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 : 29-03-2022

Date of Submission: 02-04-2022

SAP Id: 60004190053

Name : Jayesh Kavedia

Div: A

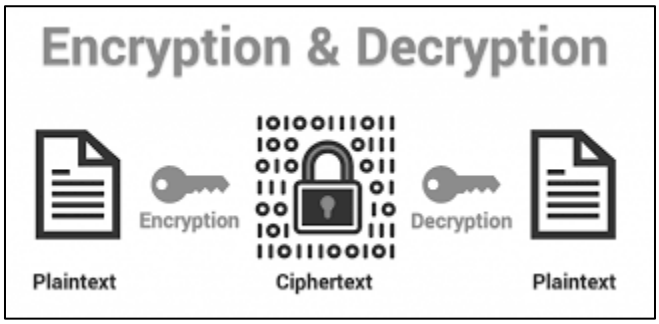
Batch : A4

Aim of Experiment

Design and Implement Encryption and Decryption Algorithm for Simplified Data Encryption Standard (S-DES).

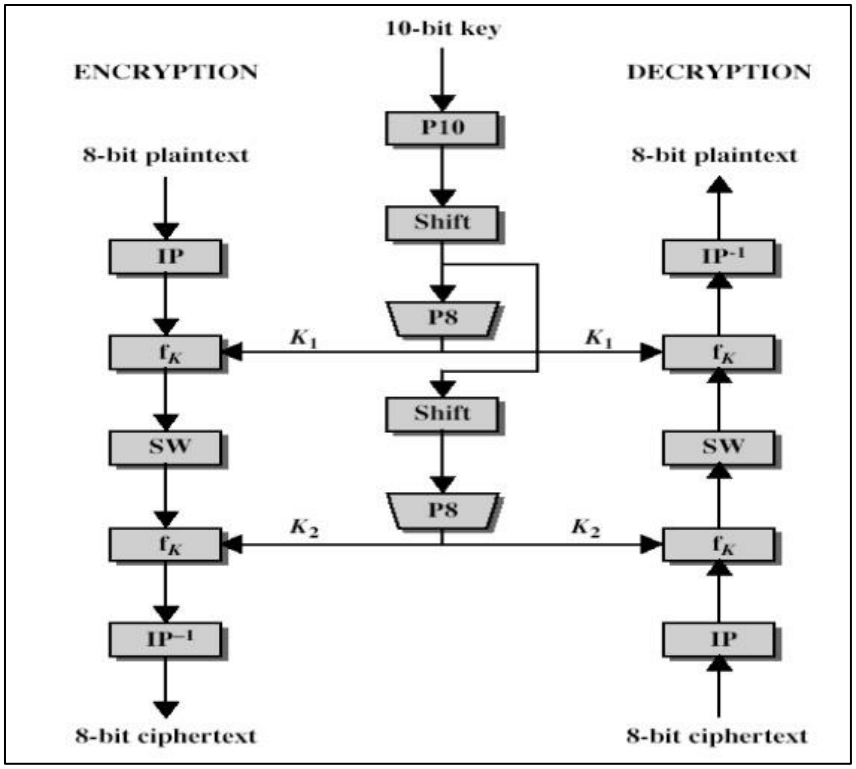
Theory / Algorithm / Conceptual Description

Encryption and Decryption



Encryption is the process of converting a readable message into an unreadable form so that it cannot be read by unauthorized parties. The process of converting an encrypted message back to its original (readable) format is known as decryption. The plaintext message is the original message. The ciphertext message is the encrypted message. Digital encryption techniques function by mathematically modifying the digital content of a plaintext message and producing a ciphertext version of the message using an encryption algorithm and a digital key. If the sender and recipient are the only ones who have access to the key, they can communicate safely.

Simplified Data Encryption Standard (S-DES)



The DES Algorithm is simplified in the Simplified Data Encryption Standard (S-DES). It's comparable to the DES algorithm, however it's a smaller version with less parameters. It is a block cipher that turns plaintext into ciphertext. It requires an 8-bit block. It's a symmetric key cipher, which means the encryption and decryption keys are the same.

The algorithm consists of five functions: an initial permutation (IP), a complex function labelled f_k that combines permutation and substitution operations and is dependent on a key input, a simple permutation function that switches (SW) the two halves of the data, the function f_k once more, and a permutation function (IP^{-1}) that is the inverse of the initial permutation. The function f_k accepts an 8-bit key as well as the data travelling through the encryption process. Two eight-bit subkeys are created from a ten-bit key. A permutation test is performed on the key initially (P_{10}). After that, a shift operation is carried out. The shift operation's output is then sent through a permutation function, which generates an 8-bit output (P_8) for the first subkey (K_1). The second subkey is created by feeding the output of the shift operation into another shift and another instance of P_8 (K_2).

Algorithm

Ciphertext = IP^{-1} (f_{K2} (SW (f_{k1} (IP (Plaintext)))))

Plaintext = IP^{-1} (f_{K1} (SW (f_{k2} (IP (Ciphertext)))))

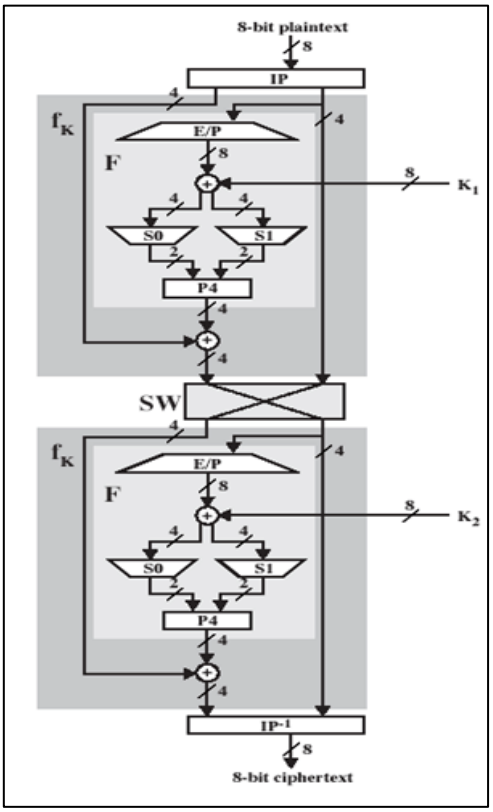
Where,

$K_1 = P_8$ (Shift (P_{10} (Key)))

$K_2 = P_8$ (Shift (Shift (P_{10} (Key))))

Complex Function f_k

The complex function f_k receives input in two halves : left half contains 4 bits as well as the right half contains 4 bits, the right 4 bits are provided as input to expansion block which provides output of 8 bits. These 8 bits are Ex-ORed with input key, the output is again divided into 2 halves. The decimal value of 1st and 4th bits of the 4 bits provide row number whereas the decimal value of 2nd and 3rd bits provide column number. The calculated row number and column number are looked into the respective substitution matrix and the value represented by the row number and column number in the matrix is converted to 2-bit binary number and provided as output. The two, two-bit outputs are concatenated and provided as input to the P_4 block, the output is then Ex-ORed with the initial left 4-bits. The output of Ex-OR is the left half of the function output whereas the right half of the function output are the initial right 4-bits.



Program

```
import numpy as np

#INITIALIZATION

initial_key = np.array([1,0,1,1,0,0,1,1,0,1]) #Example 1
# initial_key = np.array([1,0,1,0,1,0,1,1,0,1]) #Example 2
p10 = np.array([3,5,2,7,4,10,1,9,8,6])
p8 = np.array([6,3,7,4,8,5,10,9])
p4 = np.array([2,4,3,1])
ip = np.array([2,6,3,1,4,8,5,7])
ip_inv = np.array([4,1,3,5,7,2,8,6])
ep = np.array([4,1,2,3,2,3,4,1])

#KEY GENERATION"

key = initial_key[p10 - 1]
left = np.roll(key[:5],-1)
right = np.roll(key[5:], -1)
key1 = np.concatenate((left,right))
```

```
key1 = key1[p8 - 1]
print('Key 1 : ',key1)
left = np.roll(left,-2)
right = np.roll(right, -2)
key2 = np.concatenate((left,right))
key2 = key2[p8 - 1]
print('Key 2 : ',key2)

#SUBSTITUTION MATRIX

s0 = np.array([[1,0,3,2],[3,2,1,0],[0,2,1,3],[3,1,3,2]])
s1 = np.array([[0,1,2,3],[2,0,1,3],[3,0,1,0],[2,1,0,3]])

#COMPLEX FUNCTION

def complex_function(left,right,key,ep,p4,s0,s1):
  intermediate_text = right[ep - 1]
  intermediate_text = np.bitwise_xor(intermediate_text,key)
  left_s = intermediate_text[:4]
  right_s = intermediate_text[4:]
  s0_op_row = left_s[[0,3]]
  s0_op_row = [val.item() for val in s0_op_row]
  s0_op_row = int(''.join(list(map(str,s0_op_row))),2)
  s0_op_col = left_s[[1,2]]
  s0_op_col = [val.item() for val in s0_op_col]
  s0_op_col = int(''.join(list(map(str,s0_op_col))),2)
  s1_op_row = right_s[[0,3]]
  s1_op_row = [val.item() for val in s1_op_row]
  s1_op_row = int(''.join(list(map(str,s1_op_row))),2)
  s1_op_col = right_s[[1,2]]
  s1_op_col = [val.item() for val in s1_op_col]
  s1_op_col = int(''.join(list(map(str,s1_op_col))),2)
  s0_op = bin(s0[s0_op_row][s0_op_col]).replace('0b','').zfill(2)
  s1_op = bin(s1[s1_op_row][s1_op_col]).replace('0b','').zfill(2)
  intermediate_text = list(s0_op + s1_op)
  intermediate_text = np.array(intermediate_text,dtype=np.int64)
  intermediate_text = intermediate_text[p4 - 1]
  intermediate_text = np.bitwise_xor(intermediate_text,left)
  return intermediate_text, right

#ENCRYPTION
pt = 'CA' #Example1
# pt = 'FC' #Example2
```

```
print('Encryption :\nPlain Text to be Encrypted : ',pt)
pt = list(bin(int(pt,16)).replace('0b','').zfill(8))
pt = np.array(pt,dtype=np.int64)
print('8-bit Plain Text : ',pt)
intermediate_text = pt[ip - 1]
left = intermediate_text[:4]
right = intermediate_text[4:]
left,right = complex_function(left,right,key1,ep,p4,s0,s1)
left,right = right,left
left,right = complex_function(left,right,key2,ep,p4,s0,s1)
intermediate_text = np.concatenate((left,right))
ct = intermediate_text[ip_inv - 1]
print('8-bit Cipher Text : ',ct)
ct = [val.item() for val in ct]
ct = int(''.join(list(map(str,ct))),2)
ct = hex(ct).replace('0x','')
print('Encrypted Text / Cipher Text : ',ct.upper())

#DECRYPTION
ct = '2D' #Example 1
# ct = '28' #Example 2
print('Decryption :\nCipher Text to be Decrypted : ',ct)
ct = list(bin(int(ct,16)).replace('0b','').zfill(8))
ct = np.array(ct,dtype=np.int64)
print('8-bit Cipher Text : ',ct)
intermediate_text = ct[ip - 1]
left = intermediate_text[:4]
right = intermediate_text[4:]
left,right = complex_function(left,right,key2,ep,p4,s0,s1)
left,right = right,left
left,right = complex_function(left,right,key1,ep,p4,s0,s1)
intermediate_text = np.concatenate((left,right))
pt = intermediate_text[ip_inv - 1]
print('8-bit Plain Text : ',pt)
pt = [val.item() for val in pt]
pt = int(''.join(list(map(str,pt))),2)
pt = hex(pt).replace('0x','')
print('Decrypted Text / Plain Text : ',pt.upper())
```

Output

Example 1
Key 1 : [1 1 0 1 1 1 1 0]
Key 2 : [1 1 0 0 1 0 0 1]
Encryption :
Plain Text to be Encrypted : CA
8-bit Plain Text : [1 1 0 0 1 0 1 0]
8-bit Cipher Text : [0 0 1 0 1 1 0 1]
Encrypted Text / Cipher Text : 2D
Decryption :
Cipher Text to be Decrypted : 2D
8-bit Cipher Text : [0 0 1 0 1 1 0 1]
8-bit Plain Text : [1 1 0 0 1 0 1 0]
Decrypted Text / Plain Text : CA

Example 2
Key 1 : [1 1 0 0 1 1 1 0]
Key 2 : [1 1 0 1 1 0 0 1]
Encryption :
Plain Text to be Encrypted : FC
8-bit Plain Text : [1 1 1 1 1 1 0 0]
8-bit Cipher Text : [0 0 1 0 1 0 0 0]
Encrypted Text / Cipher Text : 28
Decryption :
Cipher Text to be Decrypted : 28
8-bit Cipher Text : [0 0 1 0 1 0 0 0]
8-bit Plain Text : [1 1 1 1 1 1 0 0]
Decrypted Text / Plain Text : FC

Conclusion

DES/SDES are block cipher encryption algorithm. DES encrypts blocks of 64 bits with help of 56-bit keys over 16 rounds. SDES is simplified version of DES which encrypts 8-bit blocks with help of 8-bit keys over 2 rounds. The SDES algorithm was made for understanding the DES algorithm. Since the size of key is 56 bits in DES, it takes 2^{56} attempts in brute force method also the algorithm is complex which increases the complexity to crack it.