NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

(An Institute of National Importance under MHRD, Govt. of India)
KARAIKAL – 609 609

Department of Computer Science and Engineering

Assignment Report

AES Encryption in CBC Mode

NAME:SATIRTHA SEN

ROLL:CS22B1066

COURSE:NETWORK SECURITY

COURSE CODE:CS1702

DEGREE: B.TECH

BATCH: 2022-2026

SEMESTER: 6th

ACADEMIC YEAR:2024-2025

DATE:26.04.2025

TO:
Submitted by:
DR.NARENDRAN RAJAGOPALAN
ASSOCIATE PROFESSOR
DEPT. of CSE

Assignment

SATIRTHA SEN
THIRD YEAR
DEPT. of CSE

1. Objective

The goal of this assignment is to manually implement AES-128 encryption and decryption in CBC mode using Python. The implementation focuses on understanding the internal workings of AES such as SubBytes, ShiftRows, MixColumns, AddRoundKey, key expansion, and block chaining logic.

2. Implementation Summary

- AES S-Box, inverse S-Box, and RCON are hardcoded.

- Key expansion produces 11 round keys from a 128-bit input key.

- Each block undergoes 10 rounds of transformations (SubBytes, ShiftRows, MixColumns).

- CBC mode adds XOR chaining using an Initialization Vector (IV).

- Padding is done with the PKCS#7 scheme.

3.Design Features

- Plaintext is taken from user input.

- Key is derived using SHA-256 (first 16 bytes).

- IV is generated randomly per session.

- The full cipher text is printed in hexadecimal format.

- Decryption successfully recovers the original plaintext and confirms encryption accuracy.

3. Key Expansion

The key_expansion function generates round keys for AES-128. It uses RCON and the S-Box to transform the original28-bit key into 11 round keys (each 16 bytes).

```
def key_expansion(key):
    key_symbols = list(key)
    assert len(key_symbols) == 16
    key_schedule = key_symbols.copy()
    for i in range(4, 44):
        temp = key_schedule[4*(i-1):4*i]
        if i % 4 == 0:
            temp = [S_BOX[b] for b in temp[1:] + temp[:1]]
            temp[0] ^= RCON[i//4]
        for j in range(4):
            key_schedule.append(key_schedule[4*(i-4)+j] ^ temp[j])
    return [key_schedule[16*i:16*(i+1)] for i in range(11)]
```

4. AES Encryption Block

This function applies the main AES encryption logic on a 16-byte block. It goes through 10 rounds, performing SubBytes, ShiftRows, MixColumns, and AddRoundKey, with MixColumns skipped in the final round.

```
def aes_encrypt_block(block, key):
    state = list(block)
    round_keys = key_expansion(key)
    state = add_round_key(state, round_keys[0])
    for round in range(1, 10):
        state = sub_bytes(state)
        state = shift_rows(state)
        state = mix_columns(state)
        state = add_round_key(state, round_keys[round])
    state = sub_bytes(state)
    state = shift_rows(state)
    state = add_round_key(state, round_keys[10])
    return bytes(state)
```

5. AES Decryption Block

aes_decrypt_block reverses the encryption process. It uses inverse operations like InvShiftRows, InvSubBytes, and InvMixColumns along with the reverse key schedule to recover the original plaintext block.

```
def aes_decrypt_block(block, key):
    state = list(block)
    round_keys = key_expansion(key)
    state = add_round_key(state, round_keys[10])
    for round in range(9, 0, -1):
        state = inv_shift_rows(state)
        state = inv_sub_bytes(state)
        state = add_round_key(state, round_keys[round])
        state = inv_mix_columns(state)
    state = inv_shift_rows(state)
    state = inv_sub_bytes(state)
    state = add_round_key(state, round_keys[0])
    return bytes(state)
```

6. CBC Mode Operation

In CBC (Cipher Block Chaining) mode, each plaintext block is XORed with the previous ciphertext block before encryption. A random IV is used for the first block. This chaining adds semantic security to AES.

```
def aes_encrypt_cbc(plaintext, key, iv):
    plaintext = pkcs7_pad(plaintext)
    ciphertext = b''
    previous = iv
    for i in range(0, len(plaintext), 16):
        block = plaintext[i:i+16]
        block = xor_bytes(block, previous)
        encrypted = aes_encrypt_block(block, key)
        ciphertext += encrypted
        previous = encrypted
    return ciphertext

def aes_decrypt_cbc(ciphertext, key, iv):
    plaintext = b''
    previous = iv
    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i+16]
        decrypted = aes_decrypt_block(block, key)
        decrypted = xor_bytes(decrypted, previous)
        plaintext += decrypted
        previous = block
    return pkcs7_unpad(plaintext)
```

7. Main Execution and Output

The __main__ block handles user input, performs key generation using SHA-256, encryption, and decryption, and prints a detailed view of each step.

```
if __name__ == "__main__":
    print("=================================================")
    print("       AES-128 CBC Mode Encryption/Decryption")
    print("=================================================")

    input_message = input("Enter the message to encrypt:\n> ")
    key = generate_key_from_input(input_message)
    iv = os.urandom(16)

    print("\n🔄 Generating AES Key from SHA-256 (first 16 bytes)...")

    print("🔧 AES Key (hex):", key.hex())
    print("📦 IV       (hex):", iv.hex())

    ciphertext = aes_encrypt_cbc(input_message.encode(), key, iv)
    full_output = iv + ciphertext


    print(" Ciphertext (IV + Cipher, hex):")
    print(full_output.hex())


    print(" Starting Decryption...")
    decrypted = aes_decrypt_cbc(ciphertext, key, iv)


    print(" Decrypted Message:")
    print(decrypted.decode())

    print("Decryption completed successfully!")
```

8. Sample Output

Below is a sample output generated by the program, demonstrating successful encryption and decryption:

```
=================================================
      AES-128 CBC Mode Encryption/Decryption
=================================================
Enter the message to encrypt:
> AES, or Advanced Encryption Standard, is a symmetric encryption algorithm widely used to protect sensitive data. It's a block cipher that encrypts data in blocks of

🔄 Generating AES Key from SHA-256 (first 16 bytes)...
🔧 AES Key (hex): e309617ace87a91f79eaa0d8c4ec2634
📦 IV       (hex): be82062540bce812911bf3cadf752108
 Ciphertext (IV + Cipher, hex):
be82062540bce812911bf3cadf75210862a0e007c1baf5dcf67395228fa86e2fa0eba54c2487adb7b5abdea7899a7748adec37e1ab7c3e7cd5c597069473e0a3be57dcf38309e8687bb2c6ce3d14dfa9ef68e3
 Starting Decryption...
 Decrypted Message:
AES, or Advanced Encryption Standard, is a symmetric encryption algorithm widely used to protect sensitive data. It's a block cipher that encrypts data in blocks of 1
Decryption completed successfully!
```

9. Observations

- AES rounds are implemented step-by-step without third-party libraries.
- CBC mode ensures different ciphertexts for the same message.
- The implementation uses SHA-256 for deriving a 128-bit key.
- PKCS#7 padding is used to handle variable length plaintext.

10. Limitations

- Only AES-128 supported (no 192 or 256-bit).
- CBC mode lacks authentication (no integrity protection).
- IV and ciphertext must be stored/transmitted together.

11. Conclusion

This AES implementation demonstrates a full, manual implementation of AES-128 in CBC mode. It reinforces understanding of internal AES operations, block cipher modes, and secure encryption practices.