Palestine Technical University
Faculty of Engineering
Computer Systems Engineering Department

Instructor
Dr.Anas Melhem

Students
AbdulRuhman Nabeel AbuMousa
Amjad Mohammad Abdullah
Mujahed Mahmoud Saleem

Date : 5/8/2018

Contents :

1. Simplified DES explanation

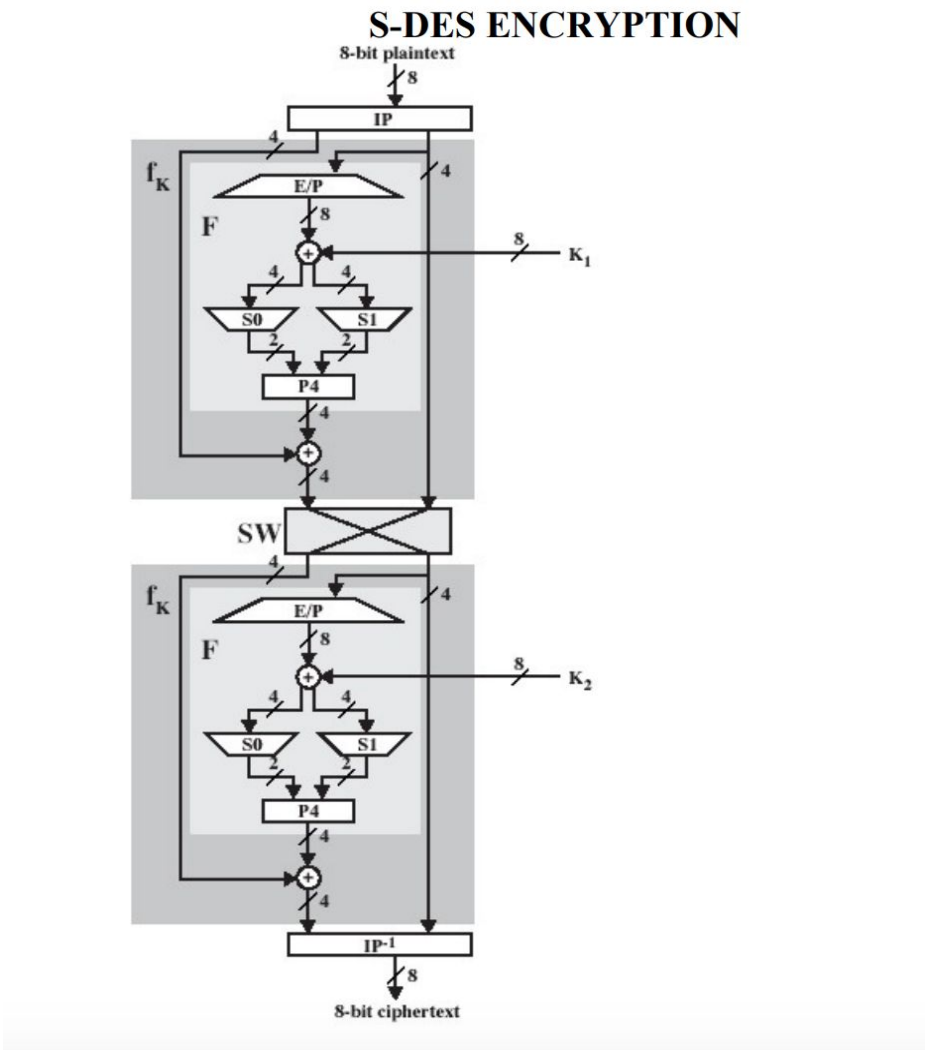


Figure illustrates the structure of the simplified DES encryption, which we will refer to as SDES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input, and produces an 8-bit block of ciphertext as output.

The encryption algorithm involves four functions:

1- An initial permutation (IP); a simple function labeled **permutation**, which involves both permutation and substitution operations and depends on a key input.

2- A Feistel function; a complex function labeled **F**, which involves both right nibble and the subkey from key generation and here we use a constant sub key as an example.

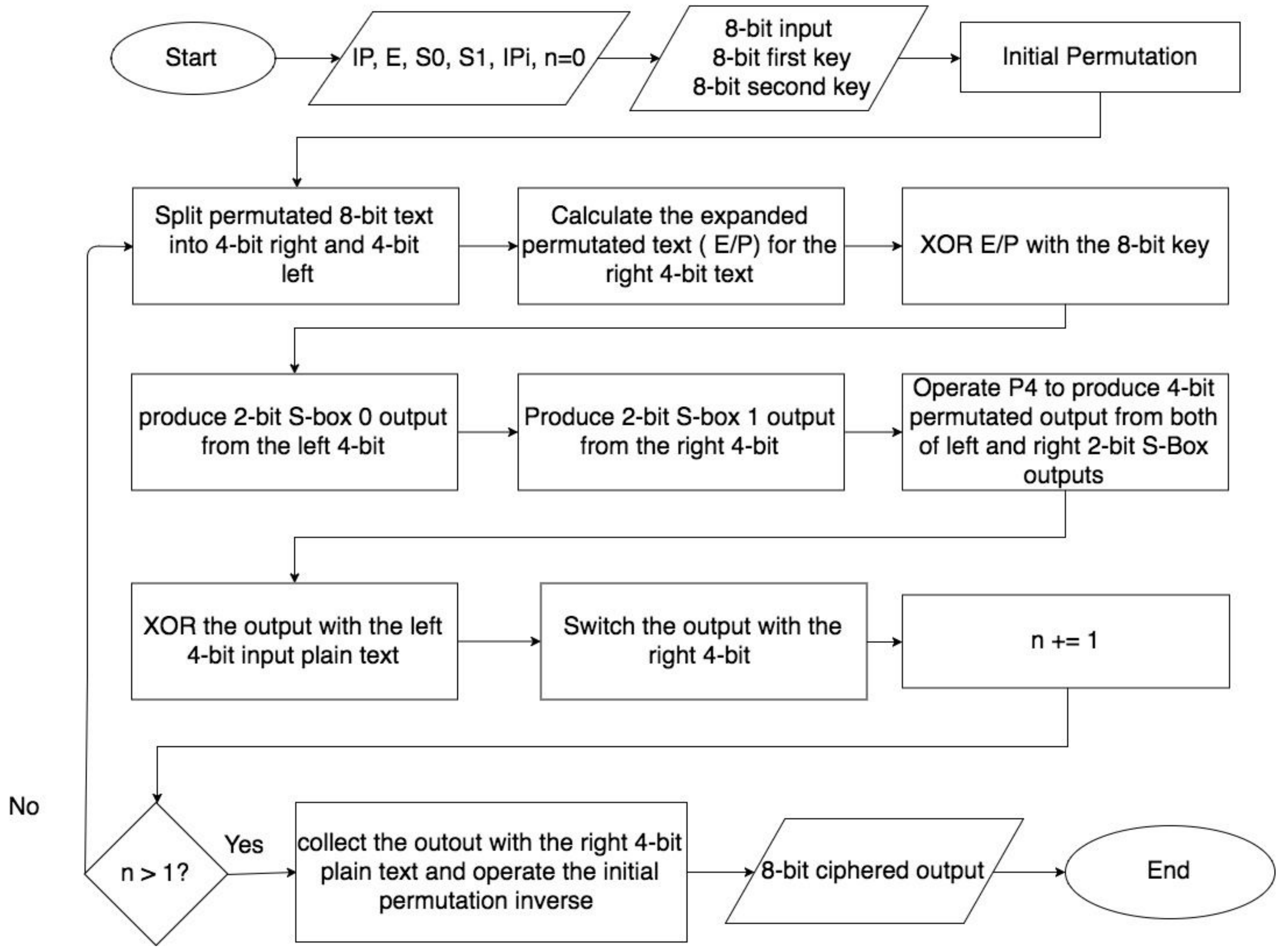
3- A S-box function ;s simple substitute function labeled **S-Box** ,which involve both input the number of S-Box

4- A fk function ;a complex function labeled **fk,** which involves both first_half nibble ,second_half nibble and the key .

The use of multiple stages of permutation and substitution results in a more complex algorithm, which increases the difficulty of cryptanalysis. The function fK takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. The algorithm could have been designed to work with a 16-bit key, consisting of two 8-bit subkeys, one used for each occurrence of fK. Alternatively, a single 8-bit key could have been used, with the same key used twice in the algorithm. A compromise is to use a 10-bit key from which two 8-bit subkeys are generated. In this case, 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). We can concisely express the encryption algorithm as a composition of functions.

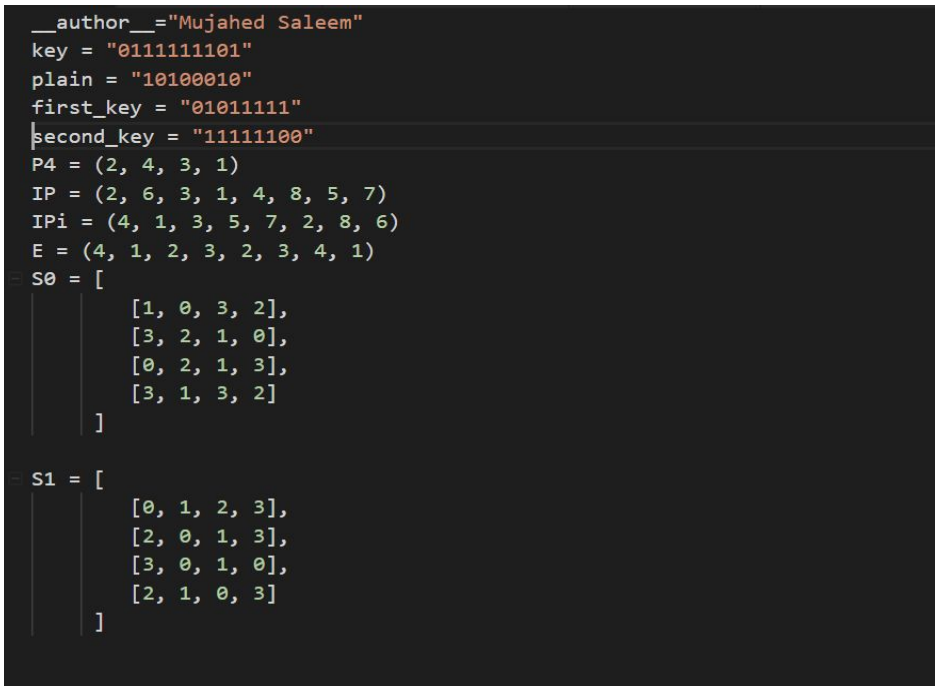
2. Project analysis

a. Flow Chart


Start
IP, E, S0, S1, IPi, n=0
8-bit input
8-bit first key
8-bit second key
Initial Permutation
Split permutated 8-bit text into 4-bit right and 4-bit left
Calculate the expanded permutated text (E/P) for the right 4-bit text
XOR E/P with the 8-bit key
produce 2-bit S-box 0 output from the left 4-bit
Produce 2-bit S-box 1 output from the right 4-bit
Operate P4 to produce 4-bit permutated output from both of left and right 2-bit S-Box outputs
XOR the output with the left 4-bit input plain text
Switch the output with the right 4-bit
n += 1
No
n > 1?
Yes
collect the outout with the right 4-bit plain text and operate the initial permutation inverse
8-bit ciphered output
End

b. Source Code Analysis.

i. Variables Declarations

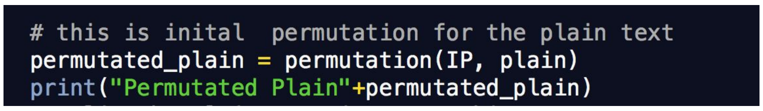


```
__author__="Mujahed Saleem"
key = "0111111101"
plain = "10100010"
first_key = "01011111"
second_key = "11111100"
P4 = (2, 4, 3, 1)
IP = (2, 6, 3, 1, 4, 8, 5, 7)
IPi = (4, 1, 3, 5, 7, 2, 8, 6)
E = (4, 1, 2, 3, 2, 3, 4, 1)
S0 = [
        [1, 0, 3, 2],
        [3, 2, 1, 0],
        [0, 2, 1, 3],
        [3, 1, 3, 2]
    ]

S1 = [
        [0, 1, 2, 3],
        [2, 0, 1, 3],
        [3, 0, 1, 0],
        [2, 1, 0, 3]
    ]
```

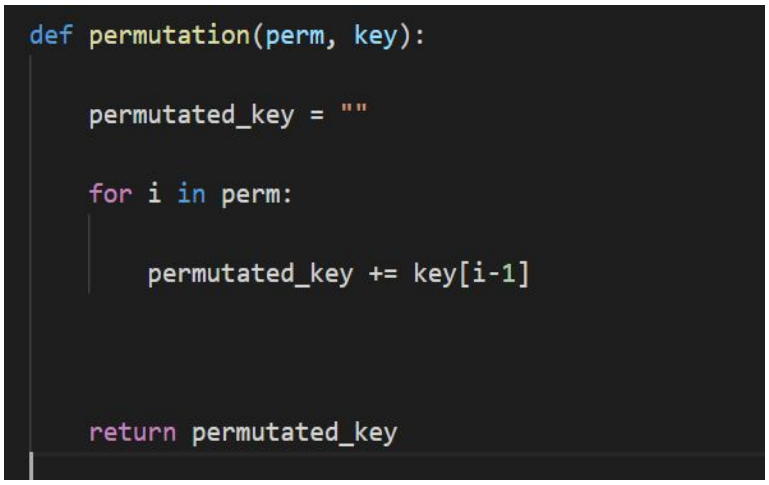
Variable Name	Description
key	the initial key in the algorithm that's shared between the sender and the receiver.
first_key	the first generated key which is predefined because of it's out of our scope.
second_key	the second generated key which is predefined because of it's out of our scope.
plaintext	the input which is intended to be encrypted.
IP	initial permutation.
IPi	initial permutation inverse.
P4	permutation that processes the output of the sbox.
E	an expansion/permutation that expands the right 4-bit input to 8-bit.
S0	s-box produces 2-bit output from the first two bits.
S1	produces another 2-bit output from the remaining 4-bits.

ii. Initial Permutation.

The input to the algorithm is an 8-bit block of plaintext, which we first permutate using the IP function: IP(1 2 3 4 5 6 7 8) = (2 6 3 1 4 8 5 7).

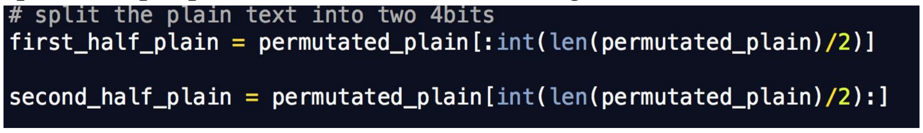


```
# this is inital  permutation for the plain text
permutated_plain = permutation(IP, plain)
print("Permutated Plain"+permutated_plain)
```



```
def permutation(perm, key):

    permutated_key = ""

    for i in perm:

        permutated_key += key[i-1]



    return permutated_key
```

iii. Split the input plain text into left 4-bit and right 4-bit.

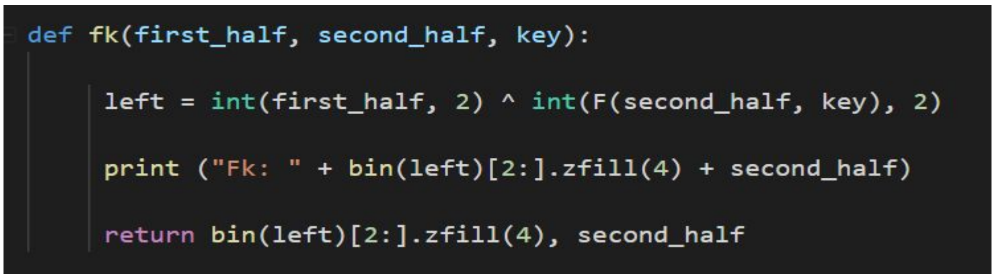


```
# split the plain text into two 4bits
first_half_plain = permutated_plain[:int(len(permutated_plain)/2)]

second_half_plain = permutated_plain[int(len(permutated_plain)/2):]
```

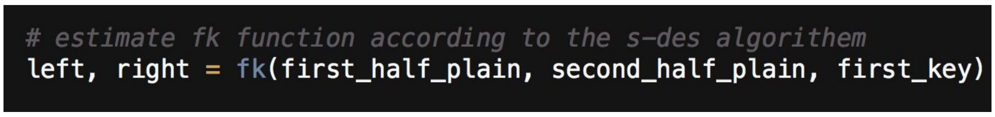
iv. Function FK.

$$f_K(L, R) = (L \oplus F(R, SK), R)$$

where SK is a subkey and $\oplus$ is the bit-by-bit exclusive-OR function.



```
def fk(first_half, second_half, key):

    left = int(first_half, 2) ^ int(F(second_half, key), 2)

    print ("Fk: " + bin(left)[2:].zfill(4) + second_half)

    return bin(left)[2:].zfill(4), second_half
```



```
# estimate fk function according to the s-des algorithem
left, right = fk(first_half_plain, second_half_plain, first_key)
```

The most complex component of S-DES is the function fK, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to fK, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings.

v. Function F.

```
left = int(first_half, 2) ^ int(F(second_half, key), 2)
```

The input is a 4-bit number R, the first operation is an expiation; which is a type of permutation so we use function permutation to do this job ,the output of this operation is a 8-bit .then we do the XOR operation with a subkey . then we divide the 8-bit output into two 4-bit R and L . R goes to S-Box function with determine which S_Box desired this function return 2-bit for each call , and here we have 2 calls so we have 4-bit output. And as a finale stage of F function we do permutation using permutation function .

```
def F(right, subkey):

    expanded_plain = permutation(E, right)

    xor_plain = bin( int(expanded_plain, 2) ^ int(subkey, 2) )[2:].zfill(8)
    print(xor_plain)
    left_xor_plain = xor_plain[:4]

    right_xor_plain = xor_plain[4:]

    left_sbox_plain = Sbox(left_xor_plain, S0)

    right_sbox_plain = Sbox(right_xor_plain, S1)

    return permutation(P4, left_sbox_plain + right_sbox_plain)
```

vi. Calculate the expanded permuted text.

```
expanded_plain = permutation(E, right)
```

vii. XOR E/P with 8-bit key.

```
xor_plain = bin( int(expanded_plain, 2) ^ int(subkey, 2) )[2:].zfill(8)
print("The output" +xor_plain)
```

viii. Produce two 2-bit output from S-Box 0 and S-Box 1

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 3 & 2 \\ 1 & 3 & 2 & 1 & 0 \\ 2 & 0 & 2 & 1 & 3 \\ 3 & 3 & 1 & 3 & 2 \end{array} \qquad S1 = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 2 & 3 \\ 1 & 2 & 0 & 1 & 3 \\ 2 & 3 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 & 3 \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the Sbox. The entry in that row and column, in base 2, is the 2-bit output.

```
def Sbox(input, sbox):

    row = int(input[0] + input[3], 2)

    column = int(input[1] + input[2], 2)

    return bin(sbox[row][column])[2:].zfill(4)
```

```
left_xor_plain = xor_plain[:4]
right_xor_plain = xor_plain[4:]
left_sbox_plain = Sbox(left_xor_plain, S0)
right_sbox_plain = Sbox(right_xor_plain, S1)
print("The output" + xor_plain)
```

ix. Produce 4-bit permuted output from P4.

```
return permutation(P4, left_sbox_plain + right_sbox_plain)
```

x. XOR the output with the left 4-bit plain input.

```
left = int(first_half, 2) ^ int(F(second_half, key), 2)
```

xi. Switch the left 4 bits and the right 4 bits then reinvoke the procedures from (iii).

```
left, right = fk(right, left, second_key)
```

xii. Collect the output with the right 4 bits and operate the initial permutation inverse then print the output.

This retains all 8 bits of the plaintext, but mixes them up. At the end of the algorithm, the inverse permutation is used: IP inverse(1 2 3 4 5 6 7 8) = (4 1 3 5 7 2 8 6).

```
print ("IP^-1(Cipher): " + permutation(IPi, left + right))
```

Function Name	**Arguments**	**Description**
Permutation	• type of permutation. • Input bits.	process the input bits due to the type of permutation.
F	• the right 4 bits of the input. • key of the round.	processes the right 4-bit input with the key of the round.

Sbox	• input of bits. • sbox index.	processes 4-bit input to 2-bit output by substitution.
Fk	• output of F. • left 4-bit of plain text.	processes XOR operation on the inputs.

c. Source Code.

```
key = "0111111101"

plain = "10100010"

P4 = (2, 4, 3, 1)

IP = (2, 6, 3, 1, 4, 8, 5, 7)

IPi = (4, 1, 3, 5, 7, 2, 8, 6)

E = (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]
    ]

def permutation(perm, key):
    permutated_key = ""
    for i in perm:
        permutated_key += key[i-1]
    return permutated_key

def F(right, subkey):
    expanded_plain = permutation(E, right)
    xor_plain = bin( int(expanded_plain, 2) ^ int(subkey, 2) )[2:].zfill(8)
    print(xor_plain)
    left_xor_plain = xor_plain[:4]
    right_xor_plain = xor_plain[4:]
    left_sbox_plain = Sbox(left_xor_plain, S0)
    right_sbox_plain = Sbox(right_xor_plain, S1)
    return permutation(P4, left_sbox_plain + right_sbox_plain)
```

```
def Sbox(input, sbox):
   row = int(input[0] + input[3], 2)
   column = int(input[1] + input[2], 2)
   return bin(sbox[row][column])[2:].zfill(4)

def fk(first_half, second_half, key):
    left = int(first_half, 2) ^ int(F(second_half, key), 2)
    print ("Fk: " + bin(left)[2:].zfill(4) + second_half)
    return bin(left)[2:].zfill(4), second_half

first_key = "01011111"
second_key = "11111100"

# this is inital  permutation for the plain text
permutated_plain = permutation(IP, plain)

# split the plain text into two 4bits
first_half_plain = permutated_plain[:int(len(permutated_plain)/2)]
second_half_plain = permutated_plain[int(len(permutated_plain)/2):]

# estimate fk function according to the s-des algorithem
left, right = fk(first_half_plain, second_half_plain, first_key)
print("SW: " + right + left)
left, right = fk(right, left, second_key) # switch left and right!
```

[*] https://github.com/AbdulRuhman/S-Des

3. Tools and Technologies.

- Python Programming Language.
- Pycharm IDE.
- Drawi.io
- Google drive