

**DES Encryption – Lab Work**

FAHS DYLAN M. FLASQUE

OBEID NASSIM PROMO 2019

Summary

[Introduction 3](#_Toc500340333)

[Aim 3](#_Toc500340334)

[Progress table 3](#_Toc500340335)

[DES Description 4](#_Toc500340336)

[Conclusion 7](#_Toc500340337)

# Introduction

The Data Encryption Standard is an algorithm based on symmetric encryption of data. The DES algorithm transforms a string block of 64 bits into another string block of the same length through a series of operations. It uses a key of 64 bits to encrypt. In fact, only 56 bits are used by the DES algorithm because 8 bits are used to check the parity. That particular key of 56 bits allows the decryption of data. Nowadays, DES is considered insecure because of its 56-bit key size which is too small. Nevertheless, the modern cryptography has been highly influenced by DES algorithm.

# Aim

The aim of this lab work is to understand and implement a symmetric cryptographic protocol in C++ programming language.

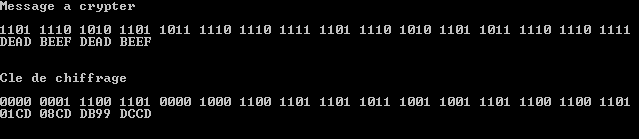
# Progress table

|  |  |  |
| --- | --- | --- |
| **Tasks** | **In progress** | **Done** |
| **Initial permutation** |  | X |
| **L0 and R0** |  | X |
| **Expand Ri** |  | X |
| **Subkey Key** |  | X |
| **SBoxs output** |  | X |
| **Permute the result** |  | X |
| **XOR operation** |  | X |
| **Results** |  | X |

# DES Description

At the beginning of the program, we initialized the message ‘M’ and the main key in the data\_tables.h and data\_messages\_examples.h files which were given in the subject.

For instance, we took the message 3 and the first key as you can see on the screen.

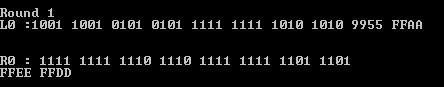


For the first step, we permuted the message 3 thanks to the permute function which use the ‘init\_perm’ array.

Here the result after permutation of the message.

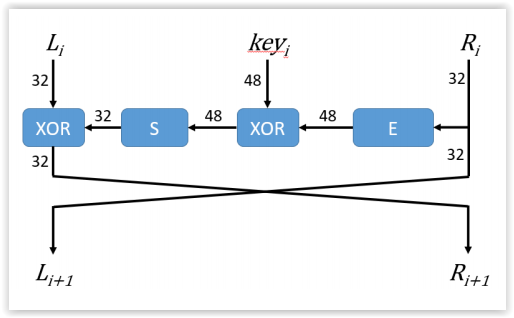


For the second step, we splited the message permuted into 2 blocks of 32 bits : left block (L0) and right block (R0). In order to split the message we used 2 functions which allow to copy the first 32 bits of the message into the left block and the others bits into the right block as following.



Then we have to repeat the following steps 16 times.

Here a schema which resume all the operations that we have to achieve.



E 🡪 expansion operation

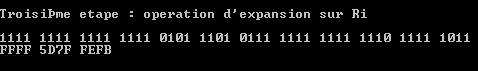
XOR 🡪 XOR operation

S 🡪 sBox operation

Let us detailed the different steps of this schema for the first round. It is the same for the 15 next.

First of all, there is the expansion operation as you can see on the right of the schema. For that step, we expanded the right block from 32 bits to 48 bits. We used the permute function and the ‘expansion\_table’ array.

As you can see below.



At the same time, an other operation is achieve. In fact, for each round we generated a new subkey of 48 bits thanks to the key given in the subject.

Let us explain you how does it work.

As the initial message, the main key is splitted into two blocks of 28 bits. However, in that case, there is also a permutation which is achieved on each block.

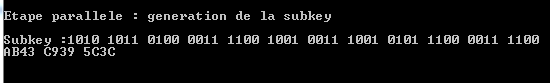
For that operation, we used the permute function and two arrays which are ‘pc\_1\_left’ array and ‘pc\_1\_right’ array.

After that, both blocks are left shifted by some number of positions, depending on the round. In fact, we implemented a funtion which need the keyshift array given in the subject and the round.

For example, for the first round, the shift is 1 bit left, for the second,the shift is 2 bits left etc.

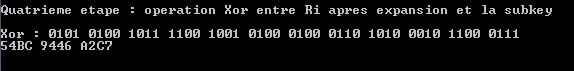
Then the two blocks were assembled and a last operation of permutation have been done.

We used again the permute function and the ‘pc\_2’ array. That operation transformed the block of 56 bits into 48 bits as following.



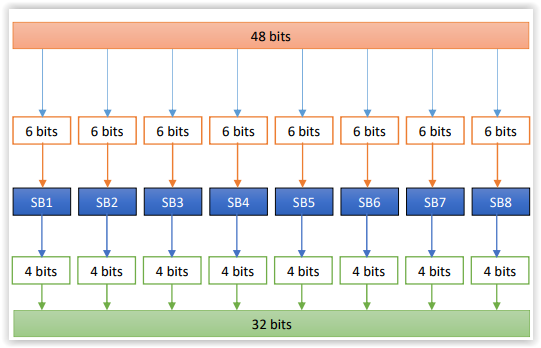
For the next step, we used the two previous results : the generated subkey and the expanded right block. We made a Xor operation between the two results. For that operation, we implemented a function which compare the bits between the two results and give 1 in output when the generated subkey bit and the expanded right block bit are the same ( two 1 or two 0).

As the result below.



After that, we implemented the sBox function which was the most difficult to realize. In fact, before implementing the function we needed time to understand how does it work.

Here a schema which explain the sBox operation.



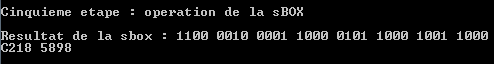
The input data of the sBox is the result of the Xor operation that we have made in the previous step. The sBox is a two dimension array. It is composed by 8 arrays : sBox1, sBox2 … each arrays contains 64 values.

We splitted the Xor operation result into 8 blocks of 6 bits. The first and the last bits of 6 bits blocks allowed to find the row of the output in the subsBox. The other 4 bits allowed to find the column of the ouput.

Then we collected the value pointed by the row and the column in the appropriate subBox thanks to the ‘valueResearchBox’ function.

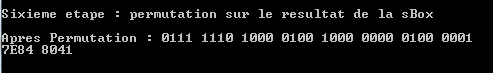
Finally, we obtained a 4 bit block in output of each subsBoxs and we assembled each 4 bits blocks together in order to have a 32 bits block.

You can see the result of the sBox for the first round below :

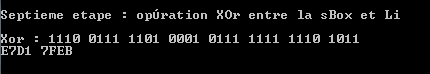


Following that operation, we needed to permute the result thanks to the function permute and the ‘permut\_32’ array.

Here the result.



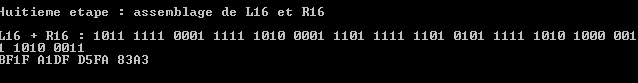
After the permutation, we made another Xor operation between the previous result and Li.



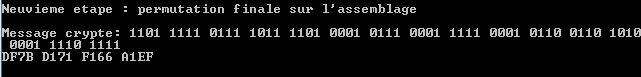
This is the end of the first round.

After 16 rounds, we obtained the left block and the right block: L16 and R16.

We used a funtion in order to assemble the two blocks. We obtained a 64 bits block.



The final step consisted on a last permutation on the previous block with the function permute and ‘reverse\_perm’ array.



# Conclusion

Thanks to that lab work we are able to implement a symmetric cryptographic protocol.