



Universitatea Tehnică "Gheorghe Asachi" din Iași  
Facultatea de Automatică și Calculatoare  
Domeniul Calculatoare și Tehnologia Informației  
Specializarea Tehnologia Informației

# **Implementarea AES utilizând limbajul Python**

Profesor îndrumător,  
Ș.l.dr.ing Alexandru Tudorache

Student,  
Bulai Iustina-Bianca  
Grupa 1410A

Iași, 2024

# Cuprins

1. Introducere
2. Despre AES (Advanced Encryption Standard)
3. Implementare
4. Interfață

## 1. Introducere

Din dorința de a securiza datele și informațiile într-un mod cât mai optim, pentru a împiedica accesul neautorizat la acestea, au fost creați mai mulți algoritmi de criptare. Printre aceștia se numără și algoritmul AES (Advanced Encryption Standard), cunoscut și sub numele de Rijndael, descris și implementat în cadrul acestui proiect. Acesta a fost adoptat ca standard de criptare în anul 2001 de către Institutul Național de Standarde și Tehnologie din Statele Unite și a devenit rapid un standard global datorită eficienței sale.

În cadrul acestui proiect, pentru implementarea algoritmului, mai exact pentru funcțiile necesare acestuia, s-a utilizat limbajul Python. Pentru testare a fost realizată o interfață, tot în Python, utilizând tkinter, cu ajutorul căreia utilizatorul are posibilitatea de a cripta/decripta mesaje.

## 2. Despre AES (Advanced Encryption Standard)

AES este un algoritm de criptare simetrică, motiv pentru care cheia este aceeași atât pentru criptare, cât și pentru decriptare. Acesta operează pe blocuri de date de 128 de biți, utilizând chei de 128, 192 sau 256 de biți. Procesul de criptare constă într-un număr de runde de transformări, în funcție de dimensiunea cheii: 10 runde, 12 runde, 14 runde, pentru chei de 128, 192, respectiv 256 de biți. În cadrul acestui proiect a fost realizată criptarea utilizând 10 runde, folosindu-se deci chei de 128 de biți.

## 3. Implementare

În ceea ce privește implementarea algoritmului, funcțiile dedicate acestuia au fost scrise în limbajul Python. Au fost realizate două script-uri, unul care conține funcțiile pentru criptare (criptare.py) și unul care conține funcțiile pentru decriptare (decriptare.py).

Algoritmul de criptare pornește de la plaintext-ul introdus, pe care îl transformă într-o matrice 4x4, iar apoi sunt folosite 4 funcții prin care se realizează criptarea. Acestea sunt:

- ➔ SubBytes(): aici are loc substituția fiecărui byte cu o valoare din tabela de substituție predefinită, S-box.
- ➔ ShiftRows(): realizează o permutare a rândurilor matricii (linia 0 rămâne aceeași, elementele de pe linia 1 se shiftează la stânga cu o poziție, cele de pe linia 2 cu 2 poziții, iar pentru linia 3, elementele sunt shiftate cu 3 poziții).
- ➔ MixColumns(): constă într-o transformare a stării care înmulțește fiecare din cele 4 coloane ale matricii cu o matrice predefinită.

Pentru realizarea acestei funcții, am plecat de la implementarea funcției xTimes(), definită ca o înmulțire a unui byte (octet) cu 2 în câmpul finit  $GF(2^8)$ . În acest context, un byte este tratat ca un polinom de gradul 7 cu coeficienți binari, iar operațiile sunt efectuate modulo polinomul primitiv  $x^8+x^4+x^3+x+1$ .

```
def xTimes(x):
    if x & 0x80: #primul bit e 1
        x = x << 1 & 0xFF #shiftam cu o pozitie la stanga
        x ^= 0x1b #facem xor cu sirul 0011011
    else:
        x = x << 1 & 0xff #doar shiftam cu o pozitie la stanga
    return x
```

**Fig. Funcția xTimes()**

Mai apoi, s-a realizat funcția mix\_a\_column(), prin care se realizează înmulțirea unei coloane cu matricea predefinită, utilizând funcția xTimes().

```
def mix_a_column(s):
    a=s[0]
    b=s[1]
    c=s[2]
    d=s[3]

    s[0] = xTimes(a) ^ xTimes(b) ^ b ^ c ^ d
    s[1] = a ^ xTimes(b) ^ xTimes(c) ^ c ^ d
    s[2] = a ^ b ^ xTimes(c) ^ xTimes(d) ^ d
    s[3] = xTimes(a) ^ a ^ b ^ c ^ xTimes(d)
```

**Fig. Funcția mix\_a\_column()**

Iar la final, s-a implementat funcția mix\_columns(), folosită pentru a aplica mix\_a\_column() pe fiecare coloană a matricii stare.

```
def mix_columns(s):
    for i in range(4):
        column=[]
        column = [s[j][i] for j in range(4)] # extrag coloana

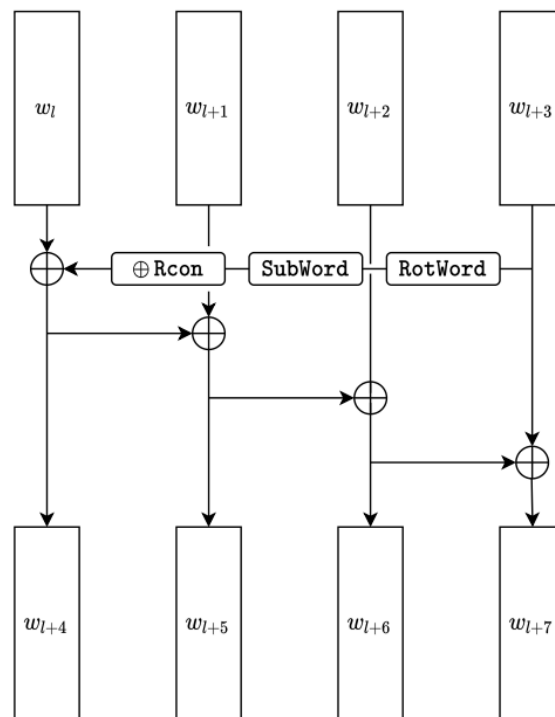
        mix_a_column(column)

        for j in range(4): # actualizez matricea s cu valorile calculate
            s[j][i] = column[j]
```

**Fig. Funcția mix\_columns()**

→ AddRoundKey(): în care se realizează o operație XOR între biții din matricea state și biții din matricea cheii.

Matricea cheii menționată anterior este extinsă la fiecare pas cu ajutorul funcției KeyExpansion(). Astfel, folosind substituții și rotații și o matrice numită Rcon, predefinită, la fiecare iterație se va folosi o nouă cheie generată din cea anterioară pentru criptarea mesajului. Astfel, se asigură faptul că la fiecare rundă se vor deriva cheile într-un mod complicat din cheia inițială. În figura Fig. 1 se poate observa cum funcționează KeyExpansion pentru AES-128 unde sunt generate noile cuvinte stocate în  $w$ .



**Fig. 1 KeyExpansion() pentru AES-128**

```

#AES-128
Nk=4 #lungime cheie
Nb=4 #marime bloc
Nr=10 #numar runde
#nk, nb se refera la nr de cuvinte din 32 de biti fiecare

def key_expansion(key):
    key_expanded = [key]
    key_prev=key_expanded[-1]
    w0 = [row[0] for row in key_prev]
    w1 = [row[1] for row in key_prev]
    w2 = [row[2] for row in key_prev]
    w3 = [row[3] for row in key_prev]

    for i in range(1,Nr+1):
        #iau w3 care reprezinta ultimul elem din key prev si aplic operatiile rotate, sub, xor cu rcon
        w3_new=w3
        #rotate
        w3_new=rot_word(w3_new)
        #iau din sbox
        w3_new=sub_word(w3_new)
        #xor cu Rcon
        rcon=Rcon[i-1]
        res=[]
        for element1, element2 in zip(w3_new,rcon):
            res.append(element1^element2)
        w3_new=res

```

Fig. Implementarea funcției key\_expansion()<sup>1</sup>

```

#calculam w4,w5,w6,w7
w4=[]
for element1, element2 in zip(w0,w3_new):
    w4.append(element1^element2)

w5=[]
for element1, element2 in zip(w1,w4):
    w5.append(element1^element2)

w6=[]
for element1, element2 in zip(w2,w5):
    w6.append(element1^element2)

w7=[]
for element1, element2 in zip(w3,w6):
    w7.append(element1^element2)

#adaugam in lista cheia generata
round_columns = [w4, w5, w6, w7]
transpusa = [[rand[i] for rand in round_columns] for i in range(len(round_columns[0]))]
key_expanded.append(transpusa)

w0=w4
w1=w5
w2=w6
w3=w7

return key_expanded

```

Fig. Implementarea funcției key\_expansion()<sup>2</sup>

Ordinea în care sunt aplicate aceste funcții descrise mai sus pentru a rezulta mesajul criptat poate fi observată în figura Fig. 2, partea stângă.

Pentru algoritmul de decriptare, sunt luate funcțiile din algoritmul de criptare, sunt inversate și apoi aplicate în ordine inversă, obținând astfel la final mesajul descifrat. Tot în figura Fig. 2, în partea dreaptă, se poate observa modul de funcționare al algoritmului de decriptare.

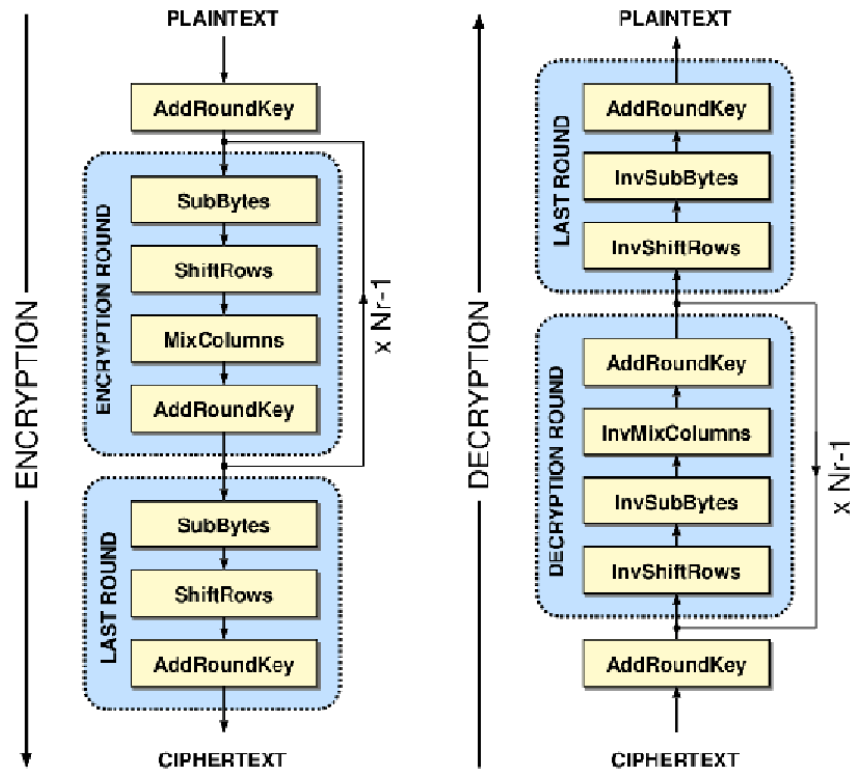


Fig. 2 Schema pentru algoritmul de criptare/decriptare

#### 4. Interfață

Pentru a-i oferi utilizatorului posibilitatea de a-și cripta/decripta propriile mesaje, a fost realizată o interfață tot în Python, utilizând tkinter. Aceasta a fost implementată în script-ul aes.py. Interfața este una destul de sugestivă, după cum se poate observa și în figura Fig. 3, oferindu-i posibilitatea utilizatorului de a introduce mesajele și cheia atât manual, cât și prin intermediul unor butoane care permit încărcarea fișierelor ce conțin mesajul și cheia.

The screenshot shows a web application window titled 'AES'. It contains two main sections: 'Criptare:' (Encryption) and 'Decriptare:' (Decryption). Each section has three input fields: 'Introduceti textul:' (Enter text), 'Introduceti cheia:' (Enter key), and 'Mesajul dupa criptare:' (Message after encryption/decryption). Below the input fields are two buttons: 'Criptare mesaj' (Encrypt message) and 'Incarcare Fisier' (Load file). The encryption section is currently empty.

**Fig. 3 Interfața**

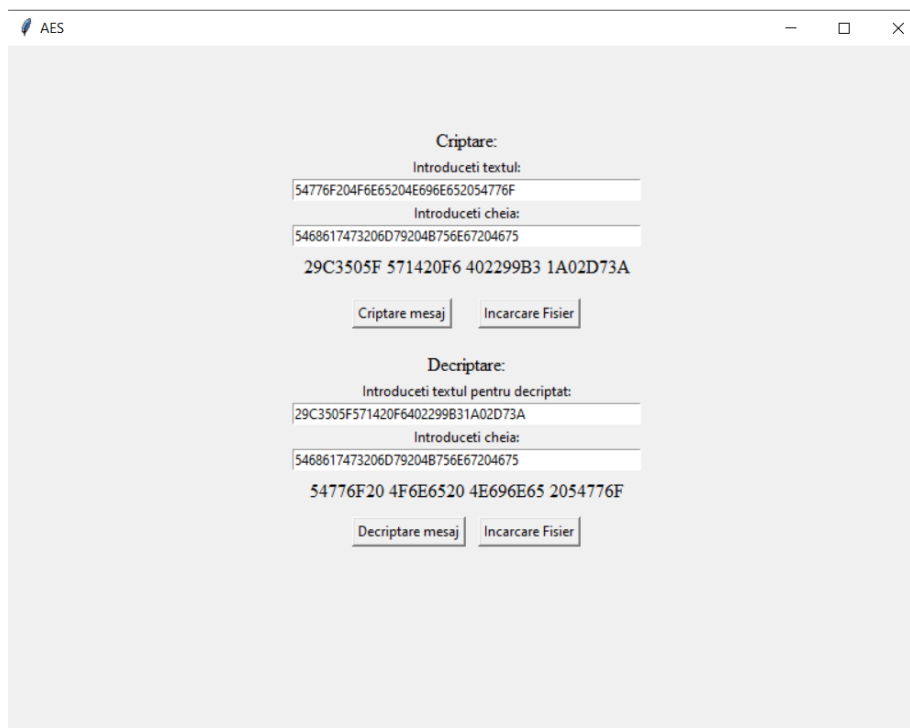
În figura Fig. 4 se poate observa testarea algoritmului de criptare și rezultatul obținut în urma aplicării operațiilor descrise mai sus.

The screenshot shows the same AES application window, but now with test data entered. In the 'Criptare:' section, the 'Introduceti textul:' field contains the hexadecimal string '54776F204F6E65204E696E652054776F', the 'Introduceti cheia:' field contains '5468617473206D79204B756E67204675', and the 'Mesajul dupa criptare:' field displays the result '29C3505F 571420F6 402299B3 1A02D73A'. The 'Decriptare:' section remains empty.

**Fig. 4 Testare criptare**



În Fig. 5 a fost realizată și introducerea mesajului criptat și a cheii pentru decriptare. Din rezultatul obținut, se poate observa că rezultatul este chiar mesajul pe care am dorit să-l criptăm în figura anterioară.



The screenshot shows a web-based application window titled "AES" with standard window controls (minimize, maximize, close). The interface is divided into two main sections: "Criptare:" (Encryption) and "Decriptare:" (Decryption).

**Criptare:**

- Label: "Introduceți textul:"
- Input field: "54776F204F6E65204E696E652054776F"
- Label: "Introduceți cheia:"
- Input field: "5468617473206D79204B756E67204675"
- Output field: "29C3505F 571420F6 402299B3 1A02D73A"
- Buttons: "Criptare mesaj" and "Incarcare Fisier"

**Decriptare:**

- Label: "Introduceți textul pentru decriptat:"
- Input field: "29C3505F571420F6402299B31A02D73A"
- Label: "Introduceți cheia:"
- Input field: "5468617473206D79204B756E67204675"
- Output field: "54776F20 4F6E6520 4E696E65 2054776F"
- Buttons: "Decriptare mesaj" and "Incarcare Fisier"

**Fig. 5 Testare criptare/decriptare**

Astfel, algoritmul AES vine în ajutorul persoanelor care doresc să-și protejeze datele, oferindu-le acestora siguranță și încredere, realizând, prin intermediul acestor funcții care îl compun, un echilibru între securitate și performanță.