1. **input.py**

class SimplifiedAES:

    """

    Simplified AES (S-AES) Implementation

    """

    # 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]

    def \_\_init\_\_(self, key):

        self.pre\_round\_key, self.round1\_key, self.round2\_key = self.key\_expansion(key)

        print("Generated Keys:")

        print("Pre-Round Key  :", self.pre\_round\_key)

        print("Round 1 Key    :", self.round1\_key)

        print("Round 2 Key    :",self.round2\_key)

        print("-" \* 40)

    def sub\_word(self, word):

        return (self.sBox[(word >> 4)] << 4) + self.sBox[word & 0x0F]

    def rot\_word(self, word):

        return ((word & 0x0F) << 4) + ((word & 0xF0) >> 4)

    def key\_expansion(self, key):

        Rcon1, Rcon2 = 0x80, 0x30

        w = [None] \* 6

        w[0], w[1] = (key & 0xFF00) >> 8, key & 0x00FF

        w[2] = w[0] ^ (self.sub\_word(self.rot\_word(w[1])) ^ Rcon1)

        w[3] = w[2] ^ w[1]

        w[4] = w[2] ^ (self.sub\_word(self.rot\_word(w[3])) ^ Rcon2)

        w[5] = w[4] ^ w[3]

        return [self.int\_to\_state((w[i] << 8) + w[i+1]) for i in range(0, 6, 2)]

    def gf\_mult(self, a, b):

        product = 0

        a, b = a & 0x0F, b & 0x0F

        while a and b:

            if b & 1:

                product ^= a

            a <<= 1

            if a & (1 << 4):

                a ^= 0b10011

            b >>= 1

        return product

    def int\_to\_state(self, n):

        return [n >> 12 & 0xF, (n >> 4) & 0xF, (n >> 8) & 0xF, n & 0xF]

    def state\_to\_int(self, m):

        return (m[0] << 12) + (m[2] << 8) + (m[1] << 4) + m[3]

    def add\_round\_key(self, s1, s2):

        return [i ^ j for i, j in zip(s1, s2)]

    def sub\_nibbles(self, sbox, state):

        return [sbox[nibble] for nibble in state]

    def shift\_rows(self, state):

        return [state[0], state[1], state[3], state[2]]

    def mix\_columns(self, state):

        return [

            state[0] ^ self.gf\_mult(4, state[2]),

            state[1] ^ self.gf\_mult(4, state[3]),

            state[2] ^ self.gf\_mult(4, state[0]),

            state[3] ^ self.gf\_mult(4, state[1]),

        ]

    def inverse\_mix\_columns(self, state):

        return [

            self.gf\_mult(9, state[0]) ^ self.gf\_mult(2, state[2]),

            self.gf\_mult(9, state[1]) ^ self.gf\_mult(2, state[3]),

            self.gf\_mult(9, state[2]) ^ self.gf\_mult(2, state[0]),

            self.gf\_mult(9, state[3]) ^ self.gf\_mult(2, state[1]),

        ]

    def encrypt(self, plaintext):

        print("---------------- Encryption ----------------------")

        state = self.int\_to\_state(plaintext)

        print("Initial State: ", state)

        state = self.add\_round\_key(self.pre\_round\_key, state)

        print("After AddRoundKey 1: ", state)

        state = self.sub\_nibbles(self.sBox, state)

        print("After SubNibbles: ", state)

        state = self.shift\_rows(state)

        print("After ShiftRows: ", state)

        state = self.mix\_columns(state)

        print("After MixColumns: ", state)

        state = self.add\_round\_key(self.round1\_key, state)

        print("After AddRoundKey 2: ", state)

        state = self.sub\_nibbles(self.sBox, state)

        print("After SubNibbles: ", state)

        state = self.shift\_rows(state)

        print("After ShiftRows: ", state)

        state = self.add\_round\_key(self.round2\_key, state)

        print("After AddRoundKey 3: ", state)

        return self.state\_to\_int(state)

    def decrypt(self, ciphertext):

        print("---------------- Decryption ----------------------")

        state = self.int\_to\_state(ciphertext)

        print("Initial Ciphertext State: ", state)

        state = self.add\_round\_key(self.round2\_key, state)

        print("After AddRoundKey 3: ", state)

        state = self.shift\_rows(state)

        print("After Inverse ShiftRows: ", state)

        state = self.sub\_nibbles(self.sBoxI, state)

        print("After Inverse SubNibbles: ", state)

        state = self.add\_round\_key(self.round1\_key, state)

        print("After AddRoundKey 2: ", state)

        state = self.inverse\_mix\_columns(state)

        print("After Inverse MixColumns: ", state)

        state = self.shift\_rows(state)

        print("After Inverse ShiftRows: ", state)

        state = self.sub\_nibbles(self.sBoxI, state)

        print("After Inverse SubNibbles: ", state)

        state = self.add\_round\_key(self.pre\_round\_key, state)

        print("After AddRoundKey 1: ", state)

        return self.state\_to\_int(state)

plaintext = int(input("Enter plaintext (16-bit binary): "), 2)

key = int(input("Enter key (16-bit binary): "), 2)

print("PlainText: ", format(plaintext,'016b'))

print("Key: ", format(key,'016b'))

ciphertext = SimplifiedAES(key).encrypt(plaintext)

print("After Encryption (Ciphertext) :", format(ciphertext, '016b'))

print("\n\n\n")

decrypted\_text = SimplifiedAES(key).decrypt(ciphertext)

print("After Decryption (Plaintext) :", format(decrypted\_text, '016b'))

1. **Output**

