Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Physics

Criptography and security

Laboratory work #1

# Intro to Cryptography. Classical ciphers. Caesar cipher.

*Author:*
Dmitrii Belih
std. gr. FAF-232

*Verified:*
Zaica M.

Chișinău 2025

# Theory Background

Caesar cipher (or Caesar). In this cipher, each letter of the plaintext is replaced by a new letter obtained by an alphabetical shift. The secret key k, which is the same for encryption and decryption, consists of the number indicating the alphabetical shift, i.e. k1, 2, 3,..., n–1, where n is the length of the alphabet. The encryption and decryption of the message with the Caesar cipher can be defined by the formulas

$$c = e_k(x) = x + k \pmod{n},$$
$$m = d_k(y) = y - k \pmod{n},$$

where x and y are the numeric representation of the respective character in the plaintext m and the ciphertext c. The function called Modulo (a mod b) returns the remainder of dividing the integer a by the integer b. This encryption method is named after Julius Caesar, who used it to communicate with his generals, using the key k = 3

## Cifrul lui Cesar + permutare

Given the low cryptoresistance of the Caesar cipher, primarily due to the key space, which consists of only 25 different keys for the Latin alphabet, it can be broken by consecutively trying all the keys. If the message was encrypted with the Caesar cipher, then one of the keys will give us a readable text in the language in which the message was written. For example, if $m$ = *BRUTE FORCE ATTACK* is a message written in English and was encrypted with the key **k = 17,** we obtain the cryptogram **c = SILKVWFITVRKKRTB**

If the cryptanalyst intercepts the encrypted message and goes through all keys 1, 2, ..., 25 – he will obtain As can be seen – only the text obtained by using the key k=17 is meaningful in English, so the message corresponding to the cryptogram is m = BRUTEFORCEATTACK. To increase the cryptoresistance of the Caesar cipher, a permutation of the alphabet can be applied by applying a keyword (not to be confused with the basic key of the cipher). This key can be any sequence of letters of the alphabet - either a word from the vocabulary, or a meaningless one. Either the second key is k2=cryptography. We apply this key to the alphabet 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 was obtained by placing the letters of k2 at the beginning, then the other letters of the alphabet follow in their natural order. We will take into account the fact that the letters will not repeat, that is, if the letter occurs several times, it is placed only once. Next, the Caesar cipher is applied, taking into account the new order of the alphabet:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | R | Y | P | T | O | G | A | H | B | D | E | F | I | J | K | M | L | N | Q | S | U |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |

*Tabelul 2. Cifrul Cezar cu cheia $k_1 = 3$ și $k_2 = cryptography$*

# The Task

1. **Task 1** Implement the Caesar algorithm for the English alphabet in one of the programming languages. Use only the letter encoding (it is not allowed to use encodings specified in the programming language, e.g. ASCII or Unicode). The key values will be between 1 and 25 inclusive and no other values are allowed. The text character values are 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 uppercase and spaces will be removed. The user will be able to choose the operation - encryption or decryption, will be able to enter the key, message or cryptogram and will obtain the cryptogram or decrypted message, respectively.

2. **Task 2** Implement the Caesar algorithm with 2 keys, keeping the conditions expressed in Task 1. In addition, key 2 must contain only letters of the Latin alphabet, and have a length of no less than 7.

# Technical implementation

The following Python program implements a simple **Caesar cipher**, where each letter of the alphabet is shifted by a given amount $k$ modulo 26:

$$E_k(x) = (x + k) \bmod 26, \quad D_k(y) = (y - k) \bmod 26$$

## Task 1

```
dict = {i-65: chr(i) for i in range(65, 65+26)}
dict_r = {chr(i): i-65 for i in range(65, 65+26)}

print(dict)
print(dict_r)

while True:
```

```
    choice = input("Enter␣1␣to␣encode,␣2␣to␣decode,␣or␣3␣to␣exit:
       ↪ ␣")

    if choice == '3':
        print("Exiting␣the␣program.")
        break

    if choice not in ['1', '2']:
        print("Invalid␣choice.␣Please␣enter␣1,␣2,␣or␣3.")
        continue

    message = input("Enter␣text␣to␣process:\n").upper().replace("
       ↪ ␣", "")
    if not message.isalpha():
        print("Text␣must␣be␣only␣in␣the␣English␣alphabet,␣no␣
           ↪ special␣chars.")
        continue

    amount = input("Enter␣shift␣coefficient:\n")
    amount = abs(int(amount)) % 26

    if choice == '1':
        # Encoding
        result = "".join([dict[(dict_r[x] + amount) % 26] for x
           ↪ in message])
        print("Encoded␣message:", result)
    else:
        # Decoding
        result = "".join([dict[(dict_r[x] - amount) % 26] for x
           ↪ in message])
        print("Decoded␣message:", result)
```

## Explanation

- Two dictionaries are created:
  - `dict`: maps numbers $0 \ldots 25$ to letters `A{Z`.
  - `dict_r`: maps letters `A{Z` back to numbers $0 \ldots 25$.
- The program repeatedly asks the user whether they want to:
  1. Encode a message.

2. Decode a message.
3. Exit the program.

- Input text is converted to uppercase and stripped of spaces. Only alphabetic input is allowed. The shift coefficient $k$ (called `amount`) is read from input and reduced modulo 26. For encoding:

$$c = (x + k) \bmod 26$$

Each letter $x$ is converted to a number, shifted by $k$, and converted back to a letter.

For decoding:

$$m = (y - k) \bmod 26$$

Each encoded letter $y$ is shifted backwards by $k$.

The final message is printed after joining all processed characters.

## Task 2

The following Python program implements a modified version of the **Caesar cipher**, where instead of using the standard alphabet order, the cipher first constructs a *custom alphabet* based on a user-defined keyword $k_2$. This keyword determines the initial order of letters, and the remaining unused letters of the alphabet are appended in sequence. After constructing this mapping, the program performs a classical Caesar shift with a user-defined key $k_1$.

Mathematically, encryption and decryption are still defined as:

$$E_{k_1}(x) = (x + k_1) \bmod 26, \quad D_{k_1}(y) = (y - k_1) \bmod 26,$$

but here $x$ and $y$ are taken from the reordered alphabet defined by $k_2$.

```
alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

while True:
    k2 = input("Please␣input␣k2\n").upper().replace("␣", "")
    if len(k2) < 7:
        print("k2␣must␣be␣of␣length␣at␣least␣7")
        continue
    if any(x not in alpha+"␣" for x in k2):
        print("k2␣can␣only␣contain␣letters␣of␣the␣Latin␣alphabet␣
        ↪ (and␣spaces␣that␣will␣be␣removed␣later)")
        continue
    break
```

```
print(len(k2))
dic = {}

for i, char in enumerate(k2):
    if char not in dic.values():
        dic[len(dic)] = char

for char in alpha:
    if char not in dic.values():
        dic[len(dic)] = char

print(dic)

dic_r = {char: i for i, char in enumerate(dic.values())}
print(dic_r)

while True:
    choice = input("Enter 1 to encode, 2 to decode, or 3 to exit:
    ↪ ")

    if choice == '3':
        print("Exiting the program.")
        break

    if choice not in ['1', '2']:
        print("Invalid choice. Please enter 1, 2, or 3.")
        continue

    message = input("Enter text to process:\n").upper().replace("
    ↪ ", "")
    if not message.isalpha():
        print("Text must be only in the English alphabet, no
        ↪ special characters.")
        continue

    amount = input("Enter shift coefficient:\n")
    amount = abs(int(amount)) % 26

    if choice == '1':
        # Encoding
        result = "".join([dic[(dic_r[x] + amount) % 26] for x in
```

```
            ↪ message])
        print("Encoded message:", result)
    else:
        # Decoding
        result = "".join([dic[(dic_r[x] - amount) % 26] for x in
            ↪ message])
        print("Decoded message:", result)
```

The program begins by defining the English alphabet `A{Z` as a reference string. The user is then asked to input a keyword $k_2$, which must contain at least seven letters. All spaces are removed, and the program verifies that the keyword includes only valid Latin letters. If the keyword fails these checks, the user is prompted to enter it again.

Once a valid keyword is provided, a new dictionary `dic` is built to represent the reordered alphabet. Each letter from the keyword is inserted sequentially into the dictionary, ensuring that duplicates are ignored. After all unique letters of the keyword have been added, the remaining unused letters from the standard alphabet are appended to complete the 26-letter mapping.

This mapping defines a new custom alphabet order, where index 0 corresponds to the first letter of $k_2$, index 1 to the second, and so on. The reverse dictionary `dic_r` is then created to allow efficient lookup of each letter's position within this reordered alphabet.

Next, the program enters an interactive loop where the user chooses between three options: encoding, decoding, or exiting the program. If the user decides to encode or decode a message, they are asked to input a text, which is automatically converted to uppercase and stripped of spaces. The program ensures that the input contains only alphabetic characters.

The user then specifies a numeric shift value $k_1$, which determines how far each letter will be shifted within the custom alphabet. The value is normalized by taking its absolute value modulo 26 to ensure it remains within valid bounds.

During the encoding process, each letter $x$ in the message is first mapped to its numerical position using `dic_r`. Then, the encryption formula

$$c = (x + k_1) \bmod 26$$

is applied, and the resulting index is converted back into a letter using `dic`. This produces the encoded text, which is displayed to the user.

Similarly, for decoding, each letter $y$ is converted into its index and processed with the inverse formula

$$m = (y - k_1) \bmod 26,$$

which restores the original message. The decryption works correctly as long as the same $k_1$ and $k_2$ values are used.

The program continues running in a loop, allowing the user to encode or decode multiple messages without restarting. It terminates only when the user selects the exit option.

Overall, this implementation combines two classical ideas: a substitution cipher based on a keyword and a Caesar shift. The keyword $k_2$ alters the alphabet order, while the key $k_1$ determines the rotational shift. Together, they create a more secure cipher than the standard Caesar cipher by introducing both a positional and structural modification to the alphabet.

# Results

*Task1*

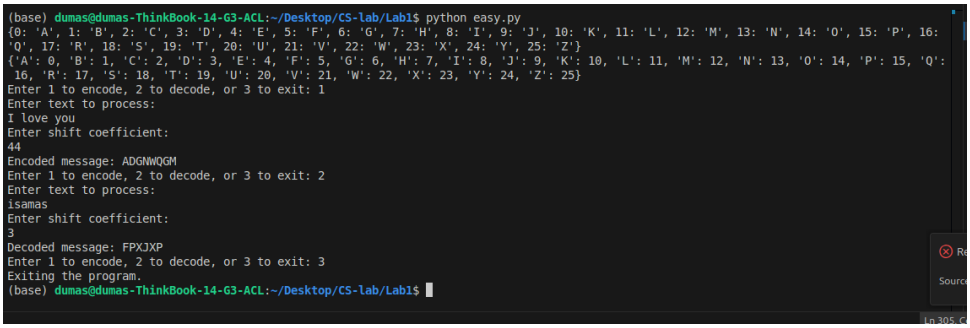


Figure 1: Result of the encoding and decoding program

*Task2*

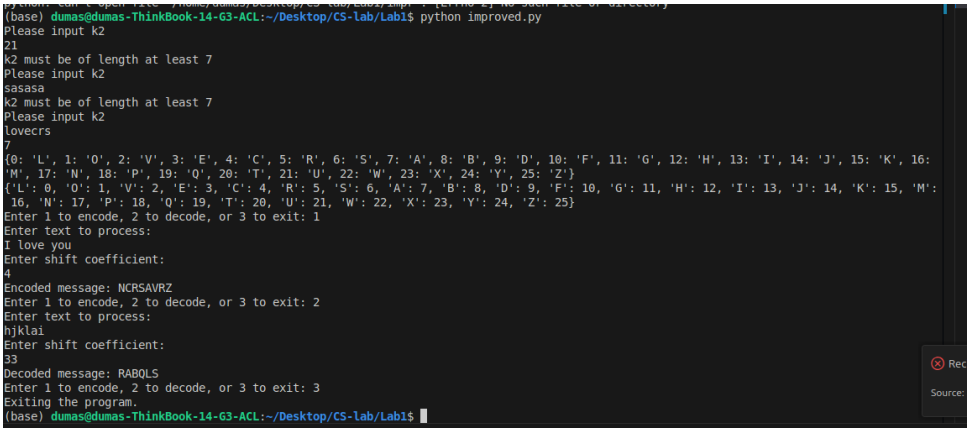


Figure 2: Result of the encoding and decoding program

# Conclusion

In this laboratory work, we explored the fundamentals of classical cryptography through the implementation of the Caesar cipher and its enhanced version with a keyword-based alphabet permutation. The Caesar cipher, a simple substitution cipher, was implemented to perform encryption and decryption by shifting letters in the English alphabet by a user-defined key $k_1$, adhering to the formulas $E_{k_1}(x) = (x + k_1) \mod 26$ and $D_{k_1}(y) = (y - k_1) \mod 26$. The program for Task 1 successfully handled user inputs, ensuring that only valid alphabetic characters and key values between 1 and 25 were processed, with text converted to uppercase and spaces removed for consistency.

For Task 2, we extended the Caesar cipher by incorporating a second key, $k_2$, which reordered the alphabet based on a user-provided keyword of at least seven letters. This permutation increased the cipher's complexity by altering the standard alphabet order before applying the Caesar shift. The implementation ensured robust input validation, rejecting non-alphabetic characters in the keyword and maintaining a unique letter mapping for the custom alphabet. Both tasks were implemented in Python, providing an interactive interface for encoding and decoding messages while demonstrating the practical application of modular arithmetic in cryptography.

The results of both implementations, as shown in the provided figures, confirmed the correctness of the encryption and decryption processes. The programs successfully transformed input texts into their corresponding ciphertexts and back, maintaining fidelity to the theoretical framework. This laboratory work highlighted the strengths and limitations of the Caesar cipher, particularly its vulnerability to brute-force attacks due to a small key space in the basic version, and how a keyword-based permutation can enhance its security.

Overall, this exercise provided valuable insights into the principles of classical ciphers, the importance of input validation in secure programming, and the role of modular arithmetic in cryptographic transformations. It also underscored the trade-offs between simplicity and security in cryptographic systems, laying a foundation for understanding more complex encryption methods.