

**Experiment 3**

**Date of Performance : 4-4-2022** **Date of Submission: 4-4-2022**

**SAP Id:** 60004190016 **Name : Aunali Patel**

**Div:** A **Batch : A3**

**Aim of Experiment**

Implement Simplified Data Encryption Standard (S-DES).

(CO1)

**Theory / Algorithm / Conceptual Description**

Simplified Data Encryption Standard (S-DES) is a simple version of the [DES Algorithm](https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/). It is similar to the [DES](https://www.geeksforgeeks.org/des-full-form/) algorithm but is a smaller algorithm and has fewer parameters than DES. t is a block cipher that takes a block of plain text and converts it into ciphertext.  It takes a block of 8 bit.

It is a symmetric key cipher i.e. they use the same key for both encryption and decryption.

**Program**

Key Generation:

|  |
| --- |
| KEY GENERATION:  def leftshift(key):    k = key    lis = [k[1],k[2],k[3],k[4],k[0]]    return lis  def p10(key):    k = key    lis = [k[3-1],k[5-1],k[2-1],k[7-1],k[4-1],k[10-1],k[1-1],k[9-1],k[8-1],k[6-1]]    return lis  def p8(key):    k = key    lis = [k[6-1],k[3-1],k[7-1],k[4-1],k[8-1],k[5-1],k[10-1],k[9-1]]    return lis  def keygeneration(mainkey)-> list:    s1 = p10(mainkey)    l = s1[0:5]    r = s1[5:]    # print("left" , leftshift(l) )    # print("right" , leftshift(r) )    l = leftshift(l)    r = leftshift(r)    ls1 = l + r    key1 = p8(ls1)    # print(p8)    l2 = leftshift(leftshift(l))    r2 = leftshift(leftshift(r))    ls2 = l2+r2    key2 = p8(ls2)    ans = [key1,key2]    return ans  mainkey = [1,0,1,0,0,0,0,0,1,0]  print(keygeneration(mainkey)[0] , "key1")  print(keygeneration(mainkey)[1], "key2")  key1 = keygeneration(mainkey)[0]  key2 = keygeneration(mainkey)[1] |

**Output**

|  |
| --- |
| [1, 0, 1, 0, 0, 1, 0, 0] key1  [0, 1, 0, 0, 0, 0, 1, 1] key2 |

**Program**

Encryption:

|  |
| --- |
| ENCRYPTION:  plaintext = [1,0,0,1,0,1,1,1]  def ip(plaintext):    pt = plaintext    lis = [pt[2-1],pt[6-1],pt[3-1],pt[1-1],pt[4-1],pt[8-1],pt[5-1],pt[7-1]]    return lis  def ep(input):    k = input    lis = [k[4-1],k[1-1],k[2-1],k[3-1],k[2-1],k[3-1],k[4-1],k[1-1]]    return lis  def xor(key,ip):    k = key    lis = [k[0]^ip[0],k[1]^ip[1],k[2]^ip[2],k[3]^ip[3],k[4]^ip[4],k[5]^ip[5],k[6]^ip[6],k[7]^ip[7]]    return lis  def xor4(key,ip):    k = key    lis = [k[0]^ip[0],k[1]^ip[1],k[2]^ip[2],k[3]^ip[3]]    return lis  def p4(key):    k = key    lis = [k[2-1],k[4-1],k[3-1],k[1-1]]    return lis  def fk(afterip,key):    print("doing fk on ",afterip)    lip = afterip[0:4]    rip = afterip[4:]    # print(rip,"rip")    # after ep    afterep = ep(rip)    print(afterep,"after Ep")    xorkey1 = xor(key,afterep)    # print(xorkey1)    # 0 1 0 0 1 1 1 1    l = xorkey1[0:4]    r = xorkey1[4:]    # S BOXES    S0box = [[1,0,3,2],[3,2,1,0],[0,2,1,3],[3,1,3,2]]    S1box =  [[0,1,2,3],[2,0,1,3],[3,0,1,0],[2,1,0,3]]    # for L    row = str(l[0]) + str(l[3])    rowint = int(row,2)    col = str(l[1]) + str(l[2])    colint = int(col,2)    S0 = S0box[rowint][colint]    S0str = format(S0,"b")    # print(S0str)    # for R    row = str(r[0]) + str(r[3])    rowint = int(row,2)    col = str(r[1]) + str(r[2])    colint = int(col,2)    S1 = S1box[rowint][colint]    S1str = format(S1,"b")    # print(S1str)    # afterSbox = S0str + S1str    afterSbox = [int(S0str[0]),int(S0str[1]),int(S1str[0]),int(S1str[1])]    # print(afterSbox)    # after p4    afterp4 = p4(afterSbox)    # print(afterp4)    rnew = xor4(lip,afterp4)    # print(rnew,"xor with left nibble after p4")    combine = [int(rnew[0]),int(rnew[1]),int(rnew[2]),int(rnew[3]),int(rip[0]),int(rip[1]),int(rip[2]),int(rip[3])]    # print(combine)    return combine  def ipinverse(lis):    k = lis    lis = [k[4-1],k[1-1],k[3-1],k[5-1],k[7-1],k[2-1],k[8-1],k[6-1]]    return lis  afterip = ip(plaintext)  afterfk = fk(afterip,key1)  print(afterfk," afterfk")  # step3 - switch  afterswitch = afterfk[4:] + afterfk[0:4]  print(afterswitch, " afterswtich")  afterfk2 = fk(afterswitch,key2)  print(afterfk2," afterfk2")  ciphertext = ipinverse(afterfk2)  print(ciphertext," is the cipher text") |

**Output**

|  |
| --- |
| doing fk on [0, 1, 0, 1, 1, 1, 0, 1]  [1, 1, 1, 0, 1, 0, 1, 1] after Ep  [1, 0, 1, 0, 1, 1, 0, 1] afterfk  [1, 1, 0, 1, 1, 0, 1, 0] afterswtich  doing fk on [1, 1, 0, 1, 1, 0, 1, 0]  [0, 1, 0, 1, 0, 1, 0, 1] after Ep  [0, 0, 1, 0, 1, 0, 1, 0] afterfk2  [0, 0, 1, 1, 1, 0, 0, 0] is the cipher text |

**Program**

Decryption:

|  |
| --- |
| DECRYPTION:  ciphertext = [0, 0, 1, 1, 1, 0, 0, 0]  def ip(plaintext):    pt = plaintext    lis = [pt[2-1],pt[6-1],pt[3-1],pt[1-1],pt[4-1],pt[8-1],pt[5-1],pt[7-1]]    return lis  def ep(input):    k = input    lis = [k[4-1],k[1-1],k[2-1],k[3-1],k[2-1],k[3-1],k[4-1],k[1-1]]    return lis  def xor(key,ip):    k = key    lis = [k[0]^ip[0],k[1]^ip[1],k[2]^ip[2],k[3]^ip[3],k[4]^ip[4],k[5]^ip[5],k[6]^ip[6],k[7]^ip[7]]    return lis  def xor4(key,ip):    k = key    lis = [k[0]^ip[0],k[1]^ip[1],k[2]^ip[2],k[3]^ip[3]]    return lis  def p4(key):    k = key    lis = [k[2-1],k[4-1],k[3-1],k[1-1]]    return lis  def fk(afterip,key):    print("doing fk on ",afterip)    lip = afterip[0:4]    rip = afterip[4:]    # print(rip,"rip")    # after ep    afterep = ep(rip)    print(afterep,"after Ep")    xorkey1 = xor(key,afterep)    # print(xorkey1)    # 0 1 0 0 1 1 1 1    l = xorkey1[0:4]    r = xorkey1[4:]    # S BOXES    S0box = [[1,0,3,2],[3,2,1,0],[0,2,1,3],[3,1,3,2]]    S1box =  [[0,1,2,3],[2,0,1,3],[3,0,1,0],[2,1,0,3]]    # for L    row = str(l[0]) + str(l[3])    rowint = int(row,2)    col = str(l[1]) + str(l[2])    colint = int(col,2)    S0 = S0box[rowint][colint]    S0str = format(S0,"b")    # print(S0str)    # for R    row = str(r[0]) + str(r[3])    rowint = int(row,2)    col = str(r[1]) + str(r[2])    colint = int(col,2)    S1 = S1box[rowint][colint]    S1str = format(S1,"b")    # print(S1str)    # afterSbox = S0str + S1str    afterSbox = [int(S0str[0]),int(S0str[1]),int(S1str[0]),int(S1str[1])]    # print(afterSbox)    # after p4    afterp4 = p4(afterSbox)    # print(afterp4)    rnew = xor4(lip,afterp4)    # print(rnew,"xor with left nibble after p4")    combine = [int(rnew[0]),int(rnew[1]),int(rnew[2]),int(rnew[3]),int(rip[0]),int(rip[1]),int(rip[2]),int(rip[3])]    # print(combine)    return combine  def ipinverse(lis):    k = lis    lis = [k[4-1],k[1-1],k[3-1],k[5-1],k[7-1],k[2-1],k[8-1],k[6-1]]    return lis    afterip = ip(plaintext)    afterip = ip(plaintext)    afterip = ip(plaintext)  afterip = ip(ciphertext)  afterfk = fk(afterip,key2)  afterswitch = afterfk[4:] + afterfk[0:4]  afterfk2 = fk(afterswitch,key1)  plaintext = ipinverse(afterfk2)  print("plaintext is",plaintext) |

**Output**

|  |
| --- |
| doing fk on [0, 0, 1, 0, 1, 0, 1, 0]  [0, 1, 0, 1, 0, 1, 0, 1] after Ep  doing fk on [1, 0, 1, 0, 1, 1, 0, 1]  [1, 1, 1, 0, 1, 0, 1, 1] after Ep  plaintext is [1, 0, 0, 1, 0, 1, 1, 1] |

**Conclusion:**

We have implemented SDES by using functions for key generation, encryption and decryption.