

Joel Tapia Salvador (1638962)
Alejandro Jaime Cabrera (1636745)

Implementación Advanced Encryption Standard (AES)

Para llevar a cabo la implementación del cifrado Advanced Encryption Standard (AES), optamos por desarrollar nuestros propios scripts en Python, enfocándonos en una estrategia modular y metódica^[1]. Este enfoque nos permitió desglosar el complejo algoritmo de cifrado en sus componentes fundamentales, abordando cada fase del proceso (desde la inicialización del estado y la expansión de la clave hasta las transformaciones de SubBytes, ShiftRows, MixColumns, y AddRoundKey) de manera individualizada.

Para garantizar la precisión y efectividad de nuestro algoritmo, adoptamos un enfoque iterativo, donde cada fase fue cuidadosamente programada y validada a través de pruebas unitarias detalladas. Esta metodología no solo facilitó una comprensión más profunda de cada operación dentro del proceso de cifrado AES, sino que también aseguró la integración fluida y sin errores de los distintos módulos.

El resultado de este proceso meticuloso se consolidó en un script principal, AES.py, que actúa como el núcleo de nuestra implementación. En este archivo, los componentes individuales se importan y ensamblan en un flujo coherente y ordenado, reflejando la estructura y secuencia prescritas por el estándar AES.

Estructura General de la Operación de Cifrado

El proceso de cifrado AES se llevó a cabo en 10 rondas, cada una compuesta por las siguientes operaciones, realizadas en el orden indicado:

AddRoundKey

Aplicación de la clave de ronda al estado mediante una operación XOR.

ByteSub (SubBytes)

Sustitución de cada byte por otro según una tabla de sustitución predefinida (S-Box).

ShiftRows

Rotación de las filas del estado; cada fila se rota un número variable de posiciones.

MixColumns

Combinación de las columnas del estado, aplicando una transformación lineal.

KeyExpansion

Expansión de la clave original para generar una clave de ronda para cada ronda de cifrado.

Descripción y Resultados de las Operaciones

Texto y Clave Utilizados

Texto a Cifrar: "JoelT AlejandroJ"

AddRoundKey

Estado de Entrada Zero

```
[  
  [0x4a, 0x6f, 0x65, 0x6c],  
  [0x54, 0x20, 0x41, 0x6c],  
  [0x65, 0x6a, 0x61, 0x6e],  
  [0x64, 0x72, 0x6f, 0x4a],  
]
```

Clave de Ronda Zero

```
[  
  [0x2b, 0x7e, 0x15, 0x16],  
  [0x28, 0xae, 0xd2, 0xa6],  
  [0xab, 0xf7, 0x15, 0x88],  
  [0x09, 0xcf, 0x4f, 0x3c],  
]
```

Estado de Salida

```
[  
  [0x61, 0x11, 0x70, 0x7a],  
  [0x7c, 0x8e, 0x93, 0xca],  
  [0xce, 0x9d, 0x74, 0xe6],  
  [0x6d, 0xbd, 0x20, 0x76],  
]
```

ByteSub (SubBytes)

```
[  
  [0xef, 0x82, 0x51, 0xda],  
  [0x10, 0x19, 0xdc, 0x74],  
  [0x8b, 0x5e, 0x92, 0x8e],  
  [0x3c, 0x7a, 0xb7, 0x38],  
]
```

ShiftRows

```
[  
  [0xef, 0x19, 0x92, 0x38],  
  [0x10, 0x5e, 0xb7, 0xda],  
  [0x8b, 0x7a, 0x51, 0x74],  
  [0x3c, 0x82, 0xdc, 0x8e],  
]
```

MixColumns

```
[  
  [0x44, 0x48, 0x81, 0xd1],  
  [0xaf, 0xb4, 0x4e, 0x76],  
  [0xa6, 0xf8, 0xcf, 0x45],  
  [0xb7, 0xd2, 0x94, 0x1d],  
]
```

Cifrado tras 10 rondas

```
[  
  [0x01, 0xca, 0xc1, 0xe1],  
  [0x8f, 0x8a, 0x67, 0xc8],  
  [0x27, 0x39, 0x17, 0x36],  
  [0x79, 0x59, 0xd5, 0x26],  
]
```

Verificación del Resultado

Definimos una constante `CAD_DEBUG`

```
CAD_DEBUG = [
    [0x32, 0x43, 0xf6, 0xa8],
    [0x88, 0x5a, 0x30, 0x8d],
    [0x31, 0x31, 0x98, 0xa2],
    [0xe0, 0x37, 0x07, 0x34]
]
```

Código 1: Definición de la constante `CAD_DEBUG`

La cual es el mismo input usado en la animación de *FormaEstudio*. (n.d.). *Rijndael Animation v4* [\[2\]](#).

Con la información de los estados correctos mostrados por la animación. Pudimos comprobar la funcionalidad de cada módulo.

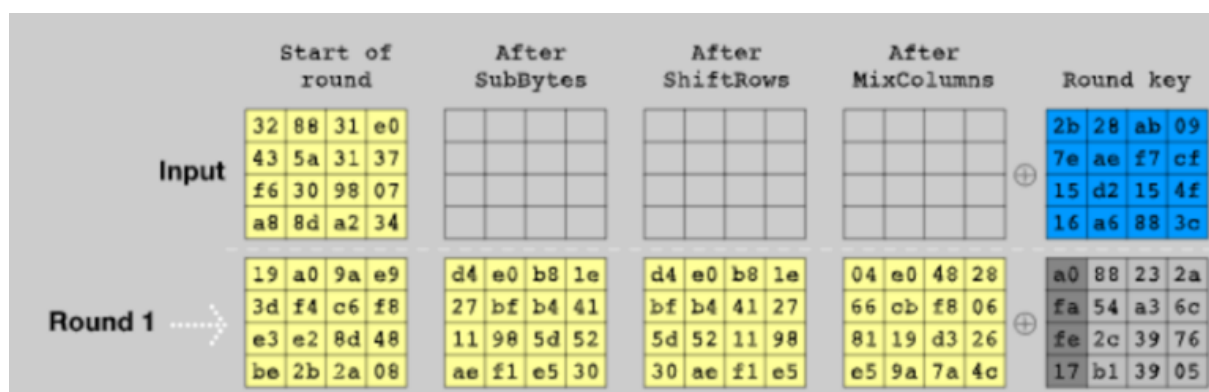


Imagen 1: Primera ronda de la animación

```
Input: = [
    [0x32, 0x43, 0xf6, 0xa8],
    [0x88, 0x5a, 0x30, 0x8d],
    [0x31, 0x31, 0x98, 0xa2],
    [0xe0, 0x37, 0x07, 0x34],
]

Clave Inicial: = [
    [0x2b, 0x7e, 0x15, 0x16],
    [0x28, 0xae, 0xd2, 0xa6],
    [0xab, 0xf7, 0x15, 0x88],
    [0x09, 0xcf, 0x4f, 0x3c],
]

Estado después de AddRoundKey 0 = [
    [0x19, 0x3d, 0xe3, 0xbe],
    [0xa0, 0xf4, 0xe2, 0x2b],
    [0x9a, 0xc6, 0x8d, 0x2a],
    [0xe9, 0xf8, 0x48, 0x08],
]

Estado después de ByteSub 1 = [
    [0xd4, 0x27, 0x11, 0xae],
    [0xe0, 0xbf, 0x98, 0xf1],
    [0xb8, 0xb4, 0x5d, 0xe5],
    [0x1e, 0x41, 0x52, 0x30],
]

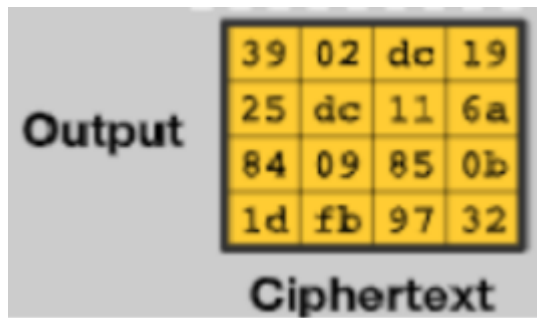
Estado después de ShiftRows 1 = [
    [0xd4, 0xbf, 0x5d, 0x30],
    [0xe0, 0xb4, 0x52, 0xae],
    [0xb8, 0x41, 0x11, 0xf1],
    [0x1e, 0x27, 0x98, 0xe5],
]

Estado después de MixColumns 1 = [
    [0x04, 0x66, 0x81, 0xe5],
    [0xe0, 0xcb, 0x19, 0x9a],
    [0x48, 0xf8, 0xd3, 0x7a],
    [0x28, 0x06, 0x26, 0x4c],
]

Estado después de AddRoundKey 1 = [
    [0xa4, 0x9c, 0x7f, 0xf2],
    [0x68, 0x9f, 0x35, 0x2b],
    [0x6b, 0x5b, 0xea, 0x43],
    [0x02, 0x6a, 0x50, 0x49],
]
```

Imagen 2 y 3: Primera ronda en nuestra implementación

Lo que a su vez nos permitió comprobar el cifrado completo:



Output

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

Ciphertext

Imagen 4: Output de la animación

```
Estado después de AddRoundKey 10 = [  
  [0x39, 0x25, 0x84, 0x1d],  
  [0x02, 0xdc, 0x09, 0xfb],  
  [0xdc, 0x11, 0x85, 0x97],  
  [0x19, 0x6a, 0x0b, 0x32],  
]  
  
Output del cifrado: = [  
  [0x39, 0x25, 0x84, 0x1d],  
  [0x02, 0xdc, 0x09, 0xfb],  
  [0xdc, 0x11, 0x85, 0x97],  
  [0x19, 0x6a, 0x0b, 0x32],  
]
```

Imagen 5: Output de nuestra implementación

Referencias

1. [GitHub - NIU1638962/Practica_AES: Practica de l'assignatura de Criptografia i Seguretat on s'implementa l'algoritme AES.](#)
2. [https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html](#)

Anexo

Código

AddRoundKey.py

```
from AES_constants_and_tools import CAD_DEBUG, KEY, print_matrix

def add_round_key(state, round_key):
    """Aplica la operación AddRoundKey al estado dado."""
    for i in range(4):
        for j in range(4):
            state[i][j] ^= round_key[i][j]
    return state

if __name__ == "__main__":
    # Usando CAD_DEBUG y KEY directamente desde el archivo auxiliar
    estado = CAD_DEBUG

    print_matrix(estado, "Estado sin cifrar:\n")

    # Imprimir la round_key
    print_matrix(KEY, "\nRound Key:\n")

    # Aplicar AddRoundKey
    new_state = add_round_key(estado, KEY)

    # Imprimir el estado después de AddRoundKey
    print_matrix(new_state, "\nEstado después de AddRoundKey:\n")
```


aes.py

```
# AES.py
from AES_constants_and_tools import KEY, CAD_DEBUG, t
from Key_Expantion import key_expantion
from AddRoundKey import add_round_key
from ByteSub import sub_bytes
from ShiftRows import shift_rows
from MixColumns import mix_columns

def write_matrix_to_file(file, matrix, title="Matriz"):
    file.write(f"{title} = [\n")
    for row in matrix:
        row_str = ', '.join(f"0x{x:02x}" for x in row)
        file.write(f"    [{row_str}],\n")
    file.write("]\n\n")

def aes_encrypt(plaintext, key, filename="aes_output.txt"):
    state = plaintext
    with open(filename, 'w') as file:
        # Aplana la matriz 4x4 de la clave para la expansión
        key_flat = [byte for row in key for byte in row]
        expanded_key = key_expantion(key_flat)
        file.write("Expansión de la llave:\n")
        write_matrix_to_file(file, [expanded_key[i:i+16] for i in
range(0, len(expanded_key), 16)], "Clave Expandida")

        # Genera las claves de ronda a partir de la clave expandida
        round_keys = []
        for r in range(11):
            round_key = expanded_key[16*r:16*(r+1)]
            round_key_matrix = [round_key[4*i:4*(i+1)] for i in
range(4)]
            round_keys.append(round_key_matrix)

        write_matrix_to_file(file, state, "Input:")
        write_matrix_to_file(file, key, "Clave Inicial:")

        # Rondas de cifrado AES
        state = add_round_key(state, round_keys[0]) # Ronda inicial
        write_matrix_to_file(file, state, "Estado después de
AddRoundKey 0")

        for i in range(1, 10):
```

```

        state = sub_bytes(state)
        write_matrix_to_file(file, state, f"Estado después de
ByteSub {i}")

        state = shift_rows(state)
        write_matrix_to_file(file, state, f"Estado después de
ShiftRows {i}")

        state = mix_columns(t(state))
        write_matrix_to_file(file, state, f"Estado después de
MixColumns {i}")

        state = add_round_key(state, round_keys[i])
        write_matrix_to_file(file, state, f"Estado después de
AddRoundKey {i}")

    # Última ronda (sin MixColumns)
    state = sub_bytes(state)
    write_matrix_to_file(file, state, f"Estado después de ByteSub
{10}")

    state = shift_rows(state)
    write_matrix_to_file(file, state, f"Estado después de ShiftRows
{10}")

    state = add_round_key(state, round_keys[10])
    write_matrix_to_file(file, state, f"Estado después de
AddRoundKey {10}")

    write_matrix_to_file(file, state, "Output del cifrado:")

# Ejemplo de uso
if __name__ == "__main__":
    plaintext = CAD_DEBUG
    aes_encrypt(plaintext, KEY)

```

AES_constants_and_tools.py

```
# Definición de la S-Box
S_BOX = [
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4,
0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5,
0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12,
0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B,
0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB,
0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9,
0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6,
0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7,
0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE,
0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3,
0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56,
0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD,
0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35,
0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E,
0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
]

# Tabla Rcon para 10 rondas de cifrado
RCON = [
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36
]
```

```

CAD = "JoelT AlejandroJ"

CAD_DEBUG = [
    [0x32, 0x43, 0xf6, 0xa8],
    [0x88, 0x5a, 0x30, 0x8d],
    [0x31, 0x31, 0x98, 0xa2],
    [0xe0, 0x37, 0x07, 0x34]
]

KEY = [
    [0x2b, 0x7e, 0x15, 0x16],
    [0x28, 0xae, 0xd2, 0xa6],
    [0xab, 0xf7, 0x15, 0x88],
    [0x09, 0xcf, 0x4f, 0x3c]
]

def print_matrix(matrix, title="Matriz"):
    """Imprime la matriz en formato de lista de Python con elementos
    hexadecimales."""
    print(f"{title} = [")
    for row in matrix:
        row_str = ', '.join(f"0x{x:02x}" for x in row)
        print(f"    [{row_str}],")
    print("]")

def t(matrix):
    """Transpone la matriz dada."""
    return [list(row) for row in zip(*matrix)]

```

ByteSub.py

```
from AES_constants_and_tools import S_BOX, print_matrix

def sub_bytes(state):
    """Aplica la operación SubBytes usando la S_BOX."""
    for i in range(4):
        for j in range(4):
            byte = state[i][j]
            # Accede a S_BOX para la sustitución
            state[i][j] = S_BOX[byte]
    return state

if __name__ == "__main__":
    # Estado después de AddRoundKey (ejemplo, reemplaza con tu estado
    real)
    estado = [
        [0x19, 0xa0, 0x9a, 0xe9],
        [0x3d, 0xf4, 0xc6, 0xf8],
        [0xe3, 0xe2, 0x8d, 0x48],
        [0xbe, 0x2b, 0x2a, 0x08],
    ]

    print_matrix(estado, "Estado antes de ByteSub:")

    # Aplicar SubBytes
    new_state = sub_bytes(estado)

    # Imprimir el estado después de SubBytes
    print_matrix(new_state, "Estado después de ByteSub:")
```

check_correct.py

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

# Configuración inicial
CAD = "JoelT AlejandroJ" # Mensaje a cifrar
clave = bytes([0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab,
0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c]) # Clave de 16 bytes

# Mensaje para cifrar
mensaje = CAD.encode() # Codifica la cadena CAD a bytes

# Cifrado con modo ECB
cipher = AES.new(clave, AES.MODE_ECB)
mensaje_cifrado = cipher.encrypt(pad(mensaje, AES.block_size)) # Añade
padding al mensaje y luego lo cifra

print("Mensaje Cifrado (en bytes):", mensaje_cifrado)

# Descifrado con modo ECB
cipher_dec = AES.new(clave, AES.MODE_ECB) # Se debe usar la misma
clave para el descifrado
mensaje_descifrado = unpad(cipher_dec.decrypt(mensaje_cifrado),
AES.block_size) # Descifra y elimina el padding

print("Mensaje Descifrado (string):", mensaje_descifrado.decode()) #
Decodifica de bytes a string
```

concatena.py

```
# Lista de archivos para concatenar
files_to_concatenate = [
    "ShiftRows.py",
    "AddRoundKey.py",
    "AES.py",
    "AES_constants_and_tools.py",
    "ByteSub.py",
    "Key_Expansion.py",
    "MixColumns.py"
]

# Nombre del archivo de salida
output_file_name = "concatenated_files.txt"

# Abrir el archivo de salida en modo de escritura
with open(output_file_name, "w") as output_file:
    # Iterar sobre cada archivo de la lista
    for file_name in files_to_concatenate:
        # Intentar abrir el archivo actual
        try:
            with open(file_name, "r") as current_file:
                # Escribir un encabezado para identificar el archivo
                # actual
                output_file.write(f"{'='*20}\n{file_name}\n{'='*20}\n\n")

                # Leer el contenido del archivo actual y escribirlo en
                # el archivo de salida
                output_file.write(current_file.read())

                # Escribir una nueva línea después de cada archivo para
                # separación
                output_file.write("\n\n")
        except FileNotFoundError:
            print(f"El archivo {file_name} no fue encontrado y será
            omitido.")

print(f"Todos los archivos han sido concatenados en
{output_file_name}.")
```

hex_to_dec.py

```
def decimal_to_hexadecimal(decimal_list):  
    """  
    Convierte una lista de valores decimales a sus equivalentes  
    hexadecimales.  
  
    Parámetros:  
    - decimal_list: Lista de números decimales.  
  
    Retorna:  
    - Lista de cadenas representando los números en formato  
    hexadecimal.  
    """  
    # Usa format() para convertir cada número decimal a hexadecimal,  
    con dos dígitos.  
    return [format(number, '02x') for number in decimal_list]  
  
decimal_list = [22, 166, 136, 60]  
hexadecimal_list = decimal_to_hexadecimal(decimal_list)  
print(hexadecimal_list)
```


idk.py

```
from Crypto.Cipher import AES
import binascii

# Clave proporcionada convertida a bytes
key_hex = '2b2878097eaef7cf15d2154f16a6883c'
key = binascii.unhexlify(key_hex)

# Texto proporcionado convertido a bytes
texto = "JoelT AlejandroJ".ljust(16)[:16] # Asegura que tiene
exactamente 16 caracteres
texto_bytes = texto.encode() # Codifica el texto a bytes

# Inicializa el cifrador AES con la clave y el modo ECB
cipher = AES.new(key, AES.MODE_ECB)

# Cifra el texto
ciphertext = cipher.encrypt(texto_bytes)

# Imprime el texto cifrado en formato hexadecimal
print("Texto cifrado (hex):", binascii.hexlify(ciphertext).decode())

# Ahora, compara este resultado con el que obteniste de tu función
`cifrar_aes`.

# Recuerda que debes convertir el resultado de tu función `cifrar_aes`
a formato hexadecimal para una comparación adecuada.
```

Key_Expansion.py

```
# Key_Expansion.py
from AES_constants_and_tools import S_BOX, RCON

def sub_word(word):
    """Aplica la S-Box a cada byte de una palabra."""
    return [S_BOX[b] for b in word]

def rot_word(word):
    """Realiza una rotación hacia la izquierda en una palabra."""
    return word[1:] + word[:1]

def key_expansion(key):
    key_size = 16 # Tamaño de clave para AES-128
    Nk = 4 # Número de palabras de 32 bits en la clave
    Nr = 10 # Número de rondas para AES-128
    expanded_key = key[:] # Comienza con la clave original

    for i in range(Nk, 4 * (Nr + 1)): # Expande hasta obtener 44
palabras de 32 bits
        temp = expanded_key[-4:] # Última palabra
        if i % Nk == 0:
            temp = sub_word(rot_word(temp)) # Aplica RotWord y SubWord
            temp[0] ^= RCON[i // Nk - 1] # Aplica XOR con el valor
RCON adecuado
            # XOR de la palabra generada con la palabra 4 posiciones antes
            expanded_key += [temp[j] ^ expanded_key[-key_size + j] for j in
range(4)]

    return expanded_key
```

LICENSE

MIT License

Copyright (c) 2024 Joel Tapia Salvador and Alejandro Jaime Cabrera

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

MixColumns.py

```
# MixColumns.py
from AES_constants_and_tools import print_matrix, t

def xtime(a):
    """Realiza la multiplicación por x (es decir, {02}) en GF(2^8)."""
    return (((a << 1) & 0xFF) ^ (0x1B if (a & 0x80) else 0x00))

def mix_single_column(a):
    """Mezcla una sola columna."""
    # Ver la especificación de AES para detalles sobre esta operación
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)
    return a

def mix_columns(state):
    """Aplica MixColumns a todo el estado."""
    for i in range(4):
        column = [state[row][i] for row in range(4)]
        column = mix_single_column(column)
        for row in range(4):
            state[row][i] = column[row]
    return t(state)

if __name__ == "__main__":

    estado = [
        [0xd4, 0xbf, 0x5d, 0x30],
        [0xe0, 0xb4, 0x52, 0xae],
        [0xb8, 0x41, 0x11, 0xf1],
        [0x1e, 0x27, 0x98, 0xe5],
    ]

    print_matrix(estado, "Estado antes de MixColumns:")

    # Aplicar MixColumns
    new_state = mix_columns(t(estado))

    # Imprimir el estado después de MixColumns
```

```
print_matrix(new_state, "Estado después de MixColumns:")
```

README.md

```
# Practica_AES
```

```
Practica de l'assignatura de Criptografia i Seguretat on s'implementa  
l'algoritme AES.
```

ShiftRows.py

```
from AES_constants_and_tools import print_matrix, t

def shift_rows(state):
    """Aplica la operación ShiftRows al estado dado."""
    # Transpone la matriz para trabajar con las columnas
    transposed_state = t(state)
    new_state = [row[:] for row in transposed_state] # Hacer una copia
del estado para evitar mutación in-place
    for i in range(4):
        new_state[i] = transposed_state[i][i:] +
transposed_state[i][:i] # Desplaza las 'filas' que ahora son columnas
    # Transpone de nuevo para restaurar la estructura original
    return t(new_state)

if __name__ == "__main__":
    estado = [
        [0xd4, 0x27, 0x11, 0xae],
        [0xe0, 0xbf, 0x98, 0xf1],
        [0xb8, 0xb4, 0x5d, 0xe5],
        [0x1e, 0x41, 0x52, 0x30],
    ]

    print("Estado antes de ShiftRows:")
    print_matrix(estado)

    # Aplicar ShiftRows
    new_state = shift_rows(estado)

    print("\nEstado después de ShiftRows:")
    print_matrix(new_state)
```

