
TP introduction à la cryptographie

AES : Comment ça marche ?

Introduction

Comme souvent, une bonne introduction sur un sujet peut être trouvée sur Wikipedia. Voici ce qu'on trouve sur l'AES :

The Advanced Encryption Standard (AES), also known by its original name Rijndael (Dutch pronunciation: [\[ˈrɛɪndɑːl\]](#)), is a specification for the [encryption](#) of electronic data established by the U.S. [National Institute of Standards and Technology](#) (NIST) in 2001.

AES is a subset of the Rijndael [block cipher](#) developed by two [Belgian](#) cryptographers, [Vincent Rijmen](#) and [Joan Daemen](#), who submitted a proposal to NIST during the [AES selection process](#). Rijndael is a family of ciphers with different key and block sizes.

For AES, NIST selected three members of the Rijndael family, each with a **block size of 128 bits**, but **three different key lengths: 128, 192 and 256 bits**.

AES has been adopted by the [U.S. government](#) and is now used worldwide. It supersedes the [Data Encryption Standard](#) (DES), which was published in 1977. The algorithm described by AES is a [symmetric-key algorithm](#), meaning the same key is used for both encrypting and decrypting the data.

In the United States, AES was announced by the NIST as U.S. [FIPS](#) PUB 197 (FIPS 197) on November 26, 2001. This announcement followed a five-year standardization process in which fifteen competing designs were presented and evaluated, before the Rijndael cipher was selected as the most suitable (see [Advanced Encryption Standard process](#) for more details).

AES became effective as a federal government standard on May 26, 2002, after approval by the [Secretary of Commerce](#). AES is included in the ISO/IEC 18033-3 standard. AES is available in many different encryption packages, and is the first (and only) publicly accessible [cipher](#) approved by the [National Security Agency](#) (NSA) for [top secret](#) information when used in an NSA approved cryptographic module (see [Security of AES](#), below).

En résumé, AES est un algorithme cryptographique:

- Symétrique : le chiffrement et le déchiffrement utilisent la même clé « secrète »
- De type bloc : il traite des données de taille 128 bits (128 bits en entrée / 128 bits en sortie)
- Utilisant des clés de 128, 192 ou 256 bits (en fonction du niveau de sécurité souhaité)

L'objectif de ce TP est de réaliser **une implémentation de l'algorithme de chiffrement AES en langage C**. Pour cela, il va vous falloir maîtriser :

- le langage C (tableaux, structures, boucles, conditions...)
- l'anglais (pour lire la spécification...)
- les représentations polynomiales (???)
- les opérations sur les corps de Galois (aïe !)
- vos connexions synaptiques (aïe aussi !) ☺
- votre esprit de synthèse

Etape 1 : Découverte du standard

La vie étant bien faite, le standard AES est totalement défini dans un document de spécification.

La première chose à faire est certainement d'en prendre connaissance. Comme il est présent dans le dossier du TP, faites-vous plaisir ☺ **(20 minutes max)**.


```
void input2state(u8 input[16], u8 state[4][4])
{
    u8 r,c;
    for (c=0; c<4; c++)
        for (r=0; r<4; r++)
            state[r][c] = input[r+4*c];
}
```

Par exemple, si input vaut :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0xAB | 0xCD | 0xEF | 0x01 | 0x23 | 0x45 | 0x67 | 0x89 | 0xBA | 0xDC | 0xFE | 0x10 | 0x32 | 0x54 | 0x76 | 0x98 |

state vaudra :

| | 0 | 1 | 2 | 3 |
|---|------|------|------|------|
| 0 | 0xAB | 0x23 | 0xBA | 0x32 |
| 1 | 0xCD | 0x45 | 0xDC | 0x54 |
| 2 | 0xEF | 0x67 | 0xFE | 0x76 |
| 3 | 0x01 | 0x89 | 0x10 | 0x98 |

Etape 3 : Développement/intégration

Le temps imparti ne permet (malheureusement !?!) pas de comprendre tous les détails. Que peut-on retenir pour nous approcher de notre objectif ?

Le §5 est décidemment le plus intéressant !

- Il existe une fonction de chiffrement (§5.1) et une fonction de chiffrement inverse (§5.2).
C'est une bonne nouvelle : on pourra déchiffrer les messages qu'on a chiffrés...
Discussion philosophique : Est-ce vraiment nécessaire ?
- Les versions AES128, AES192, AES256 se différencient par un certain nombre de paramètres (Nb, Nk, Nr).
- Il existe une notion de « key scheduling » permettant d'obtenir une clé de tour.

Les paramètres (Nk, Nb, Nr) sont définis en **figure 4**. Nous allons travailler sur le cas **AES avec une clé de 128 bits (Nr = 10 tours)**.

```
#define Nk 4  
#define Nb 4  
#define Nr 10
```

La tâche est colossale pour atteindre notre objectif : nous avons environ 200-300 lignes de codes à écrire... Nous allons donc nous organiser pour répartir la charge de travail :

- Il y aura des développeurs de fonctions élémentaires
- Il y aura des intégrateurs de ces fonctions élémentaires pour créer la fonctionnalité globale

Pour que ça fonctionne, il va falloir :

- être forts...
- **communiquer**

Communiquer, ça signifie :

- définir qui fait quoi (je vais aider en distribuant les rôles et responsabilités...)
- définir les interfaces (je vais aider en les tentant de les imposer mais, conseil, validez entre développeurs et intégrateurs vos prototypes de fonctions/formats de données)
- définir un processus de livraison :
 - il faut que les développeurs préviennent les intégrateurs de leurs livraisons
 - il faut que les développeurs livrent des fonctions élémentaires validées
- respecter le planning (parce qu'on ne va pas y passer la nuit...)

Les intégrateurs vont devoir créer les fonctions suivantes:

```
void Cipher(u8 in[4*Nb], u8 out[4*Nb], u32 sub_key[Nb*(Nr+1)])  
void InvCipher(u8 in[4*Nb], u8 out[4*Nb], u32 sub_key[Nb*(Nr+1)])
```

Pour le chiffrement, l'algorithme est décrit en §5.1 et vous pouvez le vérifier avec les données fournies en §Appendice B ou §Appendice C.1.

Pour le déchiffrement, l'algorithme est décrit en §5.3 et vous pouvez le vérifier avec les données fournies en §Appendice C.1

Les développeurs vont devoir créer les fonctions suivantes :

```
void SubBytes(u8 in_state[4][4], u8 out_state[4][4])
void ShiftRows(u8 in_state[4][4], u8 out_state[4][4])
void InvSubBytes(u8 in_state[4][4], u8 out_state[4][4])
void InvShiftRows(u8 in_state[4][4], u8 out_state[4][4])
void AddRoundKey(u8 in_state[4][4], u32 sub_key[4], u8 out_state[4][4])
u8 GF256_mult(u8 a, u8 b)
void MixColumns(u8 in_state[4][4], u8 out_state[4][4])
u32 RotWord(u32 in_word)
u32 SubWord(u32 in_word)
void KeyExpansion(u8 key[4*Nk], u32 w[Nb*(Nr+1)])
```

Quelques tips

- Le fichier `tp_aes_begin.c` vous fournit en guise de cadeau de bienvenue un certain nombre de fonctions vous permettant conversion et affichage en tout genre.
Observez les bien et n'hésitez pas à les instancier pour être certains de leur fonctionnement !
- La fonction **MixColumns** est définie en (5.6). La multiplication \bullet est une multiplication dans le corps de Galois $GF(2^8) = GF(256)$ (§4.2).
Assurez-vous (manuellement) que $\{57\} \bullet \{83\} = \{c1\}$
- Assurez-vous, en considérant que X et Y quelconques, que :
 $X = \text{InvSubBytes}(\text{SubBytes}(X))$
 $X = \text{InvShiftRows}(\text{ShiftRows}(X))$
 $X = \text{InvMixColumns}(\text{MixColumns}(X))$
 $X = \text{AddRoundKeys}(\text{AddRoundKeys}(X,Y),Y)$
- La description du paramètre Rcon est une nouvelle fois mystérieuse. Comme on approche de Noël, sa définition est donnée dans le fichier de départ.
- Vous pouvez vérifier le fonctionnement de votre code en vous référant au **§Appendice**.
- La fonction `main` qui vous est gracieusement offerte doit vous permettre de vérifier le fonctionnement de vos implémentations !

Etape 4 : Validation

Bravo ! Maintenant qu'on dispose de nos belles fonctions, on peut faire quelques essais.

Par exemple, on pourrait chiffrer les messages suivants :

```
u8 message_1[16][5]={  
{ 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },  
{ 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F },  
{ 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F },  
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F },  
{ 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F }};
```

```
u8 message_2[16][5]={  
{ 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F },  
{ 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F },  
{ 0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F },  
{ 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F },  
{ 0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F }};
```

Et alors ?

Que pourrait-on faire ?