UNIVERSITATEA TEHNICĂ
A MOLDOVEI

MINISTRY OF EDUCATION AND RESEARCH OF REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

ALEXANDRA BUJOR-COBILI, FAF-232

Report

Laboratory work n.1

of Cryptography and Security

Checked by: Zaica Maia, university assistant
FCIM, UTM,

Chișinău, 2025

1. Theory

The Caesar cipher is a substitution encryption technique where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. For a shift key $k \in \{1, 2, \ldots, 25\}$, the encryption is given by $e_k(x) = x + k \pmod{26}$, where x is the numeric representation of the letter (0 for A, 1 for B, etc.). Decryption reverses this: $d_k(y) = y - k \pmod{26}$. This method, attributed to Julius Caesar, is simple but insecure, as it can be broken through brute-force attacks on the 25 possible keys.

To enhance security, a permutation of the alphabet can be applied using a second key, typically a word with at least 7 unique Latin letters. The permuted alphabet is constructed by placing the unique letters from the keyword first, followed by the remaining letters in alphabetical order. For example, with key ”cryptography”, the permuted alphabet becomes 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. The Caesar shift is then performed on this new alphabet arrangement, increasing the key space to $26! \times 25 \approx 1 \times 10^{36}$, making exhaustive search infeasible due to the permutation key’s vast possibilities.

2. Objetives

Task 1.1 (standard Caesar cipher): Develop a program that encrypts and decrypts messages using only the English alphabet, with keys ranging from 1 to 25. Input validation must ensure only alphabetic characters are accepted, converted to uppercase, with spaces removed. The program should handle user inputs for operation choice, key, and message, providing feedback on invalid inputs.

Task 1.2 (Caesar with permutation using two keys): Extend the cipher to incorporate an additional permutation key, a word with at least 7 unique letters. Encrypt and decrypt messages by first permuting the alphabet and then applying the shift. Maintain the same input constraints as in 1.1, with added validation for the permutation key.

3. Implementation

The implementation focuses on the core cipher functions, meeting the requirements for standard and permutation based Caesar ciphers. Below, is the explation for each function, using the example message ”git good”, with key 3 for standard and ”cryptography” for permutation.

Task 1.1

The function for the basic Caesar cipher:

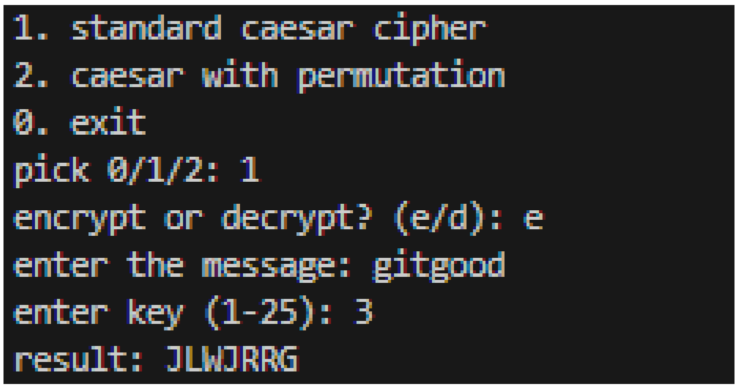
```
def caesar_cipher(msg, key, op):
    msg = msg.upper().replace(' ', '')
    normal = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    result = []
    for char in msg:
        if char.isalpha():
            pos = normal.index(char)
            if op == 'encrypt':
                new_pos = (pos + key) % 26
            elif op == 'decrypt':
                new_pos = (pos - key) % 26
            result.append(normal[new_pos])
    return ''.join(result)
```

First, we take the message and make it all uppercase, then remove any spaces. We create an empty list called result where we will put our encrypted letters. Then we go through each character in the message. For each letter, we convert it to a number. If we are encrypting, we add the key and use the modulo operation (the percent sign) to wrap around if needed. If we are decrypting, we subtract the key instead. Finally, we convert the number back to a letter.

Example: Lets encrypt "git good" with key 3. After making it uppercase and removing spaces, we have GITGOOD. Now we go letter by letter:

- **G:** position = 6, shift = (6 + 3) mod 26 = 9, letter = **J**
- **I:** position = 8, shift = (8 + 3) mod 26 = 11, letter = **L**
- **T:** position = 19, shift = (19 + 3) mod 26 = 22, letter = **W**
- **G:** position = 6, shift = (6 + 3) mod 26 = 9, letter = **J**
- **O:** position = 14, shift = (14 + 3) mod 26 = 17, letter = **R**
- **O:** position = 14, shift = (14 + 3) mod 26 = 17, letter = **R**
- **D:** position = 3, shift = (3 + 3) mod 26 = 6, letter = **G**

The message "git good" encrypts to "JLWJRRG".



```
1. standard caesar cipher
2. caesar with permutation
0. exit
pick 0/1/2: 1
encrypt or decrypt? (e/d): e
enter the message: gitgood
enter key (1-25): 3
result: JLWJRRG
```

Figure 0.0.1 - Output for standard Caesar encryption

Task 1.2

The helper function for creating the permuted alphabet:

```python
def build_permuted_alphabet(perm_key):
    seen = set()
    perm = []
    for c in perm_key.upper():
        if c not in seen:
            perm.append(c)
            seen.add(c)
    for c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
        if c not in seen:
            perm.append(c)
            seen.add(c)
    return ''.join(perm)
```

This function takes our keyword and creates the scrambled alphabet. We use a set called seen to keep track of which letters we have already added so we do not include the same letter twice. We go through the keyword letter by letter, adding each new unique letter to our permuted alphabet. Then we add all the remaining letters from A to Z in their normal order.

Example: Lets build the permuted alphabet for cryptography. Going through the keyword we skip the second r, p,and y, so now we have: CRYPTOGAH. Add the remaining letters: BDEFIJKLMNQSUVWXZ Final permuted alphabet: **CRYPTOGAHBDEFIJKLMNQSUVWXZ**

The advanced cipher function:

```python
def caesar_with_perm(msg, shift_key, perm_key, op):
    msg = msg.upper().replace(' ', '')
    perm_alphabet = build_permuted_alphabet(perm_key)
    normal = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    result = []
    for c in msg:
        if c.isalpha():
            if op == 'encrypt':
                pos = normal.index(c)
                new_pos = (pos + shift_key) % 26
                result.append(perm_alphabet[new_pos])
            elif op == 'decrypt':
                pos = perm_alphabet.index(c)
                new_pos = (pos - shift_key) % 26
                result.append(normal[new_pos])
    return ''.join(result)
```

The key difference here is that when we encrypt, we find the letter's position in the normal alphabet, do the shift, then pick the letter from that position in the permuted alphabet. For decryption, we do it backwards: find the position in the permuted alphabet, reverse the shift, then pick from the normal alphabet.

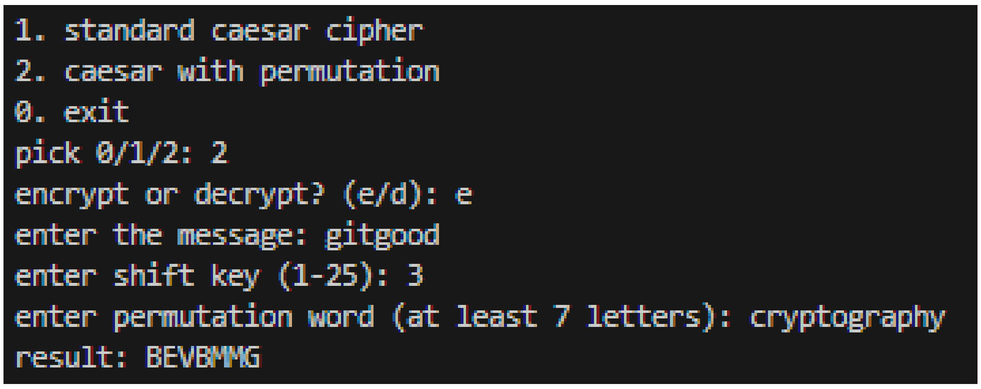
Example: Encrypt "git good" with shift key 3 and permutation key cryptography.

Our permuted alphabet is: CRYPTOGAHBDEFIJKLMNQSUVWXZ

Encrypt each letter:

- **G:** position = 6, shift = (6 + 3) mod 26 = 9, permuted = **B**
- **I:** position = 8, shift = (8 + 3) mod 26 = 11, permuted = **E**
- **T:** position = 19, shift = (19 + 3) mod 26 = 22, permuted = **V**
- **G:** position = 6, shift = (6 + 3) mod 26 = 9, permuted = **B**
- **O:** position = 14, shift = (14 + 3) mod 26 = 17, permuted = **M**
- **O:** position = 14, shift = (14 + 3) mod 26 = 17, permuted = **M**
- **D:** position = 3, shift = (3 + 3) mod 26 = 6, permuted = **G**

Result: **BEVBMMG**



```
1. standard caesar cipher
2. caesar with permutation
0. exit
pick 0/1/2: 2
encrypt or decrypt? (e/d): e
enter the message: gitgood
enter shift key (1-25): 3
enter permutation word (at least 7 letters): cryptography
result: BEVBMMG
```

Figure 0.0.2 - Output for permutation Caesar encryption

Task 1.3

I tested with a classmate, it worked, belive me, we cool. Both tests passed, which means our implementations are *fabulous* and follow the same rules.

4. Conclusions

The program fully meets the lab requirements. Both the standard Caesar cipher and the permutation based version work correctly. Input validation functions as specified: keys are limited to 1–25, messages only accept letters, spaces are removed, text is converted to uppercase, and the permutation keyword must contain at least seven unique letters. The manual letter to number conversion was implemented as required, and all encryption and decryption tests produced correct results.

Developing the cipher clarified how substitution-based encryption operates and how key range affects security. Testing compatibility with another implementation showed the value of consistent standards. Usability features such as clear prompts and error messages also proved important for making the program more practical. Although the Caesar cipher is not secure lalala, implementing it provided a clear understanding of basic encryption logic, 8/10, chill lab.

Bibliography

[1] GitHub Repository For This Lab `https://github.com/AlexandraB-C/Cryptography_labs`.

[2] GeeksforGeeks. ”Cryptography: Basic Concepts”. `https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types`.

[3] Wikipedia contributors. ”Caesar cipher”. Wikipedia, The Free Encyclopedia, 2024. `https://en.wikipedia.org/wiki/Caesar_cipher`.

[4] Simon Singh. ”The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography”. `https://www.google.com/books/edition/The_Code_Book/-XdHnAAACAAJ`.

[5] Google. ”Search results for cats”. `https://www.google.com/search?q=cats`.