

**Ecole supérieure en science et technologie de l'informatique  
et du numérique ESTIN**

---

## **Compte rendu TP 03**

**Mini AES**

---

**Réalisé par :**

**Benmedakhene Souha (G2) .**

**Encadré par :**

**Dr . Bouchoucha Lydia**

**Année universitaire : 2023/2024**

# Introduction :

Le [mini AES](#) (Advanced Encryption Standard) est une version simplifiée de l'algorithme de chiffrement AES, largement utilisé pour sécuriser les données sensibles dans divers domaines. Dans ce travail pratique, nous nous concentrons sur une version simplifiée du AES afin de comprendre ses principes fondamentaux et d'acquérir une expérience pratique dans son implémentation. La sécurité des données est une préoccupation majeure dans le monde moderne, avec une augmentation constante des cybermenaces et des violations de données. Comprendre le fonctionnement des algorithmes de chiffrement et être capable de les mettre en œuvre est essentiel pour assurer la confidentialité et l'intégrité des données. Ce travail pratique vise à fournir une introduction pratique à l'algorithme AES en se concentrant sur sa [version simplifiée](#), le mini AES, et en utilisant le langage de programmation [Python](#) pour générer des clés de chiffrement et chiffrer des messages.

## Objectifs :

Les objectifs de ce travail pratique sont les suivants :

1. Comprendre les principes de base de l'algorithme de chiffrement AES, y compris ses étapes de [génération de clés](#) et de [chiffrement](#).
2. Acquérir une expérience pratique dans l'implémentation du mini AES en utilisant le langage de programmation Python.
3. Expliquer le processus de génération de clés pour le mini AES, y compris la division de la clé initiale en blocs, les opérations de XOR et [la substitution de nibble](#) pour générer les clés de tour suivantes.
4. Mettre en œuvre le processus de chiffrement du mini AES, comprenant les étapes de substitution de nibble, de permutation de lignes, de mélange de colonnes et d'ajout de clés de tour pour sécuriser un message donné.
5. Valider la justesse de l'implémentation en testant avec différentes entrées et en comparant les résultats avec les attentes théoriques.

# Implémentation :

## Etape 1: Génération des clés

Dans l'étape de génération de clés du mini AES, la clé initiale fournie par l'utilisateur est transformée en une série de sous-clés, également appelées clés de tour, qui seront utilisées pour le chiffrement de chaque tour. Voici une explication plus détaillée de cette étape :

### 1. Division de la clé initiale :

La clé initiale est une chaîne de 16 bits (ou 4 caractères hexadécimaux), qui est divisée en quatre blocs de 4 bits chacun. Chaque bloc correspond à un mot de la clé par exemple :  $W_0, W_1, W_2, W_3$  .

### 2. Calcul des clés de tour :

Une fois la clé initiale divisée, les clés de tour sont calculées en utilisant un processus itératif. Les opérations suivantes sont effectuées pour chaque paire de mots de la clé :

#### a. Substitution de nibble :

Chaque nibble (4 bits) du mot est soumis à une substitution à l'aide d'une table de substitution prédéfinie. Cette substitution de nibble rend chaque bit de la clé dépendant de tous les autres bits, introduisant ainsi de la non-linéarité et de la confusion dans le processus de génération de clés.

#### b. Opérations de XOR :

Les nibbles des mots de clé sont XORés les uns avec les autres et avec d'autres valeurs calculées lors du processus. Cela mélange les bits de la clé et crée des différences significatives entre les clés de tour successives, renforçant ainsi la sécurité du chiffrement.

- Si on prend  $K_1 = W_4, W_5, W_6, W_7$  et  $K_2 = W_8, W_9, W_{10}, W_{11}$  .

$W_4 = W_0 \text{ XOR NibbleSub}(W_3) \text{ XOR record}(1).$

$W_5 = W_1 \text{ XOR } W_4.$

$W_6 = W_2 \text{ XOR } W_5.$

$W_7 = W_3 \text{ XOR } W_6.$

$W_8 = W_4 \text{ XOR NibbleSub}(W_7) \text{ XOR record}(2).$

$W_9 = W_5 \text{ XOR } W_8.$

$W_{10} = W_6 \text{ XOR } W_9.$

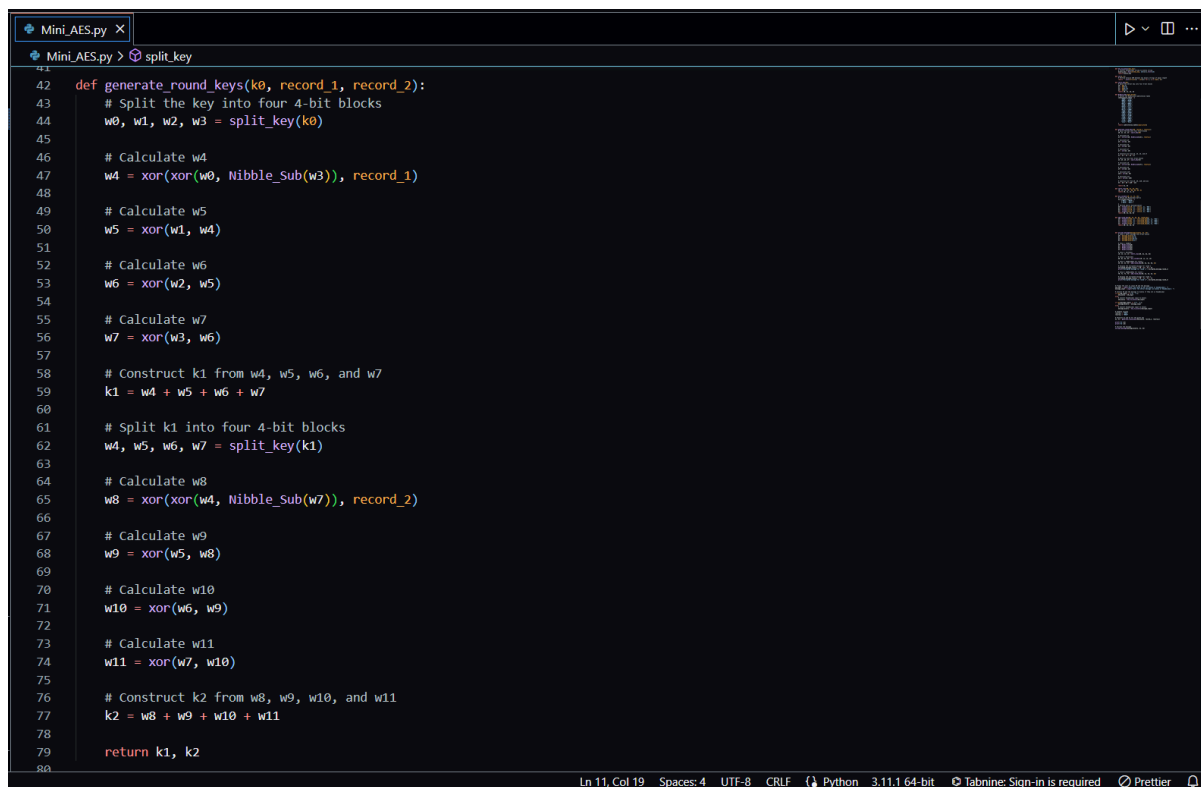
$W_{11} = W_7 \text{ XOR } W_{10}.$

3. **Clés de tour résultant** : À la fin du processus, les clés de tour résultantes sont générées. Dans le cas du mini AES, deux clés de tour sont produites, généralement désignées par  $k_1$  et  $k_2$ . Ces clés seront utilisées dans les étapes suivantes du chiffrement pour XORer avec les blocs de texte clair.

Et voici le code en Python :

```
Mini_AES.py X
Mini_AES.py > split_key

2 def hex_to_binary(hex_key):
3     # Convert hexadecimal string to binary string
4     binary_key = bin(int(hex_key, 16))[2:].zfill(16)
5     return binary_key
6
7 def xor(a, b):
8     # Perform bitwise XOR between two binary strings of equal length
9     return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))
10
11 def split_key(k0):
12     # Split the 16-bit key into four 4-bit blocks
13     w0 = k0[:4]
14     w1 = k0[4:8]
15     w2 = k0[8:12]
16     w3 = k0[12:]
17     return w0, w1, w2, w3
18
19 def Nibble_Sub(input_block):
20     # Define the Nibble_Sub substitution table
21     substitution_table = {
22         "0000": "1110",
23         "0001": "0100",
24         "0010": "1101",
25         "0011": "0001",
26         "0100": "0010",
27         "0101": "1111",
28         "0110": "1011",
29         "0111": "1000",
30         "1000": "0011",
31         "1001": "1010",
32         "1010": "0110",
33         "1011": "1100",
34         "1100": "0101",
35         "1101": "1001",
36         "1110": "0000",
37         "1111": "0111"
38     }
39     return substitution_table[input_block]
40
```



```
42 def generate_round_keys(k0, record_1, record_2):
43     # Split the key into four 4-bit blocks
44     w0, w1, w2, w3 = split_key(k0)
45
46     # Calculate w4
47     w4 = xor(xor(w0, Nibble_Sub(w3)), record_1)
48
49     # Calculate w5
50     w5 = xor(w1, w4)
51
52     # Calculate w6
53     w6 = xor(w2, w5)
54
55     # Calculate w7
56     w7 = xor(w3, w6)
57
58     # Construct k1 from w4, w5, w6, and w7
59     k1 = w4 + w5 + w6 + w7
60
61     # Split k1 into four 4-bit blocks
62     w4, w5, w6, w7 = split_key(k1)
63
64     # Calculate w8
65     w8 = xor(xor(w4, Nibble_Sub(w7)), record_2)
66
67     # Calculate w9
68     w9 = xor(w5, w8)
69
70     # Calculate w10
71     w10 = xor(w6, w9)
72
73     # Calculate w11
74     w11 = xor(w7, w10)
75
76     # Construct k2 from w8, w9, w10, and w11
77     k2 = w8 + w9 + w10 + w11
78
79     return k1, k2
80
```

## Etape 2: Chiffrement

L'étape de chiffrement dans l'algorithme Mini DES est le cœur du processus, où les données en clair sont transformées en données chiffrées à l'aide des clés de chiffrement générées précédemment (K1 et K2). Cette étape est composée de plusieurs sous-étapes, chacune contribuant à la sécurité et à la confidentialité des données chiffrées. Voici une explication détaillée de chaque aspect de l'étape de chiffrement :

### 1. Permutation initiale (PI) :

- Au début de l'étape de chiffrement, les données en clair sont soumises à une permutation initiale (PI). Cette permutation réorganise les bits du texte en clair selon une table de permutation spécifique appelée permutation initiale (PI). L'objectif de cette permutation est de mélanger les bits du texte en clair de manière à améliorer la diffusion et la confusion, deux propriétés essentielles pour renforcer la sécurité du chiffrement.

## 2. **Rounds de chiffrement** (en utilisant la fonction $F(K_i)$ ) :

- Le chiffrement dans Mini DES se déroule en deux rounds, chacun utilisant une clé de chiffrement différente ( $K_1$  pour le premier round et  $K_2$  pour le deuxième round).
- Dans chaque round, les blocs L et R sont soumis à une série d'opérations, y compris l'expansion, la substitution, la permutation et le XOR avec la clé de chiffrement correspondante.
- L'expansion consiste à étendre le bloc  $R_0$  en utilisant une table de permutation spécifique (EP), qui ajoute des bits supplémentaires au bloc pour augmenter la complexité du chiffrement.
- Ensuite, le bloc étendu est combiné avec la clé de chiffrement ( $K_1$  ou  $K_2$ ) en utilisant l'opération de XOR (OU exclusif).
- Après le XOR, le résultat est divisé en deux parties, qui sont ensuite soumises à des S-boxes (boîtes de substitution) . Les S-boxes remplacent chaque groupe de bits par un autre groupe de bits en utilisant des tables de substitution spécifiques ( $S_0$  pour la première partie et  $S_1$  pour la deuxième partie).
- Les résultats des S-boxes sont permutés à l'aide d'une autre table de permutation spécifique ( $P_4$ ), ce qui réorganise les bits pour obtenir le texte chiffré.
- Après on va faire un XOR entre le résultat et le “L” en entrée .
- Enfin , à la fin de  $F(K_1)$  on va faire un SW ( les blocs de sortie sont combinés dans un ordre spécifique) et la même chose pour  $F(K_2)$  .

## 3. **Permutation finale ( $PI^{-1}$ ) :**

- Une fois les rounds de chiffrement terminés, les blocs de sortie sont soumis à une permutation finale ( $PI^{-1}$ ) ( c'est la matrice inverse de  $PI$ ). Cette permutation inverse la permutation initiale et restaure l'ordre original des bits, produisant ainsi le texte chiffré final.

## Et voici le code en Python :

```
49 PI = [2, 6, 3, 1, 4, 8, 5, 7]
50
51 def initial_permutation(text):
52     return permute(text, PI)
53
54
55 EP = [4, 1, 2, 3]
56
57 S0 = [
58     ['01', '00', '11', '10'],
59     ['11', '10', '01', '00'],
60     ['00', '10', '01', '11'],
61     ['11', '01', '00', '10']
62 ]
63
64 S1 = [
65     ['00', '10', '10', '11'],
66     ['10', '00', '01', '11'],
67     ['11', '00', '01', '00'],
68     ['10', '01', '00', '11']
69 ]
70
71 P4 = [2, 4, 3, 1]
72
73 def xor(a, b):
74     # Perform bitwise XOR between two binary strings of equal length
75     return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))
76
77
78 def s_box(bits, sbox):
79     row = int(bits[0] + bits[3], 2)
80     col = int(bits[1] + bits[2], 2)
81     return sbox[row][col]
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102 def swap_and_permute(R0, R1):
103     return R1 + R0
104
105
106 def f_with_k2(R0, K2):
107     return f(R0, K2)
108
109
110 def swap_and_permute_final(R2, R1):
111     return R2 + R1
112
113
114 IP_1 = [4, 1, 3, 5, 7, 2, 8, 6]
115
116 def final_permutation(text):
117     return permute(text, IP_1)
118
```

```

120
121
122 def encrypt(plaintext, K1, K2):
123     # Initial permutation
124     permuted_text = initial_permutation(plaintext)
125     # Split into L0 and R0
126     L0 = permuted_text[:4]
127     R0 = permuted_text[4:]
128     # Round 1
129     R1 = f(R0, K1)
130     # Swap and permute
131     swapped_text = swap_and_permute(R0, R1)
132     # Round 2
133     R2 = f_with_k2(swapped_text[:4], K2)
134     # Swap and permute final
135     final_text = swap_and_permute_final(R2, swapped_text[4:])
136     # Final permutation
137     encrypted_text = final_permutation(final_text)
138     return encrypted_text
139
140
141 plaintext = input("Enter the plaintext (8 bits): ")
142
143
144 # Encrypt and print the result
145 encrypted_text = encrypt(plaintext, K1, K2)
146 print("Encrypted text:", encrypted_text)

```

## Etape 3: Test et Validation

La validation revêt une importance capitale lors du développement d'un algorithme de chiffrement, car elle garantit que le processus de chiffrement opère conformément aux attentes et offre un niveau de sécurité suffisant. À cet effet, divers exemples de texte en clair et de clés sont employés pour évaluer l'efficacité de l'algorithme mis en place . Le programme va prendre le texte chiffré et la clé K0 comme entrées de l'utilisateur.

Text en claire : EA0F et la clé K0 :CF03 .

```

PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\Mini_AES.py"
Enter the value of k0 (in binary or hexadecimal):

```

```

PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\Mini_AES.py"
Enter the value of k0 (in binary or hexadecimal): EA0F
Enter the 16-bit message (in binary or hexadecimal):

```



Et voici l'affichage final :

```
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\Mini_AES.py"
Enter the value of k0 (in binary or hexadecimal): EA0F
Enter the 16-bit message (in binary or hexadecimal): CF03
k1 1000001000101101
k2 0011000100111110
Encrypted message for round 1: 0101100101001001
Encrypted message for round 2: 0101100101011000
PS C:\Users\HP\Desktop\new> |
```

## Conclusion :

Dans le cadre de cette activité pratique, nous avons pu consolider notre compréhension des concepts clés du chiffrement et nous familiariser avec l'algorithme Mini AES. En implémentant cet algorithme, nous avons acquis une connaissance approfondie de son fonctionnement interne, en particulier des étapes telles que la génération des clés et le processus de chiffrement lui-même. Cette expérience pratique nous a permis de relever les défis techniques liés à la manipulation des données binaires et à la gestion des opérations de substitution et de permutation. En fin de compte, cette immersion dans le processus d'implémentation d'un algorithme de chiffrement nous a offert une perspective enrichissante sur le fonctionnement des systèmes de sécurité informatique.