



**Ministry of Education, Culture and Research of the Republic of Moldova**  
**Technical University of Moldova**  
**Department of Software and Automation Engineering**

# **REPORT**

Laboratory work no. 1  
*Cryptographic methods of information protection*

Elaborated:  
st. gr. FAF-213

Botnari Ciprian

Verified:  
asist. univ.

Cătălin Mîțu

Chișinău – 2023

## Table of Contents

Topic: Caesar Cipher .....	3
Tasks .....	3
Theoretical notes .....	3
Implementation.....	4
Results .....	4
Conclusion .....	6
Github .....	6

## Topic: Caesar Cipher

### Tasks

1. Implement the Caesar algorithm for the English alphabet in one of the programming languages. Use only the encoding of the letters as shown in Table 1 (do not allow to use the encodings specified in the programming language, e.g. ASCII or Unicode). Key values shall be between 1 and 25 inclusive and no other values are allowed. Text character values shall be between 'A' and 'Z', 'a' and 'z' and no other values are allowed. If the user enters other values - the correct range will be suggested. Before encryption the text will be converted to upper case and the spaces will be removed. The user will be able to choose the operation - encryption or decryption, enter the key, enter the message or the cryptogram and get the cryptogram or the decrypted message respectively.

2. Implement the Caesar 2-key algorithm, with the conditions expressed in the following Task 1. In addition, key 2 must contain only letters of the Latin alphabet, and must have a length not less than 7.

### Theoretical notes

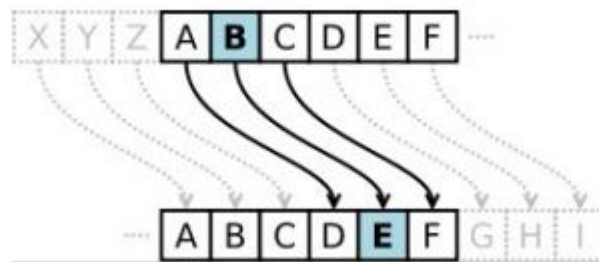
In this cipher each letter of the plain text is replaced by a new letter obtained by an alphabetic shift. The secret key  $k$ , which is the same for both encryption and decryption, is the number indicating the alphabetic shift,

$$k \in \{1, 2, 3, \dots, n - 1\}$$

where  $n$  is the length of the alphabet. Encryption and decryption of the message with the Caesar cipher can be defined by formulas:

$$\begin{aligned} c &= e_k(x) = x + k \pmod{n} \\ m &= d_k(y) = y - k \pmod{n} \end{aligned}$$

where  $x$  and  $y$  are the numerical representation of the respective character of the plaintext. The function called Modulo ( $a \bmod b$ ) returns the remainder of the integer  $a$  to the integer  $b$ . For example, for  $k = 3$  we have (fig 1):



**Figure 1** - Example of alphabetic shift

In order to increase the strength of the Caesar cipher one can apply an alphabet permutation by adding a keyword (not to be confused with the base key of the cipher). This key can be any letter sequence of the alphabet - either a word in the vocabulary or a word that makes no sense. Let

$$k_2 = \text{cryptography}$$

Apply this key to the alphabet

**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**

and we obtain:

**C R Y P T O G A H B D E F I J K M L N Q S U V W X Z**

This new order we obtained by placing  $k_2$  letters at the beginning, then follows the other letters of the alphabet in their natural order. We take into account that the letters will not repeat, i.e. if the letter is encountered several times, it is placed only once.

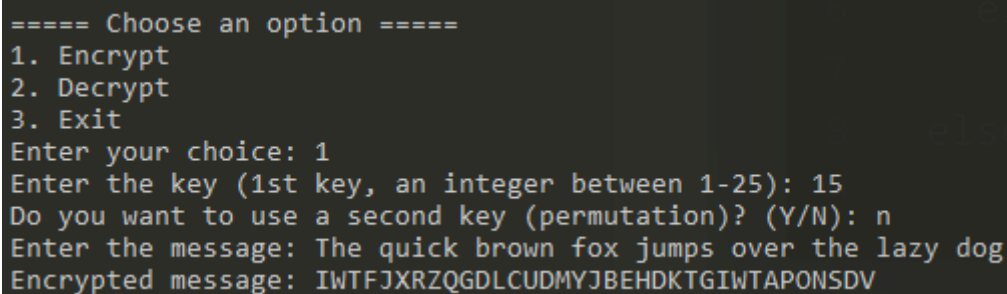
## Implementation

The code below shows how it processes each character in the text string. If a character is an alphabetical letter (A-Z or a-z), it determines its position in the alphabet (ignoring case) and then either encrypts or decrypts it based on the specified action and key. The encrypted or decrypted letter is appended to the result string.

```
for letter in text:
    if letter.isalpha():
        index = alphabet.find(letter.upper())
        if action == "encrypt":
            result += alphabet[(index + key) % 26]
        elif action == "decrypt":
            result += alphabet[(index - key) % 26]
```

## Results

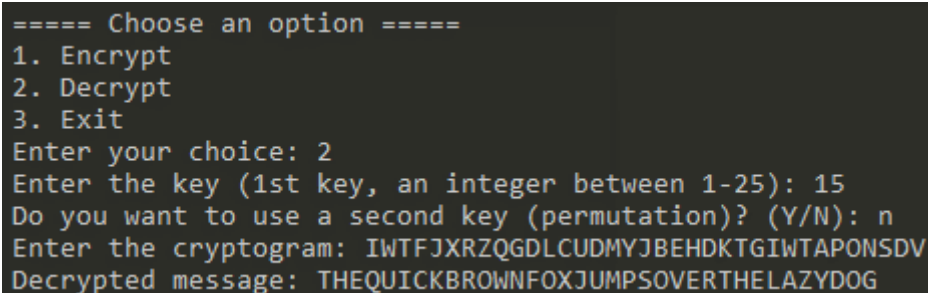
If we choose to encrypt the famous pangram „The quick brown fox jumps over the lazy dog” with 1<sup>st</sup> key being 15 the result is as follows: IUTFJXRZQGDLCUDMYJBEHDKTGIWTAPONSDV



```
==== Choose an option ====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter the key (1st key, an integer between 1-25): 15
Do you want to use a second key (permutation)? (Y/N): n
Enter the message: The quick brown fox jumps over the lazy dog
Encrypted message: IUTFJXRZQGDLCUDMYJBEHDKTGIWTAPONSDV
```

**Figure 2** – Encryption of a message

To make sure the encryption works as intended let's decrypt the cryptogram from previous figure. The key is the same, thus the decrypted message is THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG.



```
==== Choose an option ====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the key (1st key, an integer between 1-25): 15
Do you want to use a second key (permutation)? (Y/N): n
Enter the cryptogram: IUTFJXRZQGDLCUDMYJBEHDKTGIWTAPONSDV
Decrypted message: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
```

**Figure 3** – Decryption of a message

## Implementation

As the comments suggest we remove the duplicate letters from the permutation, then we delete letters from alphabet that are in the permutation and finally add the permutation to the beginning of the alphabet.

```
if permutation:
    # Remove duplicate letters from permutation
    permutation = "".join(dict.fromkeys(permutation))

    # Remove letters from alphabet that are in permutation
    for letter in permutation:
        if letter in alphabet:
            alphabet = alphabet.replace(letter, "")

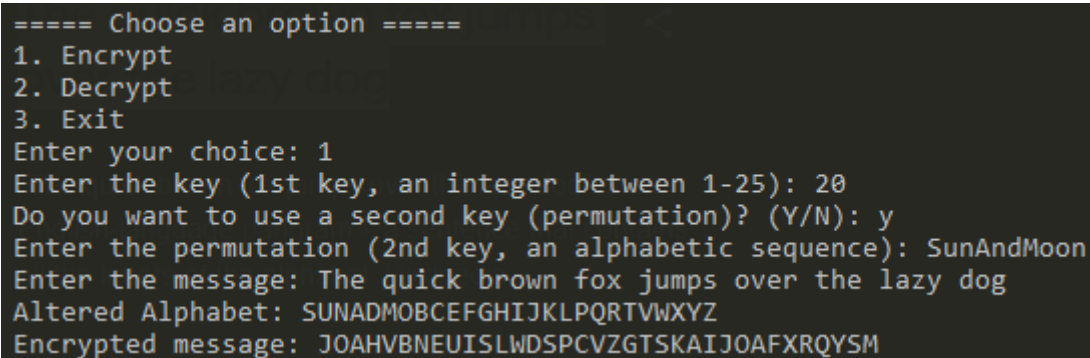
    # Add permutation to the beginning of alphabet
    alphabet = permutation + alphabet
    print("Altered Alphabet:", alphabet)
```

## Results

If we choose to encrypt the famous pangram „The quick brown fox jumps over the lazy dog” with 1<sup>st</sup> key being 15 and 2<sup>nd</sup> one being SunAndMoon the result is as follows:

**Alphabet:** SUNADMOBCEFGHIJKLPQRTVWXYZ

**Cryptogram:** JOAHVBNEUISLWDSPCVZGTSKAIJOAFXRQYSM



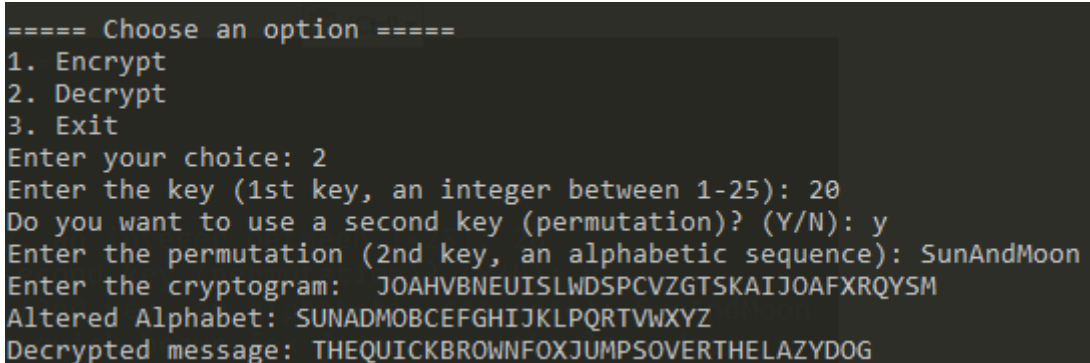
```
===== Choose an option =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter the key (1st key, an integer between 1-25): 20
Do you want to use a second key (permutation)? (Y/N): y
Enter the permutation (2nd key, an alphabetic sequence): SunAndMoon
Enter the message: The quick brown fox jumps over the lazy dog
Altered Alphabet: SUNADMOBCEFGHIJKLPQRTVWXYZ
Encrypted message: JOAHVBNEUISLWDSPCVZGTSKAIJOAFXRQYSM
```

**Figure 4** – Encryption of a message

To make sure the encryption works as intended let's decrypt the cryptogram from previous figure. The keys are the same, thus the decrypted message is

**Alphabet:** SUNADMOBCEFGHIJKLPQRTVWXYZ

**Message:** THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG



```
===== Choose an option =====
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the key (1st key, an integer between 1-25): 20
Do you want to use a second key (permutation)? (Y/N): y
Enter the permutation (2nd key, an alphabetic sequence): SunAndMoon
Enter the cryptogram: JOAHVBNEUISLWDSPCVZGTSKAIJOAFXRQYSM
Altered Alphabet: SUNADMOBCEFGHIJKLPQRTVWXYZ
Decrypted message: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
```

**Figure 5** – Decryption of a message

## Conclusion

During my laboratory experiment, I gained an understanding of how the Caesar cipher encryption algorithm functions. I conducted experiments involving both single and dual keys for message encryption and decryption. While the two-key variant may appear less complex initially, it substantially enhances the cipher's resistance to decryption compared to the single-key version. If we were to employ the brute force method, we would be required to examine a staggering  $26! * 25$  possibilities in order to decipher the message. While it serves as an excellent introduction to encryption concepts, it is not suitable for protecting sensitive or valuable information in today's digital age, where more robust encryption methods are essential to safeguard data.

## Github

[Sufferal/cryptography-labs \(github.com\)](https://github.com/Sufferal/cryptography-labs)