IMPLEMENTATION OF DATA ENCRYPTION STANDARD(DES)

BY KRISHNA J (CS22B1023)

INTRODUCTION

The Data Encryption Standard (DES) is a symmetric-key block cipher that encrypts data in 64-bit blocks using a 56-bit key. Despite being considered insecure for many modern applications due to its small key size, DES serves as a foundational algorithm in cryptography and provides valuable insights into how symmetric encryption systems work.

DES was developed in the early 1970s by IBM and later adopted by the U.S. National Institute of Standards and Technology (NIST) as a federal standard in 1977. It became the first encryption algorithm approved for public use by the U.S. government.

The algorithm operates through a series of complex permutations and substitutions structured in a Feistel network, which splits the data block into two halves and processes them through multiple rounds of transformation using subkeys derived from the main encryption key. This structure makes it computationally difficult to reverse-engineer the encryption without the key.

Although DES has been replaced by more secure algorithms like AES (Advanced Encryption Standard), it continues to be studied for its historical significance, pedagogical value, and relevance in understanding fundamental cryptographic concepts.

OBJECTIVE

The main objective is to build a simple version of the DES (Data Encryption Standard) algorithm using Python. The focus is on understanding and implementing each core component of DES. The goal is to learn how DES works internally by coding each step manually.

IMPLEMENTATION

The DES algorithm works on 64-bit blocks of data using a 56-bit key. The implementation involves several steps, each contributing to the encryption and decryption process.

- First, the text is converted into binary format. This is done using two functions: one to convert plain text into binary (text_to_bin) and another to convert binary back to text (bin_to_text). This step allows the algorithm to work on bit-level data.

- Next, permutation operations are performed using standard DES tables. These include the initial permutation (IP), the final permutation (FP), the expansion function (E), and the permutation function (P). A generic permute function applies these transformations to the bit sequences.
- The core of DES involves the use of S-boxes for substitution. There are eight S-boxes, and each takes a 6-bit input and returns a 4-bit output. The S-boxes introduce non-linearity, making the encryption more secure.
- The Feistel function is applied to the right half of the data block in each round. It includes expansion, XOR with the subkey, substitution using S-boxes, and permutation. The result is then XORed with the left half to complete one round.
- The DES algorithm performs 16 such rounds. In each round, the left and right halves are swapped, and the process repeats. This is a key characteristic of the Feistel network.
- Key generation is simplified in this implementation. The same 48-bit key is used for all 16 rounds. Normally, DES uses a key schedule to generate a unique subkey for each round.
- Finally, the encryption and decryption processes are handled by the des_encrypt_block and des_decrypt_block functions. These functions apply all the above steps to transform the data accordingly.

CODE

```
# Convert text to binary and back
def text_to_bin(text):
    return ''.join(format(ord(c), '08b') for c in text)

def bin_to_text(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0, len(binary),
8))

# Permutation helper
def permute(block, table):
    return ''.join([block[i - 1] for i in table])

# XOR helper
def xor(a, b):
    return ''.join('0' if i == j else '1' for i, j in zip(a, b))

# Use all 8 S-boxes
SBOXES = [
    # S1
    [[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
     [0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
```

```
     [4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
     [15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]],
    # S2
    [[15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],
     [3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],
     [0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],
     [13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9]],
    # S3
    [[10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],
     [13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],
     [13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],
     [1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12]],
    # S4
    [[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],
     [13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
     [10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
     [3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]],
    # S5
    [[2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
     [14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
     [4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
     [11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]],
    # S6
    [[12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],
     [10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],
     [9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],
     [4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13]],
    # S7
    [[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
     [13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],
     [1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
     [6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]],
    # S8
    [[13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],
     [1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],
     [7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],
     [2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]]
]

# Initial Permutation Table
IP = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]
```

```
# Final Permutation Table
FP = [40, 8, 48, 16, 56, 24, 64, 32,
      39, 7, 47, 15, 55, 23, 63, 31,
      38, 6, 46, 14, 54, 22, 62, 30,
      37, 5, 45, 13, 53, 21, 61, 29,
      36, 4, 44, 12, 52, 20, 60, 28,
      35, 3, 43, 11, 51, 19, 59, 27,
      34, 2, 42, 10, 50, 18, 58, 26,
      33, 1, 41, 9, 49, 17, 57, 25]

# Expansion Table
E = [32, 1, 2, 3, 4, 5, 4, 5,
     6, 7, 8, 9, 8, 9, 10, 11,
     12, 13, 12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21, 20, 21,
     22, 23, 24, 25, 24, 25, 26, 27,
     28, 29, 28, 29, 30, 31, 32, 1]

# Permutation P
P = [16, 7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26, 5, 18, 31, 10,
     2, 8, 24, 14, 32, 27, 3, 9,
     19, 13, 30, 6, 22, 11, 4, 25]

# S-box substitution
def sbox_substitution(input_bits):
    output = ''
    for i in range(8):
        block = input_bits[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        val = SBOXES[i][row][col]
        output += format(val, '04b')
    return output

# Feistel Function
def feistel(right, subkey):
    expanded = permute(right, E)
    xored = xor(expanded, subkey)
    substituted = sbox_substitution(xored)
    return permute(substituted, P)

# DES Round Function
def des_round(left, right, key):
    return right, xor(left, feistel(right, key))

# Dummy key schedule: repeat same 48-bit key for 16 rounds
```

```python
def generate_keys(key):
    key_bin = text_to_bin(key)
    return [key_bin[:48]] * 16  # Simple demo key schedule

# Encrypt 64-bit block
def des_encrypt_block(plaintext, key):
    keys = generate_keys(key)
    block = text_to_bin(plaintext)
    block = permute(block, IP)
    left, right = block[:32], block[32:]
    for k in keys:
        left, right = des_round(left, right, k)
    cipher = permute(right + left, FP)
    return cipher

# Decrypt 64-bit block
def des_decrypt_block(ciphertext, key):
    keys = generate_keys(key)[::-1]  # Reverse key schedule
    block = permute(ciphertext, IP)
    left, right = block[:32], block[32:]
    for k in keys:
        left, right = des_round(left, right, k)
    plain_bin = permute(right + left, FP)
    return bin_to_text(plain_bin)

# --- Test ---
plaintext = input("Enter plaintext (8 characters): ")
# Ensure the plaintext is exactly 8 characters
key = input("Enter key (8 characters): ")     # Exactly 8 characters

print("Plaintext:", plaintext)
cipher_bin = des_encrypt_block(plaintext, key)
print("Encrypted (binary):", cipher_bin)
decrypted = des_decrypt_block(cipher_bin, key)
print("Decrypted Text:", decrypted)
```

OUTPUT

```
Enter plaintext (8 characters): hiamfine
Enter key (8 characters): 12345543
Plaintext: hiamfine
Encrypted (binary): 1100100100100010000000000110011110101011101011101
110101000011110
Decrypted Text: hiamfine
```

CONCLUSION

This implementation helps in understanding the internal working of the DES algorithm. Each component, such as bit-level conversion, permutation, the Feistel structure, and S-box substitution, was implemented manually. Although the key schedule was simplified, the overall encryption and decryption logic followed the standard DES structure. The exercise supported the learning of symmetric encryption by applying theoretical concepts through practical coding.