

DES

Eron Romero Argumedo  
Erwin Hernandez Garcia  
3CV1

September 27, 2016

# Contents

<b>1</b>	<b>Theory</b>	<b>1</b>
1.1	DES . . . . .	1
1.2	Horst Feistel . . . . .	2
1.3	Feistel Network . . . . .	2
1.4	Permutation . . . . .	3
1.5	S-box . . . . .	5
1.6	Weak and Semi-weak keys . . . . .	7
1.7	3DES . . . . .	7
<b>2</b>	<b>Functions</b>	<b>8</b>
<b>3</b>	<b>Screenshots</b>	<b>13</b>
3.1	Usage . . . . .	13
3.2	Results . . . . .	14
<b>4</b>	<b>Signature</b>	<b>16</b>

## List of Figures

1.1	Round for DES . . . . .	2
1.2	Round for a Feistel Network . . . . .	3
1.3	Initial Permutation . . . . .	4
1.4	Inverse of the initial permutation . . . . .	4
1.5	Permutation P . . . . .	5
3.1	Usage of the program . . . . .	13
3.2	Original plaintext . . . . .	14
3.3	Ciphertext obtained after encrypt with Triple DES . . . . .	15
3.4	Plaintext obtained after decrypt with Triple DES . . . . .	16
4.1	Eron Romero Argumedo . . . . .	17
4.2	Erwin Hernández García . . . . .	18

# 1 Theory

## 1.1 DES

Originally Lucifer was intended to protect the data from a bank. Later the National Bureau of Standards make a petition to create a crypto algorithm to be made a new standard. On 1974 Lucifer was accepted and after work with the NSA, DES was created. Lucifer got modifications to its internal functions and a reduction on the key size from 112 bits to 56 bits.

There were rumours about weakness that NSA had built into the algorithm, but to this day, no evidence has been found. The first DES break was reported in 1997 and it took about three months. Currently the record for breaking DES is 10 hours.

DES is essentially a Feistel network with 16 rounds operating on blocks of 64bits and using a key of 56 bits. DES varies slightly from a Feistel network in the initial permutation that is applied before the first round starts and whose inverse is applied after the last round. It executes the following steps:

1. The right side of the block is expanded from 32bits to 48bits by doubling 16bits and permuting the block.
2. The exclusive OR of that value with the round key is computed.
3. The block of 48bits is split into eight blocks with six bit each. Each of them is the input to one of the S-boxes, Each S-box yields a 4bit output.
4. The permutation P is applied to the 32bits block yielding the round output.[1]

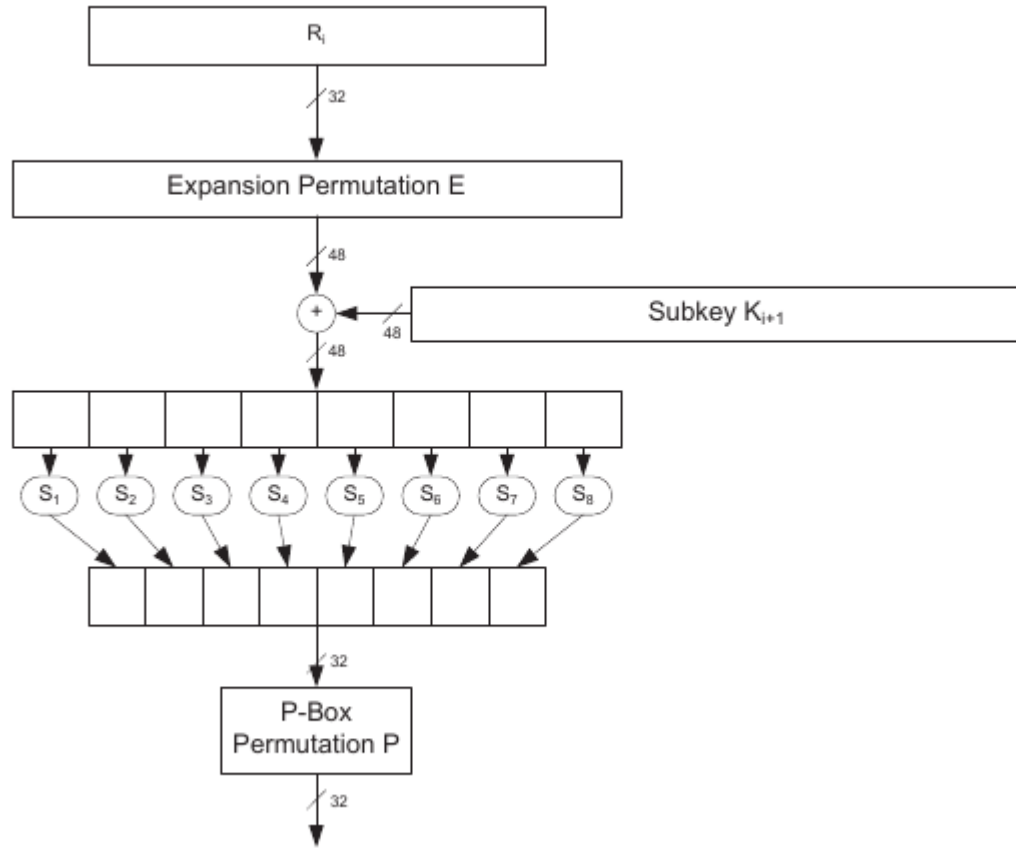


Figure 1.1: Round for DES  
[1]

## 1.2 Horst Feistel

Horst Feistel was a German cryptographer that worked designed ciphers at IBM, there he started the investigation that finished with the creation of DES. He was one of the first non-governmental cryptographers that studied the design and theory of block ciphers.

While working at IBM, he led the design of a cipher called “Lucifer” and DES.[2]

## 1.3 Feistel Network

A Feistel network is a cipher scheme created by Horst Feistel during his work on the cipher Lucifer for IBM. It defines a function  $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ . It depends on the number of rounds  $d \in \mathbb{N}$  and the round functions  $f_i, \dots, f_d : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ .

The  $F$  functions is defined as follows: It works on  $d$  rounds, the  $i$ -th round does:

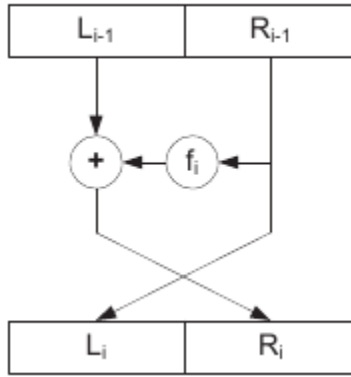


Figure 1.2: Round for a Feistel Network

1. The input is the previous round output.
2. Split by two the input:  $L_{i-1}, R_{i-1}$ .
3. Applies  $f_i$  to the right part yielding  $f_i(R_{i-1})$ .
4. Calculate  $L_{i-1} \oplus f_i(R_{i-1})$ , the result will be the right side of the output.
5. The left side of the output is the right side of the input.

[1]

## 1.4 Permutation

Listing 1: DES'Permutations

```

5  int initial_permutation[] =    {58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17,  9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7};

10 int right_sub_message_permutation[] =    {16,  7, 20, 21,
    29, 12, 28, 17,
    1, 15, 23, 26,
    5, 18, 31, 10,
    2,  8, 24, 14,
15  32, 27,  3,  9,
    19, 13, 30,  6,
    22, 11,  4, 25};

```

```

20  int final_permutation[] = {40,  8, 48, 16, 56, 24, 64, 32,
                             39,  7, 47, 15, 55, 23, 63, 31,
                             38,  6, 46, 14, 54, 22, 62, 30,
                             37,  5, 45, 13, 53, 21, 61, 29,
                             36,  4, 44, 12, 52, 20, 60, 28,
25  35,  3, 43, 11, 51, 19, 59, 27,
    34,  2, 42, 10, 50, 18, 58, 26,
    33,  1, 41,  9, 49, 17, 57, 25};

```

The  $i$ -th bit in the permutation represents the position from where the  $i$ -th bit in the message is taken. That is, for the first bit we shall take the 58th bit, for the second we shall take the 50th bit and so on.

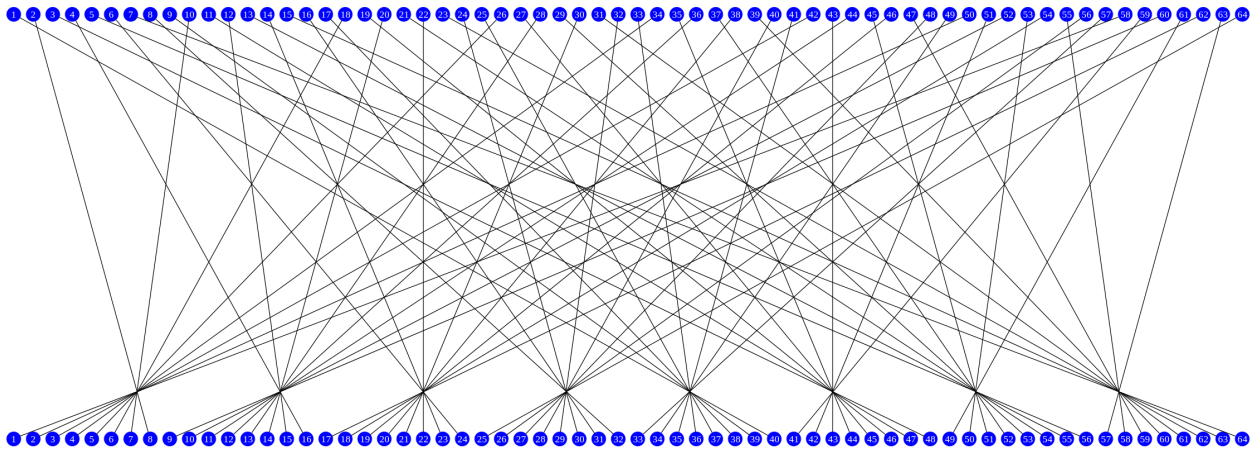


Figure 1.3: Initial Permutation

[3]

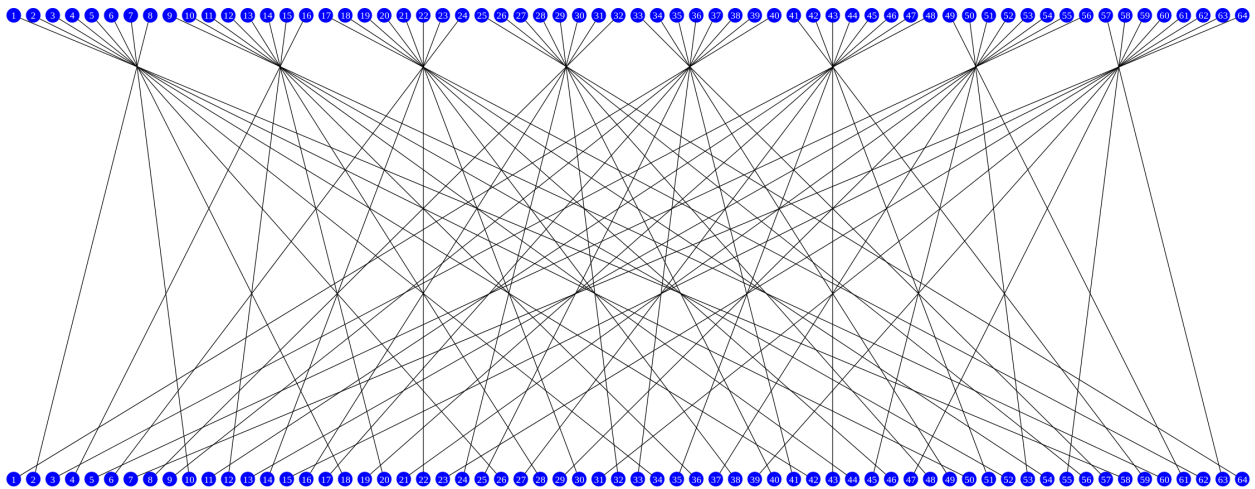


Figure 1.4: Inverse of the initial permutation

[4]

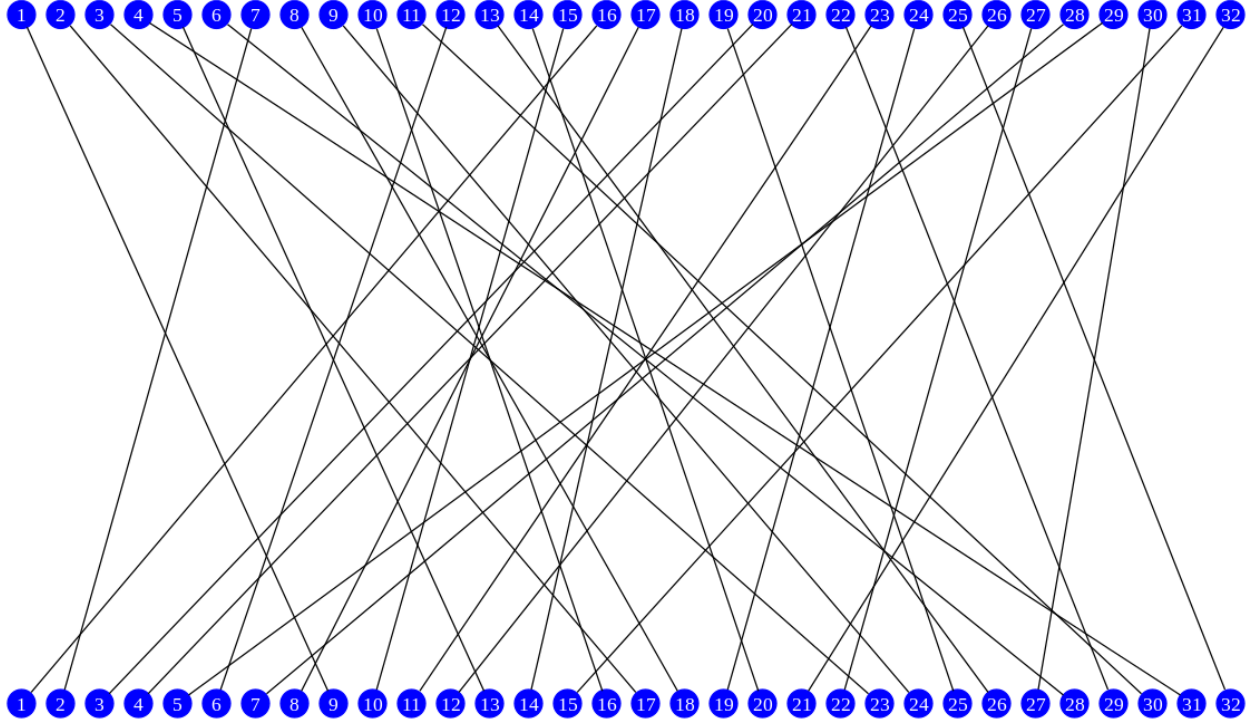


Figure 1.5: Permutation P  
[5]

## 1.5 S-box

The S-box is a table with 4 rows and 16 columns. Each S-box replaces a 6-bit input with a 4-bit output. Given a 6-bit input, the 4-bit output is found by selecting the row using the outer two bits, and the column using the inner four bits. For example, the input "011011" has outer bits "01" (second row) and inner bits "1101" (14th column). The corresponding output for S-box S5 would be "1001", the value in the second row, 14th column.

Listing 2: DES' S-boxes

```

int S1[] = {14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
            0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
            4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
            15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13};
5 int S2[] = {15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
            3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
            0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
            13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9};
int S3[] = {10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
10 13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
            13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
            1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12};
int S4[] = { 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
15 13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
            10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
            3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14};
int S5[] = { 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
            14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
            4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
20 11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3};
int S6[] = {12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
            10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
            9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
            4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13};
25 int S7[] = { 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
            13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
            1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
            6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12};
int S8[] = {13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
30 1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
            7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
            2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11};

```



## 1.6 Weak and Semi-weak keys

There are certain keys that makes DES encrypt operation behave as decryption.

- Alternating zeros and ones
- Alternating F and E
- 0xE0E0E0E0F1F1F1F1
- 0x1F1F1F1F0E0E0E0E

If the implementation doesn't account for parity bits, then

- 0x0000000000000000
- 0xFFFFFFFFFFFFFFFF
- 0xE1E1E1E1F0F0F0F0
- 0x1E1E1E1E0F0F0F0F

When using weak keys, the generated subkeys are identical, which, as DES being a Feistel network, it makes the odd rounds encrypt while the even decrypt.

The semi-weak keys will only generate two different subkeys. This keys exist in pairs, with the characteristic that:

$$E_{K_1}(E_{K_2}(M)) = M$$

- 0x011F011F010E010E y 0x1F011F010E010E01
- 0x01E001E001F101F1 y 0xE001E001F101F101
- 0x01FE01FE01FE01FE y 0xFE01FE01FE01FE01
- 0x1FE01FE00EF10EF1 y 0xE01FE01FF10EF10E
- 0x1FFE1FFE0EFE0EFE y 0xFE1FFE1FFE0EFE0E
- 0xE0FEE0FEF1FEF1FE y 0xFEE0FEE0FEF1FEF1

## 1.7 3DES

Triple DES consists in applying the following to encrypt:

$$E_{3DES}(K_1, K_2, K_3, m) = E_{DES}(K_3, D_{DES}(K_2, E_{DES}(K_1, m)))$$

and to decrypt:

$$D_{3DES}(K_1, K_2, K_3, m) = D_{DES}(K_1, E_{DES}(K_2, D_{DES}(K_3, m)))$$

[1]

## 2 Functions

Listing 3: Key generation

```
int generate_TripleDES_key(char *key)
{
    key_file = fopen(key, "wb");
    if (!key_file) {
5       printf("Could not open file to write key.");
        return 1;
    }

    unsigned int iseed = (unsigned int)time(NULL);
10    srand (iseed);

    short int bytes_written;
    unsigned char* des_key = (unsigned char*) malloc(TRIPLE_DES_KEY_SIZE*sizeof(char));
    generate_key(des_key);
    bytes_written = fwrite(des_key, 1, TRIPLE_DES_KEY_SIZE, key_file);
15    if (bytes_written != TRIPLE_DES_KEY_SIZE) {
        printf("Error writing key to output file.");
        fclose(key_file);
        free(des_key);
20    return 1;
    }
    free(des_key);
    fclose(key_file);
    return 0;
25 }
```

The key generation creates three keys that will be used by Triple DES, then they are saved in a file with a name specified by the user.

Listing 4: Encrypt/Decrypt

```

int encryptDecrypt(char *processMode, char keyNumber, char *key, char *input, char *output)
{
    unsigned long file_size;
    unsigned short int padding;
5 // Read key file
    key_file = fopen(key, "rb");
    if (!key_file) {
        printf("Could not open key file to read key.");
        return 1;
10 }

    short int bytes_read;
    unsigned char* des_key = (unsigned char*) malloc(8*sizeof(char));

15 if ( keyNumber==1 )
    {
        /* Seek to the beginning of the file */
        fseek(key_file, SEEK_SET, 0);
        bytes_read = fread(des_key, sizeof(unsigned char), DES_KEY_SIZE, key_file);
20 }
    else if ( keyNumber==2 )
    {
        /* Seek to the position 8 of the file */
        fseek(key_file, SEEK_SET, 8);
25 bytes_read = fread(des_key, sizeof(unsigned char), DES_KEY_SIZE, key_file);
    }
    else if ( keyNumber==3 )
    {
        /* Seek to the position 16 of the file */
30 fseek(key_file, SEEK_SET, 16);
        bytes_read = fread(des_key, sizeof(unsigned char), DES_KEY_SIZE, key_file);
    }
}

```

```

35     if (bytes_read != DES_KEY_SIZE) {
        printf("Key read from key file does nto have valid key size.");
        fclose(key_file);
        return 1;
    }
    fclose(key_file);

40

    // Open input file
    input_file = fopen(input, "rb");
    if (!input_file) {
45        printf("Could not open input file to read data.");
        return 1;
    }

    // Open output file
50    output_file = fopen(output, "wb");
    if (!output_file) {
        printf("Could not open output file to write data.");
        return 1;
    }

55

    // Generate DES key set
    short int bytes_written, process_mode;
    unsigned long block_count = 0, number_of_blocks;
    unsigned char* data_block = (unsigned char*) malloc(8*sizeof(char));
60    unsigned char* processed_block = (unsigned char*) malloc(8*sizeof(char));
    key_set* key_sets = (key_set*)malloc(17*sizeof(key_set));

    generate_sub_keys(des_key, key_sets);

65    // Determine process mode
    if (strcmp(processMode, ACTION_ENCRYPT) == 0) {
        process_mode = ENCRYPTION_MODE;
    }

```

```

    printf("Encrypting..\n");
} else {
70    process_mode = DECRYPTION_MODE;
    printf("Decrypting..\n");
}

// Get number of blocks in the file
75 fseek(input_file, 0L, SEEK_END);
file_size = ftell(input_file);

fseek(input_file, 0L, SEEK_SET);
number_of_blocks = file_size/8 + ((file_size%8)?1:0);

80 // Start reading input file, process and write to output file
while(fread(data_block, 1, 8, input_file)) {
    block_count++;
    if (block_count == number_of_blocks) {
85         if (process_mode == ENCRYPTION_MODE) {
            padding = 8 - file_size%8;
            if (padding < 8) { // Fill empty data block bytes with padding
                memset((data_block + 8 - padding), (unsigned char)padding, padding);
            }

90             process_message(data_block, processed_block, key_sets, process_mode);
            bytes_written = fwrite(processed_block, 1, 8, output_file);

            if (padding == 8) { // Write an extra block for padding
95                 memset(data_block, (unsigned char)padding, 8);
                process_message(data_block, processed_block, key_sets, process_mode);
                bytes_written = fwrite(processed_block, 1, 8, output_file);
            }
        } else {
100            process_message(data_block, processed_block, key_sets, process_mode);
            padding = processed_block[7];

```

```

    if (padding < 8) {
105         bytes_written = fwrite(processed_block, 1, 8 - padding, output_file);
    }
} else {
    process_message(data_block, processed_block, key_sets, process_mode);
    bytes_written = fwrite(processed_block, 1, 8, output_file);
110 }
    memset(data_block, 0, 8);
}

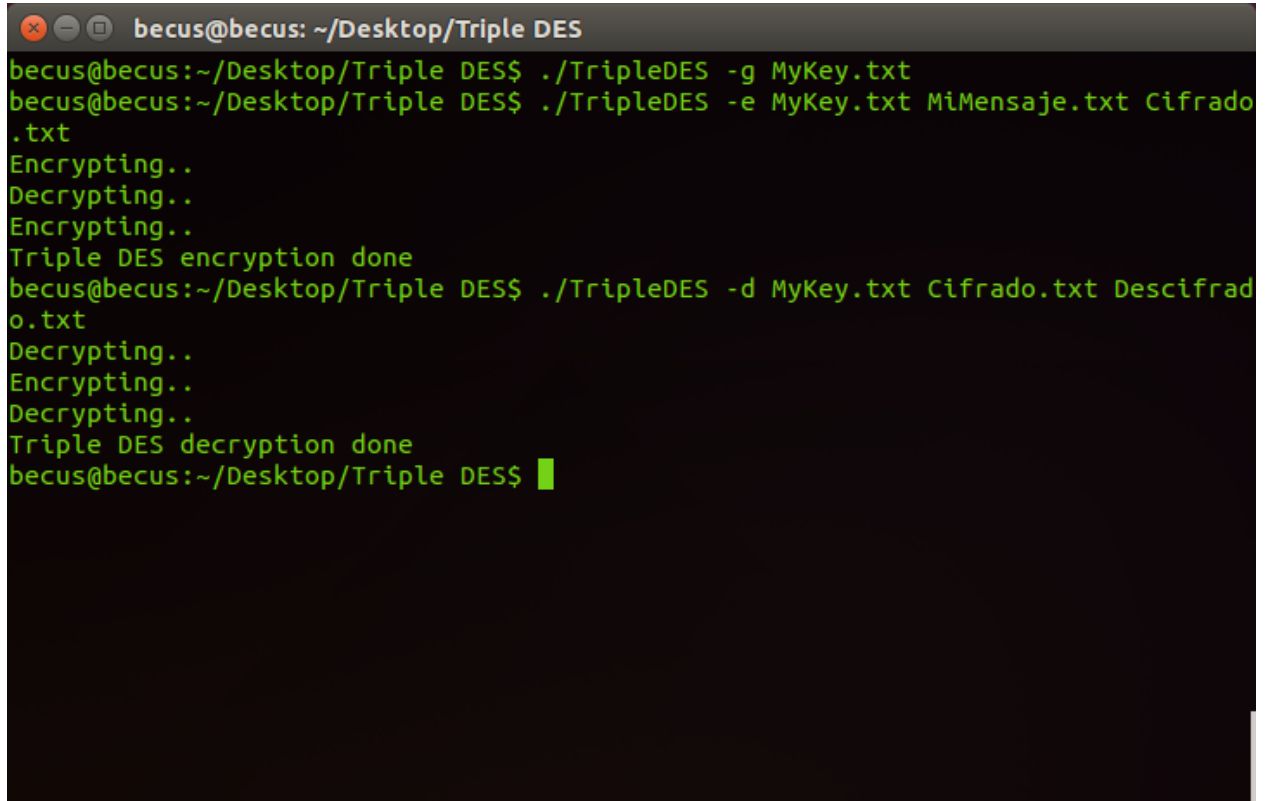
// Free up memory
115 free(des_key);
    free(data_block);
    free(processed_block);
    fclose(input_file);
    fclose(output_file);
120 return 0;
}

```

The function to encrypt/decrypt does the action specified by the user, the user must give the name of the file containing the key, the name of the file containing the cipher text or the name of the file containing the plaint text and the name of the output file.

## 3 Screenshots

### 3.1 Usage

A terminal window titled 'becus@becus: ~/Desktop/Triple DES' showing the execution of the Triple DES program. The user runs './TripleDES -g MyKey.txt' to generate a key. Then, they run './TripleDES -e MyKey.txt MiMensaje.txt Cifrado.txt' to encrypt 'MiMensaje.txt' into 'Cifrado.txt'. The output shows 'Encrypting..' and 'Decrypting..' messages, followed by 'Triple DES encryption done'. Finally, they run './TripleDES -d MyKey.txt Cifrado.txt Descifrado.txt' to decrypt 'Cifrado.txt' into 'Descifrado.txt'. The output shows 'Decrypting..' and 'Encrypting..' messages, followed by 'Triple DES decryption done'. The terminal ends with the prompt 'becus@becus:~/Desktop/Triple DES\$' and a cursor.

```
becus@becus: ~/Desktop/Triple DES
becus@becus:~/Desktop/Triple DES$ ./TripleDES -g MyKey.txt
becus@becus:~/Desktop/Triple DES$ ./TripleDES -e MyKey.txt MiMensaje.txt Cifrado
.txt
Encrypting..
Decrypting..
Encrypting..
Triple DES encryption done
becus@becus:~/Desktop/Triple DES$ ./TripleDES -d MyKey.txt Cifrado.txt Descifrad
o.txt
Decrypting..
Encrypting..
Decrypting..
Triple DES decryption done
becus@becus:~/Desktop/Triple DES$ █
```

Figure 3.1: Usage of the program

## 3.2 Results

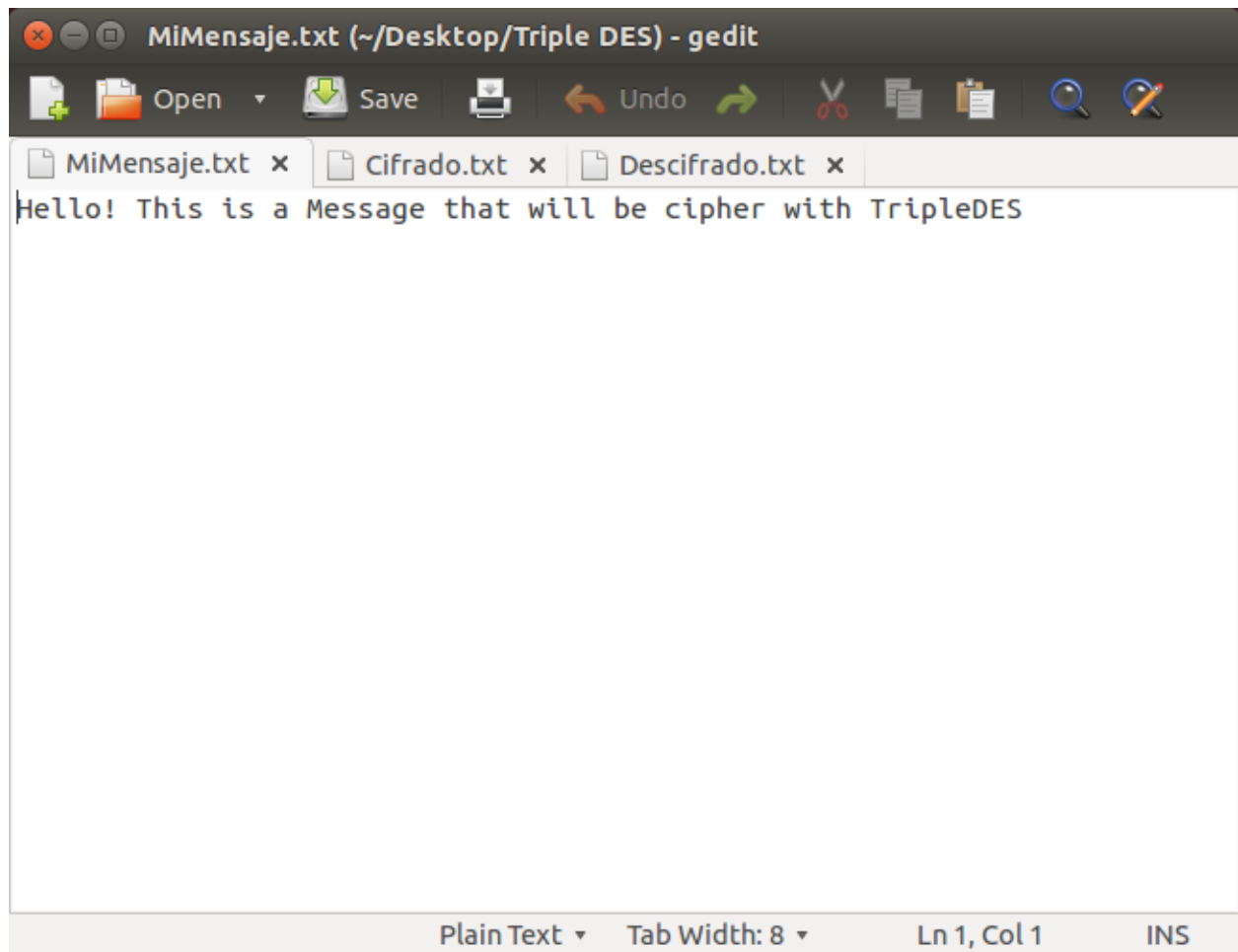


Figure 3.2: Original plaintext



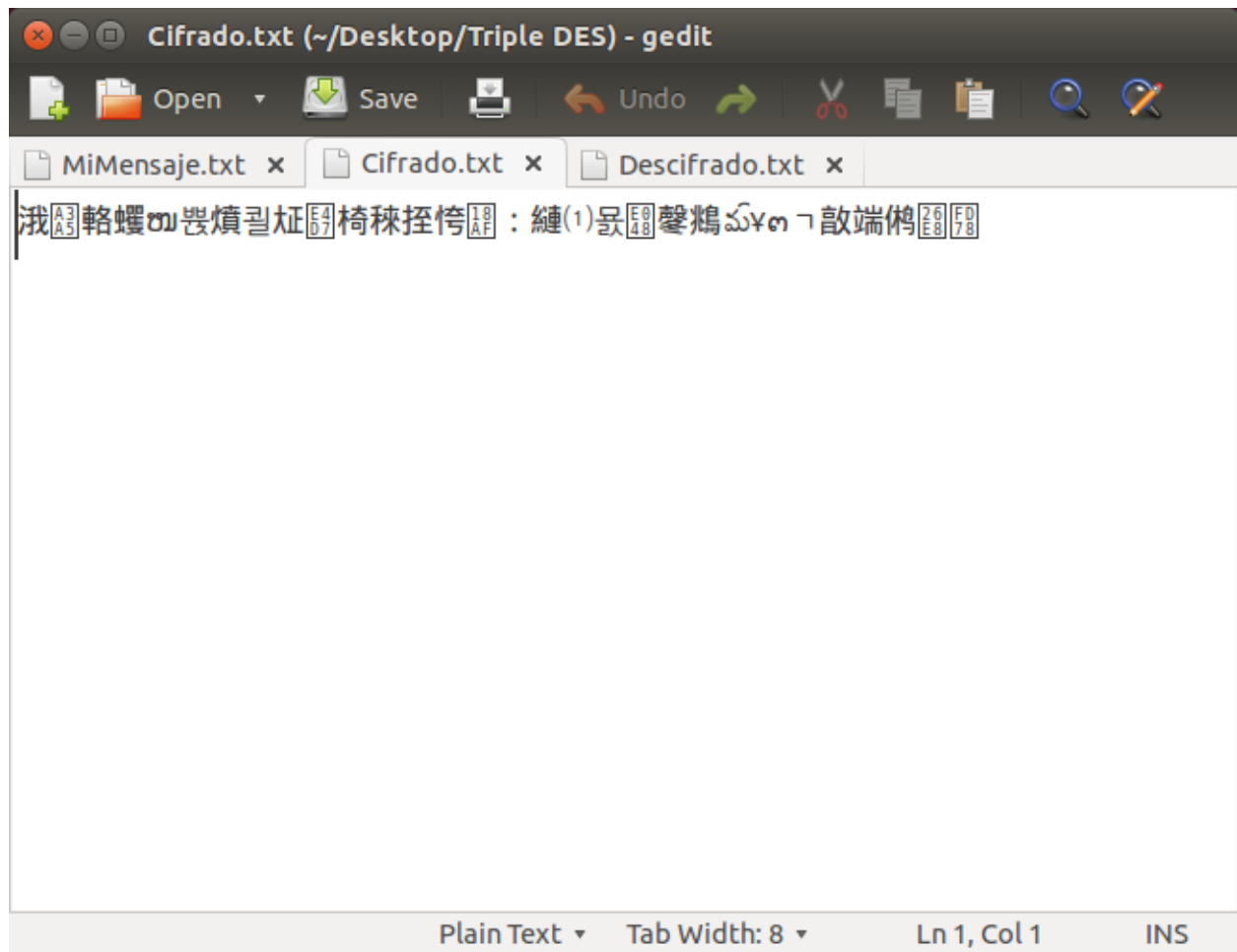


Figure 3.3: Ciphertext obtained after encrypt with Triple DES

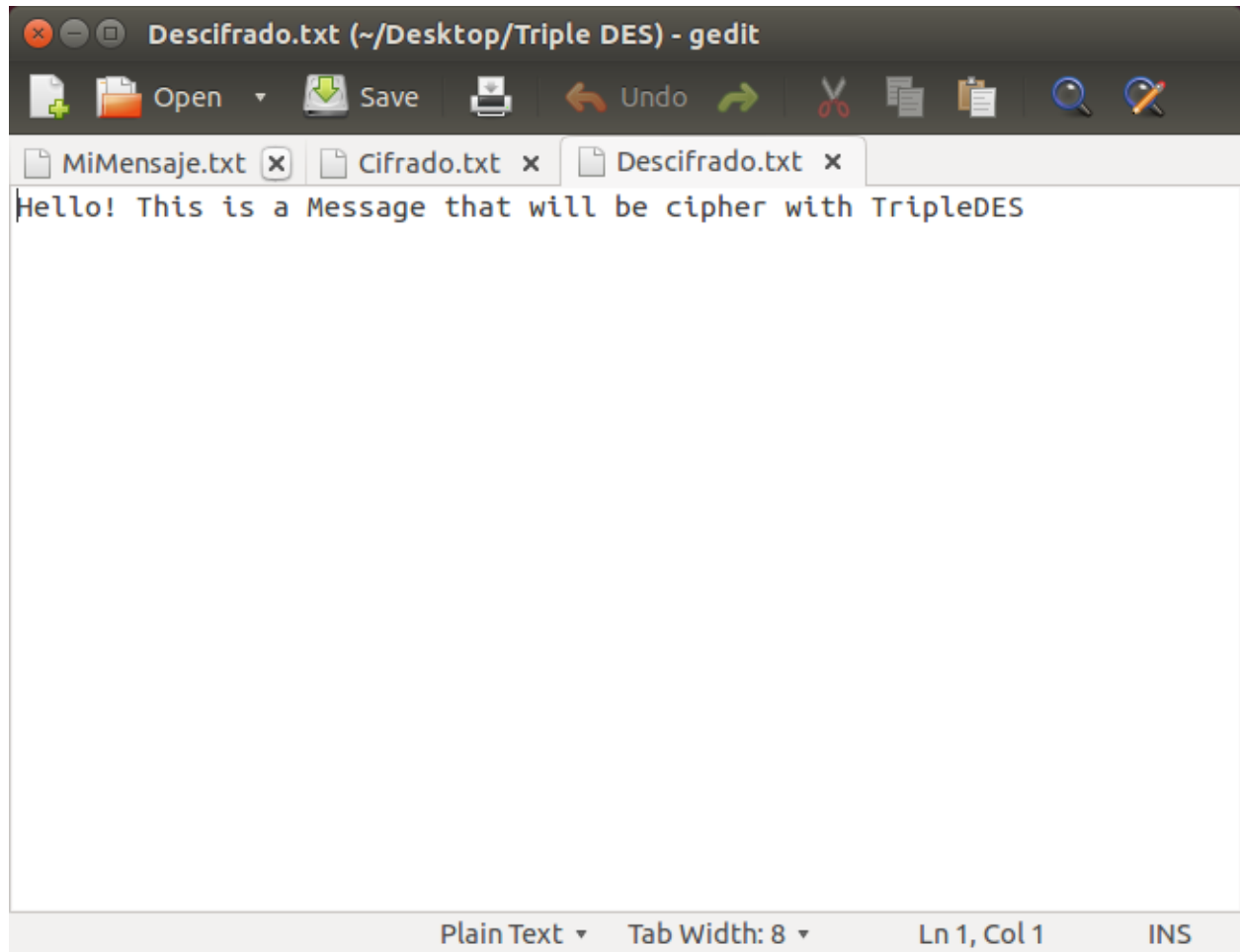


Figure 3.4: Plaintext obtained after decrypt with Triple DES

## 4 Signature

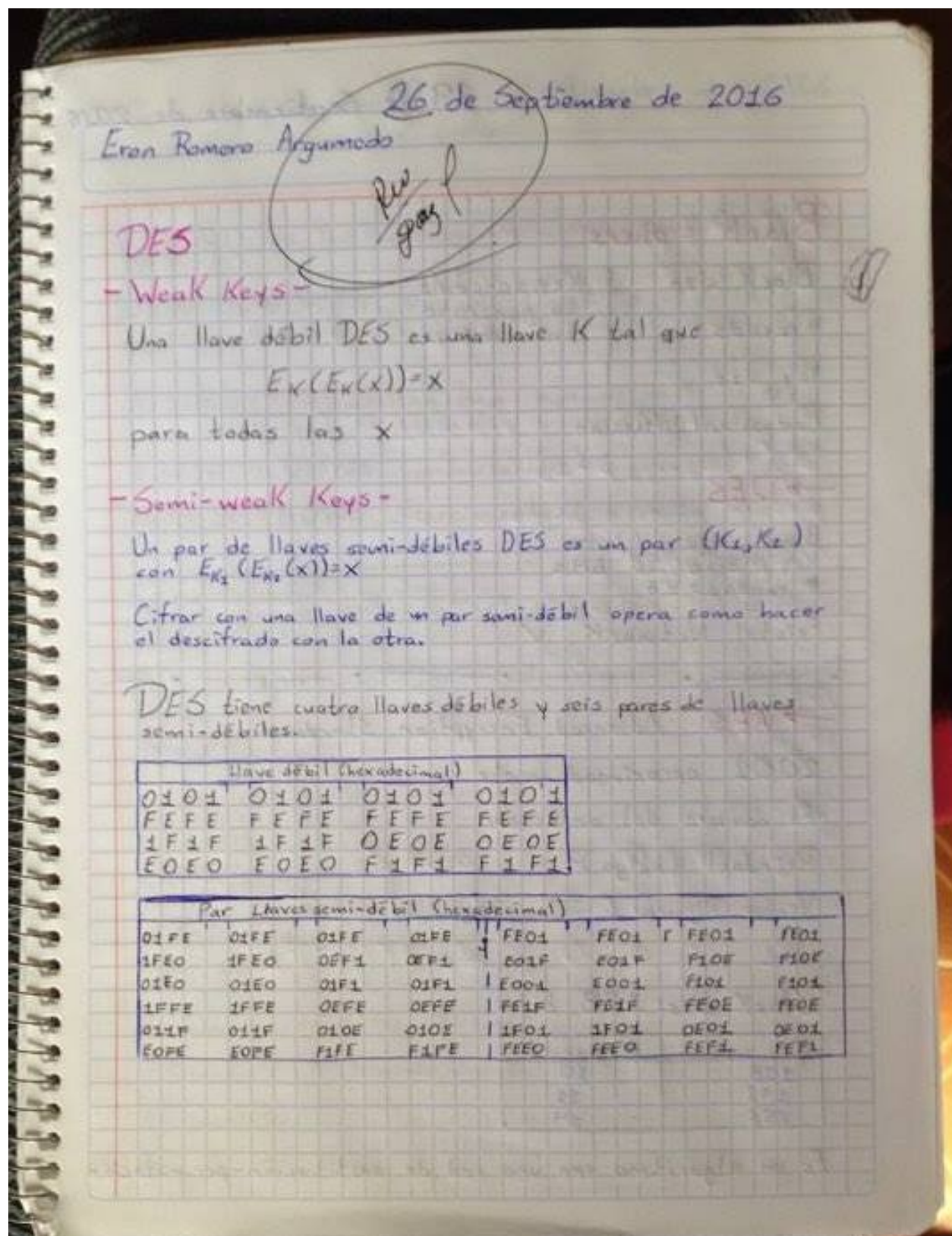


Figure 4.1: Eron Romero Argumado



Erwin Hernandez Garcia

Rev 1.27, 2016  
Sept 27, 2016  
giant!

```
#####
# Documentation #
#####
# Author: Todd Whiteman
# Date: 16th March, 2009
# Version: 2.0.0
# License: Public Domain - free to do as you wish
# Homepage: http://twhiteman.netfirms.com/des.html
# Thanks to:
# * David Broadwell for ideas, comments and suggestions.
# * Mario Wolff for pointing out and debugging some triple des CBC errors.
# * Santiago Palladino for providing the PKCS5 padding technique.
# * Shaya for correcting the PAD_PKCS5 triple des CBC errors.
#
#####
# DES #
#####
class des(_baseDes):

    # Permutation and translation tables for DES
    __pc1 = [56, 48, 40, 32, 24, 16, 8,
              0, 57, 49, 41, 33, 25, 17,
              9, 1, 58, 50, 42, 34, 26,
              18, 10, 2, 59, 51, 43, 35,
              62, 54, 46, 38, 30, 22, 14,
              6, 61, 53, 45, 37, 29, 21,
              13, 5, 60, 52, 44, 36, 28,
              20, 12, 4, 27, 19, 11, 3
    ]

    # number left rotations of pcl
    __left_rotations = [
        1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
    ]

    # permuted choice key (table 2)
    __pc2 = [
        13, 16, 10, 23, 0, 4,
        2, 27, 14, 5, 20, 9,
        22, 18, 11, 3, 25, 7,
        15, 6, 26, 19, 12, 1,
        40, 51, 30, 36, 46, 54,
        29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52,
        45, 41, 49, 35, 28, 31
    ]

    # initial permutation IP
    __ip = [57, 49, 41, 33, 25, 17, 9, 1,
            59, 51, 43, 35, 27, 19, 11, 3,
            61, 53, 45, 37, 29, 21, 13, 5,
            63, 55, 47, 39, 31, 23, 15, 7,
            56, 48, 40, 32, 24, 16, 8, 0,
            58, 50, 42, 34, 26, 18, 10, 2,
            60, 52, 44, 36, 28, 20, 12, 4,
            62, 54, 46, 38, 30, 22, 14, 6
    ]
```

Figure 4.2: Erwin Hernández García

## References

- [1] M. Backes. Block ciphers. [Online]. Available: <http://web.cs.du.edu/~ramki/courses/security/2011Winter/notes/feistelProof.pdf>
- [2] I. H. Society. Mr. horst feistel. [Online]. Available: <http://www.ithistory.org/honor-roll/mr-horst-feistel>
- [3] SebDE. [Online]. Available: <https://en.wikipedia.org/wiki/File:DES-ip.svg>
- [4] ——. [Online]. Available: <https://en.wikipedia.org/wiki/File:DES-ip-1.svg>
- [5] ——. [Online]. Available: <https://en.wikipedia.org/wiki/File:DES-pp.svg>