



FIPS 197

Introduction

Éléments importants

Structure du document

Définitions

Glossaire des termes et des définitions

Paramètres, symboles et fonctions de l'algorithme

Notation et conventions

Entrées et sorties

Octets

Tableaux d'octets

L'état intermédiaire de chiffrement (State)

Représentation de l'état intermédiaire comme un tableau de colonnes

Synthèse

Concepts mathématiques

Addition

Multiplication

Cas général

Multiplication par x

Polynômes à coefficients dans $GF(2^8)$

Spécification de l'algorithme

Chiffrement

SubBytes()

ShiftRows()

MixColumns()

AddRoundKey()

Résumé

[Expansion de la clé](#)

[Processus d'expansion de la clé](#)

[Pseudo-code pour l'expansion de clé](#)

Introduction



Le rapport FIPS 197 décrit le standard AES (Advanced Encryption Standard), qui est un algorithme de chiffrement symétrique par blocs

Éléments importants

L'algorithme AES est basé sur l'algorithme Rijndael

Il traite des blocs de données de 128 bits et utilise des clés de chiffrement de 128, 192 et 256 bits

L'algorithme Rijndael a été conçu pour gérer des tailles de blocs et de clés supplémentaires, mais ces options ne sont pas adoptées dans le standard AES

Les différentes versions d'AES sont désignées comme AES-128, AES-192 et AES-256, en fonction de la longueur de clé utilisée

Structure du document

1. Définitions des termes, acronymes, paramètres, symboles et fonctions de l'algorithme
2. Notations et conventions utilisées dans la spécification de l'algorithme, y compris l'ordre et la numérotation des bits, octets et mots
3. Propriétés mathématiques utiles pour comprendre l'algorithme
4. Spécification de l'algorithme, couvrant l'expansion de la clé, le chiffrement et le déchiffrement
5. Questions liées à l'implémentation, telles que le support de la longueur de clé, les restrictions de clé et les tailles de blocs, de clés et de tours supplémentaires

Enfin, le rapport se termine par plusieurs annexes contenant des exemples détaillés pour l'expansion de la clé et le chiffrement, des exemples de vecteurs pour le chiffrement et le déchiffrement inverse, ainsi qu'une liste de références

Définitions

Pour cette partie, on propose seulement une traduction de la norme, n'ayant pas d'éléments importants à y ajouter

Glossaire des termes et des définitions

Voici les définitions utilisées tout au long de cette norme :

AES : Advanced Encryption Standard (Standard de chiffrement avancé).

Transformation affine : Une transformation consistant en une multiplication par une matrice suivie de l'addition d'un vecteur.

Tableau (array) : Une collection énumérée d'entités identiques (par exemple, un tableau d'octets).

Bit : Un chiffre binaire ayant une valeur de 0 ou 1.

Bloc (block) : Séquence de bits binaires qui constituent l'entrée, la sortie, l'état et la clé de tour. La longueur d'une séquence est le nombre de bits qu'elle contient. Les blocs sont également interprétés comme des tableaux d'octets.

Octet : Un groupe de huit bits traité soit comme une seule entité, soit comme un tableau de 8 bits individuels.

Chiffrement (Cipher) : Série de transformations qui convertit le texte en clair en texte chiffré à l'aide de la clé de chiffrement.

Clé de chiffrement (Cipher Key) : Clé cryptographique secrète utilisée par la routine d'expansion de clé pour générer un ensemble de clés de tour ; peut être représentée comme un tableau rectangulaire d'octets, ayant quatre rangées et Nk colonnes.

Texte chiffré (Ciphertext) : Données en sortie du chiffrement ou en entrée du déchiffrement inverse.

Déchiffrement inverse (inverse Cipher) : Série de transformations qui convertit le texte chiffré en texte en clair à l'aide de la clé de chiffrement.

Expansion de clé : Routine utilisée pour générer une série de clés de tour à partir de la clé de chiffrement.

Texte en clair : Données en entrée du chiffrement ou en sortie du déchiffrement inverse.

Rijndael : Algorithme cryptographique spécifié dans ce standard de chiffrement avancé (AES).

Clé de tour (Round key) : Les clés de tour sont des valeurs dérivées de la clé de chiffrement à l'aide de la routine d'expansion de clé ; elles sont appliquées à l'état dans le chiffrement et le déchiffrement inverse.

État (State) : Résultat intermédiaire du chiffrement pouvant être représenté comme un tableau rectangulaire d'octets, ayant quatre rangées et Nb colonnes.

S-box : Table de substitution non linéaire utilisée dans plusieurs transformations de substitution d'octets et dans la routine d'expansion de clé pour effectuer une substitution un-à-un d'une valeur d'octet.

Mot (Word) : Un groupe de 32 bits traité soit comme une seule entité, soit comme un tableau de 4 octets.

Paramètres, symboles et fonctions de l'algorithme

Les paramètres, symboles et fonctions suivants sont utilisés tout au long de cette norme :

AddRoundKey() : Transformation dans le chiffrement et le déchiffrement inverse où une clé de tour est ajoutée à l'état à l'aide d'une opération XOR. La longueur d'une clé de tour est égale à la taille de l'état (c'est-à-dire, pour Nb = 4, la longueur de la clé de tour est égale à 128 bits/16 octets).

MixColumns() : Transformation dans le chiffrement qui prend toutes les colonnes de l'état et mélange leurs données (indépendamment les unes des autres) pour produire de nouvelles colonnes.

InvMixColumns() : Transformation dans le déchiffrement inverse qui est l'inverse de MixColumns().

ShiftRows() : Transformation dans le chiffrement qui traite l'état en décalant cycliquement les trois dernières rangées de l'état avec des décalages différents.

InvShiftRows() : Transformation dans le déchiffrement inverse qui est l'inverse de ShiftRows().

SubBytes() : Transformation dans le chiffrement qui traite l'état à l'aide d'une table de substitution non linéaire d'octets (S-box) qui opère sur chacun des octets de l'état indépendamment.

InvSubBytes() : Transformation dans le déchiffrement inverse qui est l'inverse de SubBytes().

K : Clé de chiffrement.

Nb : Nombre de colonnes (mots de 32 bits) composant l'état. Pour cette norme, Nb = 4.

Nk : Nombre de mots de 32 bits composant la clé de chiffrement. Pour cette norme, Nk = 4, 6 ou 8.

Nr : Nombre de tours, qui est une fonction de Nk et Nb (qui est fixé). Pour cette norme, Nr = 10, 12 ou 14.

Rcon[] : Le tableau de mots constants de tour.

RotWord() : Fonction utilisée dans la routine d'expansion de clé qui prend un mot de quatre octets et effectue une permutation cyclique.

Subword() : Fonction utilisée dans la routine d'expansion de clé qui prend un mot d'entrée de quatre octets et applique une S-box à chacun des quatre octets pour produire un mot de sortie.

XOR : Opération OU exclusif.

\oplus : Opération OU exclusif.

\otimes : Multiplication de deux polynômes (chacun de degré < 4) modulo $x^4 + 1$.

\bullet : Multiplication dans un corps fini.

Notation et conventions

Entrées et sorties

L'algorithme AES utilise des séquences de 128 bits pour les entrées et sorties, appelées blocs, et une clé de chiffrement de 128, 192 ou 256 bits. Aucune autre longueur n'est autorisée par cette norme. Les bits dans les séquences sont numérotés de 0 à un de moins que la longueur de la séquence et sont appelés indices, qui se situent dans des plages spécifiques en fonction de la longueur du bloc et de la clé.

Octets

L'unité de base pour le traitement dans l'algorithme AES est un octet (**byte**), composé de huit bits traités comme une seule entité. Les séquences d'entrée, de sortie et de clé de chiffrement sont traitées comme des tableaux d'octets, en divisant ces séquences en groupes de huit bits contigus. Les octets dans le tableau sont référencés par a_n ou $a[n]$, où n varie en fonction de la longueur de la clé et du bloc.

Les valeurs des octets dans l'algorithme AES sont présentées comme la concaténation de leurs valeurs de bits individuelles, et sont interprétées comme des éléments de corps fini à l'aide d'une représentation polynomiale. Par exemple, {01100011} identifie l'élément de corps fini $x^6 + x^5 + x + 1$.

Il est également pratique d'utiliser la notation hexadécimale pour représenter les valeurs des octets, chaque groupe de quatre bits étant représenté par un seul caractère. Ainsi, l'élément {01100011} peut être représenté par {63}.

Certaines opérations de corps fini impliquent un bit supplémentaire (b_8) à gauche d'un octet de 8 bits. Lorsque ce bit supplémentaire est présent, il apparaît comme '{01}' immédiatement avant l'octet de 8 bits.

En résumé, dans l'algorithme AES, l'unité de base est un octet, composé de huit bits. Les séquences d'entrée, de sortie et de clé de chiffrement sont traitées comme des tableaux d'octets. Les valeurs des octets sont interprétées comme des éléments de corps fini à l'aide d'une représentation polynomiale. La notation hexadécimale est souvent utilisée pour représenter les valeurs des octets, facilitant ainsi la représentation.

Tableaux d'octets

Les octets sont constitués de groupes de 8 bits consécutifs de la séquence d'entrée. Par exemple, a_0 contient les bits 0 à 7, a_1 contient les bits 8 à 15, et ainsi de suite. La même logique s'applique aux clés de 192 et 256 bits. La formule générale pour obtenir l'octet a_n est :

$$a_n = \text{input}_{8n}, \text{input}_{8n+1}, \dots, \text{input}_{8n+7}$$

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Byte number	0								1								2								...
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

La figure 2 montre comment les bits à l'intérieur de chaque octet sont numérotés, en considérant les sections 3.2 et 3.3 ensemble.

L'état intermédiaire de chiffrement (State)

On explique ici le fonctionnement interne de l'algorithme AES avec un tableau à deux dimensions appelé "State"

Le tableau *State* est composé de quatre rangées d'octets, chacune contenant Nb octets, où Nb est la longueur du bloc divisée par 32. Pour cette norme, $Nb = 4$.

Chaque octet individuel dans le tableau *State* possède deux indices : le numéro de ligne r avec ($0 \leq r < 4$) et le numéro de colonne c avec ($0 \leq c < Nb$). Ainsi, un octet individuel du tableau *State* peut être référencé sous la forme $s_{r,c}$ ou $s[r, c]$.

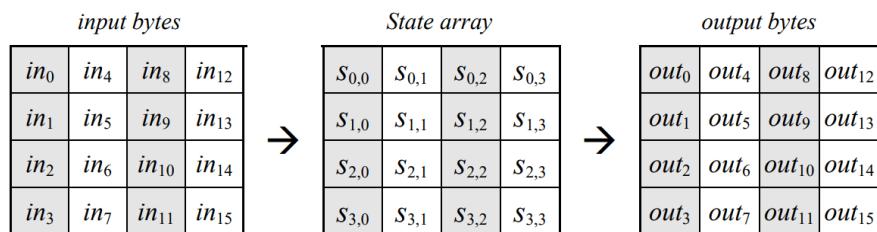


Figure 3. State array input and output.

Au début de l'algorithme, les octets d'entrée sont copiés dans le tableau *State* comme illustré dans la figure 3. Les opérations de chiffrement ou de déchiffrement sont ensuite effectuées sur ce tableau *State*, puis sa valeur finale est copiée dans les octets de sortie.

L'entrée est copiée dans le tableau *State* selon le schéma suivant :

$$s[r, c] = in[r + 4c] \text{ pour } 0 \leq r < 4 \text{ et } 0 \leq c < Nb,$$

À la fin du chiffrement ou du déchiffrement, le tableau *State* est copié dans le tableau de sortie *out* de la manière suivante :

$$out[r + 4c] = s[r, c] \text{ pour } 0 \leq r < 4 \text{ et } 0 \leq c < Nb.$$

En bref, l'algorithme AES fonctionne en interne en utilisant un tableau à deux dimensions appelé *State*. Le tableau est composé de 4 rangées et de Nb colonnes d'octets. Les opérations de chiffrement et de déchiffrement sont effectuées sur le tableau *State*. Les octets d'entrée sont d'abord copiés dans le tableau *State*, puis les opérations sont réalisées, et enfin, le contenu du tableau *State* est copié dans les octets de sortie.

Représentation de l'état intermédiaire comme un tableau de colonnes

Il est expliqué que les quatre octets de chaque colonne du tableau "State" forment des mots ([words](#)) de 32 bits, où le numéro de rangée r sert d'indice pour les quatre octets de chaque mot. Ainsi, l'état peut être interprété comme un tableau unidimensionnel de mots de 32 bits (colonnes), $w_0 \dots w_3$, où le numéro de colonne c fournit un indice pour ce tableau. Par exemple, en se basant sur la figure 3, le *State* peut être considéré comme un tableau de quatre mots, comme suit :

$$w_0 = s_{0,0} s_{1,0} s_{2,0} s_{3,0}$$

$$w_1 = s_{0,1} s_{1,1} s_{2,1} s_{3,1}$$

$$w_2 = s_{0,2} s_{1,2} s_{2,2} s_{3,2}$$

$$w_3 = s_{0,3} s_{1,3} s_{2,3} s_{3,3}$$

Synthèse

La section sur les notations et conventions d'AES aborde plusieurs aspects clés du fonctionnement de l'algorithme AES :

1 Les entrées et sorties de l'algorithme AES sont des séquences de 128 bits, et la clé de chiffrement peut être de 128, 192 ou 256 bits. Les bits sont numérotés de 0 à un nombre inférieur à la longueur de la séquence

2 L'unité de base pour le traitement dans l'algorithme AES est un octet, qui est une séquence de huit bits traitée comme une seule entité. Les octets sont représentés en notation hexadécimale pour plus de commodité

3 Les séquences de bits sont traitées comme des tableaux d'octets, en divisant les séquences en groupes de huit bits contigus. Les tableaux sont indexés par des nombres entiers dans des plages spécifiques en fonction de la longueur de la clé

4

L'algorithme AES effectue des opérations sur un tableau bidimensionnel d'octets appelé *State*. Le tableau *State* est composé de quatre rangées d'octets, chacune contenant un nombre d'octets Nb , où Nb est la longueur du bloc divisée par 32. Les octets individuels dans le tableau *State* sont référencés par deux indices, le numéro de rangée r et le numéro de colonne c

5

Les quatre octets de chaque colonne du tableau *State* forment des mots (*words*) de 32 bits, où le numéro de rangée r sert d'indice pour les quatre octets de chaque mot. L'état peut être interprété comme un tableau unidimensionnel de mots de 32 bits (colonnes)

Ces conventions et notations sont essentielles pour comprendre le fonctionnement de l'algorithme AES et faciliter la mise en œuvre de cet algorithme de chiffrement.

Concepts mathématiques

Addition



On réalise une opération d'addition entre les polynômes grâce à l'opération XOR

Table de vérité du XOR (ou exclusif) :

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

L'addition est une opération effectuée dans le cadre de la transformation [AddRoundKey](#), qui a lieu à chaque tour de l'algorithme, à la fois lors du chiffrement et du déchiffrement. L'addition dans AES est définie comme l'opération "ou exclusif" (XOR) bit à bit sur les octets (bytes) des données en entrée.

L'addition XOR est une opération commutative, ce qui signifie que l'ordre dans lequel les bits sont ajoutés n'a pas d'importance. Elle est également associative, ce qui signifie que l'ordre dans lequel les bits sont ajoutés peut être modifié sans changer le résultat final.

Il est important de noter que l'opération XOR est sa propre inverse, c'est-à-dire que si $A \text{ XOR } B = C$, alors $A = C \text{ XOR } B$ et $B = A \text{ XOR } C$. Cette propriété est utile pour le déchiffrement dans AES, car elle permet de retrouver facilement l'état précédent à partir de l'état chiffré, en utilisant simplement l'opération XOR avec la même clé de tour.

Multiplication

Cas général

La multiplication est utilisée principalement dans la transformation [MixColumns](#), qui a lieu lors de chaque tour de l'algorithme, à l'exception du dernier, lors du processus de chiffrement. Elle est également utilisée dans la transformation inverse [InvMixColumns](#) lors du déchiffrement.

Dans AES, la multiplication est définie comme une opération sur les éléments d'un corps fini appelé $GF(2^8)$, ou corps de Galois de 2^8 éléments.

Corps fini : ensemble fini d'éléments avec des opérations d'addition, de soustraction, de multiplication et de division. Dans le cas de $GF(2^8)$, les éléments sont des polynômes de degré inférieur à 8 avec des coefficients binaires (0 ou 1), et les opérations sont définies modulo un polynôme irréductible de degré 8.

La multiplication dans le champ de Galois $GF(2^8)$ est réalisée en utilisant une multiplication polynomiale. Chaque octet est considéré comme un polynôme de degré 7 dont les coefficients sont soit 0, soit 1. La multiplication de deux octets est alors réalisée en multipliant les deux polynômes correspondants, puis en réduisant le résultat modulo un polynôme irréductible de degré 8.

La réduction modulo un polynôme irréductible de degré 8 permet de ramener le résultat de la multiplication à un octet de 8 bits.

Polynôme irréductible : $x^8 + x^4 + x^3 + x + 1$

En résumé, cette opération est réalisée en multipliant deux octets considérés comme des polynômes, puis en réduisant le résultat modulo un polynôme irréductible de degré 8.

Multiplication par x

La multiplication par x dans $GF(2^8)$ est effectuée en multipliant un élément (représenté par un polynôme) par le polynôme x , puis en réduisant le résultat modulo le polynôme irréductible défini pour AES, qui est $x^8 + x^4 + x^3 + x + 1$.

En termes d'opérations sur les octets, la multiplication par x peut être réalisée en effectuant un décalage vers la gauche de l'octet, puis en effectuant une opération XOR avec le polynôme irréductible (sous forme d'octet) si le bit le plus à gauche (bit de poids fort) de l'octet initial est 1. Cette opération XOR est effectuée pour garantir que le résultat reste un polynôme de degré inférieur à 8.

Exemple :

1. Supposons que l'élément à multiplier par x soit représenté par l'octet $A = 10011011$.
2. On effectue un décalage vers la gauche : 00110110 .
3. Comme le bit le plus à gauche de A est 1, on effectue un XOR avec le polynôme irréductible (sous forme d'octet) : $00110110 \oplus 00011011 = 00101101$.

Cette multiplication par x est utilisée à plusieurs reprises dans l'algorithme AES, notamment dans les transformations `MixColumns`, `InvMixColumns` et la génération des tables de substitution pour la transformation `SubBytes`.

Polynômes à coefficients dans $GF(2^8)$

Dans cette section, il est expliqué que les polynômes à coefficients dans $GF(2^8)$ sont des polynômes dont les coefficients sont des éléments du corps fini $GF(2^8)$. Les opérations d'addition, de soustraction, de multiplication et de division sur ces polynômes sont définies en utilisant les opérations correspondantes dans $GF(2^8)$. Par exemple, pour additionner deux polynômes à coefficients dans $GF(2^8)$, on additionne les coefficients correspondants en utilisant l'opération XOR, qui est l'addition dans $GF(2^8)$.

Un aspect important de cette section est la représentation des données et des clés de l'algorithme AES sous forme de polynômes à coefficients dans $GF(2^8)$. Le bloc de données (State) et la clé de chiffrement sont représentés sous forme de matrices 4x4 d'octets, où chaque octet correspond à un élément de $GF(2^8)$. Chaque colonne de la matrice d'état peut être interprétée comme un polynôme de degré 3 avec des coefficients dans $GF(2^8)$, et chaque colonne de la matrice de clé peut être interprétée comme un polynôme de degré 3 avec des coefficients dans $GF(2^8)$.

Dans la transformation `MixColumns`, chaque colonne de la matrice d'état est traitée comme un polynôme de degré 3 sur $GF(2^8)$, et est multipliée par un polynôme fixe

$a(x) = 3x^3 + x^2 + x + 2$. Cette multiplication a pour effet de mélanger les octets dans les colonnes de la matrice d'état, renforçant ainsi la diffusion des données dans le processus de chiffrement. Lors du déchiffrement, la transformation **InvMixColumns** utilise la multiplication par le polynôme inverse de $c(x)$, c'est-à-dire $a^{-1}(x) = 11x^3 + x^2 + 9x + 14$.

Spécification de l'algorithme

Dans l'AES, la taille des blocs d'entrée, de sortie et de l'état interne est de 128 bits. Cela est représenté par $Nb = 4$, qui correspond au nombre de mots de 32 bits (nombre de colonnes) dans l'état interne.

La taille de la clé de chiffrement, Nk , est de 128, 192 ou 256 bits. La taille de la clé est représentée par $Nk = 4, 6$ ou 8 , qui correspondent au nombre de mots de 32 bits (nombre de colonnes) dans la clé de chiffrement. Le nombre de tours effectués pendant l'exécution de l'algorithme dépend de la taille de la clé. Le nombre de tours est représenté par Nr .

	Key Length (Nk words)	Block Size (Nb words)	Number of Rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figure 4. Key-Block-Round Combinations.

Pour le chiffrement et le déchiffrement (**Cipher** et **Inverse Cipher**), l'algorithme AES utilise une fonction de tour composée de quatre transformations orientées octets :

1. La substitution d'octets à l'aide d'une table de substitution (**S-Box**).
2. Le décalage des rangées du tableau d'état avec des décalages différents.
3. Le mélange des données à l'intérieur de chaque colonne du tableau d'état.
4. L'ajout d'une clé de tour (**Round Key**) à l'état (**state**)

Donc au final, l'algorithme AES utilise des blocs de 128 bits et des clés de 128, 192 ou 256 bits. Le nombre de tours dépend de la taille de la clé (10, 12 ou 14 tours). L'AES implémente des transformations spécifiques pour le chiffrement et le déchiffrement, notamment la substitution d'octets, le décalage des rangées, le mélange des colonnes et l'ajout d'une clé de tour. Les sections ultérieures décrivent

en détail ces transformations et les étapes du chiffrement, du déchiffrement et de la programmation des clés.

Chiffrement

Le chiffrement débute par la copie de l'entrée dans le tableau d'état (`State`) selon les conventions établies dans la section 3.4. Après une première addition de la clé de tour (`Round Key`), le tableau d'état subit une série de transformations en appliquant une fonction de tour 10, 12 ou 14 fois, selon la taille de la clé. La dernière transformation diffère légèrement des Nr-1 premières. Le tableau d'état final est ensuite copié vers la sortie, comme décrit dans la section 3.4.

La fonction de tour est paramétrée à l'aide d'un programme de clés (`Key Schedule`), qui est un tableau unidimensionnel de mots de quatre octets, dérivé de la routine d'expansion de clés décrite dans la section 5.2.

Le processus de chiffrement est représenté par un pseudo-code dans la Figure 5. Les transformations individuelles - `SubBytes()`, `ShiftRows()`, `MixColumns()` et `AddRoundKey()` - sont appliquées au tableau d'état et décrites dans les sous-sections suivantes. Dans la Figure 5, le tableau `w[]` contient le programme de clés, décrit dans la section 5.2.

Tous les tours Nr sont identiques, à l'exception du dernier tour qui n'inclut pas la transformation `MixColumns()`.

L'annexe B présente un exemple de chiffrement, montrant les valeurs du tableau d'état au début de chaque tour et après l'application des quatre transformations décrites dans les sections suivantes.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
```

En résumé, la partie 5.1 explique le processus de chiffrement de l'AES, avec ses différentes étapes et transformations. Les sections suivantes détaillent les méthodes de transformation et le pseudo-code fournit une représentation visuelle du processus de chiffrement.

SubBytes()

`SubBytes()` est la première étape du processus de chiffrement AES. `SubBytes()` est une substitution non-linéaire où chaque octet du tableau d'état est remplacé par un autre octet, selon une table de substitution préétablie appelée `S-Box`. La S-Box est une matrice de 16x16 contenant toutes les valeurs possibles d'un octet, et chaque élément de la matrice est obtenu à partir d'une transformation affine de l'inverse multiplicatif dans le corps fini $GF(2^8)$.

La `S-Box` est une matrice de 16x16 contenant tous les octets possibles, du 00 au FF en notation hexadécimale. La substitution est réalisée en remplaçant chaque octet du tableau d'état par l'octet correspondant dans la `S-Box`. Pour effectuer cette substitution, on décompose l'octet en deux chiffres hexadécimaux : le premier chiffre représente la ligne et le second représente la colonne de la S-Box. L'octet à l'intersection de cette ligne et colonne remplace alors l'octet d'origine dans le tableau d'état.

		y																
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
x		0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
x	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	

Par exemple, si un octet du tableau d'état est égal à "8A", on décompose cet octet en "8" (ligne) et "A" (colonne). On localise l'intersection de la ligne 8 et de la colonne A dans la S-Box, et on remplace l'octet "8A" par l'octet trouvé à cette position ("CD").

La transformation `SubBytes()` est conçue pour fournir de la confusion et de la complexité au processus de chiffrement, rendant ainsi difficile l'analyse et l'attaque du chiffrement. Cette substitution non-linéaire introduit une dissymétrie dans le chiffrement, empêchant les attaquants de déduire la clé à partir de la relation entre le texte clair et le texte chiffré.

En résumé, la transformation `SubBytes()` est une étape essentielle du chiffrement AES, où chaque octet du tableau d'état est remplacé par un autre octet en utilisant une table de substitution (`S-Box`) pour introduire de la confusion et de la complexité dans le processus de chiffrement.

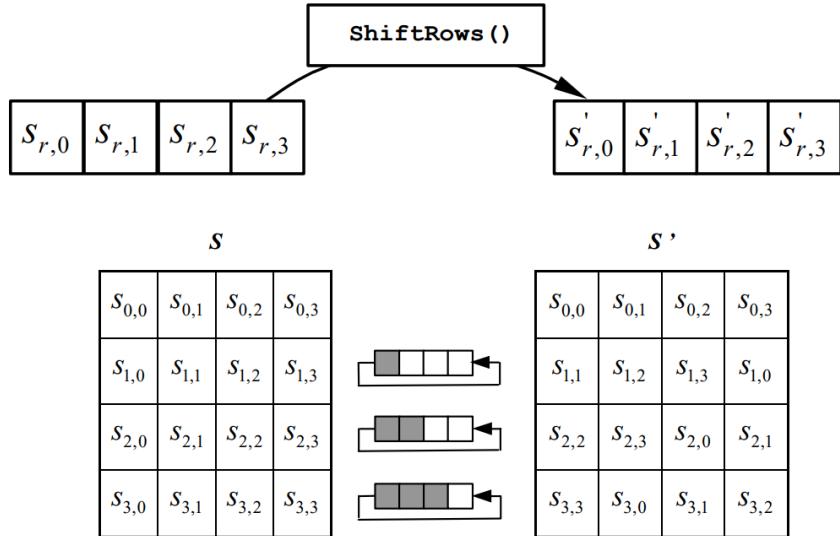
ShiftRows()

`ShiftRows()` est une opération de permutation qui agit sur les rangées du tableau d'état. Le but de cette transformation est de mélanger les octets entre les colonnes du tableau d'état pour assurer une diffusion des données.

La transformation `ShiftRows()` fonctionne de la manière suivante :

1. La première rangée (rangée 0) du tableau d'état n'est pas modifiée.
2. La deuxième rangée (rangée 1) est décalée circulairement d'une position vers la gauche.
3. La troisième rangée (rangée 2) est décalée circulairement de deux positions vers la gauche.
4. La quatrième rangée (rangée 3) est décalée circulairement de trois positions vers la gauche.

Le décalage circulaire signifie que les octets décalés hors de la rangée réapparaissent à l'autre extrémité de celle-ci. Par exemple, si une rangée était initialement [a, b, c, d], après un décalage circulaire d'une position vers la gauche, elle deviendrait [b, c, d, a].



La transformation `ShiftRows()` assure une diffusion des données à travers les colonnes du tableau d'état, ce qui contribue à la résistance de l'algorithme AES face aux attaques cryptanalytiques. Cette opération, combinée avec les autres transformations du chiffrement, rend difficile la détermination de la clé à partir des données chiffrées.

MixColumns()

`MixColumns()` est une opération de mélange des colonnes du tableau d'état, qui assure une diffusion supplémentaire des données dans les différentes colonnes.

La transformation `MixColumns()` consiste à traiter chaque colonne du tableau d'état comme un polynôme de degré 3 (on applique un modulo $x^4 + 1$) sur un corps fini ($GF(2^8)$) et à le multiplier par un polynôme fixe. Le polynôme fixe est défini comme $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Voici comment la transformation `MixColumns()` est réalisée pour chaque colonne du tableau d'état :

1. Chaque octet de la colonne est traité comme un coefficient d'un polynôme de degré 3.
2. Le polynôme de colonne est multiplié par le polynôme fixe défini ci-dessus.
3. Le résultat de la multiplication des polynômes remplace les octets originaux de la colonne du tableau d'état.

On a $s'(x) = a(x) \otimes s(x)$

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

Et on obtient

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

La multiplication des polynômes dans $GF(2^8)$ nécessite des opérations spécifiques, telles que la multiplication et l'addition modulo 2^8 , ainsi que le calcul de l'inverse multiplicatif. Ces opérations sont généralement effectuées en utilisant des tables de consultation pour plus d'efficacité.

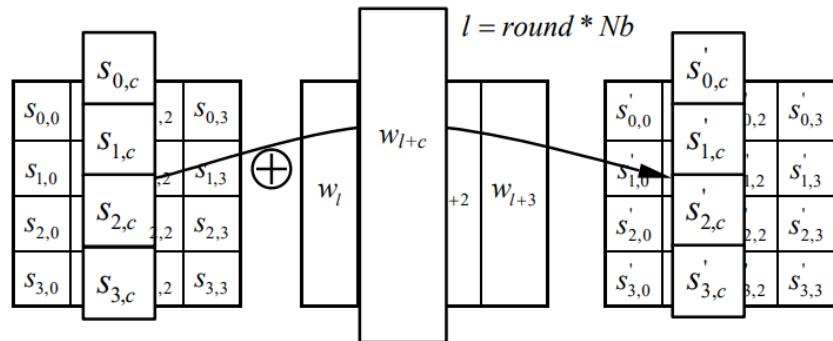
La transformation `MixColumns()` assure une diffusion supplémentaire des données à travers les colonnes du tableau d'état, renforçant ainsi la résistance de l'algorithme AES aux attaques cryptanalytiques. Cette opération, combinée aux autres transformations de chiffrement, rend difficile l'analyse des données chiffrées pour déterminer la clé secrète.

AddRoundKey()

`AddRoundKey()` est une opération simple mais cruciale pour la sécurité de l'algorithme, qui consiste à combiner le tableau d'état avec une clé de tour (`Round Key`) générée à partir du programme de clés (`Key Schedule`).

La transformation `AddRoundKey()` s'effectue de la manière suivante :

1. Chaque octet du tableau d'état est combiné avec l'octet correspondant de la clé de tour en utilisant l'opération XOR (ou exclusif).
2. Le résultat de l'opération XOR remplace l'octet d'origine dans le tableau d'état.



Il est important de noter que la transformation `AddRoundKey()` est son propre inverse. En d'autres termes, appliquer deux fois la même clé de tour à l'aide de l'opération `AddRoundKey()` annule l'effet de la première application, ce qui est crucial pour le processus de déchiffrement (`Inverse Cipher`).

La transformation `AddRoundKey()` garantit que la clé secrète est intégrée dans le processus de chiffrement à chaque tour, rendant ainsi extrêmement difficile l'analyse des données chiffrées pour déterminer la clé secrète. Cette opération, combinée aux autres transformations de chiffrement, assure la résistance de l'algorithme AES aux attaques cryptanalytiques.

Résumé

Les quatre transformations qui composent chaque tour de chiffrement dans l'algorithme AES sont les suivantes :

1 `SubBytes()` : Il s'agit d'une transformation de substitution non linéaire qui agit sur chaque octet du tableau d'état en utilisant une table de substitution appelée `S-Box`

2 `ShiftRows()` : Cette transformation consiste en une permutation des rangées du tableau d'état, où chaque rangée est décalée circulairement d'un certain nombre de positions vers la gauche

3 `MixColumns()` : c'est une opération de mélange des colonnes du tableau d'état, qui traite chaque colonne comme un polynôme sur un corps fini ($GF(2^8)$) et le multiplie par un polynôme fixe

4

`AddRoundKey()` : La dernière transformation consiste à combiner le tableau d'état avec une clé de tour (`Round Key`) générée à partir du programme de clés (`Key Schedule`) en utilisant l'opération \oplus

Expansion de la clé

L'algorithme AES prend la clé de chiffrement, K , et effectue une routine d'expansion de clé pour générer un programme de clés (`key schedule`). L'expansion de clé génère un total de $Nb(Nr + 1)$ mots : l'algorithme nécessite un ensemble initial de Nb mots, et chacun des tours Nr nécessite Nb mots de données de clé. Le calendrier de clés résultant se compose d'un tableau linéaire de mots de 4 octets, noté $[w_i]$, avec i dans l'intervalle $0 \leq i < Nb(Nr + 1)$.

Processus d'expansion de la clé

L'expansion de la clé initiale se fait selon un pseudo-code spécifique, qui utilise deux fonctions : `SubWord()` et `RotWord()`.

- `SubWord()` est une fonction qui prend un mot d'entrée de quatre octets et applique la `S-box` à chacun des quatre octets pour produire un mot de sortie.
- `RotWord()` est une fonction qui prend un mot $[a_0, a_1, a_2, a_3]$ en entrée, effectue une permutation cyclique, et renvoie le mot $[a_1, a_2, a_3, a_0]$.
- Le tableau de mots constant de round, $Rcon[i]$, contient les valeurs données par $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, où x^{i-1} sont les puissances de x (x est noté comme $\{02\}$) dans le champ $GF(2^8)$.

L'expansion de clé commence par copier la clé de chiffrement initiale dans les premiers Nk mots de la routine d'expansion de clé. Ensuite, pour chaque mot suivant, $w[i]$, on effectue un XOR entre le mot précédent, $w[i - 1]$, et le mot qui est Nk positions en arrière, $w[i - Nk]$.

Cependant, si la position actuelle i est un multiple de Nk , une étape supplémentaire est ajoutée avant le XOR. Le mot précédent $w[i - 1]$ est transformé par l'application de la fonction `RotWord()` suivie de la fonction `SubWord()`. Ensuite, un XOR est effectué entre le résultat de cette transformation et une constante de round, $Rcon[i]$.

Pour les clés de 256 bits ($Nk = 8$), il y a un cas supplémentaire à prendre en compte. Si la position actuelle $i - 4$ est un multiple de Nk , alors la fonction `SubWord()` est appliquée à $w[i - 1]$ avant de faire le XOR avec $w[i - Nk]$.

Pseudo-code pour l'expansion de clé

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
    word temp
    i = 0

    // Remplissage initial de la routine d'expansion de clé avec la clé de chiffrement
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1

    i = Nk

    // Expansion de la clé
    while (i < Nb * (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        w[i] = w[i-Nk] xor temp
        i = i + 1
```