

Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova Universitatea Tehnic**ă** a Moldovei

Facultatea Calculatoare, Informatic**ă ş**i Microelectronic**ă** Departamentul Ingineria Software și Automatica

Report

for laboratory work No. 4

course "Cryptography and Security"

Done by: Felicia LUPAȘCU, gr. FAF-212

Checked by:  
 Cǎtǎlin MÎŢU

Chișinău – 2023

**Subject:** Block ciphers. The DES algorithm

## Tasks:

To develop a program in one of the programming languages preferred for implementing an element of the DES algorithm. The task will be chosen according to the order number n of the student from the group list, according to the formula: nr\_task = n mode 11. For each task, the tables used and all intermediate steps should be displayed on the screen. Input data may be user-entered or randomly generated.

Careful! When supporting the paper, questions will be asked about the work of the entire algorithm!!!

2.4. In the DES algorithm the message (8 characters) is given. To find out L1.

**Theory :**

DES stands for Data Encryption Standard. There are certain machines that can be used to crack the DES algorithm. The DES algorithm uses a key of 56-bit size. Using this key, the DES takes a block of 64-bit plain text as input and generates a block of 64-bit cipher text.

The DES process has several steps involved in it, where each step is called a round. Depending on the size of the key being used, the number of rounds varies. For example, a 128-bit key requires 10 rounds, a 192-bit key requires 12 rounds, and so on.

The DES (Data Encryption Standard) algorithm is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology (NIST). The algorithm takes the plain text in 64-bit blocks and converts them into ciphertext using 48-bit keys.

Since it's a symmetric-key algorithm, it employs the same key in both encrypting and decrypting the data. If it were an asymmetrical algorithm, it would use different keys for encryption and decryption.

The plain text is divided into smaller chunks of 64-bit size. The IP is performed before the first round. This phase describes the implementation of the transposition process. For example, the 58th bit replaces the first bit, the 50th bit replaces the second bit, and so on. The resulting 64-bit text is split into two equal halves of 32-bit each called Left Plain Text (LPT) and Right Plain Text (RPT).

**Applications of DES Algorithm**

* It is used in random number generation
* It is deployed when not-so-strong encryption is needed
* It is used to develop a new form of DES, called Triple DES (using a 168-bit key formed using three keys)

**Implementation:**

I implemented a Python script that performs the initial permutation step in the Data Encryption Standard (DES) algorithm. This code allows me to input an 8-character message or generate a random one and then apply the initial permutation to that message to calculate L1. Here's a breakdown of the code's functionality:

***Initial Permutation (IP):*** In the DES algorithm, the initial permutation is a crucial step. It involves rearranging the bits of the input message according to a predefined permutation table. The purpose of this step is to distribute the data across both halves of the 64-bit block (L0 and R0) to prepare it for further encryption. I've defined the initial permutation table, which specifies the positions of the bits in the input message.

# Define the initial permutation table for DES  
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]

***User Input or Random Generation:*** To adhere to the requirements, I've implemented two options for input:

***Manual Input:*** The code allows me to enter an 8-character message. It checks if the message length is exactly 8 characters and if not, it asks to try again.

***Random Generation:*** Alternatively, I can choose to generate a random message. The code creates this message by combining letters, digits, and punctuation.

# Generate a random message with letters, digits, and punctuation  
characters = string.ascii\_letters + string.digits + string.punctuation  
message = ''.join(random.choice(characters) for i in range(8))  
print('Message: ', message)

***Binary Conversion***: DES operates on binary data, so I need to convert the input message into its binary representation. To do this, the code converts each character to its corresponding 8-character binary string. This binary representation is essential for the subsequent permutation.

# Convert the message to its binary representation  
message = ''.join(format(ord(i), '08b') for i in message)  
print('\nMessage in binary format:\n', message, '\n')

***Information Display:*** The code adheres to the requirement of displaying intermediate steps and tables used by providing the following information:

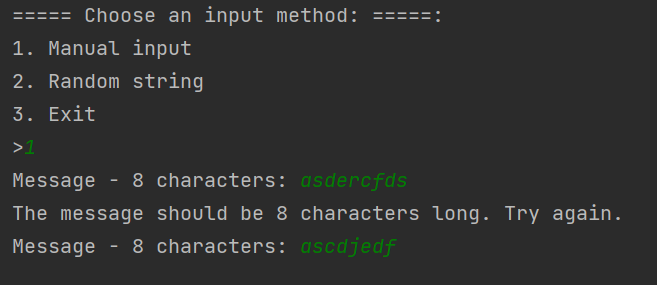
* The binary representation of my message.
* The initial permutation table, listing the bit positions and their corresponding positions in the initial permutation.
* After performing the initial permutation, the code displays the message in its new order.

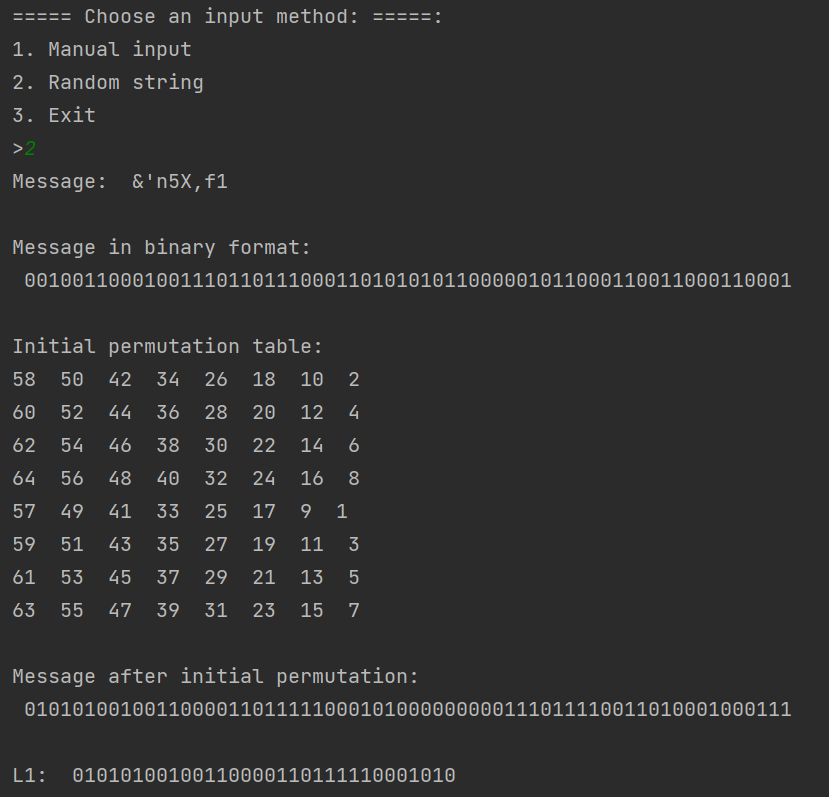
print('\nMessage after initial permutation:\n', message, '\n')

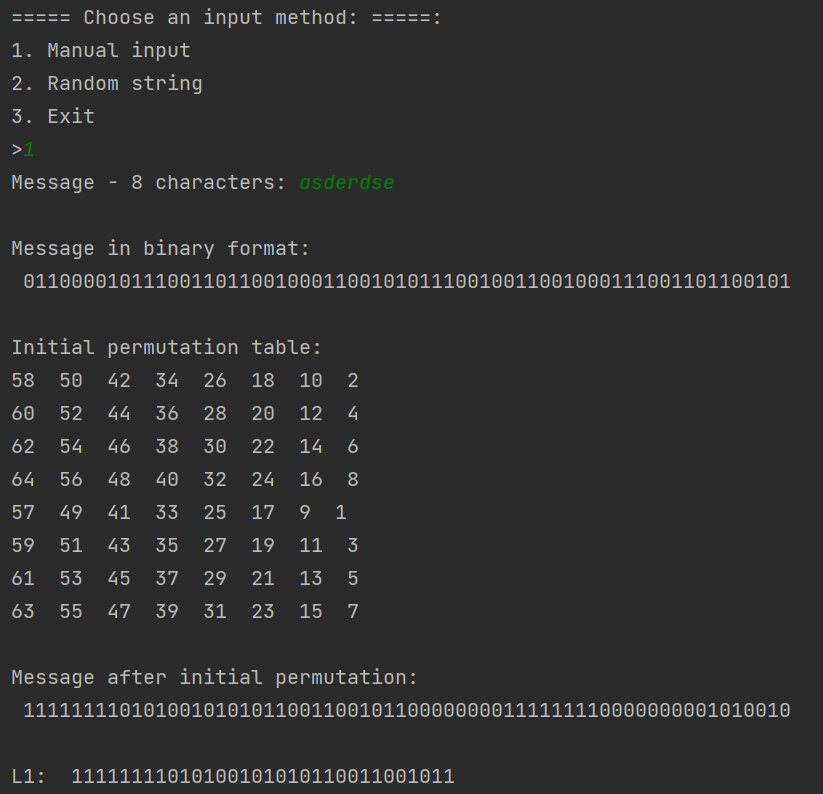
* Finally, the code extracts and displays the left half of the message, which is L1.

# Extract the left half (L1) of the message  
L1 = message[0:32]  
print('L1: ', L1, '\n')

By implementing these steps, I've ensured that the code complies with the requirement to display all intermediate steps and tables used. Additionally, it accommodates both user input and random message generation, as requested.







**Conclusion:**

Overall, this laboratory work has allowed me to gain a deeper understanding of the initial permutation step within the DES algorithm. It also reinforced the significance of ensuring that the data is properly prepared before it undergoes further encryption or decryption processes. While the code presented here serves as a simplified representation of DES, it accomplishes the goal of implementing the initial permutation, demonstrating the critical role this step plays in the broader DES encryption process.

Through this project, I have honed my skills in both algorithm implementation and code presentation, and I am better equipped to tackle more complex aspects of cryptography and computer security in future endeavors.

<https://github.com/feliciaL3/CS/tree/main/Lab4>