Master Cryptographie & Sécurité

CHIFFREMENT AUDIO

http://www.larbiguezouli.com

Présenté par Dr Larbi GUEZOULI

CHIFFREMENT AUDIO

- Introduction [1 séance]
- Chapitre I: Compression en MP3 [2 séances]
  - Introduction
  - Banque de filtres d'analyse de sous-bandes
  - Le modèle psychoacoustique
  - La transformation MDCT
  - La quantification
  - Formatage de Bitstream (ou du flux)
- Chapitre II: Chiffrement audio total en utilisant DES [2 séances]
  - Introduction
  - Algorithme Data Encryption Standard (DES)
  - Chiffrement audio total en utilisant DES
- Chapitre III: Chiffrement audio en utilisant AES [2 séances]
  - Introduction
  - Algorithme de chiffrement
  - Chiffrement audio total en utilisant AES
  - Problématique
  - Chiffrement audio sélectif en utilisant AES
- Chapitre IV: Chiffrement audio shuffle (Audio shuffle-encryption) [1 séances]
  - Introduction
  - Algorithme audio shuffle
  - Algorithme de chiffrement
  - Algorithme de déchiffrement

Mode d'évaluation :
Examen final : coef. 2
Contrôle continu : coef. 1

INTRODUCTION

Problématique:

La **transmission** des données audio demande une **réduction** dans la taille des données à envoyer ainsi qu'une **protection** de ces données.

La **réduction** de la taille des données audio est réalisée dans la plupart des cas par une **compression MP3**.

La **protection** des données audio compressées peut être faite par différentes techniques. Les plus utilisées sont: **DES total, AES total, AES sélective et le chiffrement aléatoire**.

CHAPITRE I
COMPRESSION EN MP3

PLAN

1. Introduction
2. Banque de filtres d'analyse de sous-bandes
3. Le modèle psychoacoustique
4. La transformation MDCT
5. La quantification
6. Formatage de Bitstream (ou du flux)

CHAPITRE I
COMPRESSION EN MP3

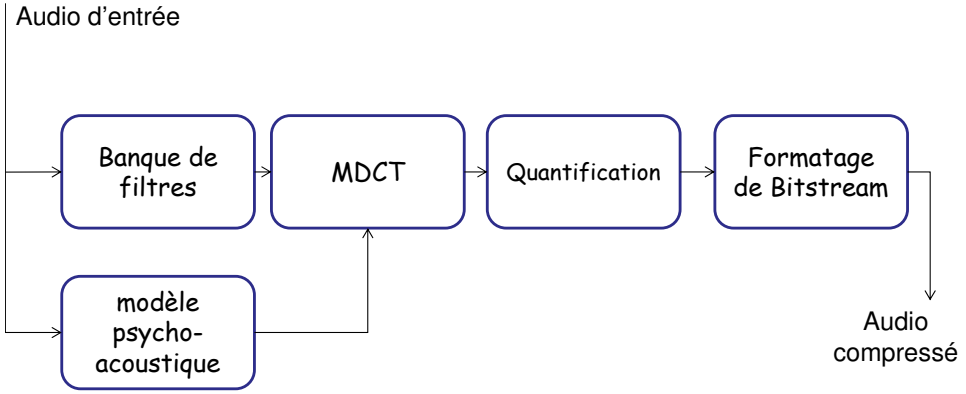
1. Introduction

L'algorithme de compression en MP3 est divisé en 5 étapes:

- Banque de filtres d'analyse de sous-bandes, pour diviser le signal en sous-bandes de fréquences;
- Application du modèle psychoacoustique, pour déterminer dans chaque sous-bande les fréquences qui peuvent être entendues et celles qui ne le peuvent pas;
- La transformation MDCT (Modified Discrete Cosine Transform)
- Quantification
- Formatage de Bitstream

CHAPITRE I
COMPRESSION EN MP3

1. Introduction (suite)



CHAPITRE I
COMPRESSION EN MP3

2. Banque de filtres d'analyse de sous-bandes

Le standard MP3 recommande l'utilisation de **filtre passe-haut**. Ce filtre utilise un seuil (entre 2Hz et 10Hz) pour faire passer les fréquences supérieurs à ce seuil.

Une **banque de filtres d'analyse de sous-bandes** est un ensemble de filtres couvrant toute la gamme de fréquences audio.

Cette banque de filtres est utilisée pour diviser le signal d'entrée avec une certaine fréquence d'échantillonnage (44.1Khz, …) en 32 sous-bandes.

CHAPITRE I
COMPRESSION EN MP3

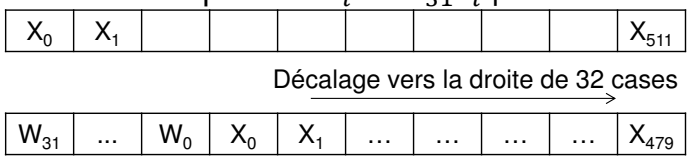
2. Banque de filtres d'analyse de sous-bandes (suite)

L'algorithme de filtre d'analyse de sous-bandes est divisé en plusieurs étapes:

- Entrée de 32 échantillons audio $W_i \mid i = 0 \rightarrow 31$
- Construire un vecteur échantillon X de 512 éléments:

$$X_i = X_{i-32} \mid i = 511 \rightarrow 32$$

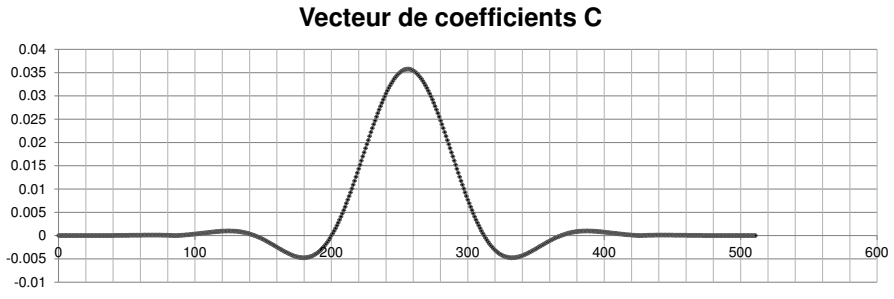
puis $X_i = W_{31-i} \mid i = 31 \rightarrow 0$



2. Banque de filtres d'analyse de sous-bandes (suite)

- Appliquer le vecteur de coefficients C, comme une fenêtre, au vecteur X

$$Z_i = C_i * X_i \mid i = 0 \; jusqu'\text{à } 511$$



2. Banque de filtres d'analyse de sous-bandes (suite)

- Calculer le vecteur Y de taille 64:

$$Y_i = \sum_{j=8i}^{8i+7} Z_j + 64 \cdot j \qquad \mid i = 0 \; jusqu'\text{à } 63$$

2. Banque de filtres d'analyse de sous-bandes (suite)

- Calculer les échantillons filtrés des 32 sous-bandes:

$$S_i = \sum_{k=0}^{63} M_{i,k} \times Y_k \mid i = 0 \; jusqu'\text{à } 31$$

tel que la matrice M est calculée comme suit:

$$M_{i,k} = cos\left[\frac{(2i+1)\cdot(k-16)\cdot\pi}{64}\right] \mid \begin{cases} i = 0 \; jusqu'\text{à } 31 \\ k = 0 \; jusqu'\text{à } 63 \end{cases}$$

3. Le modèle psychoacoustique

Généralités

L'unité de calcul de la fréquence est le **hertz** qui représente les **cycles par seconde**.

Seulement un intervalle de fréquences est **perceptible** par l'humain. Le **rang audible** est entre **20 Hz** et **20 KHz**.

La voix de l'être humain est limitée entre **80 Hz** et **850 Hz**. La **parole** normale entre **110 Hz** et **300 Hz**.

3. Le modèle psychoacoustique (suite)

La **psychoacoustique** est la branche de la psychophysique qui fait appel à **l'acoustique** (qui étudie la nature et les propriétés des ondes sonores), à la **physiologie** de l'audition et à la **psychologie** pour régler quelques paramètres et seuils.

3. Le modèle psychoacoustique (suite)

Le modèle psycho-acoustique calcule le **niveau de pression acoustique SPL** (Sound Pressure Level) qui est mesuré en décibel (**dB**).

$$SPL = 20 \cdot log\left(\frac{\Delta p}{\Delta p_0}\right) \quad en\ dB$$

tel que Δp est la pression sonore actuelle,

Δp_0 est la pression sonore de référence (= 20μPa dans l'air).

3. Le modèle psychoacoustique (suite)

Le modèle psychoacoustique calcule aussi le **taux signale-masque SMR** (Signal-to-Mask Ratio) pour chaque sous-bande.

$$SMR_{SB}(n) = L_{SB}(n) - T_{min}(n) \quad en\ (dB)$$

Pour cela, il faut calculer pour chaque sous-bande n le **niveau maximal du signale (L_{SB})** et le **seuil minimal de masquage (T_{min})**.

3. Le modèle psychoacoustique (suite)

Le **niveau maximal du signal** de la sous-bande n est calculé en (dB) comme suit:

$$L_{SB}(n) = max[P(k), 20 \cdot log(SCF_{max}(n) \cdot 32768) - 10]$$

tel que:

3. Le modèle psychoacoustique (suite)

tel que:

P(k): est le niveau de pression acoustique (sonore) de la ligne spectrale numéro (k) de la FFT (Fast Fourier Transform).

$SCF_{max}(n)$: est le maximum des trois facteurs d'échelle de la sous-bande (n).

3. Le modèle psychoacoustique (suite)

Le **seuil minimal de masquage** $T_{min}(n)$ de la sous-bande (n) est défini par l'expression:

$$T_{min}(n) = min[T_g(i)] \quad en\ dB$$

tel que: $T_g(i)$ est la fréquence du $i^{ème}$ l'échantillon de la sous-bande (n). Il est calculé et donné par une table fixe. Par exemple dans le cas d'un taux d'échantillonnage de 32 Khz:

Index (i)	1	2	3	4	5	6	7	…	130
Fréquence	43,07	86,13	129,20	172,27	215,33	258,40	301,46	…	19982,81

4. La transformation MDCT (Modified Discrete Cosine Transform)

La sortie de la banque de filtres ne permet pas une **reconstruction parfaite**. Pour cela, on utilise la transformée MDCT, et on appel cette combinaison **banque de filtres hybride**.

4. La transformation MDCT (suite)

Nous allons voir les deux phase de la MDCT, **direct** pour la transformation, et **inverse** pour la reconstruction.

4.1. MDCT direct

On utilise des blocs de taille **2M** échantillons, avec un chevauchement de 50%.

Soit un bloc de données en entrée x(n). Le vecteur de coefficients de la transformée X(k) avec $0 \leq k \leq M-1$ est calculé comme suit:

$$X(k) = \sum_{n=0}^{2M-1} x(n) \cdot h_k(n)$$

4.1. MDCT direct (suite)

tel que h_k s'appel **filtre d'analyse MDCT** ou **vecteur de base MDCT**:

$$h_k(n) = w(n) \cdot \sqrt{\frac{2}{M}} \cdot cos\left[\frac{(2n + M + 1) \cdot (2k + 1) \cdot \pi}{4M}\right]$$

tel que:

$$w(n) = sin\left(\frac{\pi}{2M} \cdot \left(n + \frac{1}{2}\right)\right)$$

4.2. MDCT inverse

La transformation MDCT **inverse** permet de faire la reconstruction en calculant la **somme** des vecteurs de base $h_k(i)$ **pondérés** par les **coefficients** de la transformée X(k).

Les M premiers échantillons du k^ième vecteur de base $h_k(i)$ avec ($0 \leq i \leq M-1$) son pondérés par le k^ième coefficient du bloc actuel X(k).

Les M échantillons suivants du k^ième vecteur de base $h_k(i)$ avec ($M \leq i \leq 2M-1$) son pondérés par le k^ième coefficient du bloc précédent $X^p(k)$.

4.2. MDCT inverse (suite)

Les M échantillons reconstruits x(k) avec ($0 \leq k \leq M-1$) sont obtenus comme suit:

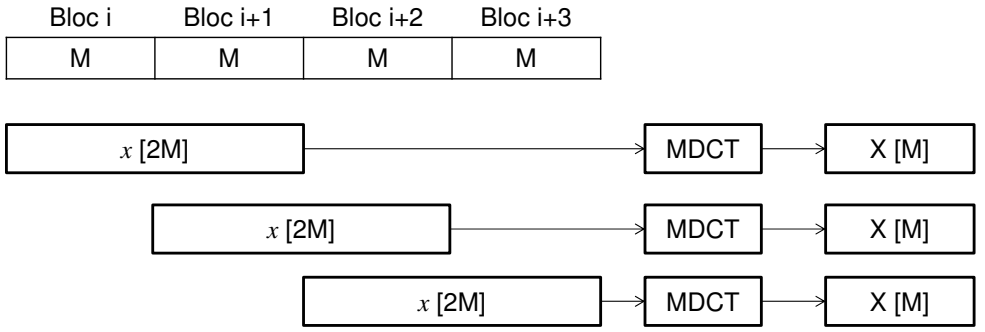
$$x(k) = \sum_{i=0}^{M-1} [X(k).h_k(i) + X^p(k).h_k(i + M)]$$

tel que: $h_k(i)$ est le i^ème échantillon du k^ème vecteur de base;

X(k) est le k^ème coefficient du bloc actuel;

$X^p(k)$ est le k^ème coefficient du bloc précédent.

4.2. MDCT inverse (suite)



5. La Quantification

La quantification permet de **supprimer** des composantes situées en **dessous** du seuil minimal de masquage $\mathbf{T_{min}}$ calculé dans le modèle psychoacoustique.

6. Formatage de Bitstream (du flux)

Le bitstream (ou le flux) est organisé en **trames** de **1152** bits (soit 26ms environ à 44,1kHz).

Chaque trame contient toutes les informations nécessaires au décodage, comme :

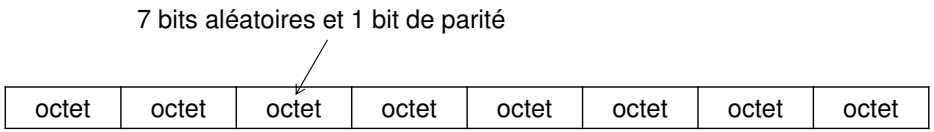
- Entête de 32 bits;
- Les informations de protection sur 16 bits;
- Les coefficients d'amplification et les informations de quantification utilisées dans cette trame sur 136 à 256 bits;
- Les données audio compressées.

PLAN

1. Introduction

DES est un **algorithme** de chiffrement/déchiffrement utilisant une **clé** de **64 bits**.

Parmi les 64 bits de la clé, **56 bits** sont générés **aléatoirement**, les **8 bits** qui restent sont utilisés pour la détection des erreurs en vérifiant la **parité** (1 bit pour les 7 bits de chaque octet).



Clé DES de 64 bits = 8 octets

2. Algorithme Data Encryption Standard (DES)

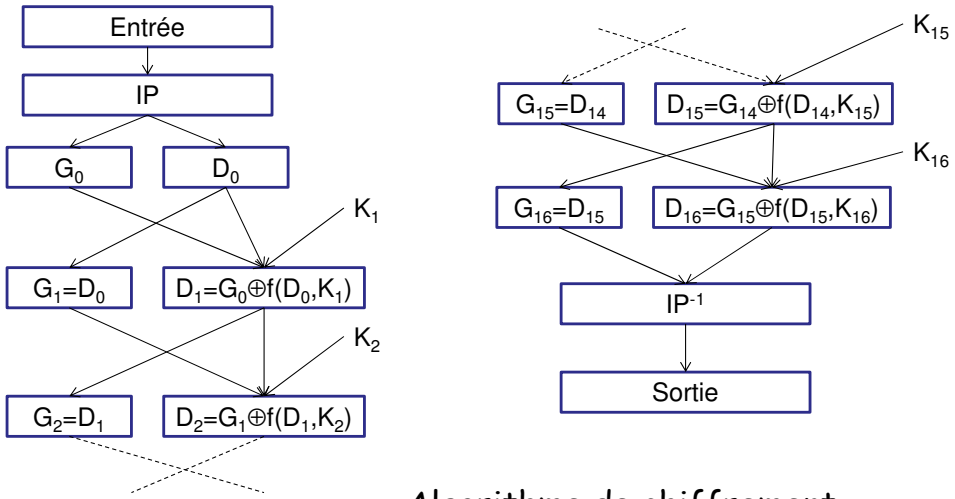
L'algorithme est conçu pour **chiffrer** et **déchiffrer** des **blocs** de **données** de **64 bits** sous le contrôle d'une **clé** de 64bits.

Le **déchiffrement** doit être fait en utilisant la **même clé** de chiffrement.

2.1. Algorithme de chiffrement DES

L'algorithme est divisé en plusieurs étapes:

- Permutation initiale du bloc à chiffrer (IP: Initial Permutation);
- Calcul complexe dépendant de la clé en utilisant une fonction de chiffrement appelée f et une fonction de planification KS (Key Schedule);
- Permutation inverse (IP^{-1}) qui est l'inverse de la permutation initiale.



Algorithme de chiffrement

2.1.1. Permutation initiale du bloc à chiffrer (IP: Initial Permutation)

La permutation initiale se fait selon le bloc suivant:

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

2.1.1. Permutation initiale du bloc à chiffrer (IP: Initial Permutation) (suite)

Le 58ème bit de l'entrée représente le 1er bit de la sortie…

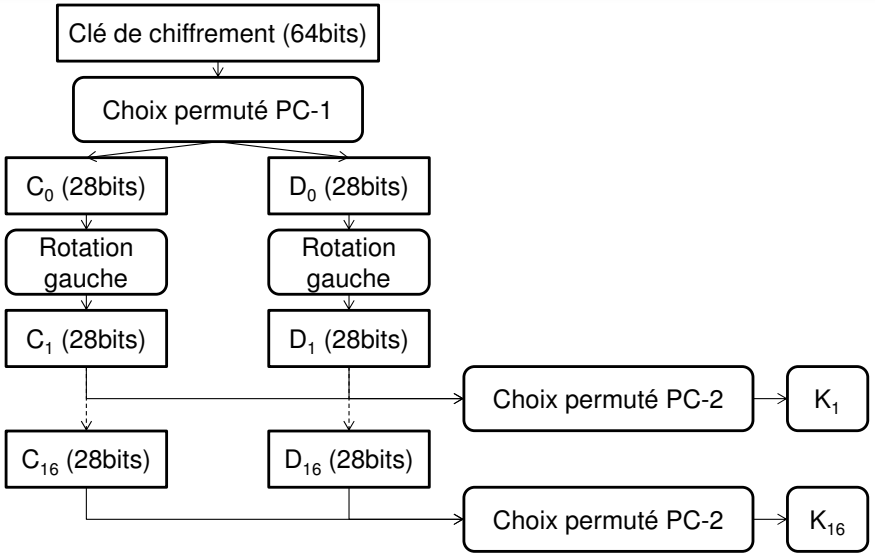
Le 7ème bit de l'entrée représente le dernier bit (64ème) de la sortie.

2.1.2. Fonction de planification KS (Key Schedule)

La **fonction de planification** KS permet de choisir 48 bits à partir de la clé de chiffrement pour générer les 16 clés:

$$K_n = KS(n, clé\ de\ chiffrement) \quad | n = 1..16$$

Cette fonction permet de **programmer** les clés pour les **itérations**.



2.1.2. Fonction de planification KS (suite)

Le choix permuté PC-1 utilise la table:

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

2.1.2. Fonction de planification KS (suite)

Les bits de C_0 sont 57, 49, …, 36 de la clé, et les bits de D_0 sont 63, 55, …, 4.

Pour définir les bits de C_1, … C_{16} et D_1, D_{16} on utilise des décalages gauches avec rotation selon la table:

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

Le décalage gauche de C_0 est sur un bit en rotation.

Le décalage gauche de C_2 est sur 2 bits en rotation.

2.1.2. Fonction de planification KS (suite)

Le choix permuté PC-2 utilise la table:

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

Le 1er bit du K_n est le 14ème bit de C_nD_n. Le 48ème bit du K_n est le 32ème bit de C_nD_n.

2.1.3. Calcul complexe dépendant de la clé

Ce calcul se fait en 16 itérations.

Dans chaque itération, on utilise la **fonction de chiffrement** f qui prend en argument un bloc de 32 bits et une clé de 48 bits choisie à partir des 64 bits de la clé de chiffrement par la **fonction de planification** KS.

2.1.3. Calcul complexe dépendant de la clé (suite)

Les 16 clés générées par la fonction de planification KS seront utilisées par la fonction de chiffrement dans les 16 itérations:

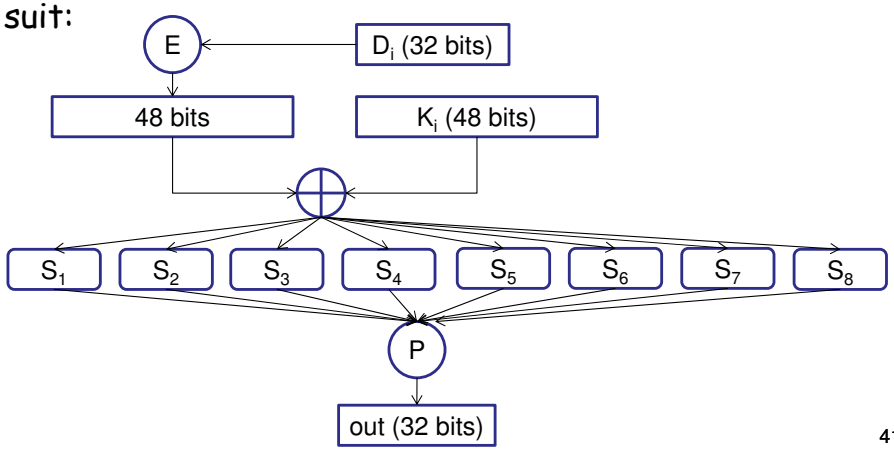
$G_n = D_{n-1}$

$D_n = G_{n-1} \oplus f(D_{n-1}, K_n)$

L'opération $\oplus$ fait l'addition bit par bit modulo 2 (XOR).

2.1.4. Fonction de chiffrement f

L'algorithme de la fonction f est représenté comme suit:



2.1.4. Fonction de chiffrement f (suite)

La fonction **E** prend en entrée un bloc de 32 bits et donne en sortie un bloc de 48 bits en utilisant la table de sélection suivante:

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

2.1.4. Fonction de chiffrement f (suite)

Le 1er bit du bloc de sortie contient le 32ème bit du bloc d'entrée;

Le 2ème bit du bloc de sortie contient le 1er bit du bloc d'entrée;…

Le 48ème bit du bloc de sortie contient le 1er bit du bloc d'entrée;

2.1.4. Fonction de chiffrement f (suite)

Ensuite, on fait **l'addition** ⊕ bit par bit modulo 2 entre le **bloc** de **sortie** de la fonction **E** (48 bits) et la **clé K_i** (48 bits) pour avoir en résultat un **nouveau bloc** de 48 bits.

Puis on fait appel aux **8 fonctions de sélection**. Chaque fonction **S_i**, prend **6 bits** du bloc précédent, et donne **4 bits** en résultat pour avoir un bloc de **32 bits**.

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection **S_1** utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

On remarque que cette table contient des nombres de **0 à 15** (1111 en binaire).

Le bloc résultat de la fonction de sélection **S_1** est $S_1[i,j]$ tel que:

i: est formé par le **1er** et **dernier** bits du bloc d'entrée;

j: est formé par les **4 bits du milieu** du bloc d'entrée;

$S_1[i,j]$ donne un bloc de 4 bits.

Les autres fonctions de sélection S_2, S_3, S_4, S_5, S_6, S_7, S_8 utilisent d'autres tables.

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection **S_2** utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection **S_3** utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection S_4 utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection S_5 utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection S_6 utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection S_7 utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

La fonction de sélection S_8 utilise la table suivante:

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

2.1.4. Fonction de chiffrement *f* (suite)

Le résultat des 8 fonctions de sélection est un bloc de 32 bits.

Ce bloc est l'entrée d'une **fonction de permutation P** qui utilise la table:

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

2.1.4. Fonction de chiffrement *f* (suite)

Le 16ème bit du bloc d'entrée représente le 1er bit du bloc de sortie; …

Le 25ème bit du bloc d'entrée représente le 32ème bit du bloc de sortie.

2.1.5. Permutation inverse (IP^{-1})

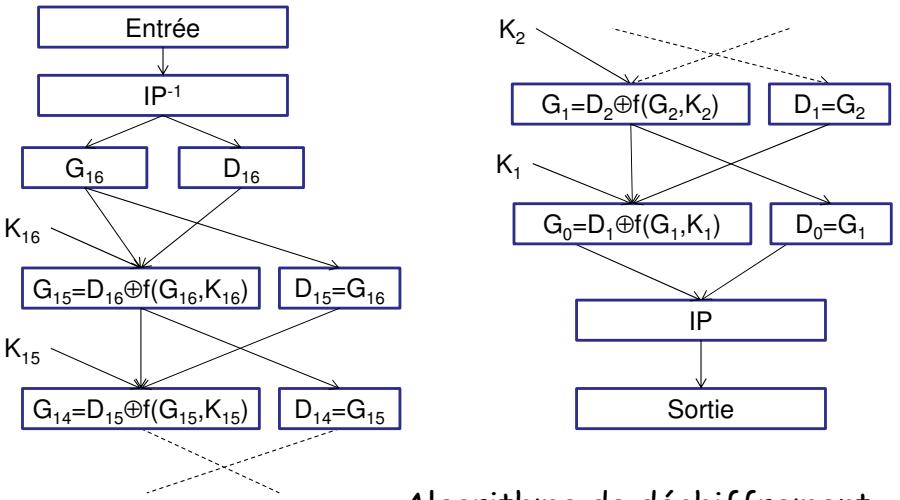
La permutation inverse prend en entrée un bloc de 64 bits et donne en sortie un bloc de 64 bits. Ça se fait selon le bloc suivant:

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

2.2. Algorithme de déchiffrement DES

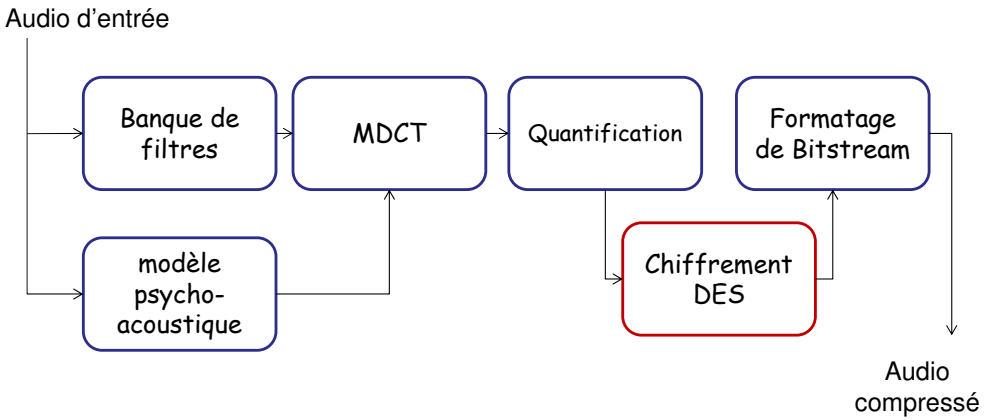
L'algorithme de déchiffrement est l'opération inverse de l'algorithme de chiffrement.

Chaque itération doit utiliser la **même** clé K_n.



Algorithme de déchiffrement

3. Chiffrement audio total en utilisant DES



PLAN

1. Introduction
2. Algorithme de chiffrement
3. Chiffrement audio total en utilisant AES
4. Problématique
5. Chiffrement audio sélectif en utilisant AES

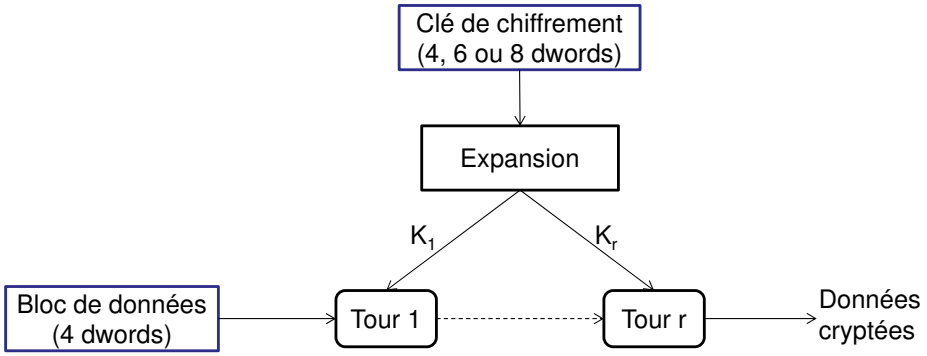
1. Introduction

AES (Advanced Encryption Standard) est un **algorithme** de chiffrement/déchiffrement. Il traite un bloc de **128 bits (4 dwords)** en utilisant une **clé** de **128 bits (4 dwords), 192 bits (6 dwords)** ou **256 bits (8 dwords)**.

Nb: 1 dword = 2 word = 4 octets

2. Algorithme de chiffrement

Le schéma générale de l'algorithme AES:



2. Algorithme de chiffrement (suite)

Le nombre de tours 'r' est calculé suivant l'algorithme du Belge RIJNDAEL comme suit:

		Bloc		
		128 bits (4 dwords)	192 bits (6 dwords)	256 bits (8 dwords)
Clé	128 bits (4 dwords)	10	12	14
	192 bits (6 dwords)	12	12	14
	256 bits (8 dwords)	14	14	14

Pour l'AES on n'utilise que les blocs de '128 bits'

2.1. L'expansion

Cette étape permet de générer le (Key Schedule) à utiliser dans les différents tours.

L'algorithme de l'expansion est divisé en plusieurs étapes. Nous allons voir le cas de chiffrement de blocs de 128 bits (4 dwords):

- Récupérer la clé de chiffrement **K** sous forme de 4 colonnes de 4 octets chacune.

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

2.1. L'expansion (suite)

- Pour construire les clés K_1 jusqu'à K_r pour les **r** tours, nous avons besoin d'utiliser deux tables: **Rcon** et **S-Box**.

2^i	01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00

Rcon

2.1. L'expansion (suite)

La table S-Box est basée sur le produit matriciel:

$$\begin{bmatrix} S'_0 \\ S'_1 \\ S'_2 \\ S'_3 \\ S'_4 \\ S'_5 \\ S'_6 \\ S'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

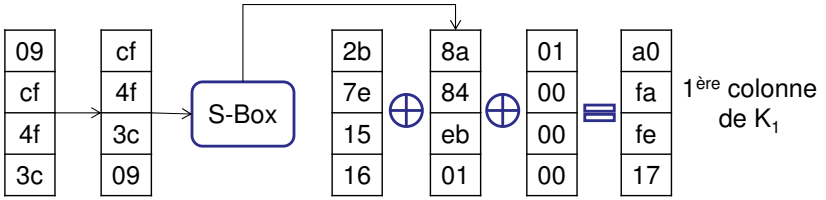
2.1. L'expansion (suite)

La table S-Box est comme suit:

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

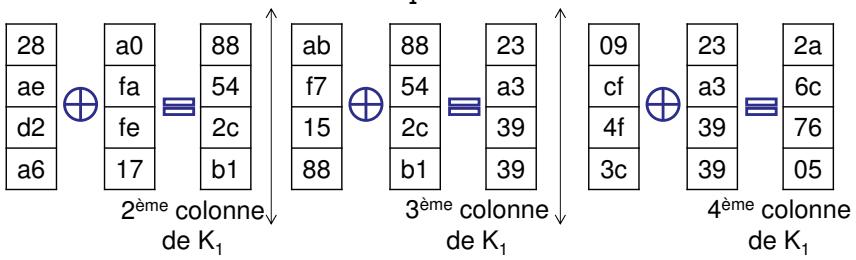
2.1. L'expansion (suite)

- Pour construire la 1ère colonne de la clé K_1 du **tour$_1$**, on utilise la 4ème colonne de la clé K, on fait une **rotation** vers le haut de 1 octet, puis on applique une transformation en utilisant la table **S-Box**, puis on applique le **XOR** entre la 1ère colonne de K, la colonne transformée et la 1ère colonne de Rcon.



2.1. L'expansion (suite)

- Pour construire les 2ème, 3ème et 4ème colonnes de la clé K_1 du **tour$_1$**, on fait le **XOR** respectivement de la 2ème colonne de K avec la 1ère colonne de K_1, la 3ème colonne de K avec la 2ème colonne de K_1 et la 4ème colonne de K avec la 3ème colonne de K_1 :

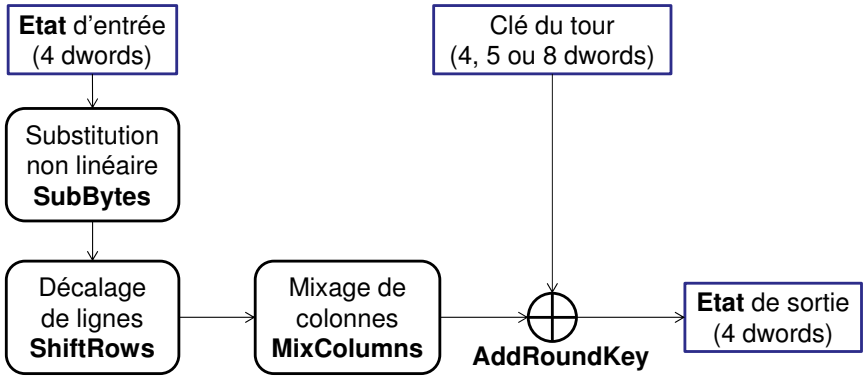


2.1. L'expansion (suite)

- Puis on boucle de la même manière pour calculer les autres K_i. On utilise la clé K_{i-1} pour calculer la clé K_i.

2.2. Les tours

Un tour peut être schématisé comme suit:



2.2.1. L'état

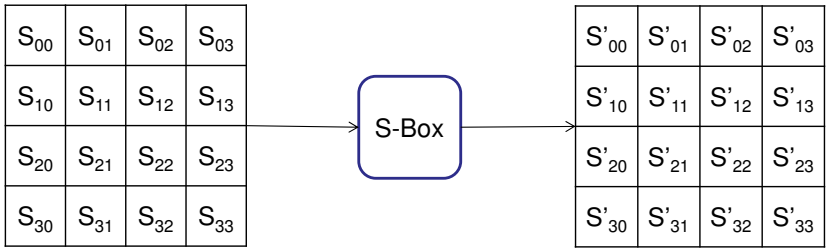
Les opérations de l'algorithme AES sont basés sur un tableau d'octets en 2 dimensions appelé **Etat** (**State**) qui est notée **S**.

L'état est une matrice de **4 colonnes** de **4 octets** chacune (= taille du bloc (128)/32).

S_{00}	S_{01}	S_{02}	S_{03}
S_{10}	S_{11}	S_{12}	S_{13}
S_{20}	S_{21}	S_{22}	S_{23}
S_{30}	S_{31}	S_{32}	S_{33}

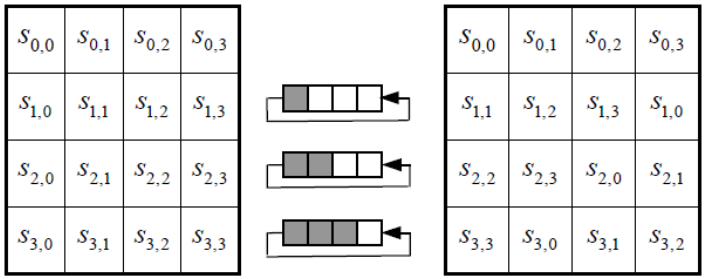
2.2.2. Substitution non linéaire `SubBytes`

Cette étape utilise la table **S-Box** pour transformer l'état d'entrée.



2.2.3. Décalage de lignes `ShiftRows`

Les lignes de l'état **S** seront décalées comme suit:



2.2.4. Mixage de colonnes `MixColumns`

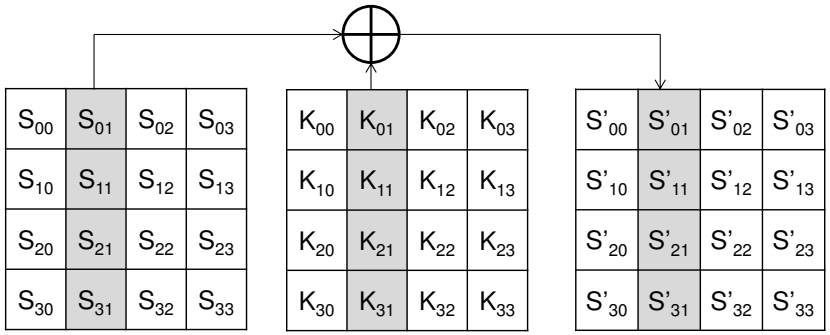
Dans cette étape on utilise le produit matriciel:

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

Avec $0 \leq c <$ taille du bloc en dwords (dans notre cas = 4)

2.2.5. Ajout de la clé de tour `AddRoundKey`

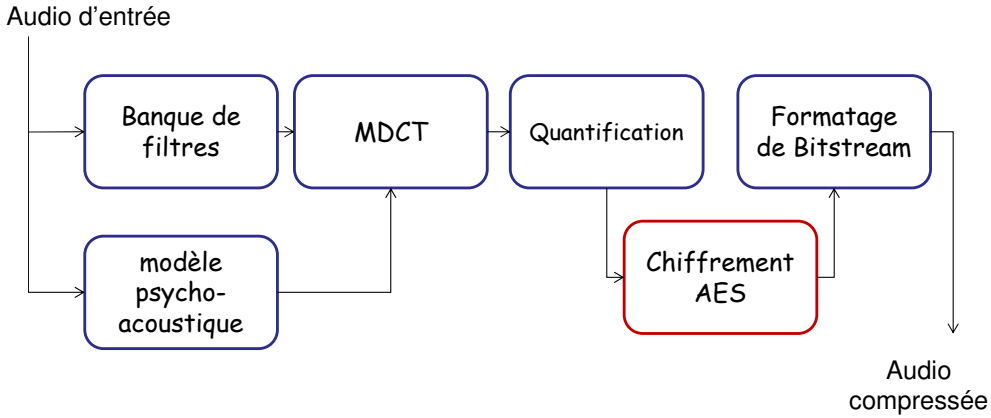
Dans cette étape, chaque **colonne** de l'état **S** est ajoutée à la **colonne** correspondante de la **clé du tour K** par un **XOR**.



3. Chiffrement audio total en utilisant AES

Pour appliquer la technique de chiffrement AES sur un fichier audio compressé en **MP3**, on va chiffrer les données **quantifiées** comme indiqué sur le schéma suivant:

3. Chiffrement audio total en utilisant AES



4. Problématique

L'application du chiffrement audio total à une séquence audio MP3 prend un **temps** considérable et **ralentit** le système.

Pour **éviter** ce problème, il a été proposé une technique de **chiffrement partiel** pour réduire le temps de traitement.

5. Chiffrement audio sélectif en utilisant AES

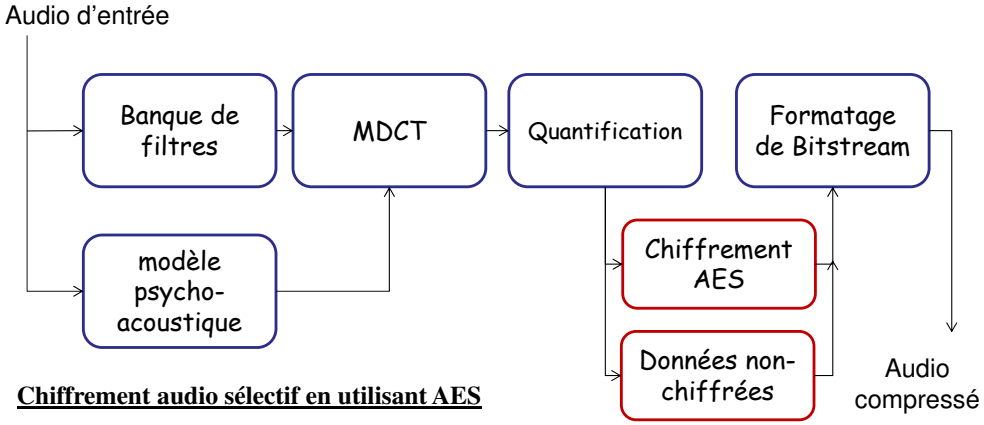
Avec cette technique, la chiffrement se fait **lors** de la **compression** en MP3.

Le chiffrement s'applique aux données **quantifiées**. Les blocs qui sont dans les **positions paires** seront **chiffrés** et les autres resteront non chiffrés.

Après le chiffrement de la partie sélectionnées les données chiffrées seront **placées** dans leurs positions d'origine.

CHAPITRE III

CHIFFREMENT AUDIO EN UTILISANT AES



Chiffrement audio sélectif en utilisant AES

CHAPITRE IV

CHIFFREMENT AUDIO SHUFFLE

PLAN

1. Introduction
2. Algorithme audio shuffle
   a. Algorithme de chiffrement
   b. Algorithme de déchiffrement

CHAPITRE IV

CHIFFREMENT AUDIO SHUFFLE

1. Introduction

Dans cet algorithme, le chiffrement est dépendant d'une **clé privée** et des **données**.

Il reçoit en entrée les données audio et une clé privée.

L'algorithme est proposé en 2014 par Abdelfatah A. Tamimi and Ayman M. Abdalla.

CHAPITRE IV

CHIFFREMENT AUDIO SHUFFLE

2. Algorithme audio shuffle

L'algorithme de chiffrement audio shuffle est divisé en deux grandes partie: chiffrement et déchiffrement.

2.1. Algorithme de chiffrement

L'algorithme de chiffrement audio shuffle est divisé en plusieurs étapes dans une boucle. Le nombre d'itération **k** de la boucle est égale au nombre d'octets de la clé.

Pour chaque itération i:

- Un **fixBit** est calculé par une fonction de hachage [**fixBit=Hash(key,i)**], par exemple le **i^{ème} octet** de la clé **modulo** 8, tel que '**i**' est le numéro de l'itération;
- On construit un vecteur S_0 contenant les **numéros** des octets de données qui ont le bit numéro **fixBit = 0**;

2.1. Algorithme de chiffrement (suite)

- On construit un vecteur S_1 contenant les **numéros** des octets de données qui ont le bit numéro **fixBit = 1**;
- Les deux vecteurs seront concaténés pour construire un vecteur Shuffle = S_0S_1;
- Puis on reconstruit les octets avec leurs nouvelles places suivant le vecteur Shuffle, tel que le i^{ème} octet se met dans la place numéro Shuffle[i];

On refait toutes les étapes **k** fois.

2.1. Algorithme de chiffrement (suite)

Exp.

Soit la suite d'octets de données suivante:

F5 2D A3 8C

Et soit la clé privée: 03 05

Les données en binaires:

11110101 00101101 10100011 10001100

fixBit = 3

S_0 = [1, 3]

S_1 = [2, 4]

2.1. Algorithme de chiffrement (suite)

Exp. (suite)

Shuffle = [1, 3, 2, 4]

Reconstruction des octets de données:

11110101, 10100011, 00101101, 10001100

F5 A3 2D 8C

On boucle pour la 2ème itération:

fixBit = 5

S_0 = [4]

S_1 = [1, 2, 3]

2.1. Algorithme de chiffrement (suite)

Exp. (suite)

Shuffle = [4, 1, 2, 3]

Reconstruction des octets de données:

10100011, 00101101, 10001100, 11110101

A3 2D 8C F5

2.2. Algorithme de déchiffrement

L'algorithme de déchiffrement audio shuffle est l'opération inverse de chiffrement. Le nombre d'itération **k** de la boucle est le nombre d'octets de la clé:

- Un **fixBit** est calculé par une fonction de hachage [**fixBit=Hash(key,i)**], par exemple le **$i^{ème}$ octet** de la clé **modulo** 8, tel que '**i**' est le numéro de l'itération;
- On construit un vecteur S_0 contenant les **numéros** des octets de données qui ont le bit numéro **fixBit = 0**;

2.2. Algorithme de déchiffrement (suite)

- On construit un vecteur S_1 contenant les **numéros** des octets de données qui ont le bit numéro **fixBit = 1**;
- Les deux vecteurs seront concaténés pour construire le vecteur Shuffle = S_0S_1;
- Puis on reconstruit les octets de l'audio originale en utilisant le vecteur Shuffle tel que l'octet numéro Shuffle[i] se met dans la $i^{ème}$ place;
- On refait toutes les étapes **k** fois.

2.2. Algorithme de déchiffrement (suite)

Exp.

On va essayer de déchiffrer la séquence d'octet chiffrée dans l'exemple précédent.

A3 2D 8C F5

Avec la même clé privée: 03 05

Les données en binaires:

10100011 00101101 10001100 11110101

fixBit = 5

S_0 = [3]

S_1 = [1, 2, 4]

2.2. Algorithme de déchiffrement (suite)

Exp. (suite)

Shuffle = [3, 1, 2, 4]

Reconstruction des octets de données:

11110101 10100011 10001100 00101101

F5 A3 8C 2D

On boucle pour la 2ème itération:

fixBit = 3

S_0 = [2, 3, 4]

S_1 = [1]

2.2. Algorithme de déchiffrement (suite)

Exp. (suite)

Shuffle = [2, 3, 4, 1]

Reconstruction des octets de données:

10100011 10001100 00101101 11110101

A3 8C 2D F5

F5 2D A3 8C