xd8,
    0x43, 0x1f, 0x2d, 0xa4, 0x76, 0x7b, 0xb7, 0xcc, 0xbb, 0x3e, 0x5a, 0xfb, 0x60,
    0xb1, 0x86, 0x3b, 0x52, 0xa1, 0x6c, 0xaa, 0x55, 0x29, 0x9d, 0x97, 0xb2, 0x87,
    0x90, 0x61, 0xbe, 0xdc, 0xfc, 0xbc, 0x95, 0xcf, 0xcd, 0x37, 0x3f, 0x5b, 0xd1,
    0x53, 0x39, 0x84, 0x3c, 0x41, 0xa2, 0x6d, 0x47, 0x14, 0x2a, 0x9e, 0x5d, 0x56,
    0xf2, 0xd3, 0xab, 0x44, 0x11, 0x92, 0xd9, 0x23, 0x20, 0x2e, 0x89, 0xb4, 0x7c,
    0xb8, 0x26, 0x77, 0x99, 0xe3, 0xa5, 0x67, 0x4a, 0xed, 0xde, 0xc5, 0x31, 0xfe,
    0x18, 0x0d, 0x63, 0x8c, 0x80, 0xc0, 0xf7, 0x70, 0x07
};
```

C Détail de la boite-S et son inverse

```
/* La fameuse Boite-S */
const F256 SBox[256] = {
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE,
    0xD7, 0xAB, 0x76, 0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4,
    0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0, 0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7,
    0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15, 0x04, 0xC7, 0x23, 0xC3,
    0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75, 0x09,
    0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3,
    0x2F, 0x84, 0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE,
    0x39, 0x4A, 0x4C, 0x58, 0xCF, 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8, 0x51, 0xA3, 0x40, 0x8F, 0x92,
    0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2, 0xCD, 0x0C,
    0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19,
    0x73, 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
    0xDE, 0x5E, 0x0B, 0xDB, 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2,
    0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79, 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5,
    0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08, 0xBA, 0x78, 0x25,
    0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86,
    0xC1, 0x1D, 0x9E, 0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E,
    0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF, 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42,
    0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};
/* la boite-S inverse */
const F256 InvSBox[256] = {
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81,
    0xF3, 0xD7, 0xFB, 0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E,
    0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB, 0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23,
    0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E, 0x08, 0x2E, 0xA1, 0x66,
    0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25, 0x72,
    0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65,
    0xB6, 0x92, 0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46,
    0x57, 0xA7, 0x8D, 0x9D, 0x84, 0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A,
    0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06, 0xD0, 0x2C, 0x1E, 0x8F, 0xCA,
    0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B, 0x3A, 0x91,
    0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6,
    0x73, 0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8,
    0x1C, 0x75, 0xDF, 0x6E, 0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F,
    0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B, 0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2,
    0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4, 0x1F, 0xDD, 0xA8,
    0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93,
    0xC9, 0x9C, 0xEF, 0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB,
    0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61, 0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6,
    0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D
};
```

Références

[Can03] Anne Canteaut. "Programmation en Langage C". INRIA - projet CODES, 2003.

[Cas98] Bernard Cassagne. "Introduction au Langage C". Laboratoire CLIPS UJF/CNRS, 1997-1998.

[DR98] Joan Daemen and Vincent Rijmen. AES Proposal : Rijndael. Technical report, 1998. `http://citeseer.ist.psu.edu/daemen98aes.html`.

[DR01] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, 1st edition, 2001. `http://www.iaik.tu-graz.ac.at/research/krypto/AES/`.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley Publishing, Inc, 1st edition, 2003. `http://www.macfergus.com/pc/`.

[Kob98] Neal Koblitz. *Algebraic Aspects of Cryptography*, volume 3 of *Algorithms and Compuitation in Mathematics*. Springer-Verlag, 1998.

[KR88] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1988. 2nd edition.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. Computer Sciences Applied Mathematics Engineering. CRC Press, Inc., 1st edition, 1996. `http://www.cacr.math.uwaterloo.ca/hac/`.

[Pil04] Jean-François Pillou. Introduction au Langage C++. Technical report, Tutorial, Encyclopédie Informatique Libre, 2004. `http://www.commentcamarche.net/cpp/cppintro.php3`.

[PKP03] Peter Prinz and Ulla Kirch-Prinz. *C Pocket Reference*. O'Reilly & Associates, 2003.

[Sch97] Bruce Schneier. *"Cryptographie Appliquée"*. Vuibert, Wiley and International Thomson Publishing, NY, 2nd edition, 1997. .

[Ste90] David Stevenson. "IEEE Std 754-1985 IEEE Standard for Binary Floating-Point Arithmetic". Technical report, IEEE Standards Association, 1990.

[Sti02] Douglas R. Stinson. *Cryptography : Theory and Practice*. Chapman & Hall/CRC Press, 2nd edition, 2002. `http://www.cacr.math.uwaterloo.ca/~dstinson/CTAP2/CTAP2.html`.

[Var05] Sébastien Varrette. *Cours de programmation avancée : le langage C*. Université du Luxembourg, janvier 2005. `http://www-id.imag.fr/~svarrett/`.