Chapitre VII

Cryptographie à clef privée

1 Introduction

Cette partie aborde le chiffrement par bloc à clef à privée (à savoir, la clef de chiffrement est la même que la clef de déchiffrement).

Nous allons voir :

- les bases théoriques qui ont conduit à considérer ce type de chiffrement,
- le concept de réseaux de cryptosystème et de chiffrement itéré,
- deux applications pratiques : le DES (obsolète) et AES.
- les modes opératoires (manière d'appliquer le chiffrement par bloc à une suite de blocs).

2 Cryptosystème produit

Il s'agit de formaliser l'idée consistant à combiner plusieurs systèmes cryptographiques afin d'essayer construire un système cryptographique plus sûr (introduit par [Sha49]).

Pour simplifier, on suppose que les cryptosystèmes sont endomorphiques, à savoir qu'ils vérifient $C = \mathcal{P}$ (les espaces des textes clairs et chiffrés sont identiques).

Définition 82 (Cryptosystème produit)**.** Soient deux cryptosystèmes endomorphiques partageant le même espace $\mathcal{P}$:
$S_1 = (\mathcal{P}, \mathcal{P}, \mathcal{K}_1, \mathcal{E}_1, \mathcal{D}_1)$,
$S_2 = (\mathcal{P}, \mathcal{P}, \mathcal{K}_2, \mathcal{E}_2, \mathcal{D}_2)$.

Le cryptosystème produit de S_1 et S_2 est le système :

$S_1 \times S_2 = (\mathcal{P}, \mathcal{P}, \mathcal{K}_1 \times \mathcal{K}_2, \mathcal{E}, \mathcal{D})$

avec :

chiffrement : $y = e_{k_1,k_2}(x) = e_{k_2}(e_{k_1}(x))$

déchiffrement : $x = d_{k_1,k_2}(y) = d_{k_1}(d_{k_2}(y))$.

On espère ainsi que la combinaison des cryptosystèmes permette d'augmenter sa sureté par l'augmentation de l'espace des clefs.

Autrement dit,

- l'espace des clefs de $S_1 \times S_2$ est constitué d'une clef $k = (k_1, k_2)$ où $k_1 \in \mathcal{K}_1$ et $k_2 \in \mathcal{K}_2$.
- on chiffre le texte clair x avec e_{k_1} (le premier cryptosystème, *i.e.* $y' = e_{k_1}(x)$), puis on en rechiffre le résultat avec e_{k_2} (le second cryptosystème, *i.e.* $y = e_{k_2}(y')$).
  Le texte chiffré est obtenu avec $y = e_{k_2}(e_{k_1}(x))$.
- le déchiffrement a lieu dans le sens inverse : on déchiffre le texte chiffré y avec d_{k_2} (le second cryptosystème, *i.e.* $y' = d_{k_2}(y)$), puis on en redéchiffre le résultat avec d_{k_1} (le premier cryptosystème, *i.e.* $x = d_{k_1}(y)$).
  Le texte clair déchiffré est obtenu avec $x = d_{k_1}(d_{k_2}(y))$.

Plus formellement, cette construction du cryptosystème produit bien le comportement attendu :

$$\begin{aligned} d_{k_1,k_2}(e_{k_1,k_2}(x)) &= d_{k_1}(d_{k_2}(e_{k_2}(e_{k_1}(x)))) && \text{(par définition)} \\ &= d_{k_1}(e_{k_1}(x)) && (\text{car } d_{k_2}(e_{k_2}(x)) = x) \\ &= x \end{aligned}$$

Définition 83 (Propriétés des cryptosystèmes produits)**.**

- l'opération produit est toujours associative i.e. : $(S_1 \times S_2) \times S_3 = S_1 \times (S_2 \times S_3)$.
- si $S_1 \times S_2 = S_2 \times S_1$, alors le cryptosystème produit est commutatif.
- un système cryptographique itéré S^n est défini par $S \times \ldots \times S$ (n fois).
- un système cryptographique est dit idempotent si $S^2 = S$.

Remarque 1 :

Par idempotent, on entend que le cryptosystème itéré S^n est identique au cryptosystème d'origine S, à la façon près de fournir la clef.

Par exemple, si S est un chiffre par décalage, alors il est clair que $e_{k_1,k_2}(x) = e_{k_1+k_2}(x)$. On a bien $S^2 = S$. L'espace des clefs n'est pas plus grand (toujours

dans $\mathbb{Z}_{26}$). Le cryptosystème produit S^2 exige de fournir deux clefs (k_1, k_2) alors que le crypotosystème équivalent S n'en a besoin que d'une seule $k = k_1 + k_2$.

Remarque 2 :

Si S_1 et S_2 sont idempotents et commutent, alors $S_1 \times S_2$ est également idempotent.

En effet :

$$\begin{aligned}(S_1 \times S_2) \times (S_1 \times S_2) &= S_1 \times S_2 \times S_1 \times S_2 && \text{(par associativité)}\\ &= S_1 \times S_1 \times S_2 \times S_2 && \text{(par commutativité)}\\ &= (S_1 \times S_1) \times (S_2 \times S_2)\\ &= S_1 \times S_2 && \text{(par idempotence)}\end{aligned}$$

Donc, si S_1 et S_2 sont idempotents, il est nécessaire qui S_1 et S_2 ne commutent pas.

Conséquences

- si un cryptosystème n'est pas idempotent, sa sécurité est augmentée en l'itérant plusieurs fois.
- le fabrication d'un système non idempotent consiste, dans la plupart des cas, à prendre le produit de deux systèmes simples (sous condition qu'ils ne commutent pas).

L'utilisation de ces principes conduit à :

- construire un cryptosystème non idempotent par produit de deux cryptosystèmes non commutatifs.
  Shannon suggère d'utiliser comme brique de base le produit d'un chiffrement par substitution et d'un chiffrement par permutation.
- itérer le cryptosystème produit.

En terme de clef

Il convient également de choisir la clef (k_1, k_2) d'un cryptosystème produit de manière indépendante, à savoir tel que :

$\Pr[(k_1, k_2)] = \Pr[k_1].\Pr[k_2]$.

afin de bénéficier, si elle a lieu, de l'augmentation de l'espace des clefs.

Ce sont ces considérations qui ont amenés aux algorithmes de chiffrement par

bloc que nous abordons maintenant.

Exercice 46: Cryptosystème produit

Pour chaque cryptosystème produit défini ci-dessous, répondre aux questions suivantes :

(a) Est-il commutatif ?
(b) Est-il idempotent ?

1. pour le cryptosystème produit de deux substitution de symboles.
2. pour le cryptosystème produit de deux chiffre de Vigenère.
   On fera la démonstration en utilisant deux chiffres avec des clefs différentes, d'abord de même longueurs, puis de longueurs différentes.
3. pour le cryptosystème produit de deux chiffres par permutation de symboles.
   On fera la démonstration en utilisant deux chiffres différentes sur un alphabet de taille 4.
4. pour le cryptosystème produit d'une substitution de symboles et d'une permutation de symboles.
5. pour le cryptosystème produit d'une substitution binaire (par exemple par paquets de 3 bits) et d'une permutation binaire (par paquets de 2 bits).

3 Cryptosystème itéré

Afin de définir un système de chiffrement itéré, nous avons besoin :

- d'une fonction d'étage qui permet d'obtenir le chiffrement à une itération donnée.
- d'un algorithme de diversification de la clef qui permet de construire une clef différente pour chaque itération.

Définissons-les maintenant plus formellement.

Définition 84 (fonction d'étage)**.** Une fonction d'étage est une fonction injective g qui prend en paramètre :

- l'état actuel w_n du chiffrement,
- une sous-clef k_n

et qui calcule un nouveau chiffrement $w_{n+1} = g(w_n, k_n)$.

Définition 85 (algorithme de diversification de la clef)**.** Algorithme D_n fixé et public qui permet à partir d'une clef binaire aléatoire K de construire un vecteur de n sous-clefs $(k_1, \ldots, k_n)$ *i.e.* $(k_1, \ldots, k_n) = D_n(K)$.

Définition 86 (Système de chiffrement itéré)**.** Un cryptosystème de chiffrement itéré à N_e étage est un cryptosystème constitué de :

- une fonction d'étage g.
- un algorithme de diversification de la clef D_{N_e}.

dont l'opération de chiffrement d'un mot clair x est obtenu comme le terme d'ordre N_e de la suite w_n définie par :

$$\begin{cases} w_0 & = x \\ w_n & = g(w_{n-1}, k_n) \end{cases}$$

où $(k_1, \ldots, k_{N_e}) = D_{N_e}(K)$.

Exemple : pour un système à 4 étages, $y = e_K(x)$ est obtenu comme :

On calcule vecteur de sous-clef à partir de clef $(k_1, k_2, k_3, k_4) = D_4(K)$.

Le chiffrement de x est obtenu par :

$w_0 = x$

$w_1 = g(w_0, k_1)$, $w_2 = g(w_1, k_2)$, $w_3 = g(w_2, k_3)$, $w_4 = g(w_3, k_4)$

$y = w_4$

d'ou $y = g(g(g(g(x, k_1), k_2), k_3), k_4)$.

Déchiffrement

La fonction g étant injective (donc inversible, g^{-1} existe), le déchiffrement consiste juste à appliquer g^{-1} dans l'ordre inverse des clefs.

Plus formellement, l'opération de déchiffrement est obtenu que le terme d'ordre N_e de la suite v_n définie par :

$$\begin{cases} v_0 & = y \\ v_n & = g^{-1}(v_{n-1}, k_{N_e - n}) \end{cases}$$

où $(k_1, \ldots, k_{N_e}) = D_{N_e}(K)$.

Exemple pour un système à 4 étages, $x = d_K(y)$ est obtenu comme :

On calcule vecteur de sous-clef à partir de clef $(k_1, k_2, k_3, k_4) = D_4(K)$.

Le déchiffrement de y est obtenu par :

$v_0 = y$

$v_1 = g^{-1}(v_0, k_4)$, $v_2 = g^{-1}(v_1, k_3)$, $v_3 = g^{-1}(v_2, k_2)$, $v_4 = g^{-1}(v_3, k_1)$

$x = v_4$

d'ou $x = g^{-1}(g^{-1}(g^{-1}(g^{-1}(x, k_1), k_2), k_3), k_4)$.

On a évidemment, $w_0 = v_4$, $w_1 = v_3$, $w_2 = v_2$, $w_3 = v_1$, $w_4 = v_0$.

Exercice 47: Système de chiffrement itéré

On considère le cryptosystème itéré à 4 étages suivant :
- la fonction d'étage est un chiffre de Vigenère,
- la fonction de diversification de la clef K de longueur $2n$ suivante : $k_1 = K_1 \dots K_n$, $k_2 = K_{n+1} \dots K_{2n}$, $k_3 = K_1 K_3 \dots K_{2n-1}$, $k_4 = K_2 K_4 \dots K_{2n}$.

1. On prend la clef K="PASTEQUE". Calculer les 4 clefs en utilisant l'algorithme de diversification.
2. Chiffrer le message "RETRAITE" avec ce cryptosystème itéré.
3. Déchiffrer le message "XMGIZPABM" produit par ce cryptosystème itéré.
4. Pourquoi l'itération est-elle inutile dans ce cas ?
5. Donner la clef équivalente et la vérifier.

4 Réseaux de substitution-permutation

Un réseau de substitution-permutation (ou SPN) est un type particulier de cryptosystème itéré destiné à coder des blocs de taille fixe.

Soit $\mathbb{B} = \{0, 1\}$ l'alphabet binaire, et $\mathbb{B}^n$ l'ensemble des mots binaires de taille n.

Il est constitué à partir des deux briques suivantes destinées à coder un bloc binaire de taille $\ell \times m$ constitué de m mots de taille ℓ :
- une substitution $\pi_S : \mathbb{B}^\ell \to \mathbb{B}^\ell$ qui effectue la substitution d'un mot M de taille ℓ par un mot $\pi_S(M)$ de même taille.
- une permutation $\pi_P : \{1, \dots, \ell.m\} \to \{1, \dots, \ell.m\}$ qui permute tous les bits d'un bloc B de taille $\ell.m$ bits.

Pour le codage d'un bloc B de taille $p = \ell \times m$ bits,
- le bloc B s'écrit comme la concaténation de m mots de ℓ bits. On note $B^{(j)}$ le $j^{\text{ème}}$ mot de ℓ bits (*i.e.* $B = B^{(1)} \dots B^{(m)}$). On a $p = \ell.m$.
- le bloc B s'écrit aussi comme une suite brute de p bits. On note $B^{[k]}$ le $k^{\text{ème}}$ bit (*i.e.* $B = B^{[1]} \dots B^{[\ell.m]}$).

Définition 87 (Fonction d'étage d'un SPN)**.** La fonction d'étage $u_n = g(u_{n-1}, k_n)$ sur un bloc de m mots de ℓ bits d'une SPN est définie comme la composition :
- d'une opération de substitution $v_n = \pi_S(u_{n-1}^{(1)}) \dots \pi_S(u_{n-1}^{(m)})$ (ou s-box).
- d'une opération de permutation $w_n = v_n^{[\pi_P(1)]} \dots v_n^{[\pi_P(\ell.m)]}$ (ou p-box).
- d'une opération d'incorporation de la clef : $u_n = w_n \oplus k_n$.
  où $\oplus$ est un "xor" binaire.

Un SPN à N_e étages sur des blocs de taille p utilise $N_e + 1$ sous-clefs k_i de longueurs p, soit obtenu directement à partir de la clef (*i.e.* $\mathcal{K} \subseteq \mathbb{B}^{(N_e+1).p}$), soit obtenu à partir d'un algorithme de diversification de clef.

Définition 88 (Chiffrement avec un SPN)**.** Un SPN à N_e étages utilise un ensemble $N_e + 1$ sous-clefs $(k_0, \ldots, k_{N_e})$ de taille p afin de chiffrer un bloc B de taille $p = m.\ell$ de la manière suivante :

- incorporer la clef : $u_0 = B \oplus k_0$
- effectuer N_{e-1} fois la fonction d'étage à partir de u_0 (on obtient u_{N_e-1}).
- puis terminer par $v_{N_e} = \pi_S(u^{(1)}_{N_e-1}) \ldots \pi_S(u^{(m)}_{N_e-1})$.
  $u_{N_e} = v_{N_e} \oplus k_{N_e}$

Algorithme de codage par bloc d'un SPN

Entrées :
$(k_0, \ldots, k_{N_e})$ = ensemble des sous-clefs.
x = bloc à chiffrer de $m \times \ell$ bits.
Sortie :
y = bloc chiffré de $m \times \ell$ bits.
// blanchissage
$u = x \oplus k_0$
// utilisation de la fonction d'étage
for $i = 1$ *à* $N_e - 1$ **do**
  // substitution
  $v = \pi_S(u^{(1)}) \ldots \pi_S(u^{(m)})$
  // permutation
  $w^{[1]} \ldots w^{[\ell.m]} = v^{[\pi_P(1)]} \ldots v^{[\pi_P(\ell.m)]}$
  // xor
  $u = w \oplus k_i$
// dernier étage (pas de permutation)
$v = \pi_S(u^{(1)}) \ldots \pi_S(u^{(m)})$
$y = v \oplus k_{N_e}$

Exemple

On utilise un alphabet hexadécimal (1 mot = 4 bits). On veut coder le bloc 16 bits $x = 4fc0_{16}$ avec la clef $K = 5fd8e206_{16}$ avec un SPN à 4 étages.

On définit :

- Diversification de clef (pour l'exercice seulement) $k_i = c_i \ldots c_{i+3}$ où c_i est obtenu en découpant $K = c_0 \ldots c_7$ en mots de 4 bits.
  $K = (0101\ 1111\ 1101\ 1000\ 1110\ 0010\ 0000\ 0110)_2$ permet d'obtenir les sous-clefs :
  $k_0 = (0101\ 1111\ 1101\ 1000)_2$

$k_1 = (1111\ 1101\ 1000\ 1110)_2$
$k_2 = (1101\ 1000\ 1110\ 0010)_2$
$k_3 = (1000\ 1110\ 0010\ 0000)_2$
$k_4 = (1110\ 0010\ 0000\ 0110)_2$

- Table de substitution : $\pi_S(x)$ (où x est un mot 4 bits), et de permutation (où x est le numéro du bit).

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi_S(x)$	6	4	10	12	9	1	14	15	7	13	11	8	3	2	5	0
$\pi_P(x)$	10	0	4	13	8	7	1	3	2	6	12	14	5	9	15	11

B	0100	1111	1100	0000
k_0	0101	1111	1101	1000
u_0	0001	0000	0001	1000
v_1	0100	0110	0100	0111
w_1	1001	0100	0101	0011
k_1	1111	1101	1000	1110
u_1	0110	1001	1101	1101
v_2	1110	1101	0010	0010
w_2	1101	1010	1000	0101
k_2	1101	1000	1110	0010
u_2	0000	0010	0110	0111
v_3	0110	1010	1110	1111
w_3	1110	0110	1101	1011
k_3	1000	1110	0010	0000
u_3	0110	1000	1111	1011
v_4	1110	0111	0000	1000
k_4	1110	0010	0000	0110
u_4	0000	0101	0000	1110

x	0	1	2	3	4	5	6	7
$\pi_S(x)$	6	4	10	12	9	1	14	15
$\pi_P(x)$	10	0	4	13	8	7	1	3

x	8	9	10	11	12	13	14	15
$\pi_S(x)$	7	13	11	8	3	2	5	0
$\pi_P(x)$	2	6	12	14	5	9	15	11

$u = B \oplus k_0$
for $i = 1$ *à* $N_e - 1$ **do**
  $v = \pi_S(u^{(1)}) \dots \pi_S(u^{(m)})$
  $w^{[1]} \dots w^{[\ell.m]} = v^{[\pi_P(1)]} \dots v^{[\pi_P(\ell.m)]}$
  $u = w \oplus k_i$
$v = \pi_S(u^{(1)}) \dots \pi_S(u^{(m)})$
$u = v \oplus k_{N_e}$

Décodage

Les opérations utilisées dans la fonction d'étage du SPN étant inversible, le décodage est lui-aussi inversible.

Il suffit d'appliquer les opérations dans l'ordre inverse où elles ont été effectuées pour le chiffrement.

On remarquera que cet méthode n'utilise qu'un petit nombre d'opérations simples, et que le coût du chiffrement est identique au coût du déchiffrement.

Variantes

Différentes variantes des SPNs ont été utilisées :

- prendre plusieurs fonctions de substitution différentes (par exemple, une par étage),
- ajouter une transformation linéaire inversible à chaque étage,

Le DES (Data Encryption Standard) est un exemple de variation de SPN (voir plus loin).

Exercice 48: Réseau de substitution-permutation

On considère un SPN à 2 étages servant à chiffrer des messages utilisant l'alphabet $\Sigma = \{a, e, n, t\}$ codé par un codage fixe dans $\mathbb{B}^2$. On utilise les deux briques de 8 bits suivantes :

1. une substitution $\pi_S : (a, e, n, t) \to (e, t, a, n)$
2. une permutation π_P qui consiste à effectuer un décalage circulaire de 1 bits vers la droite.
3. les clefs sont $(k_0, k_1, k_2) = (0xaa, 0x55, 0xcc)$

1. Chiffrer le bloc "TENTANTE".
2. Déchiffrer le bloc "AETAEAEE".

Deux types de méthode de cryptanalyse peuvent être appliqués :

- **la cryptanalyse différentielle** :
  on étudie les différences de chiffrement entre des textes similaires pour sélectionner un sous-ensemble de clefs probables.
- **la cryptanalyse linéaire** :
  on utilise des relations linéaires sur certains bits du message chiffré et du message clair pour interpoler des bits de la clef.

Sécurité

Afin de garantir un sécurité acceptable, il est nécessaire de :

- avoir une longueur de clef minimale d'au moins 128 bits.
- choisir des longueurs de bloc plus importantes (au moins 128 bits).
- choisir des largeur de mots pour la substitution d'au moins 8 bits.
- prendre au moins 10 étages (10 itérations).

5 Schéma de Feistel

Soit A, $B \in \mathbb{B}^n$. On note AB ou $[A, B]$ le concaténation de A et B (*i.e.* $AB \in \mathbb{B}^{2n}$).

Remarque préliminaire

Soit F_n l'ensemble des applications de $\mathbb{B}^n$ dans $\mathbb{B}^n$.

Pour tout élément x de l'ensemble de départ dans $\mathbb{B}^n$, il y a 2^n choix possibles d'image y dans $\mathbb{B}^n$.

En conséquence, il y a $(2^n)^{(2^n)}$ applications différentes possibles ($= |F_n|$).

Considérons maintenant une fonction quelconque $f \in F_n$ (donc $f : \mathbb{B}^n \to \mathbb{B}^n$).

Soit l'application $\Psi(f) : \mathbb{B}^{2n} \to \mathbb{B}^{2n}$ définie par :

$$\Psi(f)([L, R]) = [R, L \oplus f(R)]$$

où $L, R \in \mathbb{B}^n$.

Alors $\Phi(f)([E, F]) = [F \oplus f(E), E]$ est l'application réciproque de $\psi(f)$.

$$\begin{aligned}
\Phi(f)(\Psi(f)([L, R])) &= \Phi(f)([R, L \oplus f(R)]) \\
&= [L \oplus f(R) \oplus f(R), R] && \text{or } f(R) \oplus f(R) = 0 \\
&= [L, R] && \text{car } L \oplus 0 = L
\end{aligned}$$

Donc, pour tout application f, $\Psi(f)$ est une bijection de $\mathbb{B}^{2n}$ dans $\mathbb{B}^{2n}$, et son application réciproque est $\Phi(f)$.

Première propriété remarquable

Même si la fonction f n'est pas inversible, l'application $\Psi(f)$ peut être inversée et sa réciproque est $\Phi(f)$.

On pourrait même donc prendre pour f une fonction qui fait perdre des bits à R (par exemple en en dupliquant ou en inversant d'autres).

Les choix couramment effectués pour f utilisent des combinaisons :

- de mélanges avec la sous-clef,
- de permutations,
- de substitutions,
- d'augmentations (duplication de bits dans R)

Comme dans le cas des SPNs, nous allons utiliser une version itérée de $\Psi(f)$.

Définition 89 (Schéma de Feistel)**.** Soit $(f_1, \ldots, f_k)$, k fonctions de $\mathbb{B}^n$ dans $\mathbb{B}^n$. Soit l'application : $\Psi(f) : \mathbb{B}^{2n} \to \mathbb{B}^{2n}$ l'application paramétrée par $f \in F_n$ définie par $\Psi(f)([L, R]) = [R, L \oplus f(R)]$.

L'application $\Psi^k : \mathbb{B}^{2n} \to \mathbb{B}^{2n}$ définie par :

$$\Psi^k(f_1, \ldots, f_k) = \Psi(f_k) \circ \Psi(f_{k-1}) \circ \ldots \circ \Psi(f_2) \circ \Psi(f_1)$$

est un schéma de Feistel à k rondes.

Exemple

Un schéma de Feistel à 3 rondes est défini par l'application $\Psi^3(f_1, f_2, f_3)$:

$$\begin{aligned}
\Psi(f_1)([L,R]) &= [R\ ,\ L \oplus f_1(R)] \\
\Psi^2(f_1,f_2)([L,R]) &= \Psi(f_2) \circ \Psi(f_1)([L,R]) \\
&= \Psi(f_2)([R\ ,\ L \oplus f_1(R)]) \\
&= [L \oplus f_1(R)\ ,\ R \oplus f_2(L \oplus f_1(R))] \\
\Psi^3(f_1,f_2,f_3)([L,R]) &= \Psi(f_3) \circ \Psi(f_2) \circ \Psi(f_1)([L,R]) \\
&= \Psi(f_3)([L \oplus f_1(R)\ ,\ R \oplus f_2(L \oplus f_1(R))]) \\
&= [R \oplus f_2(L \oplus f_1(R))\ ,\ L \oplus f_1(R) \oplus f_3(R \oplus f_2(L \oplus f_1(R)))]
\end{aligned}$$

Comme Ψ a une réciproque, un schéma de Feistel à k rondes Ψ^k possède lui-aussi une réciproque.

Définition 90 (Réciproque d'un schéma de Feistel)**.** Soit l'application : $\Phi(f)$: $\mathbb{B}^{2n} \to \mathbb{B}^{2n}$ l'application paramétrée par $f \in F_n$ définie par $\Phi(f)([E,F]) = [F \oplus f(E), E]$.

Alors le schéma réciproque du schéma de Feistel à k rondes est l'application $\Phi^k : \mathbb{B}^{2n} \to \mathbb{B}^{2n}$ définie par :

$$\Phi^k(f_1, \ldots, f_k) = \Phi(f_1) \circ \Phi(f_2) \circ \ldots \circ \Phi(f_{k-1}) \circ \Phi(f_k)$$

Deuxième propriété remarquable

L'une des propriétés essentielle de ce schéma est que, dès un schéma à 3 étages, si f_1, f_2 et f_3 sont des fonctions aléatoires, alors il est difficile de distinguer $\Psi^3(f_1, f_2, f_3)$ d'une fonction aléatoire (voir le théorème de Luby-Rackoff, [BM12], p181).

Autrement dit, la cryptanalyse d'un schéma de Feistel est un problème difficile.

Exercice 49: Schéma de Feistel

On considère un schéma de Feistel sur un alphabet $\Sigma = (a, e, n, t)$ (caractères codés sur 2 bits) utilisant les trois fonctions d'étage sur 4 bits suivantes :

- $f_1(t)$ = `AND(`t`,0x5)` (on met à 0 les bits impairs).
- $f_2(t)$ = `XOR(`t`,0x5)`
- $f_3(t)$ = `OR(`t`,0x5)`

1. Chiffrer le bloc "TENTANTE" en utiliser le schéma de Feistel à 3 ronde $\Psi^3(f_1, f_2, f_3)$ sur des blocs de 8 bits.
2. Décoder le message "TNNETAET" codé avec le schéma de Feistel à 3 ronde $\Psi^3(f_1, f_2, f_3)$ sur des blocs de 8 bits.

6 DES

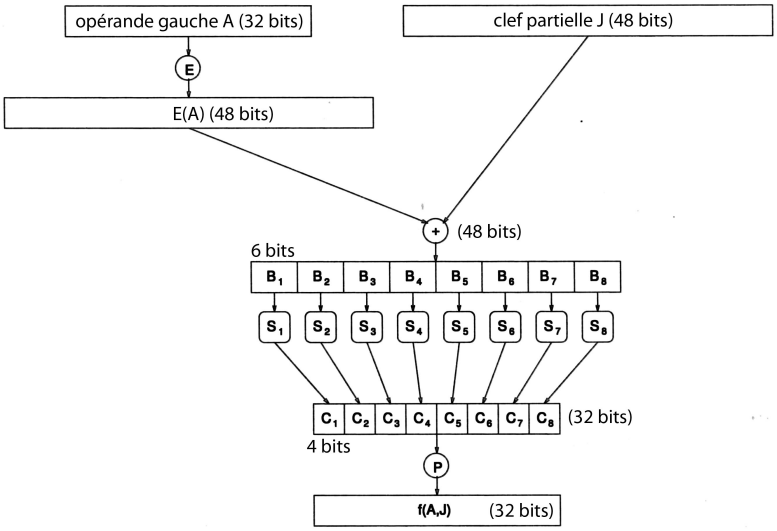
Le DES (Data Encryption Standard) est un standard de chiffrement pour les données non sensibles, développé par IBM [Cop94], et adopté par le bureau des standards américains en 1975 [Pub99].

Le DES utilise le schéma suivant pour chiffrer un bloc de 64 bits,

- application d'une permutation IP initiale $L_0R_0 = \mathrm{IP}(x)$
- application d'un schéma de Feistel à 16 étages :
  $\begin{cases} L_i & = R_{i-1} \\ R_i & = L_{i-1} \oplus f(R_{i-1}, k_i) \end{cases}$
  où L_i et R_i sont des blocs de 32 bits, et k_i est la sous-clef de l'étage.
- application de la permutation IP^{-1} finale $y = \mathrm{IP}^{-1}(R_{16}L_{16})$.

Le schéma de Feistel $f(A, J)$ est le suivant :

- augmentation de A (32 bits) en une chaîne de 48 bits avec une fonction d'expansion $E(A)$.
- incorporation de la clef $B = E(A) \oplus J$ où J est la clef partielle ($|B|$=48 bits).
- découpage du $B = B_1B_2 \dots B_8$ en 8 sous-chaines de 6 bits.
- substitution sur chaque B_i par une substitution S_i spécifique (s-box). $C_i = S_i(B_i)$ produit de 4 bits. On note $C = C_1 \dots C_8$ ($|C| = 32$ bits).
- permutation P (p-box) de la chaîne de 32 bits : $P(C)$.



Permutations initiale IP : $\mathbb{B}_{64} \to \mathbb{B}_{64}$

Cette fonction est appelée qu'une seule fois, au début du chiffrage du bloc.
$\pi_{\text{IP}} = ($ 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7)
est la table de permutation des indices (chacun n'apparaît qu'une seule fois).

Pour $x = (x_1, \ldots, x_{64})$, $\text{IP}(x) = (x_{\pi_{\text{IP}}(1)}, \ldots, x_{\pi_{\text{IP}}(64)})$.

La table π_P est constante.

Fonction d'augmentation $E : \mathbb{B}_{32} \to \mathbb{B}_{48}$

La fonction d'augmentation permet de passer le contenu du bloc sur 48 bits.

$e = ($ 32, 1, 2	3, 4, 5	4, 5, 6	7, 8, 9,
8, 9, 10	11, 12, 13	12, 13, 14	15, 16, 17,
16, 17, 18	19, 20, 21	20, 21, 22	23, 24, 25,
24, 25, 26	27, 28, 29	28, 29, 30	31, 32, 1)

est la table de la fonction d'augmentation : $\text{E}(a_1 \ldots a_{32}) = (a_{e(1)}, \ldots, a_{e(48)})$.
La table e est constante.
Noter que certains bits sont dupliqués.

Fonction de substitution (s-box) $S_i : \mathbb{B}_6 \to \mathbb{B}_4$

Une fonction de substitution est définie par un tableau de 4×16 entiers compris entre 0 et 15.
Pour une chaîne $B_j = b_1 \ldots b_6$ de 6 bits, on forme deux indices :

- $c = b_2b_3b_4b_5$ est l'indice de la colonne,
- $l = b_1b_6$ est l'indice de la ligne.

Il y a 8 tables qui définissent autant de fonction S_i différentes. Toutes ces tables sont constantes.

Exemple pour la fonction S_1

Le tableau associé à la substitution S_1 dans DES est le suivant :

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

si $B_j = 001101$, alors $c = 0110 = 6$ et $l = 01 = 1$.

Donc $S_1(B_j) = 13 = 1101$.

Fonction de permutation (p-box) $P : \mathbb{B}_{32} \to \mathbb{B}_{32}$

On utilise la table de permutation des indices constante suivante :

$\pi_P = (16, 7, 20, 21, 29, 12, 28, 17, \ 1, 15, 23, 26, \ 5, 18, 31, 10,$
$2, 8, 24, 14, 32, 27, \ 3, \ 9, 19, 13, 30, \ 6, 22, 11, \ 4, 25)$

Pour $C = (c_1, \ldots, c_{32})$, $P(C) = (c_{\pi_{P(1)}} \ldots c_{\pi_{P(32)}})$.

Cette fonction marque la fin de la fonction d'étage. Le bloc gauche et droit sont ensuite réassemblés en un mot 64 bits afin de poursuivre, soit vers l'étage suivant, soit

Permutations finale $\mathbf{IP}^{-1}$: $\mathbb{B}_{64} \to \mathbb{B}_{64}$

Cette fonction est appelée qu'une seule fois, après les 16 itérations, à la fin du chiffrage du bloc.

Elle est l'inverse de **IP**.

$\pi_{IP^{-1}} = (40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,$
$38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,$
$36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,$
$34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, \ 9, 49, 17, 57, 25)$

La diversification de la clef $(k_1, k_2, \ldots, k_{16})$ consiste en des sous-clefs de 48 bits dérivées de la clef de 56 bits où chaque k_i est une sélection permutée de K.

La clef est de 64 bits composée comme 8 paquets de 8 bits. Le dernier bit de chaque paquet est un bit de parité.

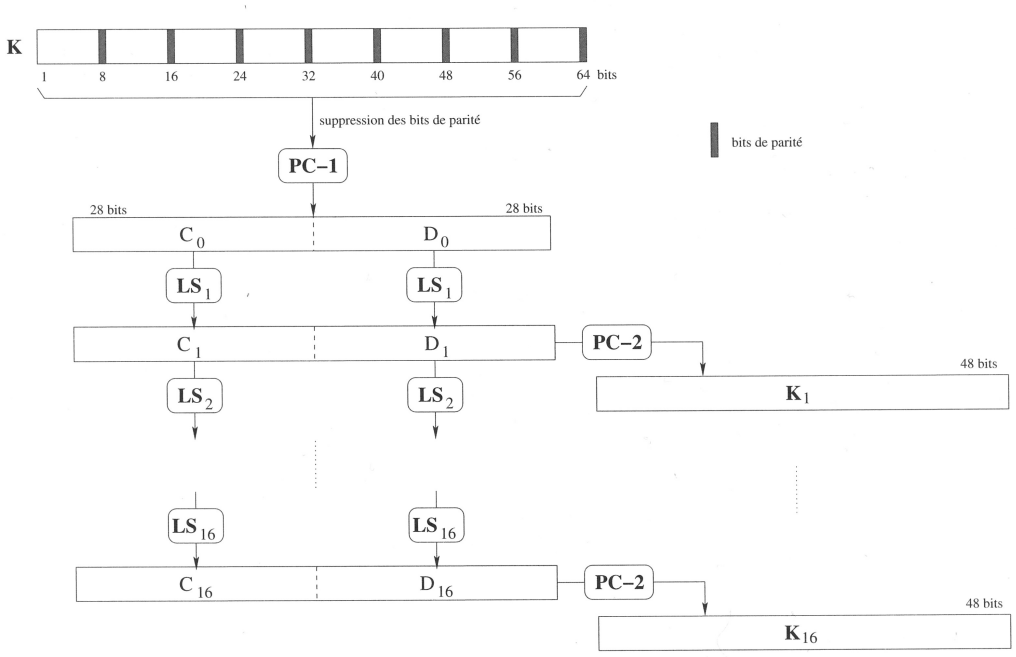
La diversification de la clef est alors obtenue de la manière suivante :

- Application d'un permutation PC-1 : $\mathbb{B}_{64} \to \mathbb{B}_{56}$ qui supprime les bits de parités, et effectue une permutation.
- Puis, le résultat est scindé en deux mots C_0 et D_0 de 28 bits.

- La clef k_i est alors obtenue comme :

$$\begin{cases} C_i & = \mathrm{LS}_i(C_{i-1}) \\ D_i & = \mathrm{LS}_i(D_{i-1}) \\ k_i & = \text{PC-2}(C_i D_i) \end{cases}$$

où
  - ◇ $\mathrm{LS}_i : \mathbb{B}_{26} \to \mathbb{B}_{28}$ est un décalage à gauche circulaire dont la valeur du décalage change en fonction de i.
  - ◇ PC-2 : $\mathbb{B}_{56} \to \mathbb{B}_{48}$ est une permutation qui supprime 8 bits.



Tables de permutation PC-1 et PC-2

PC-1							
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC-2						
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Noter que dans les deux tables, 8 bits ont disparus (i.e. sont absent de la permutation).

Table de décalages pour LS_i

i	1 2 3 4	5 6 7 8	9 10 11 12	13 14 15 16
n	1 1 2 2	2 2 2 2	1 2 2 2	2 2 2 1

Exemple

Pour LS_3, $i = 3$ la table nous donne $n = 2$.

LS_3 est donc un décalage circulaire à gauche de 2 bits.

Donc $LS_3(11001010) = 00101011$.

Remarques sur DES

- après 16 tours, le résultat de DES est statistiquement plat (*i.e.* la fréquence des différents caractères est indétectable).
- une légère modification de la clef ou du texte à chiffrer provoque des changements importants dans le texte chiffré (rend difficile l'analyse différentielle).
- repose sur des opérations facilement implémentables (on arrive à des débits de plusieurs centaines de Mo/s avec des puces spécifiques).
- la totalité des spécifications (tables, $f, \ldots$) étant connue, la sécurité repose uniquement sur le choix de la clef.

Néanmoins, la taille de la clef est trop faible, et fait que le chiffre peut être cassé en un temps relativement raisonnable.

Le standard a probablement été choisi de façon à ce que la NSA puisse travailler, mais que cela soit difficilement à la portée d'un quidam. L'évolution de la puissance des machines fait que DES n'est plus considéré comme sûr depuis la fin des années 90.

Méthodes d'attaque possible

- **Recherche exhaustive** : les clefs sont testées l'une après l'autre.
- **Précalcul exhaustif** : on stocke le résultat du chiffrement DES sur un texte T choisi pour toutes les clefs K possibles. Si on obtient un chiffre pour le texte T, on peut facilement remonter à la clef utilisée (à coût constant).
- **Cryptanalyse différentielle** : étude des différences de chiffrement entre des textes similaires pour sélectionner un sous-ensemble de clefs probables.
- **Cryptanalyse linéaire** : utilisation de relation linéaire entre certaines bits du texte clair et du chiffre afin d'interpoler les bits de la clef.

Il y a $2^{56} \leq 10^{17} = 10^8.10^9$ clef possibles. La complexité des attaques DES est

la suivante :

Méthode d'attaque	texte connu	texte choisi	stockage	calcul
Recherche exhaustive	1			2^{56}
Précalcul exhaustif		1	2^{56}	1 recherche
Cryptanalyse linéaire	$2^{47}+2^{36}$		textes	$2^{47}+2^{36}$
Cryptanalyse différentielle	2^{56}	2^{47}	textes	2^{47}

Faisabilité

En 1996, une étude a été réalisée afin savoir les performances des attaques disponibles sur DES en fonction du budget disponible aux différentes entités :

Attaquant	Budget	Outil	Temps
Hacker	300€	Logiciel	38 ans
PME	7 500€	Circuit	18 mois
Grande entreprise	225k€	ASIC	19 jours
Multinationale	7,5M€	ASIC	6 minutes
Gouvernement	225M€	ASIC	12 secondes

La sécurité du DES doit donc être considérée comme nulle si les données chiffrées intéressent un gouvernement ou une multinationale prêts à y mettre le prix.

Initialement, ce standard n'a été proposé que pour les applications non classifiées (donc, pas militaire).

Afin d'améliorer la sécurité de DES a été proposé :

- le double-DES en combinant 2 clefs (k_1, k_2), $C = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(M))$.
  Malheureusement, l'attaque (meet-in-the-middle) permet de casser le double-DES avec une complexité de l'ordre de 56×2^{56}.
  La clef équivalente est de 64 bits.
  Le double-DES n'est donc qu'environ 100 fois plus difficile (et non 2^{56} si la complexité des clefs s'additionnait).
- le triple-DES combine 3 clefs (k_1, k_2, k_3) (et triple le temps de calcul) avec $C = \mathrm{DES}_{k_3}(\mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(M)))$ (connu sous le nom 3DES-EEE).
  La clef équivalente effective est de 112 bits avec une clef de 168 bits.
- une variation consiste à prendre $C = \mathrm{DES}_{k_1}(\mathrm{DES}^{-1}_{k_2}(\mathrm{DES}_{k_1}(M)))$ (ou 3DES-EDE).
  La clef équivalente est aussi de 112 bits.

Le standard AES a pris sa place depuis les années 2000.

7 AES

AES est l'acronyme d'Advanced Encryption Standard définit en 2001 par le NIST (National Institute of Standards and Technology) pour l'encryption des données numériques, suit à un appel d'offre dans de 1997 pour remplacer le DES.

C'est un sous-ensemble du cryptosystème de Rijndael, développé par Vincent Rijmen and Joan Daemen, deux cryptographes belges. Il s'agit d'un algorithme de chiffrement itératif (n'utilise pas un schéma de Feistel).

Il est devenu un standard du gouvernement fédéral américain à partir de 2002, et est le premier standard cryptographique approuvé pour la NSA pour les informations "top secret" lorsqu'il est utilisé par un module AES approuvé par elle.

Nous décrivons dans cette section le cryptosystème de Rijndael, et AES, son cas particulier.

Comme `DES`, `AES` est un algorithme de chiffrement par bloc, à savoir que le message à chiffrer est découpé en blocs de taille fixe, puis chaque bloc est chiffrés indépendamment.

7.1 Bloc AES

Notations:

- N_b est le nombre de mots de 32 bits qui forment un bloc à chiffrer ou à déchiffrer.
- N_k est le nombre de mots de 32 bits qui constituent la clef.
- N_r est le nombre d'étages.

Un chiffre de Rijndael est défini par deux valeurs :

- **La taille des blocs :**
  pour Rijndael, $4 \leq N_b \leq 8$, donc la longueur d'un bloc est $32.N_b$ bits.
  pour AES, $N_b = 8$ est la longueur d'un bloc est toujours 256 bits.
- **La taille de la clef :**
  pour Rijndael, $4 \leq N_k \leq 8$ et restreint à 4, 6 ou 8 pour AES.
  l'espace des clefs est donc $\mathbb{Z}_2^{32.N_k}$.
  Par exemple, $N_k = 6$, la taille de la clef est $32.6 = 192$ bits.
  L'espace des clefs est $\mathbb{Z}_2^{32.N_k}$, donc de taille 2^{192} ($> 10^{57}$).
- Le nombre de tours est calculé à partir de la longueur de la clef
  $N_r = N_k + 6 = 12$ (pour $N_k = 6$).

Une ronde de chiffrement d'AES a quatre étages :

- un fonction non linéaire de substitution par octet : elle a pour but de résister aux à l'attaque linéaire et à l'attaque différentielle.
- une fonction de permutation par ligne (entre colonnes d'octets sur une même ligne),
- une fonction de permutation par colonne (entre lignes d'octets sur une même colonne),
- une fonction de blanchiment de la clef qui mélange le bloc courant avec une clef d'étage spécifique.

Les trois premiers étages ont pour but d'effectuer du mélange, de la diffusion et de la non-linéarité. Seul le dernier utilise la clef.

Chaque étage est conçu pour être facilement inversible.

La fonction de chiffrement s'applique par bloc, et prend en entrée :

- un bloc à chiffrer est de taille $N_b \times 32$bit.
  il est ensuite organisé sous forme d'une matrice d'octets (8bit) de 4 lignes et de N_b colonnes.
  Par exemple, pour $N_b = 6$, un bloc à chiffrer (ou à déchiffrer) a la forme :

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} & s_{0,4} & s_{0,5} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} & s_{2,4} & s_{2,5} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} & s_{3,4} & s_{3,5} \end{pmatrix}$$

  où chaque $s_{i,j}$ est un octet (=8 bits). Chaque colonne représente un mot de 32 bits.
- une clef diversifiée de taille $N_r + 1 \times N_b$ calculée à partir des $N_k \times 32$bit de la clef.

et retourne en sortie un bloc chiffré de la même taille que le bloc d'entré.

7.2 Algèbre sur $GF(2^8)$

AES a pour particularité d'utiliser l'algèbre dans un corps de Galois pour sa SBox et son mélange de colonnes.

Rappelons comment l'on utilise un corps de Galois afin de générer des opérations inversibles sur un ensemble à 2^8 éléments.

- $GF(2^8)$ est un corps d'ordre 2^8 (= son nombre d'éléments) dans lequel tous les éléments non nuls sont inversibles (rappel : si un élément est inversible, son inverse est unique).
- une façon de construire ce corps de Galois est de prendre un polynôme irréductible f de $\mathbb{Z}/2\mathbb{Z}[x]$, et de considérer l'ensemble des classes des résidus (= les restes) des polynômes par la division par f. Il y en a autant

que de polynômes de $\mathbb{Z}/2\mathbb{Z}[x]$ de degré inférieur à au degré de f. Donc, si f est de degré 8, alors il y a 2^8 classes de résidus.
- le polynôme f étant irréductible, alors tout représentant de classe est inversible. Donc, si $r(x)$ est un résidu, alors il existe $q(x)$ tel que $f(x) + q(x).r(x) = 1$, autrement dit $q(x).r(x) \bmod f(x) = 1 \bmod f(x)$.
- donc, l'ensemble des classes de résidus est homéomorphe à $GF(2^8)$.

On utilisera les propriétés de $GF(2^8)$ de la manière suivante :

- Pour l'opération de substitution d'octet :
  En utilise l'inversiblité des polynômes qui représentent les résidus, et en associant un octet à chaque polynôme, on construit une fonction inversible de substitution d'octet.
- Pour l'opération de mélange des colonnes :
  A chaque colonne d'octets du bloc $\begin{pmatrix} b_0 & b_1 & b_2 & b_3 \end{pmatrix}$, on associe le polynôme $b(x) = \{b_3\}.x^3 + \{b_2\}.x^2 + \{b_1\}.x + \{b_0\}$ dans $GF(2^8)[x]$ où $\{.\}$ est utilisé pour signifier que les b_i sont dans $GF(2^8)$, et utilisent donc l'algèbre de $GF(2^8)$.

  En conséquence, nous aurons besoin des règles de l'additions et de la multiplications dans $GF(2^8)$ (notées respectivement $\oplus$ et $\bullet$) lorsque nous effectuerons des additions ou des multiplications entre polynômes.

7.2.1 Inversion dans $GF(2^8)$

On peut maintenant utiliser cette propriété pour construire une fonction de substitution d'octet de la manière suivante :

- on prend le polynôme irréductible de degré 8 suivant $f(x) = x^8 + x^4 + x^3 + x + 1$ (imposé par le cryptosystème de Rijndael).
- à un octet $b = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$, on peut associer le polynôme $p_b(x) = \sum_{i=0}^{7} b_i.x^i$.
- or, tout polynôme $p_b(x)$ non nul est inversible dans l'ensemble des classes de résidus de la division par $f(x)$ dans $\mathbb{Z}/2\mathbb{Z}[x]$.
- on note $p_{b^{-1}}(x)$ le polynôme inverse de $p_b(x)$ (à savoir $p_b.p_{b^{-1}} \bmod f \equiv 1$).
- à $p_{b^{-1}}(x) = \sum_{i=0}^{7} b_i^{-1}.x^i$, on associe l'octet $b^{-1} = (b_7^{-1}, b_6^{-1}, b_5^{-1}, b_4^{-1}, b_3^{-1}, b_2^{-1}, b_1^{-1}, b_0^{-1})$.
- l'inverse de l'octet b est b^{-1}, et vice-versa puisque la transformation est inversible.
- Pour $b = 0$ (polynôme nul, donc non inversible) alors $b^{-1} = 0$ (seul octet pour lequel $GF(2^8)$ ne définit pas d'inverse).

Comme le polynôme $f(x)$ est fixe, cette fonction de substitution peut être tabulée.

Exemple

Considérons le polynôme générateur de Rijndael $f(x) = x^8 + x^4 + x^3 + x + 1$.
Soit le résidu $r(x) = x + 1$.
Appliquons l'algorithme d'Euclide étendu sur $f(x)$ et $r(x)$.

r_k	q_k	$u_k = u_{k-2} - q_k.u_{k-1}$	$v_k = v_{k-2} - q_k.v_{k-1}$
$f(x)$		1	0
$(x+1)$		0	1
1	$q(x)$	$1 - q(x).0 = 1$	$0 - q(x).1 = q(x)$
0	$x+1$	$0 - (x+1).1 = x+1$	$1 - (x+1).q(x) = f(x)$

où $q(x) = f(x)/(x+1) = x^7 + x^6 + x^5 + x^4 + x^2 + x$ et $r(x) = f(x)\%(x+1) = 1$.

Donc, $1.f(x) + q(x).(x+1) = 1$.

En conséquence, $(x+1).q(x) \bmod f(x) = 1$, et $q(x)$ et l'inverse $(x+1)$ dans les classes des résidus de $f(x)$.

Le nombre binaire de 8 bits associé à $(x+1)$ est $b = 00000011 = 02$, et à $q(x)$ est $b^{-1} = 11110110 = f6$. Par conséquent, l'inverse de $b = 02$ est $b^{-1} = f6$.

7.2.2 Addition sur $GF(2^8)$

L'opération de mélange des colonnes est construite de la manière suivante :

A chaque colonne d'octets du bloc $(\,b_0\ b_1\ b_2\ b_3\,)$, on associe le polynôme $b(x) = \{b_3\}.x^3 + \{b_2\}.x^2 + \{b_1\}.x + \{b_0\}$ dans $GF(2^8)[x]$.
où $\{.\}$ est utilisé pour signifier que les b_i sont dans $GF(2^8)$, et utilisent donc l'algèbre de $GF(2^8)$.

En conséquence, nous aurons besoin des règles de l'additions et de la multiplications dans $GF(2^8)$ (notées respectivement $\oplus$ et $\bullet$) lorsque nous effectuerons des additions ou des multiplications entre polynômes.

Attention, de simplifier les écritures, tous les octets seront écrits en hexadécimal dans les exemples suivants.

Exemple : $\{57\} \oplus \{83\}$
$= \{0101.0111\} \oplus \{1000.0011\}$
$= (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1)$
$= x^7 + x^6 + x^4 + x^2 = \{1101.0100\} = \{D4\}$

On constate que l'addition $\oplus$ dans $GF(2^8)$ est un simple `XOR` entre les représentations binaires (que l'on notait aussi $\oplus$).

7.2.3 Multiplication sur $GF(2^8)$

Pour la multiplication, on rappelle que le corps $GF(2^8)$ d'AES est construit à partir des classes des résidus de $f(x) = x^8 + x^4 + x^3 + x + 1$.

Exemple :

$$\begin{aligned}\{11\} \bullet \{9a\} &= \{0001.0001\} \bullet \{1001.1010\} \\ &= (x^4 + 1).(x^7 + x^4 + x^3 + x) \equiv f(x) \\ &= (x^{11} + x^8 + x^5 + x^4 + x^3 + x) \equiv f(x) \\ &= x^7 + x^6 + x^5 + x^4 + x^3 + 1 = \{1111.1001\} = \{f9\}\end{aligned}$$

car $x^{11} + x^8 + x^5 + x^4 + x^3 + x = (x^3 + 1).f(x) + (x^7 + x^6 + x^5 + x^4 + x^3 + 1)$ (par division Euclidienne).

Le calcul de la multiplication peut être simplifié.

Remarquons tout d'abord que si $x^8 \bmod f(x) = m(x) - x^8 = x^4 + x^3 + x + 1$. Notons que le reste de la division de x^8 par $f(x)$ est $00011011 = 0x1b$ dans $GF(2^8)$.

Soit maintenant $\{v\}$, un élément de $GF(2^8)$ quelconque. Soit $p_v(x)$ le polynôme associé à $\{v\}$.

Lorsque l'on multiplie $p_v(x)$ par 2, si le coefficient de x^7 de $p_v(x)$ (= le bit de poids le plus fort de $\{v\}$) :

- est 0, alors multiplier 2 revient à effectuer un décalage de $\{v\}$ vers la gauche (i.e. $2.p_v(x) \Leftrightarrow v << 1$).
- est 1, alors multiplier 2 revient à effectuer un décalage de $\{v\}$ vers la gauche, puis un XOR avec 00011011 sur le résultat (i.e. $2.p_v(x) \Leftrightarrow (\{v\} << 1) \oplus 00011011$).

On généralise maintenant en remarquant que pour tout b alors $v.b = \sum_i b_i.(v.2^i)$ où $b = \sum_i 2^i.b_i$ est la décomposition binaire de b. Or, $v.2^{i+1} = 2.(v.2^i)$. Donc, la méthode décrite ci-dessus peut être utilisée pour calculer tous les termes $v.2^i$, puis sommer ceux pour lesquels $b_i \neq 0$. On obtient ainsi le produit $v.b$ modulo $f(x)$.

On en déduit l'algorithme de multiplication entre deux éléments dans le $GF(2^8)$ engendré par $f(x)$:

Entrées :
a, b : les éléments de $GF(2^8)$ à multiplier
r : le reste de la division de x^8 par $f(x)$ (= ses 8 premiers bits).
On note v_i le bit numéro i de v (= celui correspondant à x^i).

```
GF(2^8) GF2multiply(GF(2^8) a, GF(2^8) b, GF(2^8) r)
    // initialisation 1 dans b
    r = ((b_0 = 1) ? a : 0)
    // décalage b_i = b_{i+1}
    b = b >> 1
    while b ≠ 0 do
        // multiplication de a par deux, avec reste si dépassement
        a = ((a_7 = 1) ? ((a << 1) ⊕ r) : (a << 1))
        // si b_i contient x^i
        if (b_0 = 1) then
            r = r ⊕ a
        // décalage b_i = b_{i+1}
        b = b >> 1
    return r
```

Exercice 50: Algèbre dans $GF(2^8)$

On se place dans le corps de Galois $GF(2^8)$ engendré par le polynôme irréductible $f(x) = x^8 + x^5 + x^3 + x + 1$.

1. calculer l'inverse de $\{31\}$.
2. calculer $\{31\} + \{c4\}$.
3. est-il possible de trouver deux nombres de $GF(2^8)$ tels que leur somme engendre une retenue sur le dernier bit ? Quelle est la conséquence de ce fait ?
4. Calculer $\{8b\} \times \{a4\}$ en utilisant la multiplication, puis la réduction modulo $f(x)$ dans $GF(2^8)$.
5. Calculer $\{8b\} \times \{a4\}$ en utilisant la méthode de multiplication rapide modulo $f(x)$ dans $GF(2^8)$.

7.3 Fonctions de chiffrement AES

La fonction de chiffrement d'AES est la suivante :

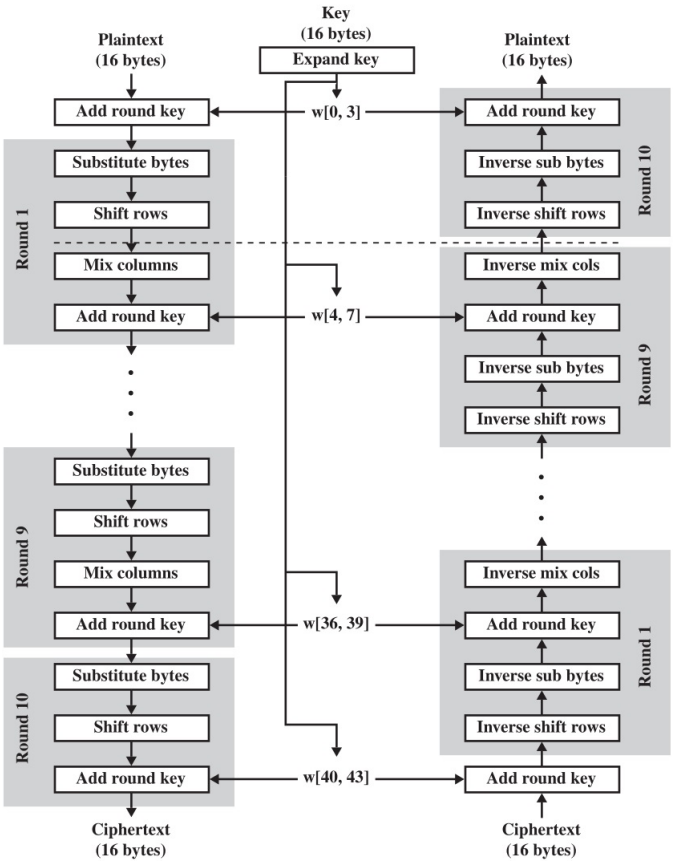
Types :
`AESblock` = type matrice $4 \times N_b$ octets = `matrix<byte>`$(4,N_b)$
`AESkey` = type matrice $N_{r+1} \times N_b$ mots 32 bit = `matrix<word>`(N_{r+1},N_b)
Entrées :
`AESblock` *Block* = bloc à chiffrer en entrée.
`AESkey` *Key* = clé diversifiée (Key_j = $j^{\text{ème}}$ vecteur de N_b mots 32bit).
Sortie : retourne le bloc chiffré (matrice AESblock)

```
AESblock AEScipher(AESblock Block, AESkey Key)
    AddRoundKey(Block,Key_0) // blanchissage
    // fonction d'étage
    for j = 1 à N_r − 1 do
        SubBytes(Block) // substitution octets
        ShiftRows(Block) // permutation des lignes
        MixColumns(Block) // permutation des colonnes
        AddRoundKey(Block,Key_j) // blanchissage
    // dernier tour
    SubBytes(Block) // substitution octets
    ShiftRows(Block) // permutation des lignes
    AddRoundKey(Block,Key_Nr) // blanchissage
    return Block
```

La fonction de déchiffrement d'AES utilise les fonctions inverses pour chacun des étages dans l'ordre exactement inverse.

Entrées :
`AESblock` *Block* = bloc chiffré à déchiffrer en entrée.
`AESkey` *Key* = clé diversifiée (Key_j = $j^{\text{ème}}$ vecteur de N_b mots 32bit).
Sortie : retourne le bloc déchiffré (matrice AESblock)

```
AESblock AESdecipher(AESblock Block, AESkey Key)
    AddRoundKey(Block,Key_Nr) // blanchissage
    // fonction d'étage
    for j = N_r − 1 à 1 do
        InvShiftRows(Block) // permutation des lignes
        InvSubBytes(Block) // substitution octets
        AddRoundKey(Block,Key_j) // blanchissage
        InvMixColumns(Block) // permutation des colonnes
    // dernier tour
    InvShiftRows(Block) // permutation des lignes
    InvSubBytes(Block) // substitution octets
    AddRoundKey(Block,Key_N0) // blanchissage
    return Block
```



7.3.1 Fonction `AddRoundKey`

La tranformation `AddRoundKey(`$Block$`,`Key`)` est la fonction qui ajoute la clef de la ronde Key au bloc courant $Block$.

En entrée,

- La clef Key est un vecteur de N_b mots de 32bit (clef pour la ronde courante).
  On note ce vecteur : $\left(k_1 k_2 \dots k_{N_b}\right)$.
- Le bloc est une matrice de $4 \times N_b$ octets : chaque colonne de la matrice est réassemblée en un mot de 32bit, pour obtenir un vecteur de N_b mots de 32bit.
  On note ce vecteur : $\left(b_1 b_2 \dots b_{N_b}\right)$.

La fonction de blanchiment de la clef est donc : $\forall j, b_j \leftarrow b_j \oplus k_j$
où l'opération $\oplus$ est un xor binaire bit à bit.

7.3.2 Fonction SubBytes

La transformation SubBytes(*Block*) est une fonction non linéaire qui transforme les octets du *Block*. Chaque octet b est transformé indépendamment en binaire de la manière suivante :

$$s(b) = A.b^{-1} \oplus c$$

où :

- b^{-1} est l'inverse de b dans $GF(2^8)$ (voir inversion sur $GF(2^8)$).
- A est une matrice binaire fixe, et c un vecteur binaire fixe.

$$\text{où } A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{et } c = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Comme l'inverse de b utilise toujours le même polynôme générateur, et que A et c sont fixes, cela signifie qu'un octet b est toujours transformé de la même manière.

Cette transformation peut donc être précalculée et tabulée (table de 256 entrées), voir les tables ci-dessous.

Tabulation de la fonction SubBytes :

s	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1x	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2x	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3x	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4x	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5x	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6x	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7x	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8x	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9x	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
ax	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
bx	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
cx	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
dx	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
ex	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
fx	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabulation de la fonction `invSubBytes` :

s^{-1}	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1x	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2x	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3x	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4x	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5x	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6x	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7x	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8x	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9x	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
ax	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
bx	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
cx	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
dx	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
ex	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
fx	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

7.3.3 Fonction `ShiftRows`

La transformation `ShiftRows`(*Block*) est une fonction qui effectue un décalage circulaire à gauche des lignes, indépendamment du nombre N_b de colonnes.

Elle décale la $i^{\text{ème}}$ ligne (avec $i = 0, ..., 3$) de i colonne vers la gauche :

$$\begin{pmatrix} \mathbf{s_{0,0}} & s_{0,1} & s_{0,2} & s_{0,3} & s_{0,4} & s_{0,5} \\ s_{1,0} & \mathbf{s_{1,1}} & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} \\ s_{2,0} & s_{2,1} & \mathbf{s_{2,2}} & s_{2,3} & s_{2,4} & s_{2,5} \\ s_{3,0} & s_{3,1} & s_{3,2} & \mathbf{s_{3,3}} & s_{3,4} & s_{3,5} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{s_{0,0}} & s_{0,1} & s_{0,2} & s_{0,3} & s_{0,4} & s_{0,5} \\ \mathbf{s_{1,1}} & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} & s_{1,0} \\ \mathbf{s_{2,2}} & s_{2,3} & s_{2,4} & s_{2,5} & s_{2,0} & s_{2,1} \\ \mathbf{s_{3,3}} & s_{3,4} & s_{3,5} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

Le but de cette transformation, lorsqu'elle est appliquée sur plusieurs rondes, est une grande diffusion de la clef sur les lignes.

7.3.4 Fonction `MixColumns`

La transformation `MixColumns` transforme chaque colonne du bloc individuellement. On considère une colonne $b_j = (\, b_{0,j}\ b_{1,j}\ b_{2,j}\ b_{3,j}\,)$ où $0 \leq j < N_b$.

La transformation est construite de la manière suivante :

- on transforme la colonne b_j en polynôme $b(x)$:
  $b_j \rightarrow b_{0,j} + b_{1,j}.x + b_{2,j}.x^2 + b_{3,j}.x^3 = b(x)$
- soit le polynôme $a(x) = \{03\}\,.x^3 + \{01\}\,.x^2 + \{01\}\,.x + \{02\}$.
- on calcule : $b(x) \bullet a(x) \bmod(x^4 + 1)$ (algèbre dans $GF(2^8)$).
- les coefficients du polynôme résultant est la colonne transformée :
  $b(x) \bullet a(x) \bmod(x^4 + 1) \rightarrow b'_j$

Comme le polynôme $(x^4 + 1)$ est irréductible, le polynôme $a(x)$ est inversible est $a^{-1}(x) = \{11\}.x^3 + \{13\}.x^2 + \{09\}.x + \{14\}$, et en conséquence, cette transformation est aussi inversible.

On peut montrer que cette transformation peut se réécrire comme la transformation linéaire suivante, pour tout j :

$$\begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix} = \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix} \begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix}$$

7.3.5 Exemple de chiffrement AES

Entrée			RoundKey0	AddRoundKey0
01 23 45 67 89 *ab cd ef* *fe dc ba* 98 76 54 32 10			0*f* 15 71 *c*9 47 *d*9 *e*8 59 0*c b*7 *ad d*6 *af* 7*f* 67 98	0*e* 36 34 *ae* *ce* 72 25 *b*6 *f*2 *6b* 17 *4e* *d*9 *2b* 55 88
SubBytes1	ShiftRows1	MixColumns1	RoundKey1	AddRoundKey1
ab 05 18 *e*4 *8b* 40 3*f* *4e* 89 7*f* *f*0 2*f* 35 *f*1 *fc c*4	*ab* 40 *f*0 *c*4 *8b* 7*f* *fc e*4 89 *f*1 18 *4e* 35 05 3*f* 2*f*	*b*9 *e*4 47 *c*5 94 *8e* 20 *d*6 57 16 *9a f*5 75 51 3*f* *3b*	*dc* 90 37 *b*0 *9b* 49 *df e*9 97 *fe* 72 3*f* 38 81 15 *a*7	65 74 70 75 0*f* *c*7 *ff* 3*f* *c*0 *e*8 *e*8 *ca* *4d d*0 *2a* 9*c*
SubBytes2	ShiftRows2	MixColumns2	RoundKey2	AddRoundKey2
4d 92 51 *9d* 76 *c*6 16 75 *ba* *9b* *9b* 74 *e*3 70 *e*5 *de*	*4d* *c*6 *9b* *de* 76 *9b* *e*5 *9d* *ba* 70 51 75 *e*3 92 16 74	*8e* *b*2 *df* *2d* 22 *f*2 80 *c*5 *db dc f*7 *1e* 12 92 *c*1 52	*d*2 *c*9 *6b* *b*7 49 80 *b*4 *5e* *de* *7e* *c*6 61 *e*6 *ff* *d*3 *c*6	*5c* *7b* *b*4 *9a* *6b* 72 34 *9b* 05 *a*2 31 *7f* *f*4 *6d* 12 94
SubBytes3	ShiftRows3	MixColumns3	RoundKey3	AddRoundKey3
4a 21 *8d* *b*8 *7f* 40 18 14 *6b* *3a* *c*7 *d*2 *bf* *3c* *c*9 22	*4a* 40 *c*7 22 *7f* *3a* *c*9 *b*8 *6b* *3c* *8d* 14 *bf* 21 18 *d*2	*b*1 *ba* *f*9 *1d* *c*1 *f*3 1*f* 19 *0b* *8b* *6a* 24 *cc* 07 *c*3 *5c*	*c*0 *af* *df* 39 89 2*f* *6b* 67 57 51 *ad* 06 *b*1 *ae* *7e* *c*0	71 15 26 24 48 *dc* 74 *7e* *5c* *da* *c*7 22 *7d* *a*9 *bd* *9c*
SubBytes4	ShiftRows4	MixColumns4	RoundKey4	AddRoundKey4
*a*3 59 *f*7 36 52 86 92 *f*3 *4a* 57 *c*6 93 *ff* *d*3 *7a* *de*	*a*3 86 *c*6 *de* 52 57 *7a* 36 *4a* *d*3 *f*7 *f*3 *ff* 59 92 93	*d*4 *3b* *cb* 19 11 44 *ab* *b*7 *fe* 06 62 07 0*f* 73 37 *ec*	2*c* 5*c* 65 *f*1 *a*5 73 0*e* 96 *f*2 22 *a*3 90 43 *8c* *dd* 50	*f*8 67 *ae* *e*8 *b*4 37 *a*5 21 0*c* 24 *c*1 97 *4c* *ff* *ea* *bc*
SubBytes5	ShiftRows5	MixColumns5	RoundKey5	AddRoundKey5
41 85 *e*4 *9b* *8d* *9a* 06 *fd* *fe* 36 78 88 29 16 87 65	41 *9a* 78 65 *8d* 36 87 *9b* *fe* 16 *e*4 *fd* 29 85 06 88	2*a* 83 84 *eb* 47 *e*8 18 10 *c*4 18 27 *0a* 48 *ba* 23 *f*3	58 *9d* 36 *eb* *fd ee* 38 *7d* 0*f* *cc* *9b* *ed* *4c* 40 46 *bd*	72 *1e* *b*2 00 *ba* 06 20 *6d* *cb* *d*4 *bc* *e*7 04 *fa* 65 *4e*
SubBytes6	ShiftRows6	MixColumns6	RoundKey6	AddRoundKey6
40 72 37 63 *f*4 6*f* *b*7 *3c* 1*f* 48 65 94 *f*2 *2d* *4d* *2f*	40 6*f* 65 2*f* *f*4 48 *4d* 63 1*f* *2d* 37 *3c* *f*2 72 *b*7 94	7*b* *1e* 94 94 05 *d*0 83 *c*4 42 20 18 43 *4a* 40 52 *fb*	71 *c*7 *4c* *c*2 *8c* 29 74 *bf* 83 *e*5 *ef* 52 *cf* *a*5 *a*9 *ef*	0*a* *d*9 *d*8 56 89 *f*9 *f*7 *7b* *c*1 *c*5 *f*7 11 85 *e*5 *fb* 14
SubBytes7	ShiftRows7	MixColumns7	RoundKey7	AddRoundKey7
67 35 61 *b*1 *a*7 99 68 21 78 *a*6 68 82 97 *d*9 0*f* *fa*	67 99 68 *fa* *a*7 *a*6 0*f* *b*1 78 *d*9 61 21 97 35 68 82	*ec* 0*c* *3b* *b*7 1*a* 50 *d*7 22 *c*0 53 00 72 80 *c*7 *ef* *e*0	37 14 93 48 *bb* *3d* *e*7 *f*7 38 *d*8 08 *a*5 *f*7 *7d* *a*1 *4a*	*db* 18 *a*8 *ff* *a*1 *6d* 30 *d*5 *f*8 *8b* 08 *d*7 77 *ba* *4e* *aa*
SubBytes8	ShiftRows8	MixColumns8	RoundKey8	AddRoundKey8
*b*9 *ad* *c*2 16 32 *3c* 04 03 41 *3d* 30 *0e* *f*5 *f*4 2*f* *ac*	*b*9 *3c* 30 *ac* 32 *3d* 2*f* 16 41 *f*4 *c*2 03 *f*5 *ad* 04 *0e*	*b*1 *3d* *0a* *9f* 1*a* 2*f* *6b* *68* 44 *ec* 2*f* *f*3 17 *b*6 42 *b*1	48 26 45 20 *f*3 *1b* *a*2 *d*7 *cb* *c*3 *aa* 72 3*c* *be* *0b* 38	*f*9 *1b* *4f* *bf* *e*9 34 *c*9 *bf* 8*f* 2*f* 85 81 *2b* 08 49 89
SubBytes9	ShiftRows9	MixColumns9	RoundKey9	AddRoundKey9
99 *af* 84 08 1*e* 18 *dd* 08 73 15 97 *0c* *f*1 30 *3b* *a*7	99 18 97 *a*7 1*e* 15 *3b* 08 73 30 84 08 *f*1 *af* *dd* *0c*	31 *ac* 46 *6a* 30 71 65 1*c* *3a* *8c* 48 31 *c*2 *c*4 *eb* 62	*fd* 0*d* 42 *cb* 0*e* 16 *e*0 1*c* *c*5 *d*5 *4a* *6e* *f*9 *6b* 41 56	*cc* *a*1 04 *a*1 *3e* 67 85 00 *ff* 59 02 5*f* *3b* *af* *aa* 34

SubBytes10	ShiftRows10		RoundKey10	AddRoundKey10
4*b* 32 *f*2 32 *b*2 85 97 63 16 *cb* 77 *cf* *e*2 79 *ac* 18	4*b* 85 77 18 *b*2 *cb* *ac* 32 16 79 *f*2 63 *e*2 32 97 *cf*		*b*4 *ba* 7*f* 86 8*e* 98 4*d* 26 *f*3 13 59 18 52 4*e* 20 76	*ff* 0*b* 84 4*a* 08 53 *bf* 7*c* 69 34 *ab* 43 64 14 8*f* *b*9

EXERCICE 51: Calcul d'un étage d'AES

Soit le bloc AES suivant à l'entrée d'une ronde :

Entrée	SubBytes	ShiftRows	MixColumns	AddRoundKey	Clef de ronde
$\begin{bmatrix} 5c & 6b & 05 & f4 \\ 7b & 72 & a2 & 6d \\ b4 & 34 & 31 & 12 \\ 9a & 9b & 7f & 94 \end{bmatrix}$	$\begin{bmatrix} 4a & 7f & 6b & bf \\ 21 & 40 & 3a & 3c \\ . & . & . & . \\ b8 & 14 & d2 & 22 \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} b1 & c1 & . & cc \\ ba & f3 & . & 07 \\ f9 & 1f & . & c3 \\ 1d & 19 & . & 5c \end{bmatrix}$	$\begin{bmatrix} 71 & 48 & . & 7d \\ 15 & dc & . & a9 \\ 26 & 74 & . & bd \\ 24 & 7e & . & 9c \end{bmatrix}$	$\begin{bmatrix} c0 & 89 & 57 & b1 \\ af & 2f & 51 & ae \\ df & 6b & ad & 7e \\ 39 & 67 & 06 & c0 \end{bmatrix}$

1. Appliquer la fonction `SubBytes` sur les éléments manquants du bloc.
2. Appliquer la fonction de décalage `ShiftRows` sur le bloc obtenu.
3. Appliquer la fonction de décalage `MixColumns` sur les éléments manquants du bloc.
4. Appliquer la clef de ronde `AddRoundKey` sur les éléments manquants du bloc.

7.4 Diversification de la clef

7.4.1 Principe

La fonction de diversification de la clef prend en entrée la clef constituée de N_k mots 32bit (notée k_i pour $i \in \{0, ..., N_k - 1\}$), et produit $(N_r + 1)$ clefs de ronde de N_b mots 32bit (w_j pour $j \in \{0, ..., (N_k + 1).N_b\}$).

Les mots sont construits de la manière suivantes :

- pour $j < N_k$, $w_j = k_j$.
- les clefs suivantes (pour $N_k \leq j < (N_k + 1).N_b$) sont ensuite codées par paquet de N_k (chaque paquet commence en $j\%N_k = 0$), à partir du paquet précédent, et l'ajout d'une fonction complexe g.

$j\%N_k$	w_j	Note
0	$w_{j-N_k} \oplus g(w_{j-1})$	1er clef du bloc
$\{1, ..., N_k - 1\}$	$w_{j-N_k} \oplus w_{j-1}$	clefs suivantes du bloc
4 (si $N_k > 6$)	$w_{j-N_k} \oplus \texttt{SubBytes}(w_{j-1})$	cas des clefs 256bit

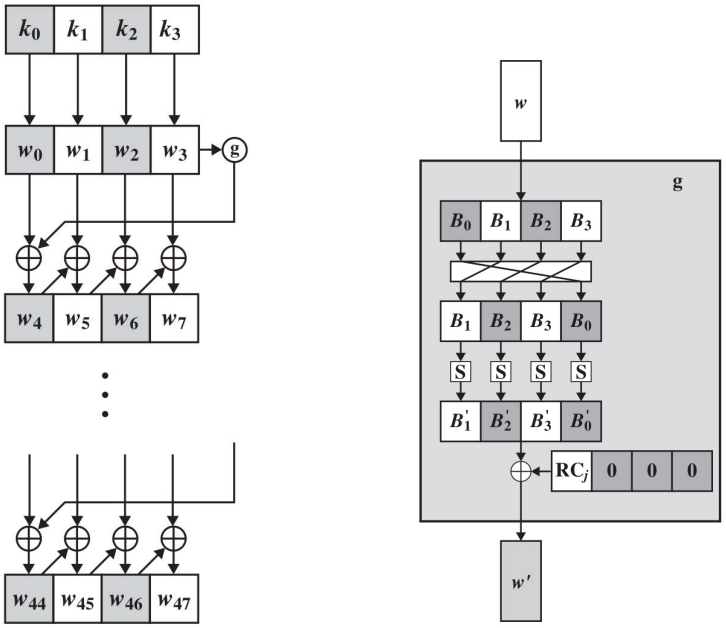
La fonction complexe g prend un mot 32bit w en entrée, et sort un mot 32bit. Elle est construite avec les étages suivants :

- un décalage à gauche d'un octet : $b_0b_1b_2b_3 \rightarrow b_1b_2b_3b_0$.
- la fonction de substitution d'octets (`SubBytes`) appliquée sur chacun des octets

- l'ajout de $\{02\}^{j/N_k}$ (calculé dans $GF(2^8)$) sur le dernier octet. Aussi appelé `RCon` dans les spécifications d'AES. Le calcul de l'exponentiel peut être tabulé (voir les 32 premières valeurs dans la table suivante).

Tabulation de la fonction `Rcon` :

Rcon	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	8d	01	02	04	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a
1x	2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39



7.4.2 Exemple

Soit la clef AES $K = \{$`0f1571c9`, `47d9e859`, `0cb7add6`, `af7f6798`$\}$.

clef ronde 0 :

clef	valeur	calcul
w_0	`0f 15 71 c9`	K_0
w_1	`47 d9 e8 59`	K_1
w_2	`0c b7 ad d6`	K_2
w_3	`af 7f 67 98`	K_3

clef ronde 1 :

w_4	dc 90 37 b0	$w_0 \oplus g(w_3)$
w_5	9b 49 df e9	$w_1 \oplus w_4$
w_6	97 fe 72 3f	$w_2 \oplus w_5$
w_7	38 81 15 a7	$w_3 \oplus w_6$

$g(w_3)$ = SubBytes(LeftShift($w3$) $\oplus$ ({02}^{1} 00 00 00)
= SubBytes(LeftShift(af 7f 67 98)) $\oplus$ ({02}^{1} 00 00 00)
= SubBytes(7f 67 98 af) $\oplus$ ({02}^{1} 00 00 00)
= (0c 59 5c 07) $\oplus$ (02 00 00 00)
= 0e 59 5c 07

clef ronde 2 :

w_8	d2 c9 6b b7	$w_4 \oplus g(w_7)$
w_9	49 80 b4 5e	$w_5 \oplus w_8$
w_{10}	de 7e c6 61	$w_6 \oplus w_9$
w_{11}	e6 ff d3 c6	$w_7 \oplus w_{10}$

$g(w_8)$ = SubBytes(LeftShift($w7$)) $\oplus$ ({02}^{2} 00 00 00)
= SubBytes(LeftShift(38 81 15 $a7$) $\oplus$ ({02}^{2} 00 00 00)
= SubBytes(81 15 $a7$ 38) $\oplus$ ({02}^{2} 00 00 00)
= (0c 59 5c 07) $\oplus$ (04 00 00 00)
= 0e 59 5c 07

...

La table complète des rondes obtenues est :

00	0f	47	0c	af	03	$c0$	89	57	$b1$	06	71	8c	83	cf	09	fd	0e	$c5$	$f9$
	15	$d9$	$b7$	7f		af	2f	51	ae		$c7$	29	$e5$	$a5$		0d	16	$d5$	6b
	71	$e8$	ad	67		df	6b	ad	7e		4c	74	ef	$a9$		42	$e0$	4a	41
	$c9$	59	$d6$	98		39	67	06	$c0$		$c2$	bf	52	ef		cb	1c	6e	56
01	dc	9b	97	38	04	2c	$a5$	$f2$	43	07	37	bb	38	$f7$	10	$b4$	ba	7f	86
	90	49	fe	81		5c	73	22	8c		14	3d	$d8$	7d		8e	98	4d	26
	37	df	72	15		65	0e	$a3$	dd		93	$e7$	08	$a1$		$f3$	13	59	18
	$b0$	$e9$	3f	$a7$		$f1$	96	90	50		48	$f7$	$a5$	4a		52	4e	20	76
02	$d2$	49	de	$e6$	05	58	fd	0f	4c	08	48	$f3$	cb	3c					
	$c9$	80	7e	ff		9d	ee	cc	40		26	1b	$c3$	be					
	6b	$b4$	$c6$	$d3$		36	38	9b	46		45	$a2$	aa	0b					
	$b7$	5e	61	$c6$		eb	7d	ed	bd		20	$d7$	72	38					

7.5 Attaques d'AES

L'algorithme de diversification de la clef a été conçu pour être résistant aux attaques cryptanalytiques connues.

En particulier, l'ajout d'une constante dépendante de la ronde (par la fonction g) permet d'éliminer la symétrie ou les similarités entre la façon dont les clefs sont générées pour chaque ronde.

On notera que :

- la connaissance d'une fraction de la clef ne permet de générer une partie importante des clefs de ronde.
- la transformation étant inversible, la connaissance de N_k clefs consécutives de rondes permet la regénération de la totalité des clefs de rondes.
- la clef d'origine se diffuse très largement dans les clefs de rondes (i.e. chaque bit de la clef se propage à travers de nombreux étages).

AES possède un **effet d'avalanche** : la modification d'un bit de la clef ou d'un bit du message provoque après deux étages le changement d'environ la moitié des bits.
- il y a suffisamment de non-linéarité pour empêcher la détermination complète d'un clef de ronde à partir de cryptanalyse différentielle.

On rappelle que l'on appelle "attaque sur un chiffre" toute méthode permettant de découvrir la clef ou de décoder le message plus rapidement qu'avec une attaque en force brute ; à savoir respectivement pour les versions 128, 192 et 256bit d'AES, 2^{128}, 2^{192} et 2^{256} clefs possibles.

Les attaques connues permettent :
- en utilisant des clefs apparentés (à savoir, une même chaine binaire de plusieurs bits identiques est utilisée au même endroit dans plusieurs clefs différentes pour coder le même message), en temps 2^{39} pour une clef de 256bit à 9 rondes. L'hypothèse de se trouver dans un cas d'utilisation de clefs apparentés est considérée comme très improbable.
- (2011, Bogdanov et al.) de trouver une clef en temps $2^{126.2}$ (resp. $2^{190.2}$, $2^{254.6}$) pour une clef de 128bit (resp. 192bit, 256bit). Outre le gain de temps très faible, l'attaque 128bit nécessiterait un espace 2^{88}.

En considérant que $2^{128} \simeq 3,4.10^{34}$, que l'ordinateur actuel le plus puissant a 10^{17} flops, que 3 années correspondent à environ 10^7 secondes, la durée de l'attaque prendrait plus de un milliard d'années. Aussi, l'espace nécessaire dépasse celui actuellement disponible sur la planète.

Il n'existe pas d'attaque actuellement connue permettant de casser AES.

D'après Snowden, la NSA mène actuellement des recherches en ce sens.

Néanmoins, notons que des attaques par canal auxiliaire sont la faiblesse majeure d'AES.

Ce type d'attaque ne considère pas qu'AES est une boite noire, mais que le programme ou le dispositif de chiffrement produit des fuites de données exploitables (liés à la faiblesse de l'implémentation matérielle ou logicielle) permettant de récupérer les clefs ou restreindre très fortement l'espace des clefs.

On pourra citer comme exemple :
- l'utilisation du nombre de cycles nécessaire à l'encodage de message,
- l'attaque d'implémentation hardware en induisant des fautes afin de révéler certains états internes,
- l'exécution de code sur le même système que celui qui effectue le chiffrement AES, y compris en mode non privilégié.

Ce type d'attaque est très efficace, et constitue un danger majeur sur toute machine qui effectue de chiffrement AES, et peut permettre la récupération de clef en des temps inférieurs à une minute. La sécurisation d'un système pour contrer ce type d'attaque devient donc critique pour continuer à garantir la solidité des outils de chiffrement.

C'est la raison pour laquelle la NSA ne certifie l'AES que sur ses propres modules.

Beaucoups de CPUs modernes (Intel et AMD) ont des instructions spécifiques pour AES afin de les prémunir contre les attaques de timing (AESENC, AESDEC, AESKEYGENASSIST).

8 Modes opératoires

Les méthodes de chiffrement par bloc exposées permettent de chiffrer des blocs de taille fixe.

Comment chiffrer un message de taille quelconque ?
Autrement dit, soit $x = x_1 x_2 \dots x_n$ une suite de blocs de texte clair.
Comment calculer le chiffre correspondant $y = y_1 y_2 \dots y_n$?
On utilise la même clef k pour chiffrer tous les blocs.

Le standard DES fixe 4 méthodes de chiffrement par bloc qui peuvent s'appliquer à toute méthode de chiffrement par bloc. Les modes sont les suivants :

- ECB (Electronic CodeBook mode) : utilisé pour transmission d'une valeur (par exemple, une clef de chiffrement),
- CFB (Cipher FeedBack mode) : utilisé pour l'échange de blocs.
- CBC (Cipher Block Chaining mode) : utilisé pour l'échange de flux.
- OFB (Output FeedBack mode) : utilisé pour l'échange de flux sur des canaux bruités.

Par ailleurs, l'utilisation d'un mode autre que ECB permet de rendre la cryptanalyse plus difficile.

Mode ECB

C'est le mode de chiffrement naïf d'une suite de blocs qui consiste à prendre $y_i = e_k(x_i)$.

Ce mode est à utiliser de préférence pour la transmission de petite quantité de données.

Propriétés du mode ECB :

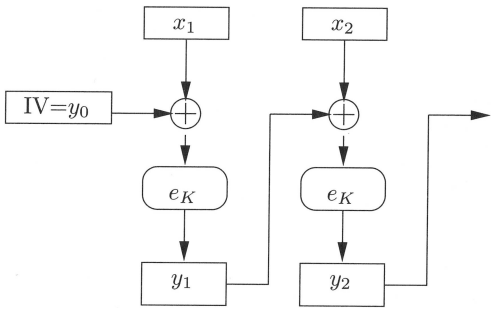
- les textes identiques chiffrés avec la même clef produisent les mêmes chiffres.
- si le message chiffré est fortement structuré ou est répété, cela facilite l'analyse.
- chainage : chaque bloc est chiffré indépendamment des autres.
- propagation d'erreur : une erreur d'un ou plusieurs bits dans un bloc n'affecte que ce bloc.
- la sécurité peut être améliorée en réservant, en fin de bloc, une partie fixe contenant des bits aléatoires pour chaque bloc.

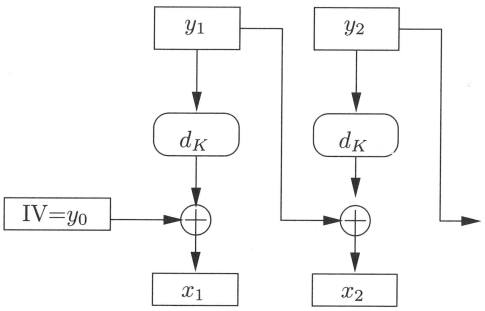
Il n'est donc pas conseillé pour les messages plus long qu'un bloc ou si la clef est utilisée plus d'une fois.

Mode CBC

Chaque cryptogramme y_i agit sur le bloc de texte clair suivant x_{i+1} avant son chiffrement par un xor.

Au départ, l'émetteur et le récepteur s'entendent sur la valeur initiale `IV` de y_0 (peut être 0).





Encodage :

$$\begin{cases} y_0 & = \texttt{IV} \\ y_i & = e_k(x_i \oplus y_{i-1}) \end{cases}$$

Décodage :

$$\begin{cases} y_0 & = \texttt{IV} \\ x_i & = d_k(y_i) \oplus y_0 \end{cases}$$

Propriété du mode CBC :

- les textes identiques chiffrés avec la même clef et le même IV produisent les mêmes chiffres.
  Donc, changer le IV, la clef, ou le premier bloc de texte (en y incluant une partie fixe aléatoire) afin de produire des chiffres différents.
- pour un chainage $(x_1, \ldots, x_n)$, la modification de x_i provoque le changement des blocs $(y_i, \ldots, y_n)$.
- le déchiffrement requiert que l'ordre des blocs soit préservé.
- une erreur d'un bit sur la transmission d'un bloc y_i, provoque une erreur sur le déchiffrement du bloc x_i (typiquement 50% aléatoire), mais n'affecte

que d'un bit le décodage du bloc x_{i+1} suivant. Le bloc au-delà du x_{i+2} ne sont pas affectés.
conséquence : un adversaire peut causer des changements de texte prévisible sur le bloc x_{i+1} en modifiant le bloc y_i.

- La valeur `IV` doit être connue de l'émetteur et du récepteur mais ne doit pas être prévisible par un tiers. Elle peut, par exemple, être transmise chiffrée en mode ECB avant l'utilisation du mode CBC. Son intégrité doit également être préservée afin d'éviter qu'y soit intégré des changements de bits prévisibles sur le premier bloc. L'utilisation d'un `IV` secret permet d'éviter ce problème.

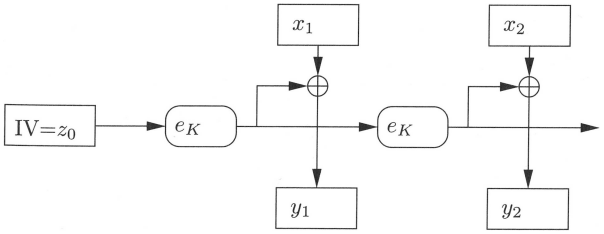
Mode OFB

On utilise la fonction de chiffrement afin de modifier itérativement la clef. La clef est alors utilisée pour coder le texte clair avec un simple xor.

Encodage :
$$\begin{cases} z_0 &= \texttt{IV} \\ z_i &= e_k(z_{i-1}) \\ y_i &= x_i \oplus z_i \end{cases}$$

Décodage :
$$\begin{cases} z_0 &= \texttt{IV} \\ z_i &= e_k(z_{i-1}) \\ x_i &= y_i \oplus z_i \end{cases}$$

Le schéma d'encodage est reproduit ci-dessous :



Propriété du mode OFB :

- des textes identiques chiffrés avec la même clef et le même IV produisent les mêmes chiffres.
  Donc, changer IV ou la clef, ou tous les blocs de texte (en y incluant une partie fixe aléatoire) afin de produire des chiffres différents.
- IV n'a pas besoin d'être secret, mais doit être changé chaque fois que la même clef k est réutilisée (si une cryptanalyse a permis de déterminer les couples (x_i, y_i), les z_i sont obtenus avec des simples $\oplus$).
- chainage : le flux de clef est indépendant du texte clair.
- correction d'erreur : une erreur dans un bloc n'affecte que le caractère sur lequel l'erreur se produit.

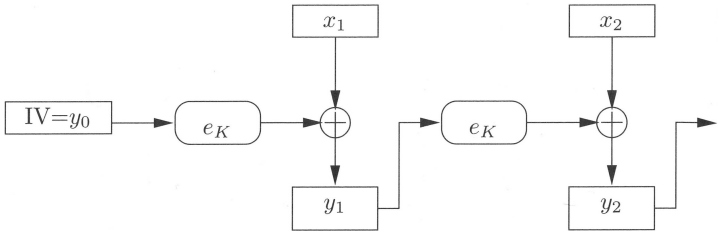
Mode CFB

C'est une variation du mode OFB.

On utilise la chaine cryptée obtenue comme clef du chiffrement suivant. Le chiffre obtenu est utilisé pour coder le texte clair avec un simple xor.

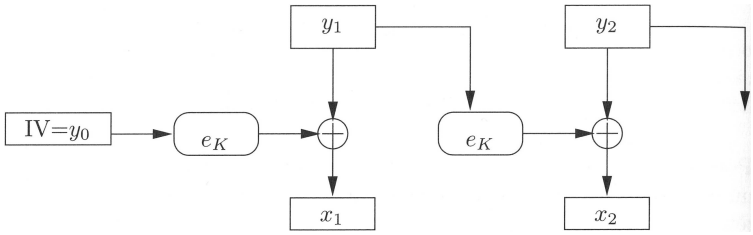
Encodage :

$$\begin{cases} y_0 & = \texttt{IV} \\ y_i & = x_i \oplus e_k(y_{i-1}) \end{cases}$$



Décodage :

$$\begin{cases} y_0 & = \texttt{IV} \\ x_i & = y_i \oplus e_k(y_{i-1}) \end{cases}$$



Propriété du mode CFB :

- des textes identiques chiffrés avec la même clef et le même IV produisent les mêmes chiffres.
  Donc, changer IV ou la clef, ou tous les blocs de texte (en y incluant une partie fixe aléatoire) afin de produire des chiffres différents.
- IV n'a pas besoin d'être secret, mais ne doit pas être prévisible.
- chainage : le chiffre y_i est dépendant des blocs de textes précédents $(x_1, \ldots, x_i)$ (similaire à CBC). Le déchiffrement correct du bloc y_i exige que tous les blocs de chiffre $(y_1, \ldots, y_{i-1})$ soit correct.
- correction d'erreur : si un bloc y_i est incorrect, seuls les bits sur lesquels il y une erreur sont changés dans x_i. En revanche, l'ensemble des blocs suivants seront déchiffrés de manière incorrectes.
- **remarque :** une variation de CFB (nommée CBF-r) permettant de récupérer sur des erreurs existe (en ajoutant remplaçant partiellement le contenu de y_i à partir d'un décalage à gauche de y_{i-1}. En conséquence, au bout d'un nombre r fixé de blocs corrects, on retrouve un bloc "propre").

Exercice 52: Modes opératoires

On travaille sur des blocs de 4 bits. On prend comme fonction d'encodage $e_k(x)$ (pour l'exemple), un chiffre par décalage avec un clef $k = \texttt{0xC}$.
On prend comme chaîne à coder la suite de blocs :

$$B = \{\texttt{0x1}, \texttt{0x2}, \texttt{0x4}, \texttt{0x8}\}$$

On prendra IV=`0x5`.

1. Coder cette chaine par bloc en mode ECB.
2. Coder cette chaine par bloc en mode CFB.
3. Coder cette chaine par bloc en mode CBC.
4. Coder cette chaine par bloc en mode OFB.

9 Conclusion

Les notions abordées ici vous permettent de comprendre et appliquer les méthodes de chiffrement par bloc.

Nous avons vu que le chiffrement par bloc :

- est bien adapté à la sécurisation des données en milieu contrôlé, à savoir dans un cadre où la distribution de la clef ne constitue pas un problème de sécurité.
- utilise des opérations de logiques binaires (donc, chiffrement/déchiffrement rapide).

Autrement dit, ce type d'algorithme constitue le choix préférentiel :

- pour crypter au vol des données sur un disque,
- pour transmettre des informations sur un réseau privé ou semi-privé,
- pour transmettre des informations sur un réseau public, si une méthode de d'échange de clef publique robuste a été utilisée pour échanger les clefs privées.
- ...