INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Cryptography

"Permutation"

Abstact

The P and P-1 functions of 32 bits.

By:

Alan Garduño Velazquez

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

November 2017

To validate this report it is necessary to include the corresponding seal

Index

Introduction:

In this document we are going to explain how the P and P-1 function works in the DES algorithm but in this case, we are going to use only 32 bits instead of 64 and the order of the bits is not the same it was invented during the class.

Literature review:

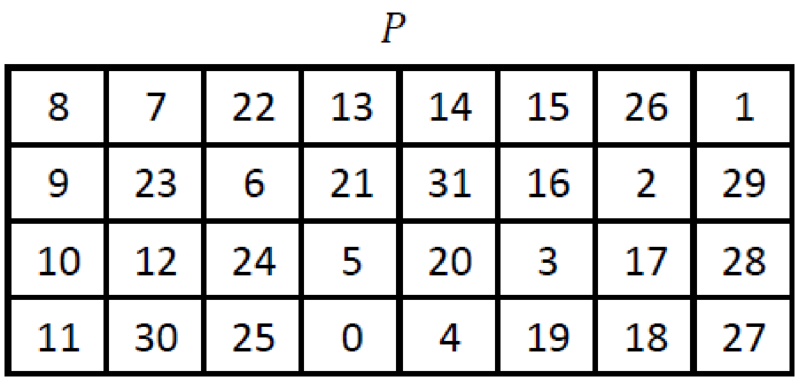
As we saw in class the P is a permutation, so it consists in mix the bits of the message in the process of decrypt and decrypt and P-1 consist in return P to the original positions.

We have the original message in bit representation in the Figure 1.1

7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8
23	22	21	20	19	18	17	16
31	30	29	28	27	26	25	24

Figure 1.1 Original bits.

When we apply the permutation, we have the positions of P in the Figure 1.2



P

8	7	22	13	14	15	26	1
9	23	6	21	31	16	2	29
10	12	24	5	20	3	17	28
11	30	25	0	4	19	18	27

Figure 1.2 P bits positions. (Bits permuted)

In P-1 we must return to the original representation so as we can see in the Figure 1.2 (P) **0** is in the position **28 so the first element of P-1 will be 28, we have 1 in the position 0, so the second value of P-1 will be 0 and the have value 2 in the position 9 of P, so the third value of P-1 will be 9, basically we must match every in position P to recover the original representation (see Figure 1.1 and figure 1.3).**

So, the positions of P-1 will be

6	13	20	27	18	9	0	28
2	3	4	22	31	23	15	7
14	5	12	19	26	25	17	10
11	30	8	16	24	1	29	21

Figure 1.3 P-1 bit position

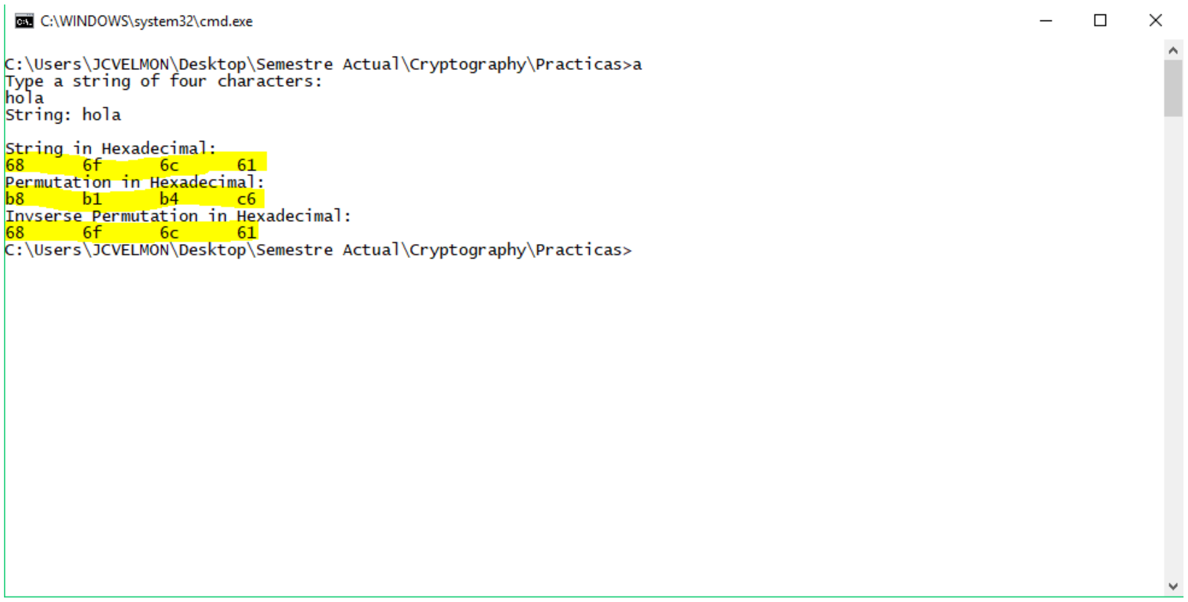
As we can see in the Figure 1.3 we already recover the original positions (see figure 1.1) but with the values of the position based in the Permutation.

Software

Visual Studio Code.

Result

I implement this permutation; the results are the next (In Figure 1.4)



```
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas>a
Type a string of four characters:
hola
String: hola

String in Hexadecimal:
68      6f      6c      61
Permutation in Hexadecimal:
b8      b1      b4      c6
Invserse Permutation in Hexadecimal:
68      6f      6c      61
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas>
```

Figure 1.4 Output of the program

Discussion:

Following the literature review if we have the string “hola” that in Hex is 0x68 0x6F 0x6C 0x61

If we put the bits of the string in a representation like in the Figure 1.5 (View the Figure 1.1 in Literature Review for reference).

0	1	1	0	1	0	0	0
0	1	1	0	1	1	1	1
0	1	1	0	1	1	0	0
0	1	1	0	0	0	0	1

Figure 1.5 Original bits of string “hola”.

When we apply the permutation (View Figure 1.2 in Literature Review for reference and figure 1.6 for the result).

1	0	1	1	1	0	0	0
1	0	1	1	0	0	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	1	1	0

Figure 1.6 Permutation of string “hola”.

And in HEX: 0xB8 0xB1 0xB4 0xC6

And finally, when we apply the P-1 we recover the original representation (View Figure 1,3 for reference and figure 1.7 for the result.)

0	1	1	0	1	0	0	0
0	1	1	0	1	1	1	1
0	1	1	0	1	1	0	0
0	1	1	0	0	0	0	1

Figure 1.7 P-1 of string “hola”.

And in HEX: 0x68 0x6F 0x6C 0x61

So if you see the Figure 1.4 the result in hex is the same for the string “hola” that we obtained doing the process.

Conclusions:

In conclude I learn how the permutation works, this process applies in DES algorithm and its very In conclusion, I learned how they are stirred and how the bits in the permute function are recovered using the DES algorithm, although in this case we only use 32 bits instead of 64 bits, but I think it was quite useful to see how simple it is. I implemented it although I had some difficulties when implementing it in the order of printing and to convert the binary values into hexadecimal, although in the end I found it.

On the other hand, this implementation is so simple that it has no application in the real world, as I said the algorithm DES uses the same process but with 64 bits at the end a it is the same but this is only a small part of the algorithm by itself It is not very useful.

References:

The algorithm, and the order of P and P-1 was taken from the class.

Code

```c
#include<stdio.h>
#include<math.h>
unsigned char per_ord[] = {1, 26, 15, 14, 13, 22, 7, 8, 29, 2, 16, 31, 21, 6, 23, 9, 28, 17, 3, 20, 5, 24, 12, 10, 27, 18, 19, 4, 0, 25, 30, 11};
unsigned char p[32],p_1[32];

int main(){
    unsigned char cad[4]={'0'};
    int i;
    unsigned char position, exponent, aux;
    printf("Type a string of four characters:\n");
    scanf("%s",cad);
    printf("String: %s \n",cad);
    printf("\nString in Hexadecimal:\n%x\t%x\t%x\t%x\n",cad[0],cad[1],cad[2],cad[3]);
for (i = 0; i < 32; ++i)
    p[i]=0;

for (i = 0; i < 32; ++i){
    position = per_ord[i]/8;
    exponent = per_ord[i]%8;
    aux=pow(2,exponent);
    if(cad[position]&aux)
        p[i]=1;
}
for(i=32;i>4;i-=4){
        if(i%8==0)
```

```
            printf("\t");
        printf("%x",8*p[i]*p[i-1]+2*p[i-2]+p[i-3]);
    }
```

```
  printf("\nPermutation inverse:\n");

for(i=0;i<32;i++){
        p_1[p_ord[i]]=p[i];
}
for(i=32;i>0;i-=4){
        if(i%8==0)
            printf("\t");
        printf("%x",8*p_1[i]*p_1[i-]+2*p_1[i-2]+p_1[i-3]);
    }

  return 0;
}
```