|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Cryptography**

**“IP and IP-1”**

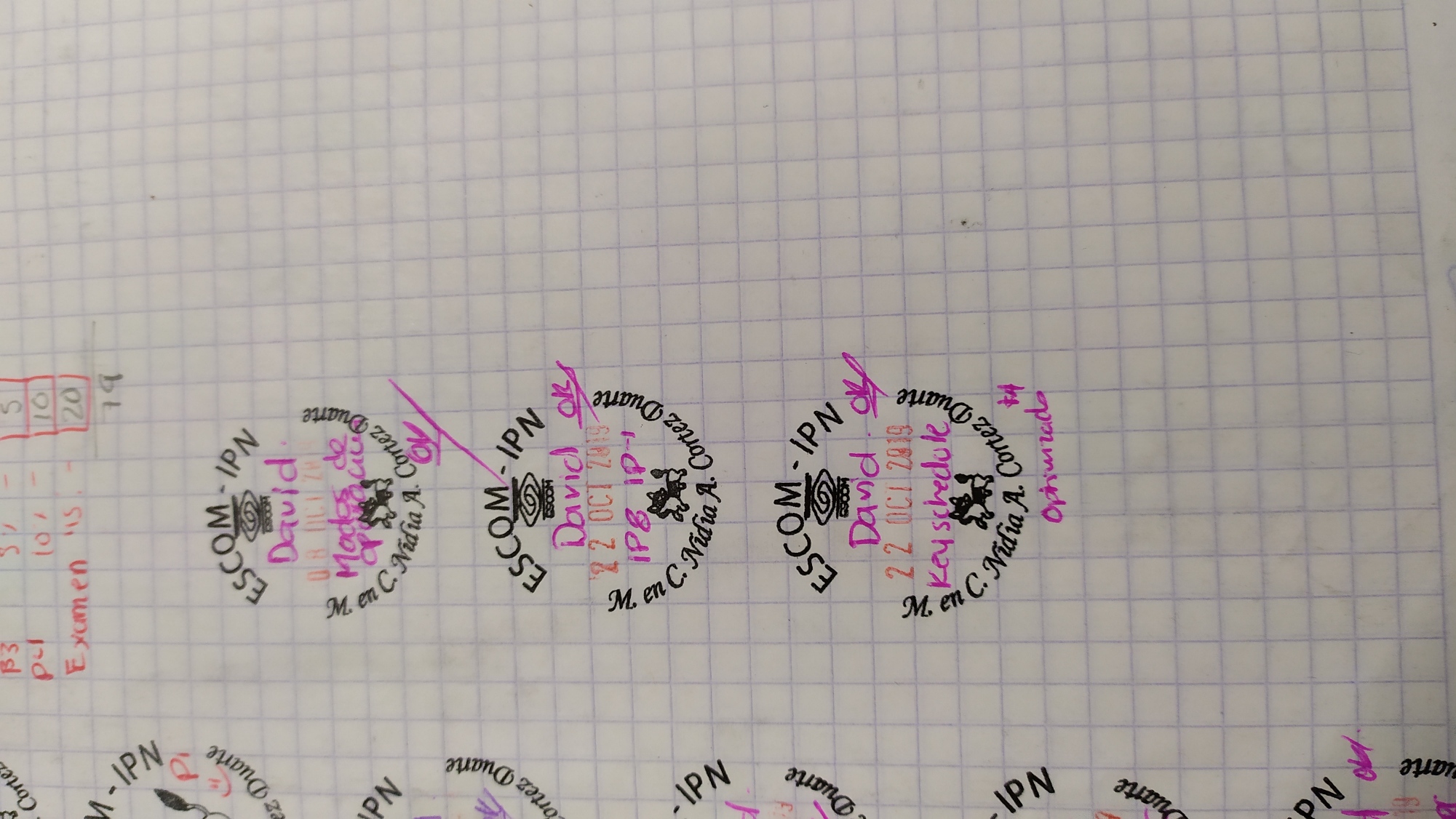
**By:**

**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

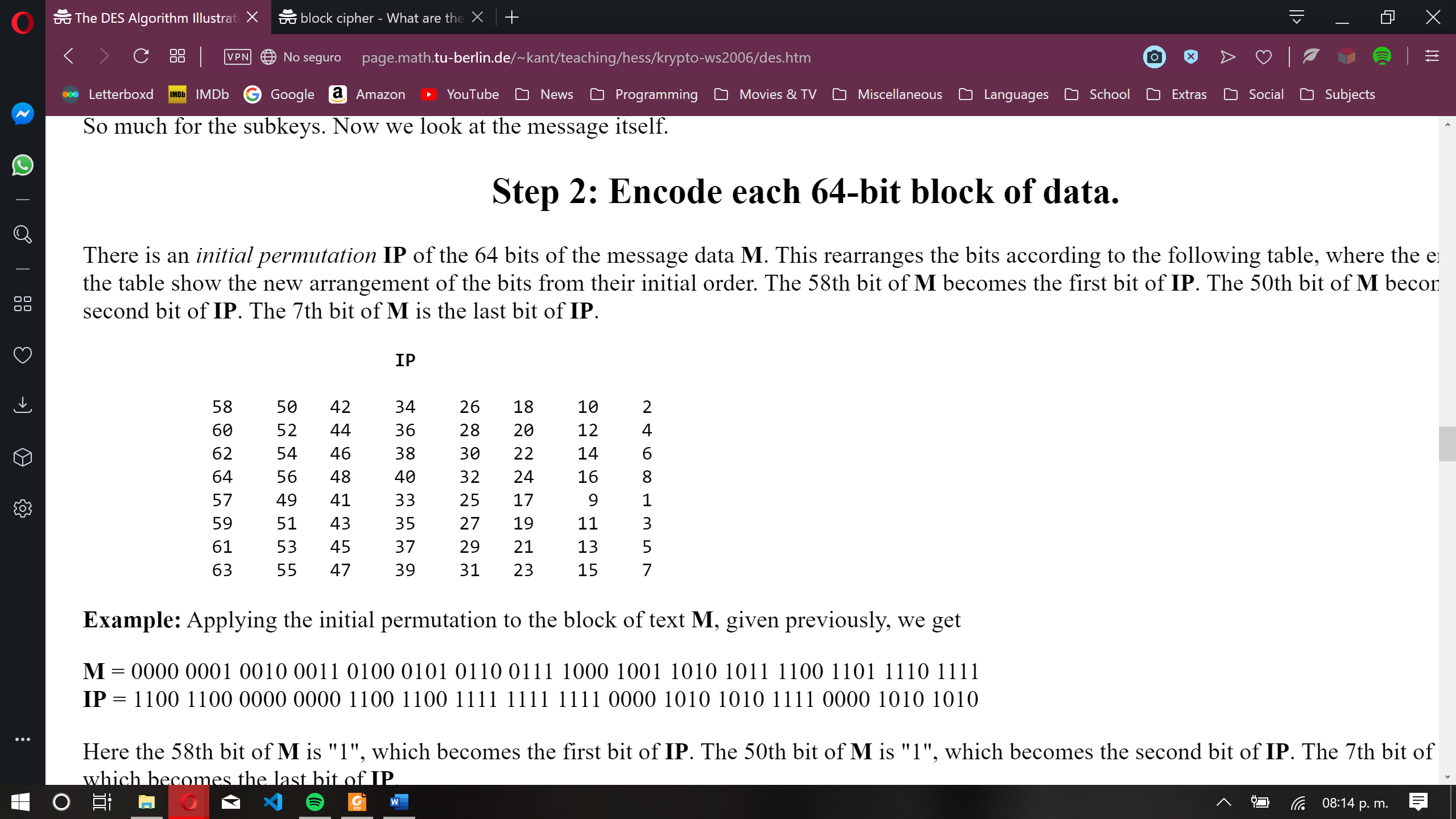


Table 1: Initial Permutation

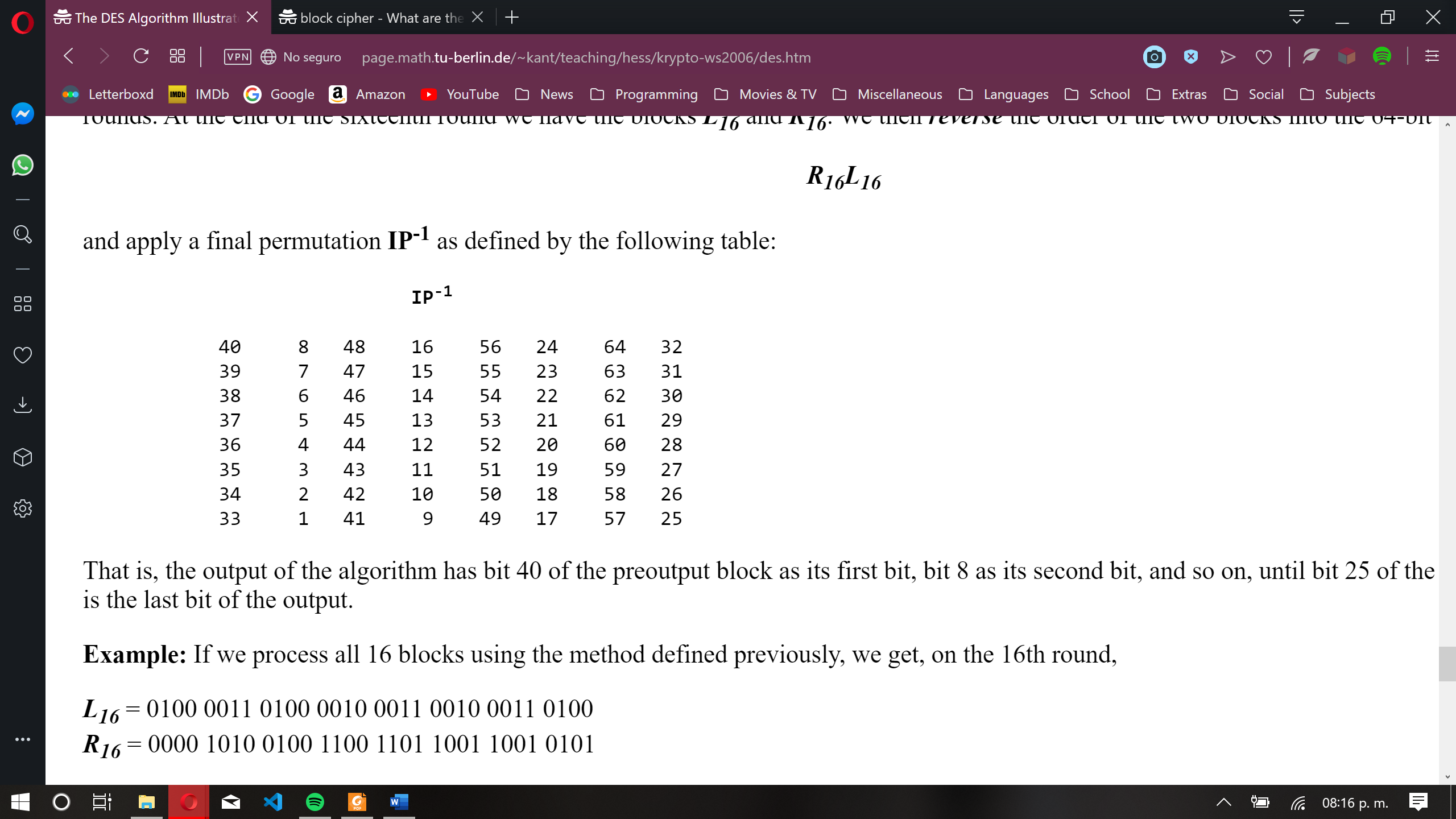


Table 2: Inverse Initial Permutation

# Results

In this test I used, as asked by the teacher, to user 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.

It 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.

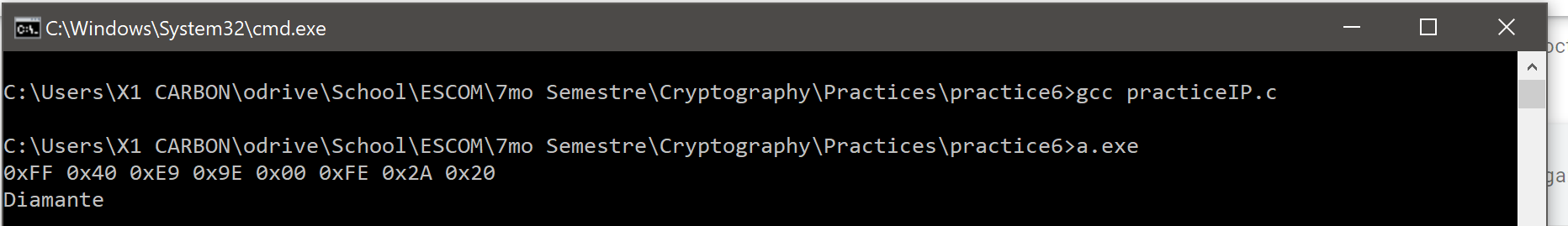


Figure 1: Exectution of the program

# 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;  } |