Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Experiment 3

Date of Performance : **Date of Submission:**

SAP Id: 60004190057 **Name : Junaid Altaf Girkar**

Div: A **Batch : A4**

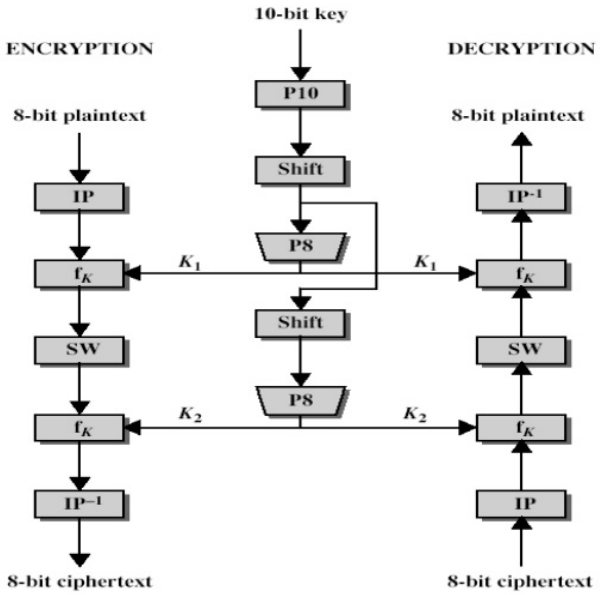
Aim of Experiment

Design and Implement Encryption and Decryption Algorithm for S-DES

Theory / Algorithm / Conceptual Description

Simplified Data Encryption Standard is a simple version of Data Encryption Standard having a 10-bit key and 8-bit plain text. It is much smaller than the DES algorithm as it takes only 8-bit plain text whereas DES takes 64-bit plain text. It is a block cipher algorithm and uses a symmetric key for its algorithm i.e. they use the same key for both encryption and decryption. It has 2 rounds for encryption which use two different keys.

The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.



The encryption algorithm involves five functions:

- An initial permutation (IP)
- A complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input
- A simple permutation function that switches (SW) the two halves of the data the function f_k again
- A permutation function that is the inverse of the initial permutation

The function fk takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition of functions: IP-1 o fK2 o SW o fk1 o IP

Which can also be written as

```
Ciphertext = IP-1 (fK2 (SW (fk1 (IP (plaintext)))))
```

Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as

```
Plaintext = IP-1 (fK1 (SW (fk2 (IP (ciphertext)))))
```

CODE:

```
def apply_table(inp, table):
    res = ""
    for i in table:
        res += inp[i - 1]
    return res


def left_shift(data):
    return data[1:] + data[0]


def XOR(a, b):
    res = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            res += "0"
        else:
            res += "1"
    return res


def apply_sbox(s, data):
    row = int("0b" + data[0] + data[-1], 2)
    col = int("0b" + data[1:3], 2)
    return bin(s[row][col])[2:]


def function(expansion, s0, s1, key, message):
    left = message[:4]
    right = message[4:]
```

```
    temp = apply_table(right, expansion)
    temp = XOR(temp, key)
    l = apply_sbox(s0, temp[:4])
    r = apply_sbox(s1, temp[4:])
    l = "0" * (2 - len(l)) + l
    r = "0" * (2 - len(r)) + r
    temp = apply_table(l + r, p4_table)
    temp = XOR(left, temp)
    return temp + right

def key_generation_1(key, table):
    k = table_shift(key, table)
    key_merge = split_and_merge(k)
    return table_shift(key_merge, table)

def key_generation_2(key, table): return split_and_merge(key)


if __name__ == "__main__":

    key = key = str('0001101101')#input("Enter 10 bit key: ")
    message = "10101010"#input("Enter 8 bit message: ")
    print("Plain text before decryption is : " + str(message))

    p8_table = [6, 3, 7, 4, 8, 5, 10, 9]
    p10_table = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    p4_table = [2, 4, 3, 1]
    IP = [2, 6, 3, 1, 4, 8, 5, 7]
    IP_inv = [4, 1, 3, 5, 7, 2, 8, 6]
    expansion = [4, 1, 2, 3, 2, 3, 4, 1]
    s0 = [[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]]
    s1 = [[0, 1, 2, 3], [2, 0, 1, 3], [3, 0, 1, 0], [2, 1, 0, 3]]

    # key generation
    temp = apply_table(key, p10_table)
    left = temp[:5]
    right = temp[5:]
    left = left_shift(left)
```

```
    right = left_shift(right)
    key1 = apply_table(left + right, p8_table)
    left = left_shift(left)
    right = left_shift(right)
    left = left_shift(left)
    right = left_shift(right)
    key2 = apply_table(left + right, p8_table)


    # encryption
    temp = apply_table(message, IP)
    temp = function(expansion, s0, s1, key1, temp)
    temp = temp[4:] + temp[:4]
    temp = function(expansion, s0, s1, key2, temp)
    CT = apply_table(temp, IP_inv)
    print("Cipher text is:", CT)

    # decryption
    temp = apply_table(CT, IP)
    temp = function(expansion, s0, s1, key2, temp)
    temp = temp[4:] + temp[:4]
    temp = function(expansion, s0, s1, key1, temp)
    PT = apply_table(temp, IP_inv)
    print("Plain text after decrypting is:", PT)
```

OUTPUT:

```
Plain text before decryption is : 10101010
Cipher text is: 00011111
Plain text after decrypting is: 10101010
```

CONCLUSION

With the increasing amount of data being generated, it is very important that confidential information does not get leaked and is read by the intended recipient.We learnt about the Simplified DES algorithm and we then wrote a python program to implement it.