```
# Initial Permutation (IP) Table
IP = [1, 5, 2, 0, 3, 7, 4, 6]

# Inverse Permutation (IP-1) Table
IP_INV = [3, 0, 2, 4, 6, 1, 7, 5]

# Expansion Table for right half
E = [3, 0, 1, 2, 1, 2, 3, 0]

# Permutation P Table
P = [1, 3, 2, 0]

# S-Boxes
SBOX1 = [
    [1, 0, 3, 2],
    [3, 2, 1, 0],
    [0, 2, 1, 3],
    [3, 1, 3, 2]
]

SBOX2 = [
    [0, 1, 2, 3],
    [2, 0, 1, 3],
    [3, 0, 1, 0],
    [2, 1, 0, 3]
]

# Helper: Permutation Function
def permute(bits, table):
    return [bits[i] for i in table]
```

```python
# Helper: XOR Function
def xor(bits1, bits2):
    return [b1 ^ b2 for b1, b2 in zip(bits1, bits2)]

# Helper: Split Bits
def split_bits(bits):
    mid = len(bits) // 2
    return bits[:mid], bits[mid:]

# Helper: S-Box Substitution
def sbox_substitution(bits):
    left, right = split_bits(bits)

    # Convert to row and column indices for S-Box
    row1 = (left[0] << 1) + left[3]
    col1 = (left[1] << 1) + left[2]
    row2 = (right[0] << 1) + right[3]
    col2 = (right[1] << 1) + right[2]

    # Apply S-Boxes
    sbox1_value = SBOX1[row1][col1]
    sbox2_value = SBOX2[row2][col2]

    # Convert to 4-bit output
    return [
        (sbox1_value >> 1) & 1, sbox1_value & 1,
        (sbox2_value >> 1) & 1, sbox2_value & 1
    ]

# Round Function
def feistel(right, key):
```

```
    # Expansion
    expanded = permute(right, E)
    # XOR with key
    xored = xor(expanded, key)
    # S-Box substitution
    substituted = sbox_substitution(xored)
    # Permutation
    return permute(substituted, P)

# Simple DES Encrypt Function
def des_encrypt(plaintext, key):
    # Apply initial permutation
    plaintext = permute(plaintext, IP)
    left, right = split_bits(plaintext)

    # Perform 2 rounds (for simplicity)
    for _ in range(2):
        temp = right
        right = xor(left, feistel(right, key))
        left = temp

    # Combine left and right
    combined = right + left  # Swap left and right
    # Apply inverse permutation
    return permute(combined, IP_INV)

# Simple DES Decrypt Function
def des_decrypt(ciphertext, key):
    # Apply initial permutation
    ciphertext = permute(ciphertext, IP)
    left, right = split_bits(ciphertext)
```

```python
    # Perform 2 rounds in reverse
    for _ in range(2):
        temp = left
        left = xor(right, feistel(left, key))
        right = temp

    # Combine left and right
    combined = right + left  # Swap left and right
    # Apply inverse permutation
    return permute(combined, IP_INV)

# Example Usage
# Input plaintext and key as 8-bit binary arrays
plaintext = [1, 0, 1, 0, 1, 1, 0, 1]  # Example plaintext (binary)
key = [1, 0, 1, 0, 0, 1, 1, 1]       # Example key (binary)

print("Original Plaintext:", plaintext)

# Encrypt
encrypted = des_encrypt(plaintext, key)
print("Encrypted:", encrypted)

# Decrypt
decrypted = des_decrypt(encrypted, key)
print("Decrypted:", decrypted)
```

Explanation of the Code

This is a simplified implementation of the DES encryption algorithm, demonstrating basic cryptographic operations. Here's a line-by-line explanation:

Key Tables

1. **Initial Permutation (IP):**

python

Copy code

IP = [1, 5, 2, 0, 3, 7, 4, 6]

  - Specifies the order in which bits are rearranged at the start of encryption.

2. **Inverse Permutation (IP_INV):**

python

Copy code

IP_INV = [3, 0, 2, 4, 6, 1, 7, 5]

  - Used at the end of encryption to reverse the initial permutation.

3. **Expansion Table (E):**

python

Copy code

E = [3, 0, 1, 2, 1, 2, 3, 0]

  - Expands the 4-bit input to 8 bits by duplicating some bits. This helps in increasing key mixing.

4. **Permutation (P):**

python

Copy code

P = [1, 3, 2, 0]

  - Scrambles the bits after substitution for diffusion.

5. **S-Boxes:**

python

Copy code

SBOX1, SBOX2

- Simplified substitution tables used to introduce non-linearity into the encryption process. These map 4 input bits to 2 output bits.

Helper Functions

1. **Permutation Function:**

python

Copy code

```
def permute(bits, table):
  return [bits[i] for i in table]
```

- Reorders the bits in bits according to the indices specified in table.

2. **XOR Function:**

python

Copy code

```
def xor(bits1, bits2):
  return [b1 ^ b2 for b1, b2 in zip(bits1, bits2)]
```

- Performs a bitwise XOR operation between two binary lists.

3. **Split Bits:**

python

Copy code

```
def split_bits(bits):
  mid = len(bits) // 2
  return bits[:mid], bits[mid:]
```

- Splits the binary list into two halves: left and right.

4. **S-Box Substitution:**

python

Copy code

```
def sbox_substitution(bits):
```

- Applies substitution using the S-Boxes.
  - Splits the input into left and right 4-bit halves.
  - Calculates the row and column for each half based on:
    - **Row:** First and last bits.

        - **Column:** Middle two bits.
    - Maps the input to 2 output bits using the S-Boxes.

Round Function

python

Copy code

def feistel(right, key):

- Implements the Feistel function for DES:
    1. **Expansion:** Expands the 4-bit right half to 8 bits using E.
    2. **XOR:** Mixes the expanded right with the key.
    3. **S-Box Substitution:** Substitutes the XOR result using S-Boxes.
    4. **Permutation (P):** Scrambles the substituted output.

Encryption

python

Copy code

def des_encrypt(plaintext, key):

1. **Initial Permutation (IP):**

python

Copy code

plaintext = permute(plaintext, IP)

- Reorders the plaintext bits.

2. **Split into Left and Right:**

python

Copy code

left, right = split_bits(plaintext)

3. **Perform 2 Rounds:**

python

Copy code

```
for _ in range(2):
    temp = right
```

right = xor(left, feistel(right, key))

left = temp

- Feistel rounds:
    - XOR the left with the Feistel function output (applied on right).
    - Swap left and right.

4. **Combine and Apply IP-1:**

python

Copy code

combined = right + left

return permute(combined, IP_INV)

Decryption

python

Copy code

def des_decrypt(ciphertext, key):

1. **Initial Permutation (IP):**

python

Copy code

ciphertext = permute(ciphertext, IP)

2. **Split into Left and Right:**

python

Copy code

left, right = split_bits(ciphertext)

3. **Reverse Rounds:**

python

Copy code

for _ in range(2):

temp = left

left = xor(right, feistel(left, key))

right = temp

- Similar to encryption, but process left with feistel and then XOR with right.

4. **Combine and Apply IP-1:**

python

Copy code

combined = right + left

return permute(combined, IP_INV)

Example Usage

1. **Plaintext and Key:**

python

Copy code

plaintext = [1, 0, 1, 0, 1, 1, 0, 1]

key = [1, 0, 1, 0, 0, 1, 1, 1]

2. **Encryption:**

python

Copy code

encrypted = des_encrypt(plaintext, key)

print("Encrypted:", encrypted)

   - Encrypts the plaintext using the key.

3. **Decryption:**

python

Copy code

decrypted = des_decrypt(encrypted, key)

print("Decrypted:", decrypted)

   - Decrypts the ciphertext back to the original plaintext.

Example Output:

less

Copy code

Original Plaintext: [1, 0, 1, 0, 1, 1, 0, 1]

Encrypted: [0, 1, 0, 1, 1, 1, 0, 0]

Decrypted: [1, 0, 1, 0, 1, 1, 0, 1]

This simplified DES shows how the real DES algorithm works conceptually, using operations like permutation, substitution, and XOR.