Name: Sagar Patil
Roll No: 8061

Assignment No.2

Title:

Implementation of S-AES (Advanced Encryption Standard)

Problem Definition:

Implementation of S-AES

Prerequisite:

Basic of Python3, Concept of Advanced Encryption Standard

Software Requirements:

Python 3

Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn How to Apply Advanced Encryption Standard Algorithm to encryption of given data.

Outcomes:

After completion of this assignment students are able Implement code for **Advanced Encryption Standard Algorithm** for given data and find the encrypted data of the given data.

Theory Concepts:

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six times faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C ,Java and Python

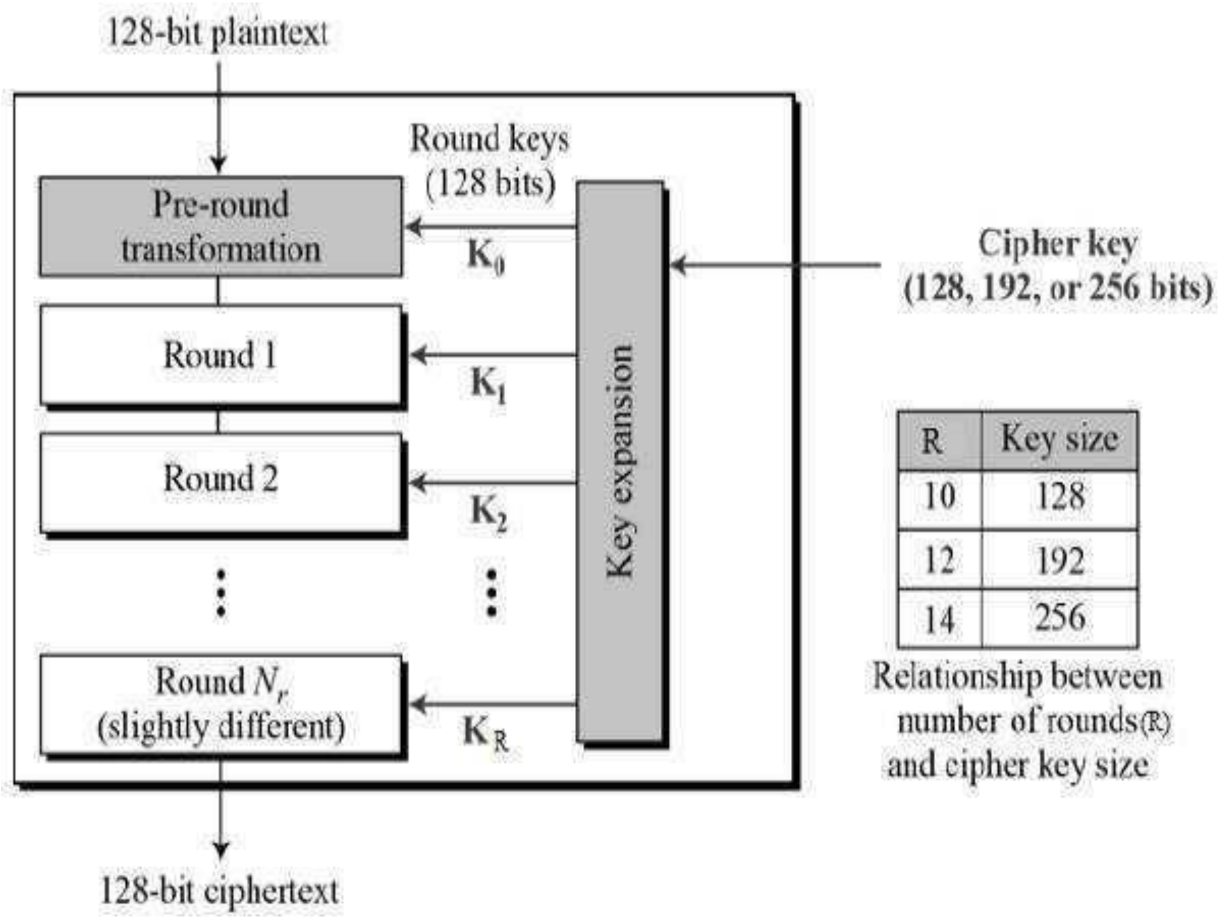
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −



R	Key size
10	128
12	192
14	256

Figure 6.1 AES structure

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −

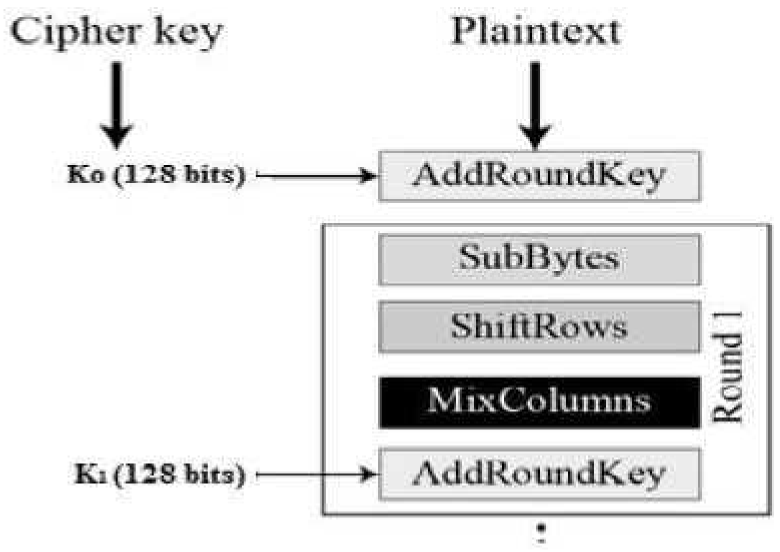


Figure 6.2 Encryption Process

Conclusion: Successfully learned and implemented S-AES algorithms.

Code:

```
import sys

# S-Box
sBox = [0x9, 0x4, 0xa, 0xb, 0xd, 0x1, 0x8, 0x5,
        0x6, 0x2, 0x0, 0x3, 0xc, 0xe, 0xf, 0x7]

# Inverse S-Box
sBoxI = [0xa, 0x5, 0x9, 0xb, 0x1, 0x7, 0x8, 0xf,
         0x6, 0x0, 0x2, 0x3, 0xc, 0x4, 0xd, 0xe]

# Round keys: K0 = w0 + w1; K1 = w2 + w3; K2 = w4 + w5
w = [None] * 6

def mult(p1, p2):
    """Multiply two polynomials in GF(2^4)/x^4 + x + 1"""
    p = 0
    while p2:
        if p2 & 0b1:
            p ^= p1
        p1 <<= 1
        if p1 & 0b10000:
            p1 ^= 0b11
        p2 >>= 1
    return p & 0b1111

def intToVec(n):
    """Convert a 2-byte integer into a 4-element vector"""
    return [n >> 12, (n >> 4) & 0xf, (n >> 8) & 0xf,  n & 0xf]

def vecToInt(m):
    """Convert a 4-element vector into 2-byte integer"""
    return (m[0] << 12) + (m[2] << 8) + (m[1] << 4) + m[3]

def addKey(s1, s2):
    """Add two keys in GF(2^4)"""
    return [i ^ j for i, j in zip(s1, s2)]

def sub4NibList(sbox, s):
    """Nibble substitution function"""
    return [sbox[e] for e in s]

def shiftRow(s):
    """ShiftRow function"""
    return [s[0], s[1], s[3], s[2]]

def keyExp(key):
    """Generate the three round keys"""
    def sub2Nib(b):
        """Swap each nibble and substitute it using sBox"""
        return sBox[b >> 4] + (sBox[b & 0x0f] << 4)

    Rcon1, Rcon2 = 0b10000000, 0b00110000
    w[0] = (key & 0xff00) >> 8
    w[1] = key & 0x00ff
    w[2] = w[0] ^ Rcon1 ^ sub2Nib(w[1])
    w[3] = w[2] ^ w[1]
    w[4] = w[2] ^ Rcon2 ^ sub2Nib(w[3])
    w[5] = w[4] ^ w[3]

def encrypt(ptext):
    """Encrypt plaintext block"""
    def mixCol(s):
        return [s[0] ^ mult(4, s[2]), s[1] ^ mult(4, s[3]),
                s[2] ^ mult(4, s[0]), s[3] ^ mult(4, s[1])]

    state = intToVec(((w[0] << 8) + w[1]) ^ ptext)
    state = mixCol(shiftRow(sub4NibList(sBox, state)))
    state = addKey(intToVec((w[2] << 8) + w[3]), state)
    state = shiftRow(sub4NibList(sBox, state))
    return vecToInt(addKey(intToVec((w[4] << 8) + w[5]), state))

def decrypt(ctext):
    """Decrypt ciphertext block"""
    def iMixCol(s):
        return [mult(9, s[0]) ^ mult(2, s[2]), mult(9, s[1]) ^ mult(2, s[3]),
                mult(9, s[2]) ^ mult(2, s[0]), mult(9, s[3]) ^ mult(2, s[1])]

    state = intToVec(((w[4] << 8) + w[5]) ^ ctext)
    state = sub4NibList(sBoxI, shiftRow(state))
    state = iMixCol(addKey(intToVec((w[2] << 8) + w[3]), state))
    state = sub4NibList(sBoxI, shiftRow(state))
    return vecToInt(addKey(intToVec((w[0] << 8) + w[1]), state))

if__name__== '_main_':


    plaintext = 0b1101011100101000
    key = 0b0100101011110101
    ciphertext = 0b0010010011101100
    keyExp(key)
    try:
        assert encrypt(plaintext) == ciphertext
    except AssertionError:
        print("Encryption error")
        print(encrypt(plaintext), ciphertext)
        sys.exit(1)
    try:
        assert decrypt(ciphertext) == plaintext
    except AssertionError:
        print("Decryption error")
        print(decrypt(ciphertext), plaintext)
        sys.exit(1)
    print("Test ok!")
    sys.exit()
```