

## School of Computer Science and Engineering

**OPERATING SYSTEM**

#### PROJECT COMPONENT

**PROTECTION OF FILE SYSTEM IN OPERATING SYSTEM USING ENCRYPTION METHOD**

SUBMITTED TO:

Dr. Kalyanraman P

SUBMITTED BY:

Ashutosh Choudhari - 16BEI0037

Rohit Mehta – 18BCE0199

Peddi Sai Prabhat – 19BCI0234

##### ABSTRACT:

###### Storage systems are increasingly subject to attacks. Cryptographic document frameworks alleviate the threat of uncovering information by utilizing encryption and integrity security strategies and ensure end-to-end security for their customers. This paper describes the general architecture of cryptographic file systems and their implementation in the SAN file system (SAN). Key administration is incorporated with the meta-information administration of the SAN file system. The two systems have been executed in the customer record framework driver. Benchmarks illustrate that the overhead is recognizable for some falsely developed utilize cases, however that it is little for normal document framework applications.

INTRODUCTION:

Data Security has always remained an integral and indispensable part of operating system. OS security refers to the steps or steps used to protect the OS from threats, viruses, worms, malware or remote hacker access. OS security includes all security control techniques, which protect any computer assets that are stolen, edited or deleted when OS security is compromised. Therefore, the computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

* Authentication
* One Time passwords
* Program Threats
* System Threats
* Computer Security Classifications

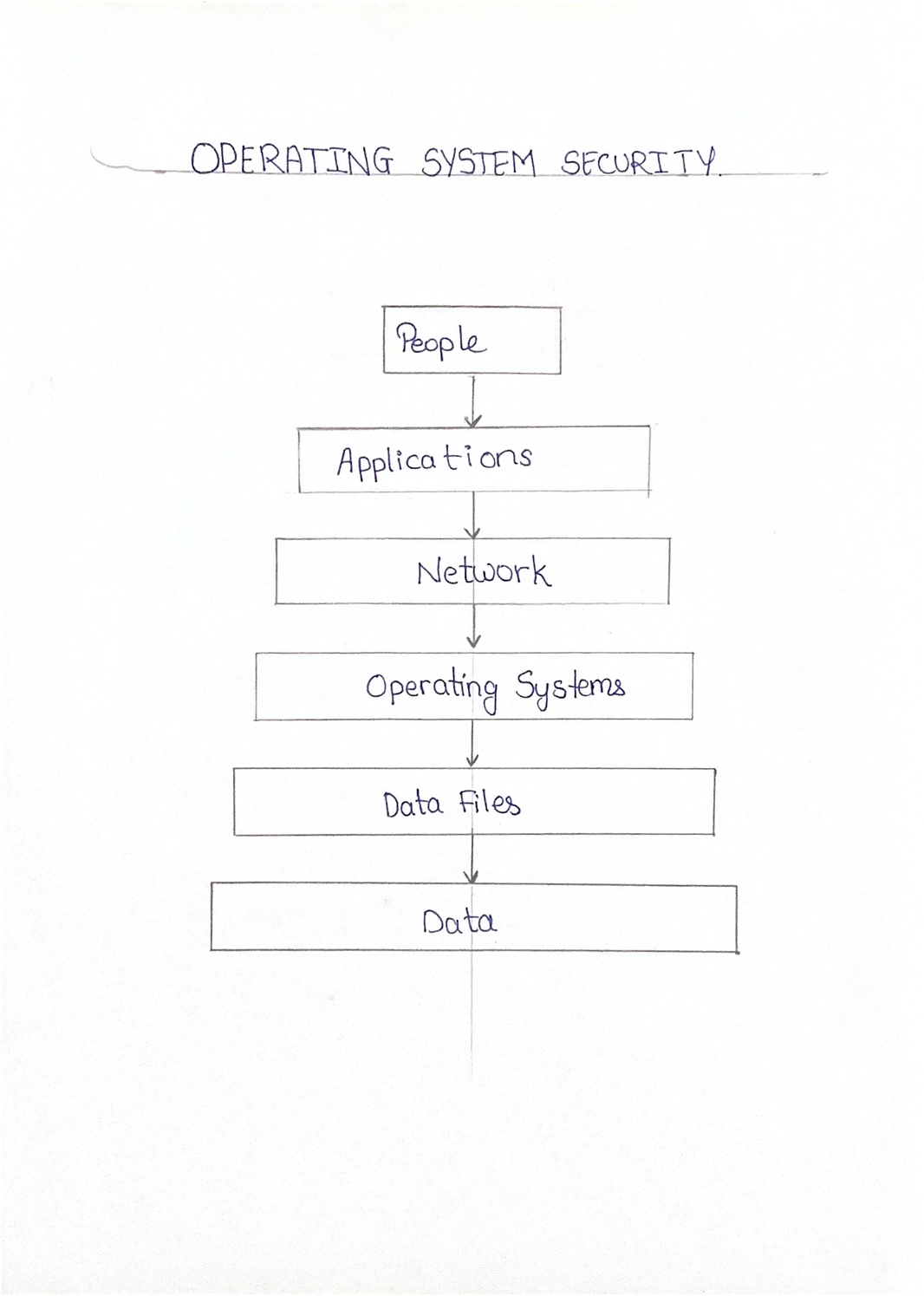
The basis of OS protection is to partition. The classification can be of four different types:

- Physical: physical objects, such as CPUs, printers, etc.

- Temporary: killed at different times

- Logical: domains, each user receives an idea

- Cryptographic: hiding data, so that some users may not understand it



Key areas that need to security measures in operating system:

1. User Accounts
2. Account Policies
3. File System
4. Network Services

If we develop some basic cryptosystem that has the objective to encrypt and decrypt the given data then few fundamentals that must be kept in mind are as follows:

**Plaintext:** It is data that should be protected while transmitting.

Encryption Algorithm: It is a mathematical process that generates a ciphertext for any specific rendering and encryption. It is a cryptographic algorithm that takes plaintext and encryption keys as input and produces ciphertext.

Contents: It is a complex version of text generated by an encryption algorithm using a specific encryption key. The appendix is ​​not monitored. It flows through a public channel. It may be restricted or restricted to anyone who has access to a communication channel.

**Contents:** It is a complex version of text generated by an encryption algorithm using a specific encryption key. The appendix is ​​not monitored. It flows through a public channel. It may be restricted or restricted to anyone who has access to a communication channel.

**Decryption Algorithm:** It is a mathematical process, which produces a unique image of any given input and key. It's a cryptographic algorithm that takes ciphertext and encryption keys as input, and comes out short. The decryption algorithm actually returns the encryption algorithm and is therefore closely related to it.

**Encryption Key:** It is a known value for the sender. The sender inserts the encryption key in the encryption algorithm and theintext to compile the collection.

**Decryption Key:** The value known to the recipient. The translation key is related to the compose button, but it is not always the same. The receiver includes a translation key in the compilation algorithm and a prefix for the compiled link.

Literature Review:

The DES algorithm is acquired as the Federal Information Processing Standard. DES encrypts 64-bit blocks with a 56-bit key. The use of triple encryption with two different keys alleviates concerns of cryptanalysis using an exhaustive key search. The algorithm consists of 16 cycles or iterations, each of which consists of 32 left turns, as well as permchas, substitutions and XOR operations. Note that all permutation and expansion operations are implemented with wire crossings and fanout. A different subset of the key bits is used in each round Decryption is similar to encryption, except that subkeys are generated in reverse order..

This algorithm is implemented to provide encryption in Virtex FPGAs made by Xilinx. According to Patterson (2000), the Virtex architecture efficiently implements the DES primitive operations, and permits a high degree of pipelining JBits provides a Java-based Application Programming Interface (API) for running time creation and bitching of configuration. This allows for stronger circuit flexibility based on a specific key and mode (secret or translation). The main schedule is completely calculated on the software, and is part of the flow. Because of this, all cryptographic and subkey generation keystrokes are removed from a completely unregistered database. When combined with a speed efficient layout, the result is a throughput of over 10 Giga-bits per second. This exceeds the performance of then recently announced DES ASIC.

In the paper by Coppersmith (May 1994), the paper focuses on the strengths of the algorithm against attacks. In the paper, the DES algorithm is first explained. Then the attacks to the DES algorithm are then described. The paper later explains the disclosure of the design criteria of the S boxes and permutation. The paper later goes on to explain why the algorithm has failed against the attacks.

After understanding the working and applications of DES algorithm, we turn to SDES algorithm. SDES is a simplified version of DES, hence the SDES algorithm is used in the J component. The SDES is a simplified version of a well-known Data Encryption Standard (DES). It was designed in 1996 by Edward Schaefer for general academic purposes. SDES is a symmetric cipher, that is, the same key is used to encrypt and decrypt information. It is a block cipher. The length of a single encrypted message block equals 8 bits and the data is encrypted by a 10-bit key. The SDES operates on strings of bits instead of normal characters. The decryption process uses the same encrypting algorithm but the subkeys are provided in reverse order. SDES is a two-round algorithm, based on two basic operations: combination done with permutations and dispersion

Today, data security is based on ciphers and cryptographic systems. One that examines the quality of such security measures is cryptanalysis The paper by Dworak(2015), presents a new cryptanalysis attack aimed at a ciphertext generated with the use of the SDES (Simplified Data Encryption Standard). The attack was carried out with a modified version of the BPSO (Binary Particle Swarm Optimization) algorithm. A well-adjusted version of this method can have a positive effect on the quality of the results obtained in a given period of time.

In recent years network security has become an important issue. Encryption has come up as a solution, and plays an important role in information security system. Many techniques are needed to protect the shared data. In the paper by Mahajan (2013), the work focuses on cryptography to secure the data while transmitting in the network. Firstly the data which is to be transmitted from sender to receiver in the network must be encrypted using the encryption algorithm in cryptography. Secondly, by using decryption technique the receiver can view the original data. In the paper they have implemented three encrypt techniques like AES, DES and RSA algorithms and compared their performance of encrypt techniques based on the analysis of its stimulated time at the time of encryption and decryption. Experiments results are given to analyses the effectiveness of each algorithm.

The cryptanalysis of the simplified process of data encryption can be performed as an NP-Hard integration problem.In the paper by Garg(2009), Firstly a study about how evolutionary computation techniques can efficiently solve the NP-Hard combinatorial problem is conducted. To achieve this goal, a few integrative integration techniques such as the memetic algorithm, genetic algorithm and cryptanalysis estimation of standard normal data encoding (SDES). And secondly comparisons were made between the memetic algorithm, the genetic algorithm and the annealing was performed to investigate the effectiveness of cryptanalysis in SDES. Methods were tested and multiple diagnostic results indicate that the memetic algorithm works better than genetic algorithms and is actively monitoring some form of NP-Hard integration problem. This paper represents effort toward efficient memetic algorithm for the cryptanalysis of SDES.

In the paper by Dworak(2016), evolutionary algorithms are exploited for cryptanalysis and focus is on a chosen-plaintext attack model, in which the attacker is able to access both the ciphertext and the plaintext. The aim of this attack is to determine the decryption key for the Simplified Data Encryption Standard (SDES), so that other encrypted texts can be easily deciphered. We propose extracting key using genetic and memetic algorithms (the latter being a combination of evolutionary strategies and specific analytical processes).An extensive experimental study, coupled with the sensitivity analysis on method components and statistical tests, show the convergence capabilities and prove they are very competitive compared with other state-of-the-art algorithms.

Objective:

* + Understanding the need of data security and integrity to be included during design of distributed operating system.
  + Types of cryptography modules used by different operating system store data and authorize users.
  + Implement a file encryption technique using DES for the windows

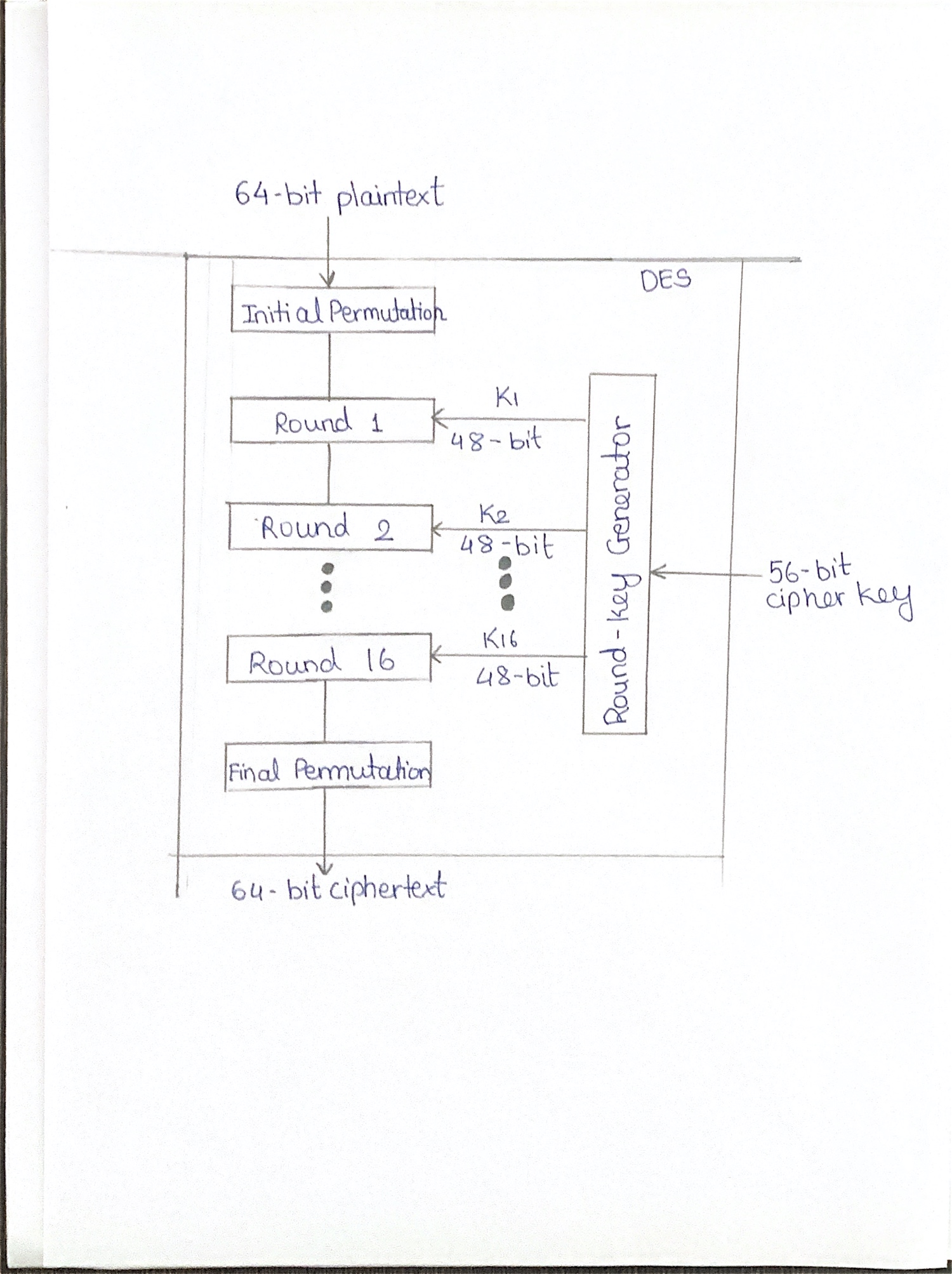
platform.

* + Comparing the time and space aspects of decrypted file and encrypted file.

**ABOUT DES:**

The Data Encryption Standard (DES) is an icmmetric-key block cipher issued by the National Institute of Standards and Technology (NIST).

DES implementation of Feistel Cipher. Uses 16 Feistel circular structure. The block size is 64-bit. Or, the key length is 64-bit, DES has a valid key of 56 bits, since 8 of the 64 key bits are used by the encryption algorithm (functionality only checks).



**SCOPE:**

• First, the user completes the agreement form of the program and then the user either files a text massage or files a text massage.

• Encryption key as input as a result, the program will encrypt and decrypt the text massage and save it to a file.

**ANALYSIS OF S-DES Algorithm:**

**Key generator**

We analyzed the project that we would take the input (encryption key) from the user, we would convert this key to 10 bit binary. Then we will divide the 10 bit binary key in 5-5 bits .we will do LS-1(LEFT [1bit]) of these 5-5 bits .we will merge and arrange these bits according to the following rule

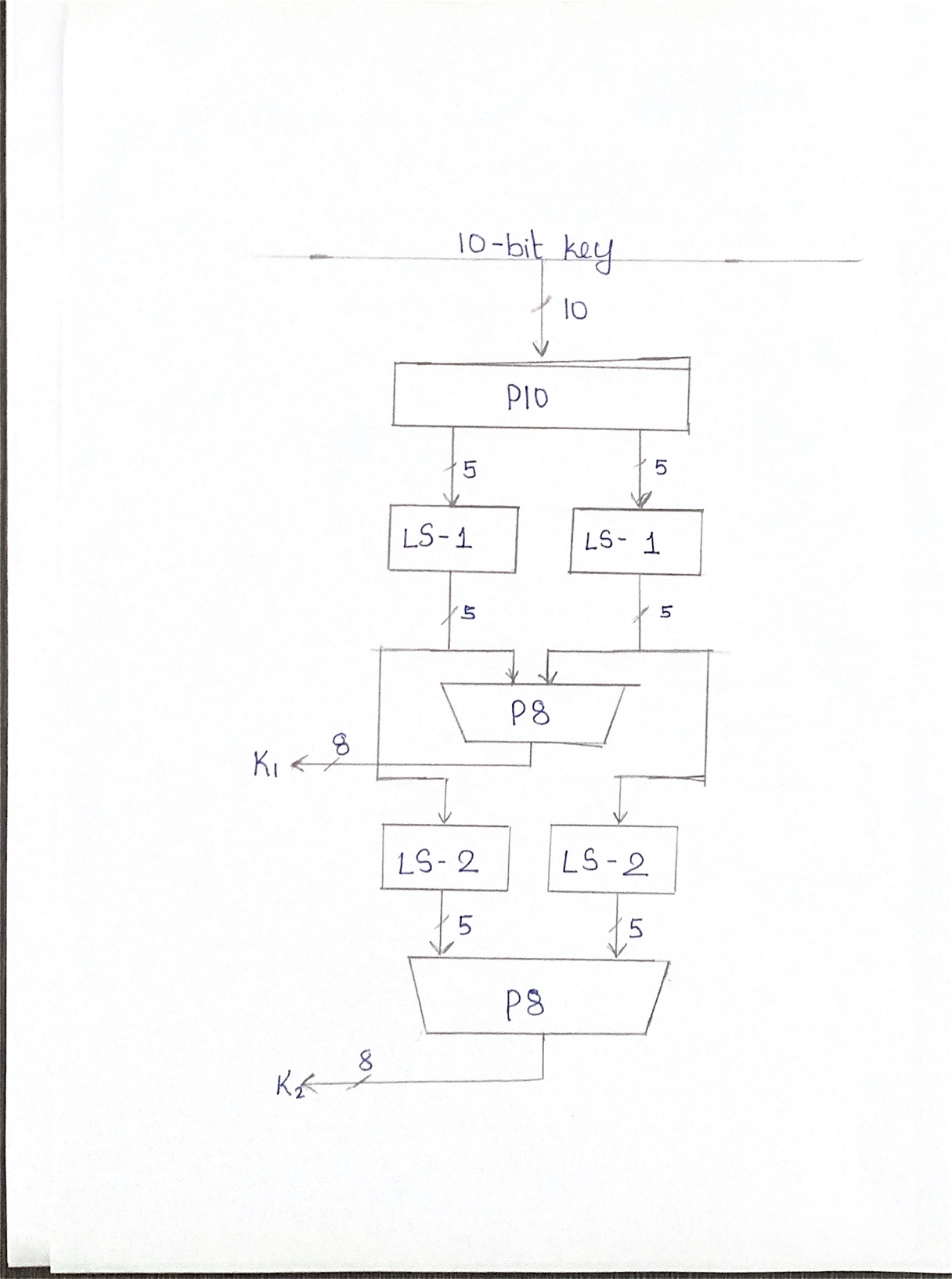
Through this we will get a key(K1) of 8 bit.

P8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 3 | 7 | 4 | 8 | 5 | 10 | 9 |

From those 5-5 pieces (LS-1), as well as key generating (K1),

we will do LS-2(left shift [2bit]) by the help of above rule (P8) we have merged (LS-2),we will get key(key2).



Flow chart of key generator

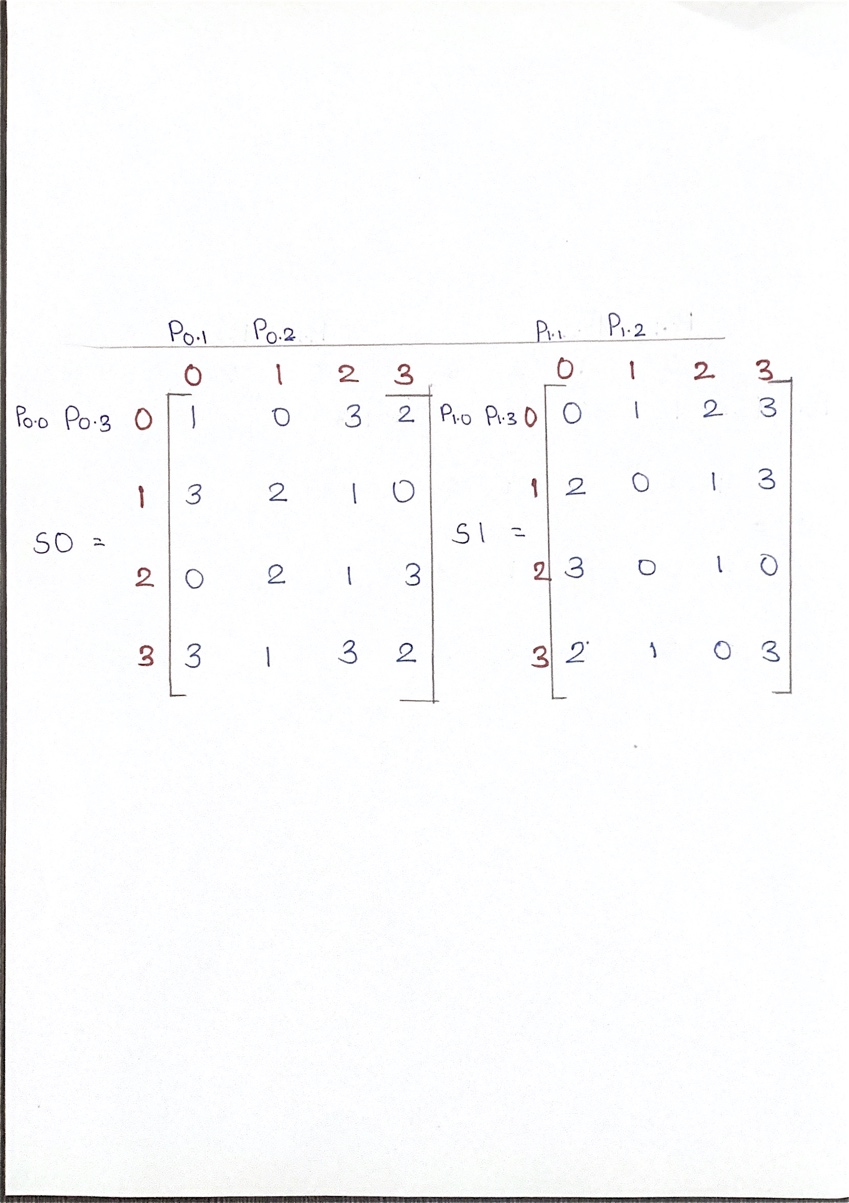
**Encryption detail**

We will take any alphabet or integer from the user and will convert into an 8 bit (IP BINARY), we will divide this 8-bit binary in to 4- 4 bit as left 4bit and right 4bit **IP.** We will convert right 4bit binary into 8 bit binary and arrange it according to the following rule.

F/P

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 1 | 2 | 3 | 2 | 3 | 4 | 1 |

After solving S0 and S1, we will get 2-2 bit binary and arrange them with RULE P4.Now we will add 4 IP to the left with this set of 2-2 bit binary. by using the switch function we will repeat this method and use key 2 instead of key 1 for the rest of this program, the user input provided will encrypt.

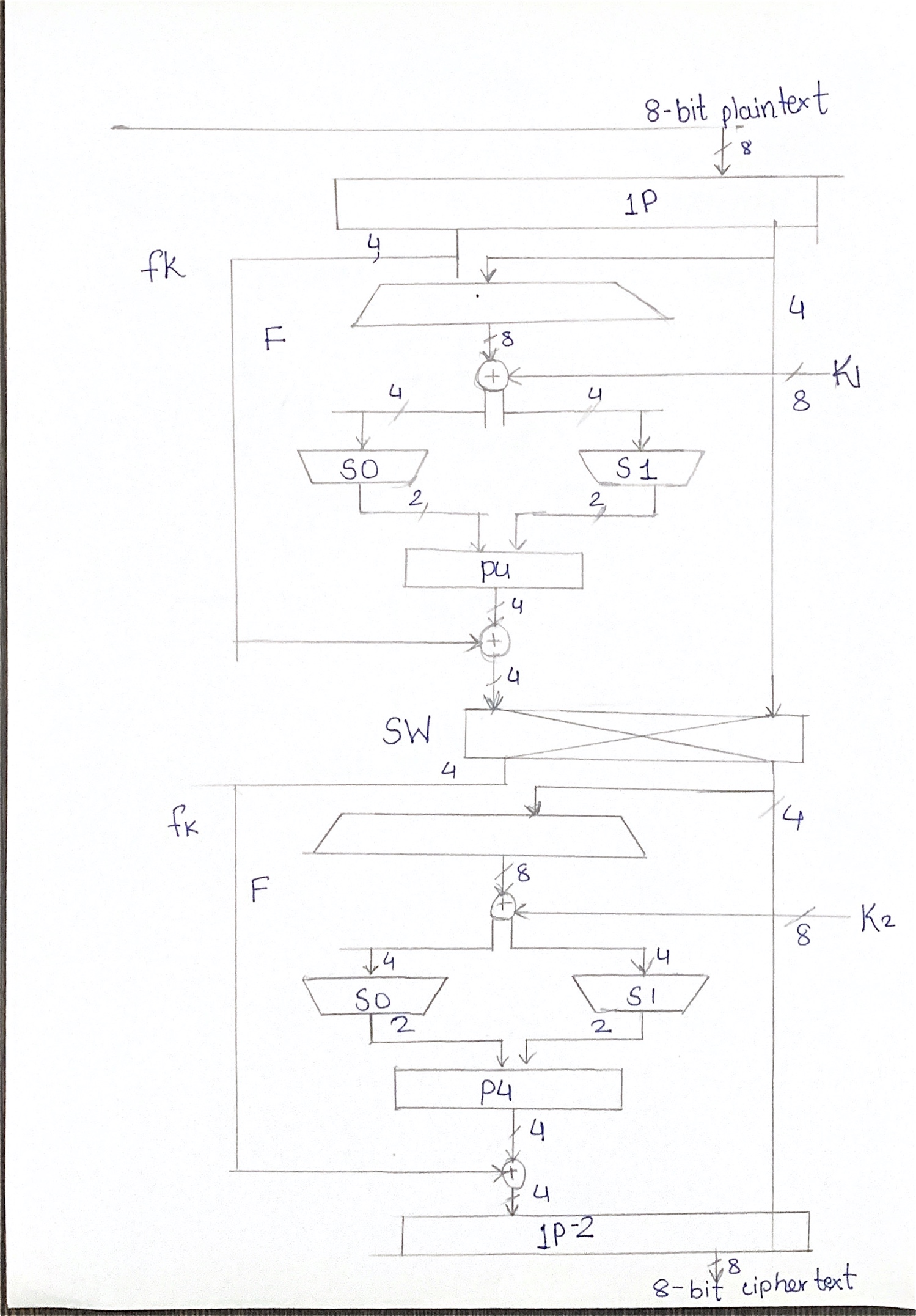


After solving S0 and S1 ,we will get 2-2 bit binary and arrange them through RULE P4.

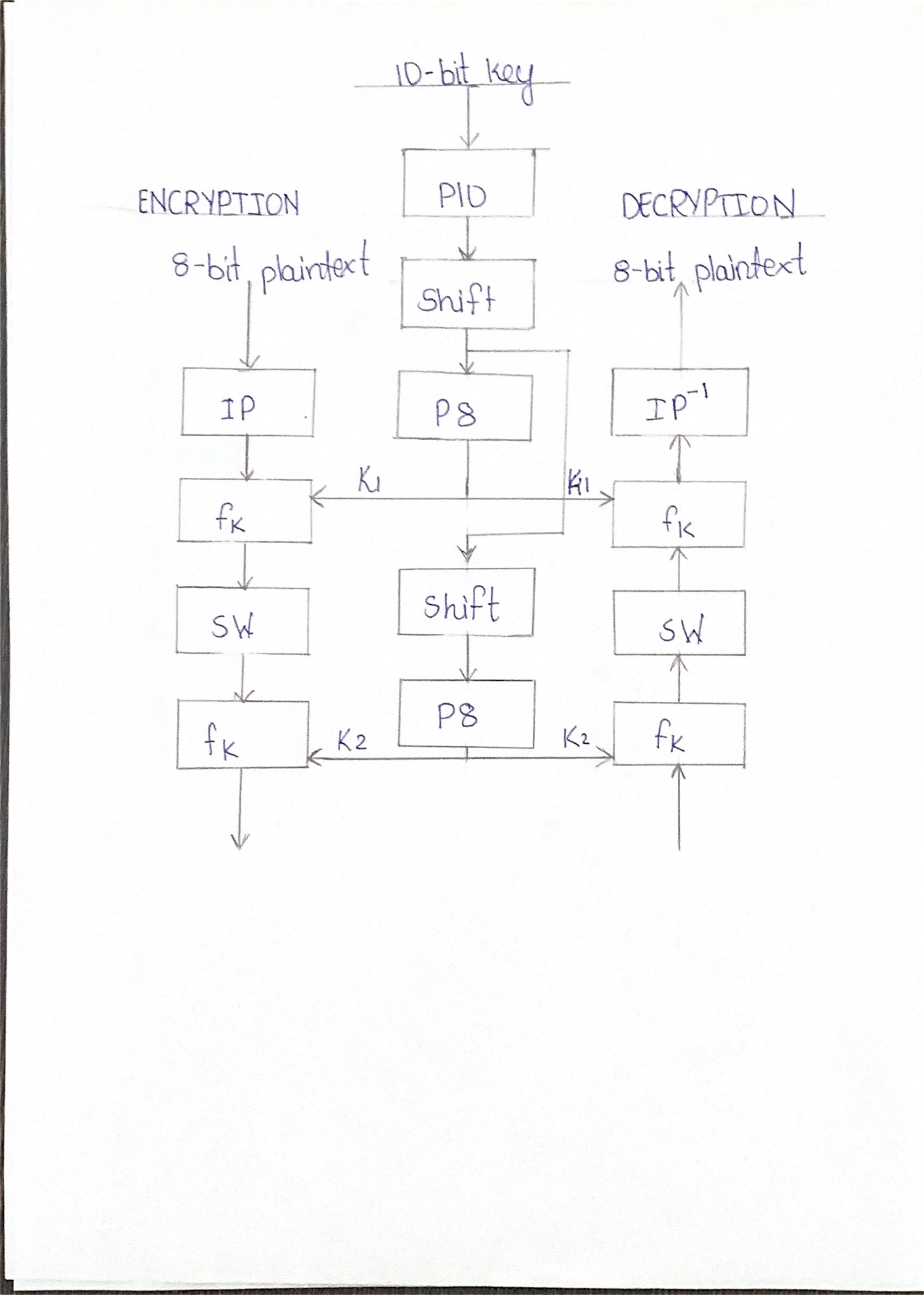
P4

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 3 | 1 |

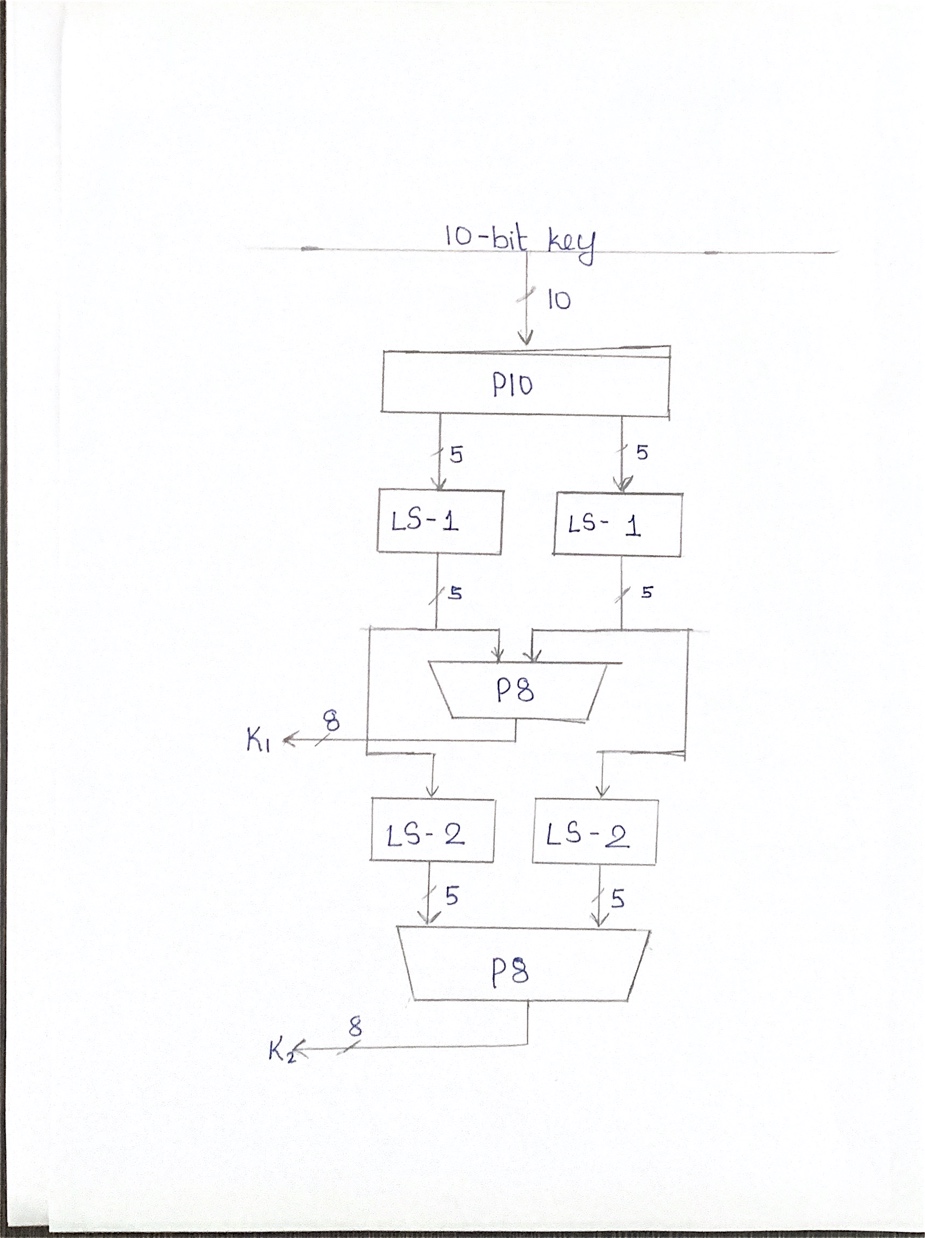
Now we will add the left 4 bit IP with this arranged 2-2 bit binary. by using switch function we will repeat this method and use key 2 instead of key 1 as a result of this whole scheme the user ‘s given massage will encrypt.

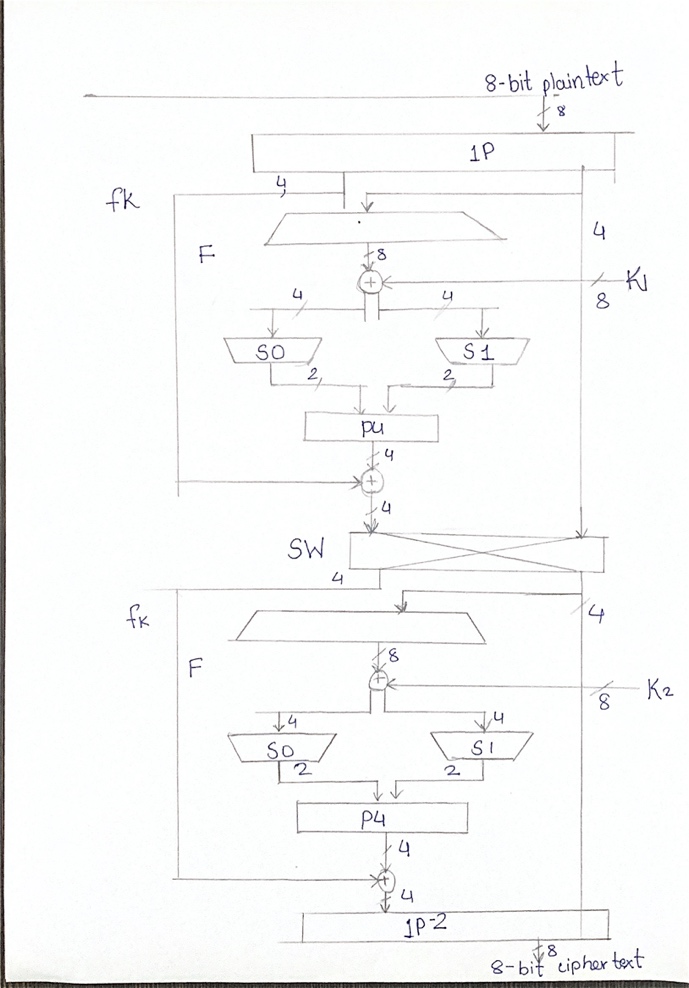


**STRUCTURE OF ENCRYPTION AND DECRYPTION**



**CONTROL FLOW OF ALGORITHM:**





## CODE:

# #include <iostream>

# #include <fstream>

# int f99=0,f98=0,f97=0;

# using namespace std;

# void binary(int num,int bin[]){

# int i,loop=9;

# while(loop>=0){

# bin[loop]=num%2;

# num=num/2;

# loop--;

# }

# }

# void arr\_p10(int bin[],int p10[]){

# int i;

# p10[0]=bin[2];

# p10[1]=bin[4];

# p10[2]=bin[1];

# p10[3]=bin[6];

# p10[4]=bin[3];

# p10[5]=bin[9];

# p10[6]=bin[0];

# p10[7]=bin[8];

# p10[8]=bin[7];

# p10[9]=bin[5];

# if(f98==0)

# {

# cout<<"After Permutation "; for(i=0;i<=9;i++)

# {

# cout<<bin[i];

# }

# cout<<endl<<endl; f98=1;

# }

# }

# void left\_shift\_1(int p10[],int left\_left\_shift\_1[],int right\_left\_shift\_1[])

# {

# int i;

# left\_left\_shift\_1[0]=p10[1];

# left\_left\_shift\_1[1]=p10[2];

# left\_left\_shift\_1[2]=p10[3];

# left\_left\_shift\_1[3]=p10[4];

# left\_left\_shift\_1[4]=p10[0];

# right\_left\_shift\_1[0]=p10[6];

# right\_left\_shift\_1[1]=p10[7];

# right\_left\_shift\_1[2]=p10[8] ;

# right\_left\_shift\_1[3]=p10[9] ;

# right\_left\_shift\_1[4]=p10[5];

# if(f97==0){

# cout<<"After Left Part Left Shift ";

# for(i=0;i<=4;i++)

# {

# cout<<left\_left\_shift\_1[i];

# }

# cout<<endl<<endl;

# cout<<"After Right Part Left Shift ";

# for(i=0;i<=4;i++)

# {

# cout<<right\_left\_shift\_1[i];

# }

# cout<<endl<<endl;

# f97=1;

# }

# }

# void per\_8(int left\_left\_shift\_1[],int right\_left\_shift\_1[],int subkey1[])

# {

# subkey1[0]=right\_left\_shift\_1[0] ;

# subkey1[1]=left\_left\_shift\_1[2];

# subkey1[2]=right\_left\_shift\_1[1] ;

# subkey1[3]=left\_left\_shift\_1[3];

# subkey1[4]=right\_left\_shift\_1[2] ;

# subkey1[5]=left\_left\_shift\_1[4];

# subkey1[6]=right\_left\_shift\_1[4] ;

# subkey1[7]=right\_left\_shift\_1[3];

# }

# void left\_shift\_2(int left\_left\_shift\_1[],int right\_left\_shift\_1[],int left\_left\_shift\_2[],int right\_left\_shift\_2[])

# {

# left\_left\_shift\_2[0]=left\_left\_shift\_1[2] ;

# left\_left\_shift\_2[1]=left\_left\_shift\_1[3] ;

# left\_left\_shift\_2[2]=left\_left\_shift\_1[4] ;

# left\_left\_shift\_2[3]=left\_left\_shift\_1[0] ;

# left\_left\_shift\_2[4]=left\_left\_shift\_1[1];

# right\_left\_shift\_2[0]=right\_left\_shift\_1[2] ;

# right\_left\_shift\_2[1]=right\_left\_shift\_1[3] ;

# right\_left\_shift\_2[2]=right\_left\_shift\_1[4] ;

# right\_left\_shift\_2[3]=right\_left\_shift\_1[0] ;

# right\_left\_shift\_2[4]=right\_left\_shift\_1[1];

# }

# void key\_generation(int num,int subkey1[],int subkey2[])

# {

# int i,bin[10];

# binary(num,bin);

# if(f99==0){

# for(i=0;i<=9;i++){

# cout<<bin[i]<<" ";

# }

# f99=1;

# cout<<endl<<endl;

# }

# int p10[10];

# arr\_p10(bin,p10);

# int left\_left\_shift\_1[5],right\_left\_shift\_1[5];

# left\_shift\_1(p10,left\_left\_shift\_1,right\_left\_shift\_1);

# per\_8(left\_left\_shift\_1,right\_left\_shift\_1,subkey1);

# int left\_left\_shift\_2[5],right\_left\_shift\_2[5];

# left\_shift\_2(left\_left\_shift\_1,right\_left\_shift\_1,left\_left\_shift\_2,right\_left\_shift\_2);

# per\_8(left\_left\_shift\_2,right\_left\_shift\_2,subkey2);

# }

# void inner\_permutate(int bin[],int inner\_permutate\_arrange[])

# {

# inner\_permutate\_arrange[0]=bin[3] ;

# inner\_permutate\_arrange[1]=bin[7] ;

# inner\_permutate\_arrange[2]=bin[4] ;

# inner\_permutate\_arrange[3]=bin[2] ;

# inner\_permutate\_arrange[4]=bin[5] ;

# inner\_permutate\_arrange[5]=bin[9] ;

# inner\_permutate\_arrange[6]=bin[6] ;

# inner\_permutate\_arrange[7]=bin[8];

# }

# void seperate(int inner\_permutate\_arrange[],int left\_inner\_permutate[],int right\_inner\_permutate[])

# {

# left\_inner\_permutate[0]=inner\_permutate\_arrange[0];

# left\_inner\_permutate[1]=inner\_permutate\_arrange[1];

# left\_inner\_permutate[2]=inner\_permutate\_arrange[2];

# left\_inner\_permutate[3]=inner\_permutate\_arrange[3];

# right\_inner\_permutate[0]=inner\_permutate\_arrange[4] ;

# right\_inner\_permutate[1]=inner\_permutate\_arrange[5] ;

# right\_inner\_permutate[2]=inner\_permutate\_arrange[6] ;

# right\_inner\_permutate[3]=inner\_permutate\_arrange[7];

# }

# void FP\_arrange(int right\_inner\_permutate[],int fp[])

# {

# fp[0]=right\_inner\_permutate[3] ;

# fp[1]=right\_inner\_permutate[0] ;

# fp[2]=right\_inner\_permutate[1] ;

# fp[3]=right\_inner\_permutate[2] ;

# fp[4]=right\_inner\_permutate[1] ;

# fp[5]=right\_inner\_permutate[2] ;

# fp[6]=right\_inner\_permutate[3] ;

# fp[7]=right\_inner\_permutate[0];

# }

# void zor(int fp[],int subkey1[],int left\_zor[4],int right\_zor[4])

# {

# if(fp[3]==subkey1[0]){

# left\_zor[0]=0;

# }

# else if (fp[3]!=subkey1[0]){

# left\_zor[0]=1;

# }

# if(fp[0]==subkey1[1]){

# left\_zor[1]=0;

# }

# else if (fp[0]!=subkey1[1]){

# left\_zor[1]=1;

# }

# if(fp[1]==subkey1[2]){

# left\_zor[2]=0;

# }

# else if (fp[1]!=subkey1[2]){

# left\_zor[2]=1;

# }

# if(fp[2]==subkey1[3]){

# left\_zor[3]=0;

# }

# else if (fp[2]!=subkey1[3]){

# left\_zor[3]=1;

# }

# ///// right\_zor /////

# if(fp[1]==subkey1[4]){

# right\_zor[0]=0;

# }

# else if (fp[1]!=subkey1[4])

# {right\_zor[0]=1;

# }

# if(fp[2]==subkey1[5])

# {right\_zor[1]=0;

# }

# else if (fp[2]!=subkey1[5]){

# right\_zor[1]=1;

# }

# if(fp[3]==subkey1[6])

# {right\_zor[2]=0;

# }

# else if (fp[3]!=subkey1[6])

# {right\_zor[2]=1;

# }

# if(fp[0]==subkey1[7])

# {right\_zor[3]=0;

# }

# else if (fp[0]!=subkey1[7]){

# right\_zor[3]=1;

# }

# }

# void S\_box(int left\_zor[],int right\_zor[],int s0[] ,int s1[] )

# {

# int s\_box0[4][4]={{1,0,3,2},{3,2,1,0},{0,2,1,3},{3,1,3,2}};

# int row=(left\_zor[0]\*2)+(left\_zor[3]\*1),col=(left\_zor[1]\*2)+(left\_zor[2]\*1);

# int num;

# num=s\_box0[row] [col];

# for(int l=1;l>=0;l--)

# {

# s0[l]=num%2;

# num=num/2;

# }

# int s\_box1[4][4]={{0,1,2,3},{2,0,1,3},{3,0,1,0},{2,1,0,3}};

# int row1=(right\_zor[0]\*2)+(right\_zor[3]\*1),col1=(right\_zor[1]\*2)+(right\_zor[2]\*1);

# num=s\_box1[row1][col1];

# for(int l=1;l>=0;l--)

# {

# s1[l]=num%2;

# num=num/2;

# }

# }

# void P4\_arrange(int s0[],int s1[],int p4[])

# {

# p4[0]=s0[1];

# p4[1]=s1[1];

# p4[2]=s1[0];

# p4[3]=s0[0];

# }

# void P4\_zor(int left\_inner\_permutate[],int p4[],int sw\_right\_inner\_permutate[])

# {

# for(int l=0;l<4;l++)

# {

# if(left\_inner\_permutate[l]==p4[l]){

# sw\_right\_inner\_permutate[l]=0;

# }

# if(left\_inner\_permutate[l]!=p4[l]){

# sw\_right\_inner\_permutate[l]=1;

# }

# }

# }

# void inner\_permutate\_invers(int sw\_zor[],int sw\_right\_inner\_permutate[],int inner\_permutate\_inv[])

# {

# inner\_permutate\_inv[0]=sw\_zor[3];

# inner\_permutate\_inv[1]=sw\_zor[0];

# inner\_permutate\_inv[2]=sw\_zor[2];

# inner\_permutate\_inv[3]=sw\_right\_inner\_permutate[0];

# inner\_permutate\_inv[4]=sw\_right\_inner\_permutate[2];

# inner\_permutate\_inv[5]=sw\_zor[1];

# inner\_permutate\_inv[6]=sw\_right\_inner\_permutate[3];

# inner\_permutate\_inv[7]=sw\_right\_inner\_permutate[1];

# }

# int E(int leter\_num,int key\_num )

# {

# int subkey1[8],subkey2[8];

# key\_generation(key\_num,subkey1,subkey2);

# int bin[10];

# binary(leter\_num,bin);

# int inner\_permutate\_arrange[8];

# inner\_permutate(bin,inner\_permutate\_arrange);

# int left\_inner\_permutate[4],right\_inner\_permutate[4];

# seperate(inner\_permutate\_arrange,left\_inner\_permutate,right\_inner\_permutate);

# int fp[8]; FP\_arrange(right\_inner\_permutate,fp);

# int left\_zor[4],right\_zor[4];

# zor(fp,subkey1,left\_zor,right\_zor);

# int s0[2],s1[2];

# S\_box(left\_zor,right\_zor,s0,s1);

# int p4[4];

# P4\_arrange(s0,s1,p4);

# int sw\_right\_inner\_permutate[4];

# P4\_zor(left\_inner\_permutate,p4,sw\_right\_inner\_permutate);

# int sw\_fp[8];

# FP\_arrange(sw\_right\_inner\_permutate,sw\_fp);

# int sw\_left\_zor[4],sw\_right\_zor[4];

# zor(sw\_fp,subkey2,sw\_left\_zor,sw\_right\_zor);

# int sw\_s0[2],sw\_s1[2];

# S\_box(sw\_left\_zor,sw\_right\_zor,sw\_s0,sw\_s1);

# int sw\_p4[4];

# P4\_arrange(sw\_s0,sw\_s1,sw\_p4);

# int sw\_zor[4];

# P4\_zor(right\_inner\_permutate,sw\_p4,sw\_zor);

# int inner\_permutate\_inv[8];

# inner\_permutate\_invers(sw\_zor,sw\_right\_inner\_permutate,inner\_permutate\_inv);

# int sum;

# sum=(inner\_permutate\_inv[0]\*128)+(inner\_permutate\_inv[1]\*64)+(inner\_permutate\_inv[2]\*32)+(inner\_permutate\_inv[3]\*16)+(inner\_permutate\_inv[4]\*8)+(inner\_permutate\_inv[5]\*4)+(inner\_permutate\_inv[6]\*2)+(inner\_permutate\_inv[7]\*1);

# return(sum);

# }

# int D(int leter\_num,int key\_num )

# {

# int subkey1[8],subkey2[8];

# key\_generation(key\_num,subkey1,subkey2);

# int bin[10];

# binary(leter\_num,bin);

# int inner\_permutate\_arrange[8];

# inner\_permutate(bin,inner\_permutate\_arrange);

# int left\_inner\_permutate[4],right\_inner\_permutate[4];

# seperate(inner\_permutate\_arrange,left\_inner\_permutate,right\_inner\_permutate);

# int fp[8];

# FP\_arrange(right\_inner\_permutate,fp);

# int left\_zor[4],right\_zor[4];

# zor(fp,subkey2,left\_zor,right\_zor);

# int s0[2],s1[2];

# S\_box(left\_zor,right\_zor,s0,s1);

# int p4[4];

# P4\_arrange(s0,s1,p4);

# int sw\_right\_inner\_permutate[4];

# P4\_zor(left\_inner\_permutate,p4,sw\_right\_inner\_permutate);

# int sw\_fp[8];

# FP\_arrange(sw\_right\_inner\_permutate,sw\_fp);

# int sw\_left\_zor[4],sw\_right\_zor[4];

# zor(sw\_fp,subkey1,sw\_left\_zor,sw\_right\_zor);

# int sw\_s0[2],sw\_s1[2];

# S\_box(sw\_left\_zor,sw\_right\_zor,sw\_s0,sw\_s1);

# int sw\_p4[4];

# P4\_arrange(sw\_s0,sw\_s1,sw\_p4);

# int sw\_zor[4];

# P4\_zor(right\_inner\_permutate,sw\_p4,sw\_zor);

# int inner\_permutate\_inv[8];

# inner\_permutate\_invers(sw\_zor,sw\_right\_inner\_permutate,inner\_permutate\_inv);

# int sum;

# sum=(inner\_permutate\_inv[0]\*128)+(inner\_permutate\_inv[1]\*64)+(inner\_permutate\_inv[2]\*32)+(inner\_permutate\_inv[3]\*16)+(inner\_permutate\_inv[4]\*8)+(inner\_permutate\_inv[5]\*4)+(inner\_permutate\_inv[6]\*2)+(inner\_permutate\_inv[7]\*1);

# return(sum);

# }

# void get\_file\_data(char message\_file[],char key\_file[],char user\_message[],char user\_key[])

# {

# fstream myline(message\_file,ios::out | ios::app);

# myline.close();

# int loop=0;

# fstream gline(message\_file,ios::in);

# if(!gline){

# cout<<"There is some error so that's way file can not be open .\n\n";

# }

# else

# {

# while(!gline.eof())

# {

# user\_message[loop]=gline.get();

# loop++;

# }

# gline.close();

# loop=0;

# }

# fstream key(key\_file,ios::in);

# if(!key){

# cout<<"There is some error so that's way file can not be open .\n\n";

# }

# else

# {

# while(!key.eof())

# {

# user\_key[loop]=key.get();

# loop++;

# }

# key.close();

# }

# }

# int edchoice()

# {

# char cchoice;

# int ichoice;

# cout<<"(1) Encrypt my message.\n\n";

# cout<<"(2) Decrypt my message.\n\n";

# cout<<"\n\nEnter your choice :";

# cin>>cchoice;

# ichoice=cchoice;

# return(ichoice);

# }

# void key\_store(char user\_key[])

# {

# char key\_file[500];

# int loop=0;

# cout<<"Type your file name in which your key will be stored:";

# cin.getline(key\_file,500);

# fstream myfile(key\_file,ios::out | ios:: app);

# while(user\_key[loop]!=-52)

# {

# myfile<<user\_key[loop];

# loop++;

# }

# myfile.close();

# }

# void Encryption\_and\_Decryption(char user\_message[],char user\_key[],char ED\_choice)

# {

# char output\_file[500];

# cout<<"Type your file name in which your output data will be stored :";

# cin.getline(output\_file,500);

# int line;

# int key;

# char output[99999];

# int loop=0;

# char ch;

# if(ED\_choice==49){

# cout<<" Encrypted message \n\n\n"; while((user\_message[loop]!='\0')&&(user\_key[loop]!='\0')&&(user\_message[loop]!=- 52)&&(user\_key[loop]!=-52))

# {

# line=user\_message[loop

# ];

# key=user\_key[loop];

# ch=E(line,key);

# output[loop]=ch;

# fstream save\_file(output\_file,ios::out | ios::app);

# save\_file<<ch;

# save\_file.close();

# cout<<ch;

# loop++;

# }

# }

# loop=0;

# if(ED\_choice==50)

# {

# cout<<" Decrypted message \n\n\n";

# while((user\_message[loop]!='\0')&&(user\_key[loop]!='\0')&&(user\_message[loop]!=- 52)&&(user\_key[loop]!=-52))

# {

# line=user\_message[loop];

# key=user\_key[loop];

# if(line<0)

# line=line+256;

# ch=D(line,key);

# fstream key(output\_file,ios::out | ios::app);

# key<<ch;

# key.close();

# cout<<ch;

# loop++;

# }

# }

# }

# void message\_and\_key(int ichoice)

# {

# char user\_message[99999];

# char user\_key[99999];

# char message\_file[500];

# char key\_file[500];

# int ED\_choice;

# ED\_choice=edchoice();

# if(ichoice==49)

# {

# cout<<"Enter your message file name :";

# cin.sync();

# cin.getline(message\_file,500);

# cout<<"\n\n";

# cout<<"Enter your key file name :";

# cin.getline(key\_file,500);

# get\_file\_data(message\_file,key\_file,user\_message,user\_key);

# Encryption\_and\_Decryption(user\_message,user\_key,ED\_choice);

# }

# else if(ichoice==50)

# {

# cout<<" Type your message\n\n";

# cin.sync();

# cin.getline(user\_message,99999);

# cout<<" Type your key\n\n";

# cin.getline(user\_key,99999);

# key\_store(user\_key);

# Encryption\_and\_Decryption(user\_message,user\_key,ED\_choice);

# }

# }

# void file\_choice()

# {

# char cchoice;

# int ichoice;

# cout<<"(1) Include my files for taking message and key.\n\n";

# cout<<"(2) Don't include my files, I will type my message and key.\n\n";

# cout<<"\n\nEnter your choice:";

# cin>>cchoice;

# ichoice=cchoice;

# message\_and\_key(ichoice);

# }

# int main()

# {

# char cchoice;

# int ichoice;

# cchoice='1';

# ichoice=cchoice;

# if(ichoice==49)

# {

# file\_choice();

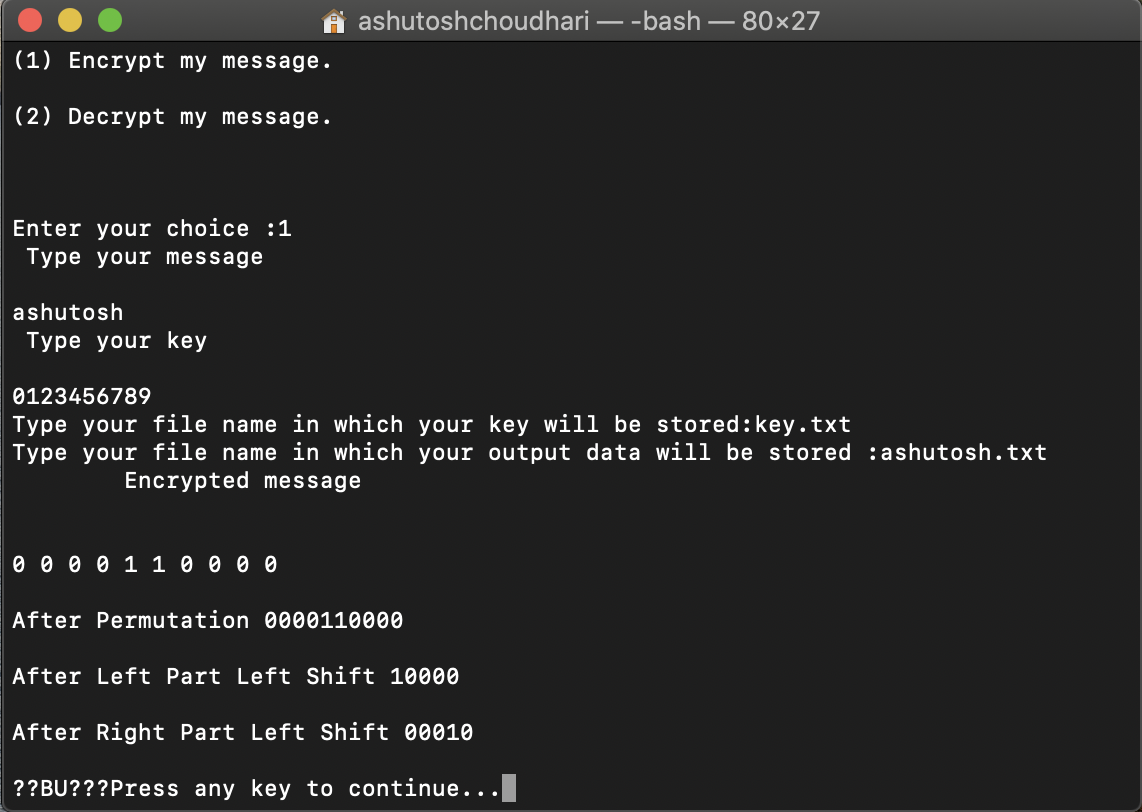
# }

# return 0;

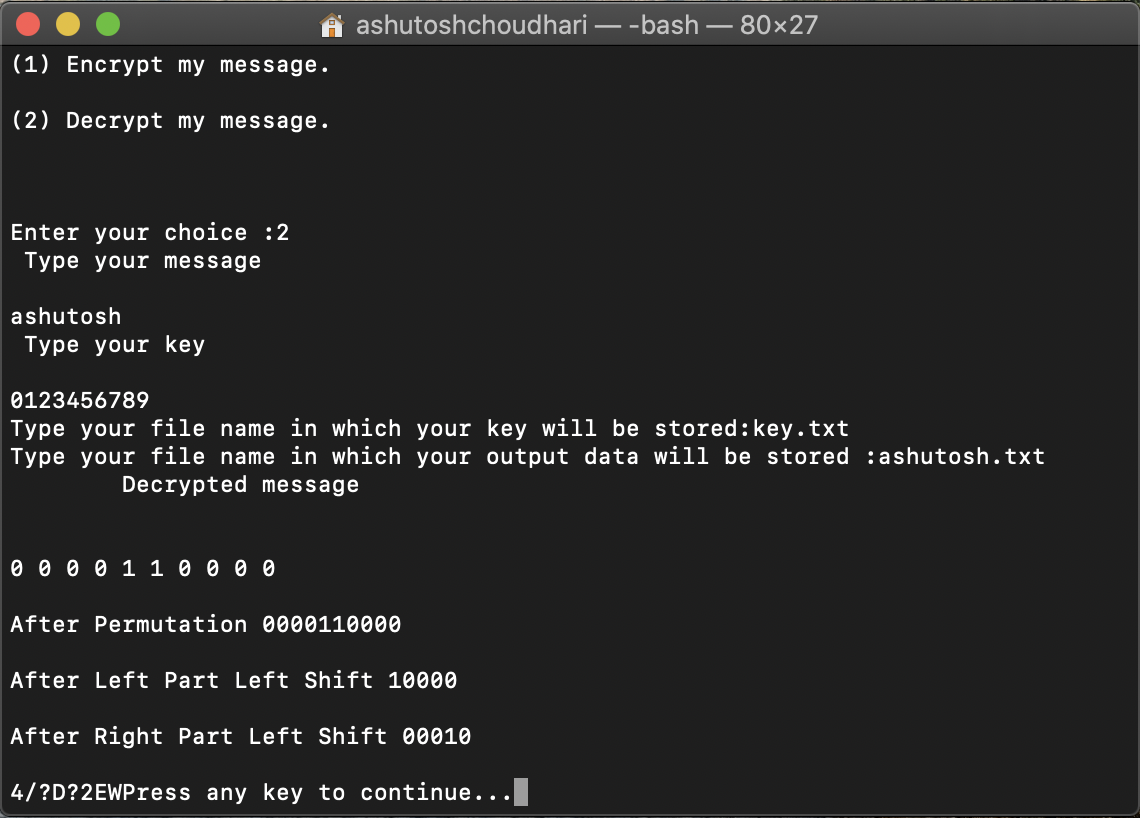
# }

# OUTPUT:

1. Encryption Result:



### Decryption Result:



**REFRENCES:**

[1] C. Patterson, "High performance DES encryption in Virtex/sup TM/ FPGAs using JBits/sup TM/," *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*, Napa Valley, CA, USA, 2000, pp. 113-121, doi: 10.1109/FPGA.2000.903398.

[2] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," in IBM Journal of Research and Development, vol. 38, no. 3, pp. 243-250, May 1994, doi: 10.1147/rd.383.0243.

[3] Dworak K., Boryczka U. (2015) Cryptanalysis of SDES Using Modified Version of Binary Particle Swarm Optimization. In: Núñez M., Nguyen N., Camacho D., Trawiński B. (eds) Computational Collective Intelligence. Lecture Notes in Computer Science, vol 9330. Springer, Cham

[4] PRERNA MAHAJAN, ABHISHEK SACHDEVA, Dr.. A Study of Encryption Algorithms AES, DES and RSA for Security. Global Journal of Computer Science and Technology, [S.l.], dec. 2013. ISSN 0975-4172.

[5] Poonam Garg (2009) .Cryptanalysis of SDES via evolutionary computation techniques International Journal of Computer Science and Information Security [arXiv:0906.5123v1](https://arxiv.org/abs/0906.5123v1)

[6] Dworak K., Nalepa J., Boryczka U., Kawulok M. (2016) Cryptanalysis of SDES Using Genetic and Memetic Algorithms. In: Król D., Madeyski L., Nguyen N. (eds) Recent Developments in Intelligent Information and Database Systems. Studies in Computational Intelligence, vol 642. Springer, Cham