Name: Tejas Redkar

Batch: C2; Roll no: 44

PRN: 1032210937

**ICS Lab Assignment 3**

**Lab A3:** Implement simplified AES symmetric key algorithm using python or java or C++

**Objective of Lab:**

1. To understand symmetric encryption techniques, block cipher encryption and how the S-AES algorithm works

**Theory :**

**Example of Using Simplified AES**

Let's encrypt the plaintext "1100110011001100" (16 bits) using a 16-bit key "1010101010101010."

1. Key Expansion:

- Initial Key: 1010101010101010

2. Round 1: Encryption

- Initial Round Key: 1010101010101010

- Add the Initial Round Key to the plaintext:

- Plaintext: 1100110011001100

- Round Key: +1010101010101010

- Result: 0110011001100110

- Apply an S-Box substitution:

- S-Box: (mapping is simple)

- 00 -> 10

- 11 -> 01

- 01 -> 00

- 10 -> 11

- Result after S-Box: 1001100101011000

3. Round 2: Encryption

- Swap the positions of the first and second bytes:

- Result after swapping: 0010110010011001

- Add the Round Key 2:

- Round Key 2: 0101010101010101

- Result: 0010110010011001 + 0101010101010101

- Result: 0110000111001100

The resulting ciphertext is "0110000111001100."

To decrypt, you would reverse the process by applying the decryption steps for S-AES, including reversing the byte swapping, the inverse S-Box operation, and subtracting the round keys.

**Algorithm:**

The Simplified Advanced Encryption Standard (S-AES) algorithm is a basic, educational variant of the widely used Advanced Encryption Standard (AES) encryption algorithm. It is designed to simplify the core concepts of AES and provide a clear illustration of how encryption and decryption work. S-AES is not suitable for secure real-world encryption due to its simplicity and lack of security. Here are the key components and steps of S-AES:

Block Size: S-AES typically uses a 16-bit (2-byte) block size, making it less secure compared to the 128-bit block size used in AES.

Key Size: S-AES uses an 8-bit (1-byte) key, which is significantly smaller than the 128, 192, or 256-bit keys used in AES.

Rounds: S-AES has two rounds, which is fewer than the 10, 12, or 14 rounds used in AES, depending on the key length.

Key Expansion: S-AES has a simplified key expansion process compared to AES. It involves rotating and applying a simple S-Box to the key to generate round keys for each round.

Substitution: S-AES uses a basic S-Box substitution operation, where each nibble (4 bits) of the block is substituted using a predefined substitution table.

Permutation: S-AES involves a permutation layer, which is a simple permutation of bytes in the block.

Encryption Process: The S-AES encryption process consists of adding the round key, applying the S-Box substitution, and then the permutation for each round.

Decryption Process: The S-AES decryption process is the reverse of the encryption process, involving the inverse S-Box substitution, reversing the permutation, and subtracting the round key.

**Code:**

lookup = {

    1: ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"],

    2: ["0", "2", "4", "6", "8", "A", "C", "E", "3", "1", "7", "5", "B", "9", "F", "D"],

    4: ["0", "4", "8", "C", "3", "7", "B", "F", "6", "2", "E", "A", "5", "1", "D", "9"],

    9: ["0", "9", "1", "8", "2", "B", "3", "A", "4", "D", "5", "C", "6", "F", "7", "E"],

}

def hex\_to\_bin(k, size=16):

    return bin(int(k, 16)).replace("0b", "").zfill(size)

def bin\_to\_hex(k, size=4):

    return hex(int(k, 2)).replace("0x", "").zfill(size)

def dec\_to\_hex(k, size=2):

    return hex(k).replace("0x", "").zfill(size)

def xor(a, b):

    return "".join([hex(int(a[i], 16) ^ int(b[i], 16))[2:] for i in range(len(a))])

def str\_to\_matrix(s):

    return [

        [int(s[:4], 2), int(s[8:12], 2)],

        [int(s[4:8], 2), int(s[12:], 2)]

    ]

def matrix\_to\_str(m):

    return hex\_to\_bin(m[0][0], 4) + hex\_to\_bin(m[1][0], 4) + hex\_to\_bin(m[0][1], 4) + hex\_to\_bin(m[1][1], 4)

def rotate\_nibble(x):

    return x[4:] + x[:4]

def substitute\_nibble(k):

    s\_box = [

        ["9", "4", "A", "B"],

        ["D", "1", "8", "5"],

        ["6", "2", "0", "3"],

        ["C", "E", "F", "7"]

    ]

    n = len(k) // 4

    res = ""

    for i in range(n):

        x = int(k[4 \* i: 4 \* i + 2], 2)

        y = int(k[4 \* i + 2: 4 \* i + 4], 2)

        res += hex\_to\_bin(s\_box[x][y], 4)

    return res

def inv\_substitute\_nibble(k):

    inv\_s\_box = [

        ["A", "5", "9", "B"],

        ["1", "7", "8", "F"],

        ["6", "0", "2", "3"],

        ["C", "4", "D", "E"]

    ]

    n = len(k) // 4

    res = ""

    for i in range(n):

        x = int(k[4 \* i: 4 \* i + 2], 2)

        y = int(k[4 \* i + 2: 4 \* i + 4], 2)

        res += hex\_to\_bin(inv\_s\_box[x][y], 4)

    return res

def generate\_keys(key):

    key\_in\_binary = hex\_to\_bin(key)

    w = []

    rcon = ["10000000", "00110000"]

    w.append(key\_in\_binary[:8])

    w.append(key\_in\_binary[8:])

    yield bin\_to\_hex(w[0] + w[1])

    for i in range(2):

        sub\_op = substitute\_nibble(rotate\_nibble(w[-1]))

        left = xor(sub\_op, xor(w[-2], rcon[i]))

        right = xor(left, w[-1])

        w.extend([left, right])

        yield bin\_to\_hex(left + right)

def shift\_rows(matrix):

    return matrix[:4] + matrix[12:] + matrix[8:12] + matrix[4:8]

def inv\_shift\_rows(matrix):

    return matrix[:4] + matrix[12:] + matrix[8:12] + matrix[4:8]

def mix\_columns(s):

    s = str\_to\_matrix(s)

    M = [[1, 4], [4, 1]]

    \_s = [[0, 0], [0, 0]]

    def mmul(i, j):

        return lookup[i][j]

    \_s[0][0] = xor(mmul(M[0][0], s[0][0]), mmul(M[0][1], s[1][0]))

    \_s[0][1] = xor(mmul(M[0][0], s[0][1]), mmul(M[0][1], s[1][1]))

    \_s[1][0] = xor(mmul(M[1][0], s[0][0]), mmul(M[1][1], s[1][0]))

    \_s[1][1] = xor(mmul(M[1][0], s[0][1]), mmul(M[1][1], s[1][1]))

    return matrix\_to\_str(\_s)

def inv\_mix\_columns(s):

    s = str\_to\_matrix(s)

    M = [[9, 2], [2, 9]]

    \_s = [[0, 0], [0, 0]]

    def mmul(i, j):

        return lookup[i][j]

    \_s[0][0] = xor(mmul(M[0][0], s[0][0]), mmul(M[0][1], s[1][0]))

    \_s[0][1] = xor(mmul(M[0][0], s[0][1]), mmul(M[0][1], s[1][1]))

    \_s[1][0] = xor(mmul(M[1][0], s[0][0]), mmul(M[1][1], s[1][0]))

    \_s[1][1] = xor(mmul(M[1][0], s[0][1]), mmul(M[1][1], s[1][1]))

    return matrix\_to\_str(\_s)

def enc\_round\_i(ci, k, round):

    if round == 0:

        return xor(ci, k)

    subNib = substitute\_nibble(hex\_to\_bin(ci))

    shRow = shift\_rows(subNib)

    mixCol = shRow

    if round != 2:

        mixCol = mix\_columns(shRow)

    return xor(bin\_to\_hex(mixCol), k)

def dec\_round\_i(ci, k, round):

    if round == 0:

        return xor(ci, k)

    shRow = inv\_shift\_rows(hex\_to\_bin(ci))

    subNib = inv\_substitute\_nibble(shRow)

    addKey = xor(subNib, hex\_to\_bin(k))

    mixCol = addKey

    if round != 2:

        mixCol = inv\_mix\_columns(addKey)

    # print(mixCol)

    return bin\_to\_hex(mixCol)

key = "4AF5"

keys = [k for k in generate\_keys(key)]

def encrypt():

    plain\_text = input("Enter 16-bit plain text in hexadecimal: ").upper()

    current\_iteration = plain\_text

    # 1101 0111 0010 1000

    for i, keyI in enumerate(keys):

        current\_iteration = enc\_round\_i(current\_iteration, keyI, i)

    cipher\_text = current\_iteration.upper()

    print("Cipher text: ", cipher\_text)

def decrypt():

    cipher\_text = input("Enter 16-bit cipher text in hexadecimal: ").upper()

    current\_iteration = cipher\_text

    # 1101 0111 0010 1000

    for i, keyI in enumerate(keys[::-1]):

        current\_iteration = dec\_round\_i(current\_iteration, keyI, i)

    plain\_text = current\_iteration.upper()

    print("Plain text: ", plain\_text)

c = input("Enter choice: \n1)Encrypt \n2)Decrypt \n")

if c == "1":

    encrypt()

elif c == "2":

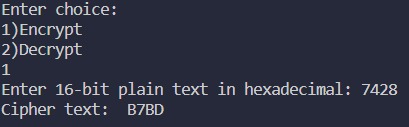
    decrypt()

else:

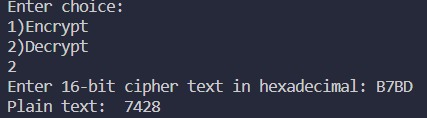
    print("Exiting...")

**Output Screen shots**:

Encrypt:



Decrypt:



**Conclusion**:

We successfully implemented simplified AES symmetric key algorithm, in Python. This implementation provides a practical demonstration of how this algorithm works for encrypting and decrypting messages.

# FAQs:

# 

# What is S-AES algorithm and how it is different from AES algorithm

# Explain Key generation in S-AES

# Explain Encryption in S-AES

# Explain Decryption in S-AES