Cryptanalyse du chiffrement AES

UVSQ - UFR des Sciences

Amel LASSAL, Enzo REALE
Dihia DERBAL, Celine DJADEL

8 mai 2023

Sommaire

Qu'est-ce que l'AES ?

- Advanced Encryption Standard (ou Rijndael) inventé par Vincent Rijmen et Joan Daemen est un algorithme de chiffrement par bloc. Il est encore aujourd'hui le chiffrement symétrique le plus utilisé au monde. Développé après un appel international à candidatures lancé en janvier 1997 par le NIST, il fut adopté en 2001, puis a finalement été approuvé par la NSA pour remplacer le DES.
- Nécessite l'utilisation d'une clé et de message en bloc :
  - Clé de 128 bits qui sert à chiffrer le message.
  - Message de 256 bits qui passe par dix tours puis est renvoyé en un message chiffré.

Algorithme de l'AES

Algorithme 1 : Chiffrement

Entrées : Etat (message clair), clé
Sorties : Message chiffré
Appliquer AddRoundKey(state, w[0, 10-1]);
for *round de 1 à 9* **do**
    Appliquer SubBytes(Etat);
    Appliquer ShiftRows(Etat);
    Appliquer MixColumns(Etat);
    Appliquer AddRoundKey(Etat, w[round*Nb, (round+1)*Nb-1]);
Appliquer SubBytes(Etat);
Appliquer ShiftRows(Etat);
Appliquer AddRoundKey(Etat, w[10*Nb, (10+1)*10-1]);
retourner "Message chiffré";

SubBytes

- La première fonction principale est SubBytes :

```
def subBytes(etat):
    """La fonction prend en entrée un tableau 4x4 représentant l'état actuel
    et effectue la substitution de bytes en utilisant la fonction "SubWord()"
    sur chaque élément de l'état."""
    etat_change =  np.copy(etat)
    for i in range(len(etat_change)):
        etat_change[i] = SubWord(etat[i])
    return etat_change
```

ShiftRows

- La seconde est la fonction ShiftRows :

```
def ShiftRows(etat):
    """La fonction prend en entrée un tableau 4x4 représentant l'état actuel et
    applique une rotation horizontale de chaque ligne de l'état en fonction de
    l'indice de la ligne. Plus précisément, la première ligne n'est pas modifiée,
    la deuxième ligne est décalée d'une position vers la gauche, la troisième ligne
    est décalée de deux positions vers la gauche et la quatrième ligne est décalée de
    trois positions vers la gauche."""
    etat_change =  np.copy(etat)
    for i in range(len(etat)):
        etat_change[i] = np.roll(etat[i],-i)
    return etat_change
```

MixColumns

- Nous avons ensuite la fonction MixColumns ci-dessous :

```python
def MixColumns(etat):
    """Cette fonction implémente l'étape de mélange de colonnes de l'algorithme AES.
    Elle prend en entrée un état sous forme d'une matrice 4x4, effectue des opérations
    arithmétiques sur chaque colonne, et renvoie l'état résultant.

    Plus précisément, la fonction effectue les opérations suivantes pour chaque colonne de l'état :

    - Elle multiplie le premier élément par 2, le deuxième élément par 3, le troisième élément par 1,
    et le quatrième élément par 1.
    - Elle additionne les quatre produits obtenus, modulo 256.
    - Enfin, elle remplace chaque élément de la colonne par le résultat obtenu.

    Cette étape permet de mélanger les bits dans chaque colonne, de sorte que les bits d'un octet affectent
    les bits de tous les autres octets de la même colonne."""
    etat_change = np.copy(etat)
    for i in range(4):
        un0, un1, un2, un3 = etat[0, i], etat[1, i], etat[2, i], etat[3, i]
        etat_change[0, i] = format_hex(multiplication_by_2[int(un0, 16)]^multiplication_by_3[int(un1, 16)]^int(un2, 16)^int(un3, 16))
        etat_change[1, i] = format_hex(multiplication_by_2[int(un1, 16)]^multiplication_by_3[int(un2, 16)]^int(un0, 16)^int(un3, 16))
        etat_change[2, i] = format_hex(multiplication_by_2[int(un2, 16)]^multiplication_by_3[int(un3, 16)]^int(un0, 16)^int(un1, 16))
        etat_change[3, i] = format_hex(multiplication_by_2[int(un3, 16)]^multiplication_by_3[int(un0, 16)]^int(un2, 16)^int(un1, 16))
    return etat_change
```

AddRoundKey

- Et enfin la fonction AddRoundKey :

```
def AddRoundKey(etat, round):
    """La fonction prend en entrée un état et une clé de ronde (aussi appelée clé d'expansion),
    et retourne l'état résultant après avoir effectué un XOR entre chaque colonne de la clé de
    ronde et la colonne correspondante de l'état. Cela permet de mélanger la clé et l'état avant
    de passer à l'étape suivante."""
    etat_change = np.copy(etat)
    for i in range(len(etat)):
        etat_change[i] = XorWord(etat[i], round[i])
    return etat_change
```

Chiffrement

- La fonction complète de chiffrement est la suivante :

```
def encrypt(texte, key):
    """Effectue le chiffrement AES complet en utilisant les fonctions
    précédemment définies. La fonction prend en entrée le texte à
    chiffrer et la clé puis retourne le message chiffré."""
    key = keyExpansion(key)
    message_crypte = printState(texte)
    message_crypte = AddRoundKey(message_crypte, key[:, :4])
    for i in range(1, 10):
        message_crypte = subBytes(message_crypte)
        message_crypte = ShiftRows(message_crypte)
        message_crypte = MixColumns(message_crypte)
        message_crypte = AddRoundKey(message_crypte, key[:, i*4:i*4+4])
    message_crypte = subBytes(message_crypte)
    message_crypte = ShiftRows(message_crypte)
    message_crypte = AddRoundKey(message_crypte, key[:, -4:])
    return message_crypte
```

Déchiffrement

Algorithme 2 : Déchiffrement

Entrées : Etat (message chiffré), clé
Sorties : Message déchiffré en clair
clé ← Expansion de la clé;
message chiffré ← AddRoundKey avec comme arguments le message chiffré et les 4 premieres colonnes de la clé;
message chiffré ← ShiftRowsInverse du message chiffré;
message chiffré ← SubBytesInverse du message chiffré;
for *i de 1 à 9* **do**
    message chiffré ← AddRoundKey avec comme arguments le message chiffré et la clé les colonnes allant de i*4 à i*4+4;
    message chiffré ← MixColumnsInverse du message chiffré;
    message chiffré ← ShiftRowsInverse du message chiffré;
    message chiffré ← SubBytesInverse du message chiffré;
message clair ← AddRoundKey avec comme arguments le message chiffré et la clé les 4 dernieres colonnes;
retourner "Message clair";

ShiftRowsInverse

- La fonction ShiftRowsInverse reprend la fonction ShiftRows :

```
def ShiftRowsInverse(etat):
    """ Fonction inverse de ShiftRows, retourne l'état après avoir
    décale selon une rotation horizontale vers la droite de i positions chaque lignes de l'état """
    etat_change = np.copy(etat)
    for i in range(len(etat)):
        etat_change[i] = np.roll(etat[i], i)
    return etat_change
```

SubBytesInverse

- Utilise la fonction SubWordInverse afin de retrouver l'inverse de SubBytes

```
def subBytesInverse(etat):
    """  Fonction inverse de SubBytes, Retourne l'état après avoir appliqué la fonction
    SubWord inverse à chaque bits de l'état """
    etat_change =  np.copy(etat)
    for i in range(len(etat_change)):
        etat_change[i] = SubWordInverse(etat[i])
    return etat_change
```

SubWordInverse

- Appliquée avec la table S-box

```python
def SubWordInverse(tab):
    """Fonction inverse de SubWord, effectue une substitution de chaque élément
    du tableau en utilisant l'index des éléments de la table de substitution Sbox"""
    return np.array([format_hex(Sbox.index(int(elem, 16))) for elem in tab])
```

MixColumnsInverse

- Et enfin la fonction MixColumnsInverse :

```
def MixColumnsInverse(etat):
    """ Fonction inverse de la fonction Mix Columns """
    etat_change = np.copy(etat)
    for i in range(4):
        un0, un1, un2, un3 = etat[0, i], etat[1, i], etat[2, i], etat[3, i]
        etat_change[0, i] = format_hex(multiplication_by_14[int(un0, 16)]^multiplication_by_11[int(un1, 16)]
                                       ^multiplication_by_13[int(un2, 16)]^multiplication_by_9[int(un3, 16)])
        etat_change[1, i] = format_hex(multiplication_by_9[int(un0, 16)]^multiplication_by_14[int(un1, 16)]
                                       ^multiplication_by_11[int(un2, 16)]^multiplication_by_13[int(un3, 16)])
        etat_change[2, i] = format_hex(multiplication_by_13[int(un0, 16)]^multiplication_by_9[int(un1, 16)]
                                       ^multiplication_by_14[int(un2, 16)]^multiplication_by_11[int(un3, 16)])
        etat_change[3, i] = format_hex(multiplication_by_11[int(un0, 16)]^multiplication_by_13[int(un1, 16)]
                                       ^multiplication_by_9[int(un2, 16)]^multiplication_by_14[int(un3, 16)])
    return etat_change
```

Déchiffrement

- La fonction de déchiffrement à 10 tours :

```python
def decrypt(message_crypte, key):
    """ Fonction qui retourne le message clair à partir d'un chiffré et de sa clé (chiffrement inverse) """
    key = keyExpansion(key)
    message_crypte = AddRoundKey(message_crypte, key[:, -4:])
    message_crypte = ShiftRowsInverse(message_crypte)
    message_crypte = subBytesInverse(message_crypte)
    for i in range(9, 0, -1):
        message_crypte = AddRoundKey(message_crypte, key[:, i*4:i*4+4])
        message_crypte = MixColumnsInverse(message_crypte)
        message_crypte = ShiftRowsInverse(message_crypte)
        message_crypte = subBytesInverse(message_crypte)
    message_crypte = AddRoundKey(message_crypte, key[:, :4])
    return printStateInverse(message_crypte)

# Test de la fonction déchiffrement
texte = 'This is one text'
message_crypte = encrypt(texte, key)
key = '2b7e151628aed2a6abf7158809cf4f3c'
print('Le message déchiffré obtenu est :\n', decrypt(message_crypte, key))
```

Attaque intégrale - (Square Attack)

- Création de clé étendue pour l'AES à partir d'une clé initiale donnée

```python
def attaque(DeltaSets):
    """Prend une liste de delta sets clair/chiffré en entrée et tente de casser l'AES sur 4tours
    Si la version claire encryptée est égale à la version chiffrée, elle retourne la clé utilisée."""
    cle_expend = []
    for pos in range(16):
        index= (pos % 4, pos // 4)
        for DeltaSet in DeltaSets:
            validate_guess = []
            for guess in range(0x100):
                guessHex = format_hex(guess)
                reversed_bytes = reverseState(guessHex, index, DeltaSet[1])
                if checkGuess(reversed_bytes):
                    validate_guess.append(guessHex)
            if len(validate_guess) == 1:
                break
        cle_expend.append(validate_guess[0])
    cle = create_key_inversed(InvertKeyExpansion(4, np.array(cle_expend).reshape((4, 4), order='F')))
    if np.array_equiv(EncryptWithRound(DeltaSets[0][0][0], cle, 4), DeltaSets[0][1][0]):
        return cle
```

Setup

- Initialisation de tableaux delta set clair et delta set chiffre aléatoires pour le chiffrement de clé de blocs AES

```
def setup(cleP):
    """Prend une clé en entrée et renvoie un couple de tableaux delta_set_clair
    et delta_set_chiffre aléatoires tous les deux de de taille de 256 bits."""
    x = format_hex(random.randint(0, 255))
    delta_set_clair = np.array([[[format_hex(i), x, x, x], [x, x, x, x], [
                        x, x, x, x], [x, x, x, x]] for i in range(256)])
    delta_set_chiffre = np.array([EncryptWithRound(delta_set_clair[i], cleP, 4)
                        for i in range(len(delta_set_clair))])
    return delta_set_clair, delta_set_chiffre
```

CheckKeyGuess

- Utilise la fonction XorBit qui calcule entre deux bits le où exclusif

```python
def checkGuess(v_set):
    """Effectue un XOR entre les éléments de l'ensemble donné en entrée
    et retourne True si la somme des valeurs obtenues est égale à 0."""
    result = '00'
    for i in v_set:
        result = XorBit(result, i)
    return int(result, 16) == 0
```

ReverseState

- Voici la fonction reverseState utilisée :

```
def reverseState(key, pos_key, d_set):
    """Prend en entrée une clé sous forme hexadécimale, une position
    de clé (ligne, colonne), et un ensemble de 256 valeurs.Elle retourne une liste
    r_bits obtenue en faisant un XOR entre chaque élément de l'ensemble delta à la
    position de clé donnée et la clé donnée en entrée, puis en inversant le résultat
    avec la fonction InverseBit."""
    r_bits = []
    for elem in d_set:
        x = XorBit(elem[pos_key], key)
        x = InverseBit(x)
        r_bits.append(x)
    return r_bits
```

InvertKeyExpansion

- Et enfin la fonction KeyExpansion :

```
def InvertKeyExpansion(index, cle_ronde):
    """Prend en entrée un nombre de tours index et une clé de tour cle_ronde.
    La fonction retourne ensuite les 4 premières colonnes de la clé principale à
    partir de laquelle la clé ronde a été générée."""
    for i in range(index, 0, -1):
        for _ in range(3):
            cle_ronde = np.c_[XorWord(cle_ronde[:, 2], cle_ronde[:, 3]), cle_ronde]
        x = cle_ronde[:, 2]
        x = RotWord(x)
        x = SubWord(x)
        x = XorWord(XorWord(cle_ronde[:, 3], Rcon(i)), x)
        cle_ronde = np.c_[x, cle_ronde]
    return cle_ronde[:, 0:4]
```

Interface interactive

- Il est possible pour l'utilisateur de choisir la fonction qu'il souhaite exécuter

```
Bienvenue dans le mode intéractif!
Que souhaitez-vous faire?
 Tapez 'c' pour chiffrer
 Tapez 'd' pour déchiffrer
 Tapez 'a' pour lancer une attaque
 Tapez 'q' pour quitter
 c
Entrez le message : This is one text
Entrez la clé : 0b47e23def9fa6e24e9a8253a6241ecc
Le message chiffré obtenu est : »É→ wäm(ü4/4gBx
Voulez-vous relancer une commande ?
Tapez Oui/Non.
```

Interface interactive

- Il peut également relancer une autre commande

```
Voulez-vous relancer une commande ?
Tapez Oui/Non.
 Oui
Bienvenue dans le mode intéractif!
Que souhaitez-vous faire?
 Tapez 'c' pour chiffrer
 Tapez 'd' pour déchiffrer
 Tapez 'a' pour lancer une attaque
 Tapez 'q' pour quitter
 a
Clé trouvée par l'attaque : 8f3992d899c7ab4ed34aa6db9e6c81ec
```

Choix d'implémentation

- Nous avons choisi d'utiliser la bibliothèque **NumPy** afin de pouvoir stocker plus facilement la clé utilisée ainsi que le texte clair sous forme de matrices, mais également, car cet outil permet d'effectuer des calculs logiques et mathématiques bien plus rapidement et efficacement que de simples listes Python.
- La clé utilisée est également générée aléatoirement avec l'utilisation du module **secrets** de Python spécialement conçu pour la sécurité et la cryptographie contrairement au module par défaut random qui lui est conçu pour la modélisation et la simulation.

Fin

Merci pour votre attention ! :)
Si vous avez des questions, n'hésitez pas.