# Feuille d'exercices 7

Chiffrement par bloc

## Modes opératoires des chiffrements par bloc

**Exercice 1.** On suppose que lors de la transmission de N blocs de textes chiffrés, un bloc (qui n'est pas le dernier) est mal transmis.

1. Quel est le nombre de blocs qui ne pourront être déchiffrés correctement suivant que l'on est en mode ECB, CBC, OFB, CTR, CFB ?
2. On suppose que seul un bit du bloc transmis a été modifié, préciser les conséquences pour chacun des 5 modes précédents.

Ces 5 modes (et la solution de l'exercice) sont décrits dans le document du NIST (National Institute of Standards and Technology, agence fédérale américaine) « *Recommendation for Block Cipher Modes of Operation Methods and Techniques* » :

`http://dx.doi.org/10.6028/NIST.SP.800-38A`

**Exercice 2.** OpenSSL (voir `http://www.openssl.org`) est une implémentation libre du protocole SSL/TLS largement utilisée sur internet. L'outil en ligne de commande `openssl` permet d'utiliser les fonctions cryptographiques de la bibliothèque OpenSSL. La documentation est accessible par `man 1 openssl` et, pour l'utilisation des chiffres, par `man 1 enc`. Voir aussi : `openssl help`, `openssl enc -help` (plus complète à une époque en source, pour les chiffres utilisables).
Le fichier `tpbloc/couverture.pgm` (accessible à partir de la page du cours) est un fichier image (scan d'une couverture de livre) au format « portable graymap », qui est un format bitmap très simple pour une image en niveaux de gris. Il contient un header en ascii – qui fait dans ce cas particulier 4 lignes –, et indique comment interpréter le reste du fichier – dans ce cas particulier comme une suite d'octets telle que chaque octet décrit un pixel, le nombre de pixels par lignes et le nombre de lignes étant donnés par le header. Pour plus de détails, voir `http://netpbm.sourceforge.net/doc/pgm.html`.
Utiliser les commandes `head`, `tail` pour séparer le header des données dans le fichier image. Chiffrer les données en mode EBC avec le chiffrement DES puis le chiffrement AES (chiffrer par mot de passe), puis en mode CBC pour l'un de ces deux chiffrements. Recoller (commande `cat`) le header aux fichier obtenus et les visualiser. Expliquer le résultat.

## DES

**Exercice 3 (double DES).** On note $e$ et $d$ les fonctions de chiffrement et de déchiffrement du DES, qui est un chiffrement par bloc de n=64 bits, la taille des clefs étant de p=56 bits. On s'intéresse au *double DES*, qui est le chiffrement obtenu en composant deux fois le DES, pour deux clefs différentes :

$$c = e_{k_2}(e_{k_1}(m)) \; ; \;\; m = d_{k_1}(d_{k_2}(c))$$

1. Que pensez-vous de l'intérêt de ce chiffrement si DES était un « groupe » : pour toutes clefs $k_1$ et $k_2$, il existe une clef $k$ tellle que $e_{k_2} \circ e_{k_1} = e_k$ ? En fait DES n'est pas un groupe, la taille du groupe engendré est supérieure à $10^{2499}$ (handbook of applied cryptography, Menezes, van Oorschot and Vanstone, ch 7).
2. A priori combien d'essais sont nécessaires pour casser un double DES par recherche exhaustive ?
3. On suppose que l'attaquant connait deux couples clair-chiffré :

   $$c = e_{k_2}(e_{k_1}(m)) \; ; \;\; c' = e_{k_2}(e_{k_1}(m'))$$

   l'attaquant calcule la liste des $e_k(m)$ pour toutes les clefs $k$ possibles, puis la liste des $d_k(c)$ pour toutes les clefs $k$ possibles. Il cherche un élément commun aux deux listes, soit $(e_{l_1}(m), d_{l_2}(c))$, qui de plus vérifie que $c' = e_{l_2}(e_{l_1}(m'))$

   **a.** Evaluer le nombre d'opérations nécessaire : le nombre d'encryptions, le nombre de comparaisons, ainsi que l'espace mémoire nécessaire.

   **b.** Evaluer la probabilité de succès, c'est à dire que $(l_1, l_2) = (k_1, k_2)$.

   Le double DES n'a pas été utilisé en raison de cette attaque, appelée « attaque par rencontre au milieu » (*meet in the middle attack*), même si, telle quelle, elle est n'est pas très réaliste. C'est un exemple de compromis temps-mémoire (*time-memory tradeoff*), il en existe de plus directement utilisables.

   On a utilisé par contre le triple DES, avec une clef qui peut être de 168 bits : 3 clefs de 56 bits différentes, ou de 112 bits : la même clef de 56 bits est utilisé pour le premier DES et pour le troisième DES.

# AES

Le chiffrement AES (Advanced Encryption Standard) est un chiffrement par bloc de 128 bits. Les clefs peuvent être de 128 bits, 196 bits ou 256 bits. C'est un chiffrement de type réseau de permutations et substitutions (SPN), le nombre de tours dépend de la longueur de la clef :

| clef (nb bits) | tours |
|---|---|
| 128 | 10 |
| 192 | 12 |
| 256 | 14 |

Le chiffrement découpe chaque bloc de 128 bits en 16 octets, un bloc est réprésenté sous forme d'un matrice 4x4. Chaque octet est traité comme un élément du corps $\mathbb{F}_{2^8}$. Plus précisément la suite des bits (du poids fort au poids faible) est vu comme la suite des coefficients, classés par ordre de puissance décroissante, d'un polynôme de degré 7, qui représente un élément de $\mathbb{F}_{2^8}$. La somme est la somme bit à bit (xor). Le produit est le produit polynomial modulo le polynôme irréductible de degré 8 suivant :

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

On note "$*$" l'opération de multiplications (sur les octets). L'inverse de l'octet $b$ pour cette opération est noté $b^{-1}$. Par convention $0^{-1} = 0$.
Un octet peut être représenté par deux chiffres en hexadécimal, on le note $\{XY\}$.
Chaque vecteur colonne (32 bits) est vu, pour certaines opérations, comme un élément de l'anneau quotient $A = \mathbb{F}_{2^8}[X]/X^4 + 1$. On utilisera l'élément $a$ de $A$ suivant, qui est inversible dans $A$ :

$$a(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\}\ ;\ \ a^{-1}(X) = \{0B\}X^3 + \{0D\}X^2 + \{09\}X + \{0E\}\ .$$

**Exercice 4.** **1.** Donner la représentation en hexadécimal des éléments de $\mathbb{F}_{2^8}$ suivant :

$$1,\ X,\ X^2,\ X^3,\ X^4,\ X+1,\ X^4+X^3+X+1$$

**2.** Soit $b$ un octet, $b(X)$ le polynôme correspondant. Exprimer avec l'opérateur `x?y:z`, et les opérations bit à bit `xtime(b)` la multiplication de $b(X)$ par $X$.

**Exercice 5.** **1.** Calculer les inverses dans le corps $\mathbb{F}_{2^8}$ pour la représentation choisie de $X$, et de $1+X$. Traduire en hexadécimal.

**2.** Quel est l'algorithme général qui permet de calculer l'inverse d'un élément dans le corps $\mathbb{F}_{2^8}$ (pour la représentation choisie) ? Cette opération peut être implémentée par une table. Quelle est la taille de cette table ? La calculer.

**Exercice 6.** On se place maintenant dans l'anneau $A$.

**1.** Montrer que $A$ n'est pas un corps, mais que (la classe de) $a(X)$ est inversible dans $A$, d'inverse (la classe de) $a^{-1}(X)$.

**2.** Soient deux polynômes

$$P(X) = c_3X^3 + c_2X^2 + c_1X + c_0$$
$$Q(X) = b_3X^3 + b_2X^2 + b_1X + b_0\ .$$

Montrer que modulo $X^4+1$ :

$$P(X)\cdot Q(X) = \sum_{k=0}^{3} \sum_{i+j=k \bmod 4} a_i b_j X^k$$

**3.** Donner, pour $P(X)$ fixé, une expression matricielle du produit par $P(X)$. Implémenter le produit dans l'anneau (en C on peut prendre comme type d'un élément de l'anneau une union d'un entier de 32 bits et d'un tableau de 4 octets).

**4.** Donner l'expression matricielle du produit par $a(X)$. Pour chacun des éléments $\varepsilon$ de $\mathbb{F}_{2^8}$ qui apparaissent dans cette matrice, donner un algorithme rapide à implémenter pour calculer $\varepsilon * b$. On peut également implémenter ces multiplications à l'aide de tables. Quelle est la taille des tables nécessaires pour le chiffrement ? Le déchiffrement ?

Le chiffrement AES utilise de façon répétée (en dehors de la génération de clefs) les 4 procédures suivantes dans l'ordre indiqué (en vis-à-vis les procédures inverses pour le déchiffrement, à appliquer dans l'ordre opposé) :

```
SubBytes(state)                 invSubBytes(state)
ShiftRows(state)                invShiftRows(state)
MixColumns(state)               invMixColumns(state)
AddRoundKey(state, RoundKey)    AddRoundKey(state, RoundKey)
```

l'entrée `state` est un bloc de 128 bits (vue comme une matrice 4x4 de 16 octets). Ce bloc est initialement le bloc à chiffrer auquel on additionne bit à bit une clef de tour initiale. A chaque tour, il est modifié successivement par chacune des procédures dans l'ordre où elles sont énoncées, sauf au dernier tour où l'on omet la procédure `MixColumns(state)`. L'entrée `RoundKey` est également un bloc de 128 bits, différent pour chaque tour. Cette suite de clefs (y compris celle ajoutée initialement) est engendrée par un algorithme d'ordonnancement et de diversification des clefs (*key schedule*) à partir de la clef initiale.
La procédure de déchiffrement consiste à appliquer dans l'ordre inverse les inverses de chacune des procédures, l'algorithme d'ordonnancement des clefs étant bien entendu modifié pour engendrer celles-ci dans l'ordre inverse.

— La procédure `SubBytes`, décrite à l'exercice 7, est la seule qui ne corresponde pas à une fonction linéaire.
— La procédure `ShiftRows` applique une permutation circulaire différente à chaque ligne de 4 octets de la matrice d'état. On en déduit la procédure inverse `InvShiftRows`.

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{bmatrix} \mapsto \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 & b_0 \\ c_2 & c_3 & c_0 & c_1 \\ d_3 & d_0 & d_1 & d_2 \end{bmatrix}$$

— La procédure `MixColumns` est la multiplication de chaque vecteur colonne de la matrice d'état, vu comme un élément de l'anneau $A$, par le polynôme $a(X)$ (voir dernière question de l'exercice 6). La procédure inverse `InvMixColumns` correspond donc à la multiplication par $a^{-1}(X)$.
— La procédure `AddRoundKey` additionne bit à bit à l'état `State` la clef de tour `RoundKey`. La procédure inverse est donc identique, les clefs devant être engendrées dans l'ordre inverse.

La procédure d'ordonnancement des clefs prend en entrée la clef de chiffrement et produit en sortie autant de clefs de tour que nécessaire, chacune de 128 bits. Plus précisément la procédure engendre une suite `W[i]` de mots de 32 bits, et les clefs de tour sont obtenues en les regroupant consécutivement 4 par 4. Elle utilise

— une substitution `SubWord()` consistant à appliquer la S-Box de l'AES (voir ex. 7) à chacun des 4 octets du mot ;
— une permutation circulaire `RotWord()`, des 4 octets d'un mot de 32 bits ($a_0a_1a_2a_3 \mapsto a_1a_2a_3a_0$) ;
— des constantes de tours `Rcon[i]`, pour $i \geq 1$ `Rcon[i]` $:= X^{i-1}\{00\}\{00\}\{00\}$ dans le corps $\mathbb{F}_{2^8}$ représenté comme indiqué ci-dessus (modulo $m(X)$).

Supposons la clef constituée de `Nk` mots de 32 bits. Alors aux `Nk` premiers mots `W[i]` sont affectés dans l'ordre les `Nk` mots de la clef. Les mots `W[i]` sont obtenus ensuite en nombre suffisant pour engendrer toutes les clefs de tour par :

— `W[i]:=SubWord(RotWord(W[i-1])) xor Rcon[i/Nk]` si `i mod Nk == 0`
— `W[i]:=SubWord(W[i-1]) xor W[i-Nk]` si `i mod Nk == 4` (clef de 256 bits seulement)
— `W[i]:=W[i-1] xor W[i-Nk]` sinon

**Exercice 7.** La procédure `SubBytes` agit indépendemment sur chacun des 16 octets de la matrice, en appliquant la même S-Box. Sur chaque octet, la S-Box est composée deux transformations :

— l'opération : $x \mapsto x^{-1}$ dans $\mathbb{F}_{2^8}$ prolongée en 0 (par $0^{-1} = 0$ !), c'est encore l'application : $x \mapsto x^{254}$ ;
— une transformation affine sur $\mathbb{F}_{2^8}$ (comme $\mathbb{F}_2$-ev) donnée par, $c_i$ étant le i-ème bit de l'octet {63} :

$$b'_i = b_i + b_{i+4 \bmod 8} + b_{i+5 \bmod 8} + b_{i+6 \bmod 8} + b_{i+7 \bmod 8} + c_i, \quad 1 \leq i < 8 .$$

1. Écrire cette transformation affine sous forme matricielle.
2. Cette transformation pourrait être implémentée à l'aide d'une table (S-box). Quelle est la taille de cette table ? La calculer.
3. Décrire la transformation réciproque `InvSubBytes`, et son implémentation.

**Exercice 8.** Une proposition des auteurs d'AES pour implémenter la transformation `MixColumns` est la suivante (`a` est un tableau de 4 octets) :

```
Tmp = a[0] ^ a[1] ^ a[2] ^ a[3] ;
Tm = a[0] ^ a[1] ; Tm = xtime(Tm); a[0] ^= Tm ^ Tmp;
Tm = a[1] ^ a[2] ; Tm = xtime(Tm); a[1] ^= Tm ^ Tmp;
Tm = a[2] ^ a[3] ; Tm = xtime(Tm); a[2] ^= Tm ^ Tmp;
Tm = a[3] ^ a[0] ; Tm = xtime(Tm); a[3] ^= Tm ^ Tmp;
```

1. Justifiez.
2. Cette méthode est recommandée par les auteurs pour les processeurs 8 bits (c'est le cas des cartes à puce de l'époque, et de certaines actuelles). Expliquez pourquoi dans ce cas la procédure de déchiffrement est moins rapide que celle de chiffrement. Pour quels modes opératoires du chiffrement cela a-t-il une incidence ?

**Exercice 9.** Sur les processeurs d'au moins 32 bits, les auteurs d'AES proposent d'implémenter l'algorithme par recherche de valeur dans des tables d'entiers de 32 bits (un entier de 32 bits peut correspondre à une ligne ou une colonne de la matrice d'état). Soient $s_0$, $s_1$, $s_2$, $s_3$ les 4 vecteurs lignes et $t_0$, $t_1$, $t_2$, $t_3$ les 4 vecteurs colonnes de la matrice d'état au

début d'un tour de AES. On les note $s'_i$, $t'_i$ à la fin du tour. Soient $k_0$, $k_1$, $k_2$, $k_3$ les vecteurs colonnes correspondant à la clef générée au tour considéré. On note comme d'habitude $s_{i,j}$ la $j$-ième composante du vecteur $s_i$, et $s_{i,j-p}$ est la $k$-ième composante du vecteur $s_i$ ($0 \le k \le 3$), où $k = j - p \bmod 4$. Enfin la S-Box sur les octets utilisée par la procédure `SubBytes` (voir exercice 7) est notée $S$.

1. Indiquer comment calculer « efficacement » avec la représentation choisie les 4 vecteurs $s_0$, $s_1$, $s_2$, $s_3$ en fonction de $t_0$, $t_1$, $t_2$, $t_3$ ?
2. Montrer que pour $0 \le j \le 3$, si le tour n'est pas le dernier :

$$t'_j = S(s_{0,j}) \cdot \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} + S(s_{1,j-1}) \cdot \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} + S(s_{2,j-2}) \cdot \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + S(s_{3,j-3}) \cdot \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} + k_j$$

3. En déduire que l'on peut implémenter un tour d'AES (qui n'est pas le dernier) en utilisant 4 tables $T_i$, $0 \le i \le 3$, de 256 entiers sur 32 bits. Estimer le nombre d'opérations nécessaires (en les distinguant suivant leur nature).
4. Indiquer comment réaliser l'implémentation du dernier tour (sans `MixColumns`) en utilisant ces 4 tables et du masquage.
5. Calculer les 4 tables en question.

L'implémentation du chiffrement AES était réalisée en utilisant ces 4 tables. On les trouve dans les sources de ssl, fichier `aes_core.c` (voir, à partir de la page web du cours, `tpbloc/openssl_aes` ).
Les processeurs 64 bits modernes disposent d'instructions dédiées (par exemple `AES-NI` depuis 2010 pour Intel) qui permettent encore d'accélérer le calcul.

6. Il est possible, par exemple en cas de place mémoire restreinte, de n'utiliser que l'une de ces 4 tables (c'est une proposition des auteurs de l'AES). Indiquer comment on peut alors engendrer les 3 autres par une procédure simple à partir de l'une d'entre elles.

**Exercice 10 (déchiffrement).**

1. Décrire rapidement une procédure de déchiffrement qui appliquerait, dans l'ordre inverse, l'inverse de chaque procédure de chiffrement.
2. Montrer que l'on peut permuter les procédures `InvShiftRow` et `InvSubBytes`.
3. Indiquer comment modifier la clef de tour correspondante pour pouvoir permuter les procédures `AddRoundKey` et `InvMixColumns`.
4. donner un découpage en tours pour le déchiffrement qui applique à chaque tour les procédures inverses des procédures de chiffrement dans le même ordre que les procédures initiales, en précisant comment modifier l'expansion de clefs.

En procédant de cette façon, on peut utiliser le même algorithme pour le déchiffrement que l'algorithme de chiffrement décrit à l'exercice précédent (avec 4 nouvelles tables), et les temps de calcul du chiffrement et du déchiffrement sont alors analogues.

**Exercice 11.** Soit $\mathbb{F}_{2^n}$ le corps à $2^n$ éléments. Soient $\alpha$ et $\beta$ deux éléments de $\mathbb{F}_{2^n}$. Le but est d'étudier la S-Box de l'AES. On prolonge donc l'inverse en 0 par $0^{-1} = 0$, et il faut tenir compte de ceci dans les questions qui suivent.

1. Soient $\alpha$ et $\beta$ deux éléments de $\mathbb{F}_{2^n}$ non tous les deux nuls. Montrer que

$$(x+\alpha)^{-1} + x^{-1} = \beta \qquad (*)$$

(avec l'interprétation ci-dessus de l'inverse) a au plus 2 solutions si $\beta^{-1} \neq \alpha$, et au plus 4 si $\beta^{-1} = \alpha$. Quel est le nombre de solutions quand $\alpha = \beta = 0$ ?

2. Montrer que 3 est un diviseur de $2^n - 1$ si et seulement si $n$ est pair.
3. En déduire que l'équation $(*)$ a au plus 2 solutions si $n$ est impair, et 4 solutions si $n$ est pair (élever au carré l'équation).
4. On appelle $s$ la substitution associée à la S-Box de l'AES, qui est obtenue en composant l'inverse sur $\mathbb{F}_{2^8}$ prolongée par 0 en 0 et une transformation affine. Montrez que $\alpha$ et $\beta$ étant deux éléments non tous les deux nuls de $\mathbb{F}_{2^8}$, l'équation

$$s(x+\alpha) + s(x) = \beta$$

a au plus 4 solutions.

Ce résultat permet, en faisant l'hypothèse que $x$ et $y$ sont distribués uniformément, de donner une bonne majoration (laquelle ?) de la probabilité que pour deux éléments $x$ et $y$ de différence $\alpha$, la différence de sortie $s(x)+s(y)$ soit $\beta$ ($(\alpha,\beta) \neq (0,0)$), ce qui est utile pour la résistance aux attaques différentielles.

Le chiffrement AES a été proposé en 1998 sous le nom de *Rijndael*, et adopté comme standard en 2002 par le National Institute of Standards and Technology – NIST (agence du gouvernement fédéral des USA). Il est décrit dans les documents :

`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
`http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`