Project 1: AES_IMPLEMENTATION

Krishna Somani 2021058
Mayank Gupta 2021065

Our implementation contains 2 files table.py and aes_implementation.py.

tables.py

The r_con table provides round constant values used in key expansion, while s_box and inv_s_box represent the substitution boxes for forward and inverse substitutions. These tables are crucial for the byte substitution and key schedule generation steps in AES, contributing to its cryptographic strength.

```
def get_r_con_value(index):

    return r_con[index]

def get_inv_s_box_value(row, col):

    return inv_s_box[row * 16 + col]

def get_s_box_value(row, col):

    return s_box[row * 16 + col]
```

This file is just for easy access and keeping the code more organized.

AES_implementation.py

In this AES encryption implementation, a 128-bit key ('ABCDabcd12344321') is utilized to encrypt three separate 128-bit plaintexts. The primary focus is on ensuring that the output of the first round during encryption and the ninth round during decryption match.
We tested our system with three different inputs each of 16 bytes or 128 bits each and we have displayed the outputs in the screenshots below.
In the output we have displayed the state matrix after each encryption and decryption stages for better readability and understanding of it. The ciphertext after the 9th state output in encryption indicates the final ciphertext after 10th round.

1)

```
Input : Two One Nine Two
The intial Plaintext taken : b'Two One Nine Two'
The key used for AES : b'ABCDabcd12344321'

Initial_state : 15352c642e0c06447f5b5d511467455e

1th state : a3df6cd69f3f1337c990bda9a84fe776
2th state : 6f485746ab81d48c728af9158ef7f955
3th state : 6b8b775796027aacf994c3508fbd332a
4th state : 76383d9babd20cdd262900cd7f2fb8fc
5th state : a948aa781959a033aea739830b3f9eb5
6th state : dc4498aaf4cb808300e97eee6527dc25
7th state : 8086aedf240fca604e4a8914820e03b3
8th state : 06ded0ffa2f0d1633b8ee6ccdcd559c1
9th state : 4de27cfc66def703455f53732ec13bd9

Ciphertext : b6c941e5ae3bcc5b3abfbbc6656128b2

----------------------------------------------------

Starting Decryption ...
Decryption starting state : b6c941e5ae3bcc5b3abfbbc6656128

1th state : 4de27cfc66def703455f53732ec13bd9
2th state : 06ded0ffa2f0d1633b8ee6ccdcd559c1
3th state : 8086aedf240fca604e4a8914820e03b3
4th state : dc4498aaf4cb808300e97eee6527dc25
5th state : a948aa781959a033aea739830b3f9eb5
6th state : 76383d9babd20cdd262900cd7f2fb8fc
7th state : 6b8b775796027aacf994c3508fbd332a
8th state : 6f485746ab81d48c728af9158ef7f955
9th state : a3df6cd69f3f1337c990bda9a84fe776

Final state : 15352c642e0c06447f5b5d511467455e

----------------------------------------------------

Original Plaintext : Two One Nine Two
```

Here we can see the 1st state of encryption matches the 9th state of decryption and output of 9th encryption round is same as output of the 1st decryption round.

2)

```
Input : Four Five Eleven
The intial Plaintext taken : b'Four Five Eleven'
The key used for AES : b'ABCDabcd12344321'

Initial_state : 072d363641240a12541276585145575f

1th state : 60059cdf6c38a1d1f57486873e9dc97e
2th state : 2282c798b1113cfc9098785700e0362b
3th state : 5f9208fe111c011dadedec6d0e58f2ed
4th state : bed8553cb0231d4c37740e7b2d4e2e6b
5th state : 109bed81a7d8363ec0ccee5ff7f7c875
6th state : 6d71e54576abaef95860e2f3ce2dd37e
7th state : 612fd4b25738e6ae30be9c02a4891af5
8th state : b418002d0c5f85e416f3cc11ea27e0b8
9th state : 848041e556bd756e25f33263e0b7d57f

Ciphertext : 0a9cd28cc9ce28d9f4cbea0482b1f953

-----------------------------------------------------

Starting Decryption ...
Decryption starting state : 0a9cd28cc9ce28d9f4cbea0482b1f953

1th state : 848041e556bd756e25f33263e0b7d57f
2th state : b418002d0c5f85e416f3cc11ea27e0b8
3th state : 612fd4b25738e6ae30be9c02a4891af5
4th state : 6d71e54576abaef95860e2f3ce2dd37e
5th state : 109bed81a7d8363ec0ccee5ff7f7c875
6th state : bed8553cb0231d4c37740e7b2d4e2e6b
7th state : 5f9208fe111c011dadedec6d0e58f2ed
8th state : 2282c798b1113cfc9098785700e0362b
9th state : 60059cdf6c38a1d1f57486873e9dc97e

Final state : 072d363641240a12541276585145575f

-----------------------------------------------------

Original Plaintext : Four Five Eleven

-----------------------------------------------------
```

3)

```
Input : Ones Twos Threes
The intial Plaintext taken : b'Ones Twos Threes'
The key used for AES : b'ABCDabcd12344321'

Initial_state : 0e2c26374136140b4212675c46565742

1th state : 31b01e135417d0fbf89d0377ba090414
2th state : cc9a215cc3417b6db846bfd0a8abb677
3th state : 254619053cfbf88fb3074fb536d09755
4th state : 4ee5087d74c1bddaeb74ed8f968f1e3d
5th state : 022a6b4569d6d596a203a4678098a50f
6th state : 4c295b86cef38f2ef43e82fa370be814
7th state : 8b27623f30329b5dea7320b58b51e3ed
8th state : db1c12ff06e891049b7a79c9ce550509
9th state : dbfd42a5cee2b226a19254c6941851fb

Ciphertext : ec057d532b6440e3f784e7465f72fd81

-----------------------------------------------------

Starting Decryption ...
Decryption starting state : ec057d532b6440e3f784e7465f72fd81

1th state : dbfd42a5cee2b226a19254c6941851fb
2th state : db1c12ff06e891049b7a79c9ce550509
3th state : 8b27623f30329b5dea7320b58b51e3ed
4th state : 4c295b86cef38f2ef43e82fa370be814
5th state : 022a6b4569d6d596a203a4678098a50f
6th state : 4ee5087d74c1bddaeb74ed8f968f1e3d
7th state : 254619053cfbf88fb3074fb536d09755
8th state : cc9a215cc3417b6db846bfd0a8abb677
9th state : 31b01e135417d0fbf89d0377ba090414

Final state : 0e2c26374136140b4212675c46565742

-----------------------------------------------------

Original Plaintext : Ones Twos Threes

-----------------------------------------------------
```

AES implementation has three main functions which are key expansion , encryption and decryption.

1)Key Expansion:

The Key Expansion process involves transforming a short key into an expanded key schedule. This ensures that each round of AES encryption uses a unique key. The keyExpansion function in the implementation utilizes the Rijndael key schedule, combining rotation, substitution, and XOR operations. The r_con table provides round constant values, and the S-box (s_box) is applied for byte substitution.

```
def keyExpansion(key):
    key = bytes2matrix(key)
    Nr = 10
    Nk = 4
    i = 0

    while len(key) < 4 * (Nr + 1):
        word = list(key[-1])
        if i % 4 == 0:
            word.append(word.pop(0))
            for index in range(len(word)):
                word[index] = get_s_box_value(word[index] // 0x10,
word[index] % 0x10)
            word[0] ^= get_r_con_value(i)
            i += 1
        word = xor_bytes(word, key[-4])
        key.append(list(word))

    return key
```

2)Encryption:

In the encryption process, the plaintext is transformed into ciphertext through a series of rounds. The encrypt function applies SubBytes, ShiftRows, MixColumns, and AddRoundKey operations. SubBytes involves substituting each byte with a corresponding value from the S-box. ShiftRows shifts the rows of the state matrix, MixColumns performs a column-wise mixing operation, and AddRoundKey XORs the state with a round key.

```python
def shift_rows(state):
    state[1] = [state[1][1], state[1][2], state[1][3], state[1][0]]
    state[2] = [state[2][2], state[2][3], state[2][0], state[2][1]]
    state[3] = [state[3][3], state[3][0], state[3][1], state[3][2]]
    return state
```

```python
def addRoundKey(state, key):
    Nk = 4
    new_state = []
    for i in range(Nk):
        temp_row = []
        for j in range(Nk):
            temp_row.append(state[i][j] ^ key[i][j])
        new_state.append(temp_row)
    return new_state
```

```python
def sub_bytes(s):
    new_state = []
    for i in range(4):
        temp_row = []
        for j in range(4):
            temp_row.append(get_s_box_value(s[i][j] // 0x10, s[i][j] % 0x10))
        new_state.append(temp_row)
    return new_state
```

```python
def mix_columns(state):
    new_state = []
    for i in range(4):
        new_column = []
        for j in range(4):
            temp_xor = state[i][j % 4] ^ state[i][0] ^ state[i][1] ^ state[i][2] ^ state[i][3] ^ xtime(state[i][j % 4] ^ state[i][(j + 1) % 4])
            new_column.append(temp_xor)
        new_state.append(new_column)
    return new_state
```

```python
def encrypt(plaintext, key):

    state=bytes2matrix(plaintext)
    expanded_key=keyExpansion(key)

    state=addRoundKey(state,expanded_key[0:4])
    print(f"Initial_state : {matrix2bytes(state).hex()}")
    round_states = {}
    print()
    for round in range(1,10):
        state=sub_bytes(state)
        state = shift_rows(state)
        state = mix_columns(state)
        state=addRoundKey(state,expanded_key[4*round:4*(round+1)])
        new_state=state
        round_states[round]=new_state
```

```
        print(f"{round}th state : {matrix2bytes(new_state).hex()}")

    state = sub_bytes(state)
    state = shift_rows(state)
    state = addRoundKey(state, expanded_key[10*4:11*4])
    print()
    return state,round_states
```

3)Decryption:

Decryption is the reverse process of encryption. The decrypt function applies the inverse operations of encryption: InvShiftRows, InvSubBytes, InvMixColumns, and AddRoundKey. The decryption process ensures the original plaintext is reconstructed accurately.

```
def inv_sub_bytes(s):
    new_state = []
    for i in range(4):
        temp_row = []
        for j in range(4):
            temp_row.append(get_inv_s_box_value(s[i][j] // 0x10, s[i][j] %
0x10))
        new_state.append(temp_row)
    return new_state

def inv_shift_rows(state):
    state[1] = [state[1][3], state[1][0], state[1][1], state[1][2]]
    state[2] = [state[2][2], state[2][3], state[2][0], state[2][1]]
    state[3] = [state[3][1], state[3][2], state[3][3], state[3][0]]
    return state

def inv_mix_columns(state):
    for i in range(4):
        x = xtime(state[i][0] ^ state[i][2])
        y = xtime(state[i][1] ^ state[i][3])
        state[i][0] ^= xtime(x)
        state[i][1] ^= xtime(y)
        state[i][2] ^= xtime(x)
        state[i][3] ^= xtime(y)
```

```
    state=mix_columns(state)
    return state
```

```
def decrypt(ciphertext, key):

    print("Starting Decryption ...")

    state = ciphertext
    expanded_key = keyExpansion(key)
    print(f"Decryption starting state : {matrix2bytes(state).hex()}")
    print()
    state = addRoundKey(state,expanded_key[10*4:11*4])
    state = inv_shift_rows(state)
    state = inv_sub_bytes(state)
    round_states = {}

    for round in range(9, 0, -1):
        print(f"{10-round}th state : {matrix2bytes(state).hex()}")
        state = addRoundKey(state, expanded_key[round*4:(round+1)*4])
        state = inv_mix_columns(state)
        state = inv_shift_rows(state)
        state = inv_sub_bytes(state)
        round_states[round] = state
    print()
    print(f"Final state : {matrix2bytes(state).hex()}")
    state = addRoundKey(state, expanded_key[0:4])
    return state,round_states
```