CRYPTOGRAPHY AND NETWORK SECURITY
LAB – 4 - AES

Name of the Student: SreeDananjay S
Registration Number: 21BAI1807
Slot: L31+L32
Course Code: BCSE309P
Programme: Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning
School: School of Computer Science and Engineering(SCOPE)

Q) AES algorithm implementation

Consider a sender and receiver who need to exchange data confidentially using symmetric encryption. Write program that implements AES encryption and decryption using a 64/128/256 bits key size and 64 bit block size

Code:

```
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,
```

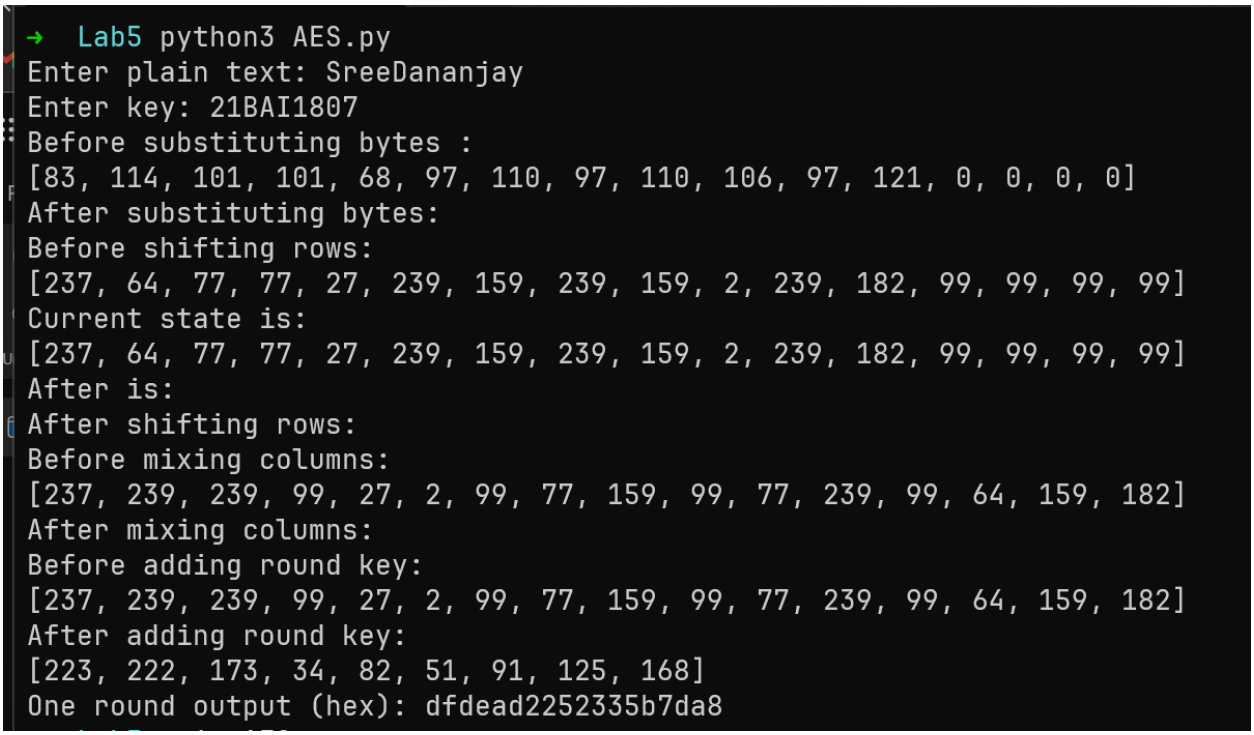
```
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4,
0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae,
0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b,
0x8a,
    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
]

# Constants
BLOCK_SIZE = 16  # AES block size in bytes (128 bits)
KEY_SIZE = 16    # AES key size in bytes (128 bits)

# Helper function to convert string to bytes
def string_to_bytes(text):
    return [ord(c) for c in text]

# Key expansion (Key Schedule) - simple version for one round
def key_expansion(key):
    # Expand the key into a round key for this example (simple replication)
    return key[:BLOCK_SIZE]  # Keep it 16 bytes for simplicity

# SubBytes step using the AES S-Box
```

```
def sub_bytes(state):
    return [S_BOX[b] for b in state]

# ShiftRows step (simplified for educational purposes)
def shift_rows(state):
    print("Current state is: ")
    print(state)
    print("After is: ")
    return [
        state[0], state[5], state[10], state[15],
        state[4], state[9], state[14], state[3],
        state[8], state[13], state[2], state[7],
        state[12], state[1], state[6], state[11]
    ]

# MixColumns step (simplified for educational purposes)
def mix_columns(state):
    # This is a simplified version and does not implement the full matrix multiplication in GF(2^8)
    return state  # No actual transformation for simplicity

# AddRoundKey step - XOR the state with the round key
def add_round_key(state, round_key):
    return [s ^ k for s, k in zip(state, round_key)]

# AES one round operation
```

```
def aes_one_round(plaintext, key):
    # Convert plaintext and key to bytes
    plaintext_bytes = string_to_bytes(plaintext)
    key_bytes = string_to_bytes(key)

    # Pad plaintext to 16 bytes if necessary
    if len(plaintext_bytes) < BLOCK_SIZE:
        plaintext_bytes += [0] * (BLOCK_SIZE - len(plaintext_bytes))

    # Key expansion (for simplicity, no real key scheduling)
    round_key = key_expansion(key_bytes)

    # Perform one round of AES (SubBytes, ShiftRows, MixColumns, AddRoundKey)
    state = plaintext_bytes[:BLOCK_SIZE]  # First block of the plaintext
    print("Before substituting bytes : ")
    print(state)
    state = sub_bytes(state)
    print("After substituting bytes: ")
    print("Before shifting rows: ")
    print(state)
    state = shift_rows(state)
    print("After shifting rows: ")
    print("Before mixing columns: ")
    print(state)
    state = mix_columns(state)
    print("After mixing columns: ")
```

```
    print("Before adding round key: ")
    print(state)
    state = add_round_key(state, round_key)
    print("After adding round key: ")
    print(state)
    return state

# Test with Name and Roll Number
pt = input("Enter plain text: ")
key = input("Enter key: ")

# Perform one round of AES
one_round_output = aes_one_round(pt, key)
print("One round output (hex):", ''.join([f'{b:02x}' for b in one_round_output]))
```

Output Screenshots:



Result:

Thus, the DES algorithm has been successfully executed and verified.