



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



Cryptography

“IP and IP^{-1} ”

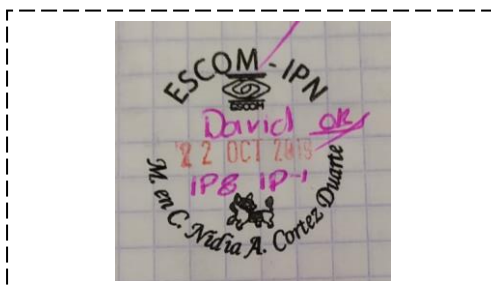
By:

Jose David Portilla Martinez

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

October 2019



Importance of the Initial Permutation

There is an initial permutation IP of the 64 bits of the data. This rearranges the bits according to the table shown in the following Tables section, where the entries in the table show the new arrangement of the bits from their initial order. The initial permutation is a key element in the DES Algorithm, as far as I can see, it gives an extra layer of arbitrariness so that the data is not passed just as it is.

But while investigation, I found that it does absolutely nothing to increase the security in the DES algorithm, in fact is just so the implementation is easier in most contexts, such in the hardware implementation, which the data there is received over a 8-bit bus; it can accumulate the bits into eight shift registers, which is more efficient (in terms of circuit area) than a single 64-bit register.

Tables

IP

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

Table 1: Initial Permutation

IP⁻¹

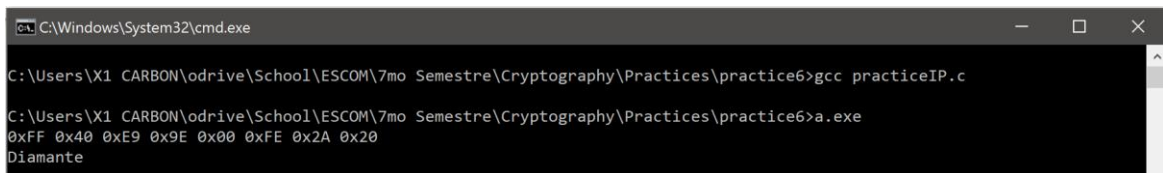
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 2: Inverse Initial Permutation

Results

In this test I used, as asked by the teacher, to use the word Diamantes, but as this word is made of 9 bytes and the data is packed in blocks of 8 bytes, I passed the word Diamante.

In the next screenshot, it can be seen in the first line (after the execution of the program) how the initial permutation works, in the next line it verifies, in some way, that the initial permutation was correctly done by execution the inverse initial permutation.



```
C:\Windows\System32\cmd.exe
C:\Users\X1 CARBON\odrive\School\ESCOM\7mo Semestre\Cryptography\Practices\practice6>gcc practiceIP.c
C:\Users\X1 CARBON\odrive\School\ESCOM\7mo Semestre\Cryptography\Practices\practice6>a.exe
0xFF 0x40 0xE9 0x9E 0x00 0xFE 0x2A 0x20
Diamante
```

Figure 1: Execution of the program

Code

I omitted the initialization of the the IP and IP^{-1} tables, everything else is exactly as it was compiled.

It's just a function and the main calls that function.

```
#include <stdio.h>
#define mod(a, b) (((a % b) + b) % b)
#define printh(a, b) printf("0x%2.2X%c", a, b)

void
makePermutation(unsigned char *m, unsigned char *r,
                unsigned char *IP, unsigned char exp)
{
    unsigned char i, j = 0;
    unsigned char row, column, count = 0;
    for (i = 0; i < 64; i++)
    {
        if (count == 8) {j++; count = 0;}
        row = IP[i] >> 3;
        column = mod(IP[i], 8);
        if (j == exp) row--;
        if ((m[row] & (128 >> mod(column - 1, 8))))
            r[j] |= (128 >> count);
        count++;
    }
}
```

```

int main()
{
    unsigned char i;
    unsigned char r[8] = {0}, ir[8] = {0};

    makePermutation("Diamante", r, arrIP, 3);

    for (i = 0; i < 8; i++)
        printh(r[i], ' ');

    printf("\n");
    makePermutation(r, ir, arrIP_1, 0);

    for (i = 0; i < 8; i++)
        printf("%c", ir[i]);
    printf("\n");
    return 0;
}

```