Ministry of Education and Research of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Cryptography and Security

Laboratory work #1

Intro to Cryptography. Classical ciphers. Caesar cipher.

Author:
Mihai Gurduza
std. gr. FAF-233

Verified:
M. Zaica

Chișinău 2025

Purpose

The purpose of this laboratory work is to study and implement classical cryptographic methods, specifically the Caesar cipher and its enhanced version with alphabet permutation. Through this practical work, I aim to understand the basic principles of symmetric encryption, explore the concept of cryptographic keys, and analyze the security limitations of classical ciphers. The lab also focuses on implementing these algorithms using only custom character mapping without relying on built-in ASCII or Unicode encoding functions.

Theory Background

The Caesar cipher, also known as the shift cipher, is one of the oldest and simplest encryption techniques in cryptography. Named after Julius Caesar, who reportedly used this method to communicate with his generals using a shift of 3 positions, this cipher belongs to the family of substitution ciphers.

The mathematical foundation of the Caesar cipher is based on modular arithmetic. For encryption, each letter in the plaintext is shifted by a fixed number of positions in the alphabet according to the formula:

$$c = e_k(x) = (x + k) \bmod n$$

For decryption, the reverse operation is performed:

$$m = d_k(y) = (y - k) \bmod n$$

Where:

- x and y are the numerical representations of chars in plaintext m and ciphertext c
- k is the secret key (shift value) where $k \in \{1, 2, 3, ..., n - 1\}$
- n is the length of the alphabet (26 for English)

The modulo operation ensures that the shift wraps around the alphabet. For example, with $k = 3$: - $e_k(S) = 18 + 3 \bmod 26 = 21 = V$ - $d_k(V) = 21 - 3 \bmod 26 = 18 = S$

The main weakness of the basic Caesar cipher is its extremely small key space of only 25 possible keys, making it vulnerable to brute force attacks. An enhanced version uses alphabet permutation with a keyword to increase security while maintaining the same shifting principle.

The Tasks

This laboratory work consists of two main tasks:

Task 1.1: Basic Caesar Cipher Implementation

Implement the Caesar cipher algorithm for the English alphabet using a custom character mapping system. The implementation must meet the following requirements:

- Use only the character encoding shown in Table 1 (no ASCII or Unicode functions allowed)
- Key values must be between 1 and 25 inclusive
- Input text should contain only letters ('A'-'Z', 'a'-'z')
- Before encryption, convert text to uppercase and remove spaces

Task 1.2: Caesar Cipher with Alphabet Permutation

Enhance the basic Caesar cipher by implementing alphabet permutation using a keyword. This version requires:

- All requirements from Task 1.1
- Second key (keyword) containing only Latin alphabet letters
- Minimum keyword length of 7 characters
- Alphabet rearrangement based on the keyword before applying Caesar shift

The enhanced version creates a permuted alphabet by placing unique letters from the keyword at the beginning, followed by remaining alphabet letters in their natural order. This increases the total number of possible keys to 26!, making brute force attacks computationally infeasible.

Technical implementation

The implementation consists of a main `CaesarCipher` class and a user interface module. The solution is designed to handle both basic Caesar cipher and the enhanced version with alphabet permutation.

Core Algorithm Structure

The `CaesarCipher` class contains the following key components:

- `alphabet`: Standard English alphabet string for character mapping
- `_create_cipher_alphabet()`: Creates permuted alphabet when keyword is provided
- `encrypt()`: Performs encryption with optional keyword support
- `decrypt()`: Performs decryption with optional keyword support

Alphabet Permutation Algorithm

When a keyword is provided, the algorithm creates a new alphabet arrangement:

Listing 1: Alphabet Permutation Implementation

```
def _create_cipher_alphabet(self, offset, key_string=""):
    offset = offset % 26

    unique_key_chars = []
    seen = set()
    for char in key_string.upper():
        if char in self.alphabet and char not in seen:
            unique_key_chars.append(char)
            seen.add(char)

    remaining_chars = [char for char in self.alphabet if char
        not in seen]

    cipher_alphabet = "".join(unique_key_chars) + "".join(
       remaining_chars)
    shifted_alphabet = cipher_alphabet[offset:] +
       cipher_alphabet[:offset]
    return shifted_alphabet
```

Encryption Process

The encryption method implements the mathematical formula with proper input validation:

Listing 2: Encryption Implementation

```
def encrypt(self, message, offset, key_string=""):
    if key_string and len(key_string) < 7:
        raise ValueError("Key string must be at least 7
           characters long")

    for char in message:
        if not (char.isalpha() or char.isspace()):
            raise ValueError("Message can only contain
               English letters and spaces")

    cipher_alphabet = self._create_cipher_alphabet(offset,
       key_string)
    result = []

    for char in message:
        if char.isalpha():
            char_upper = char.upper()
            pos = self.alphabet.index(char_upper)
            encrypted_char = cipher_alphabet[pos]
            result.append(encrypted_char)

    return "".join(result)
```

Results

The implemented Caesar cipher program was tested with both simple and complex key configurations. The following sections demonstrate the actual results obtained from running the program.

Simple Caesar Cipher (Key = 13)

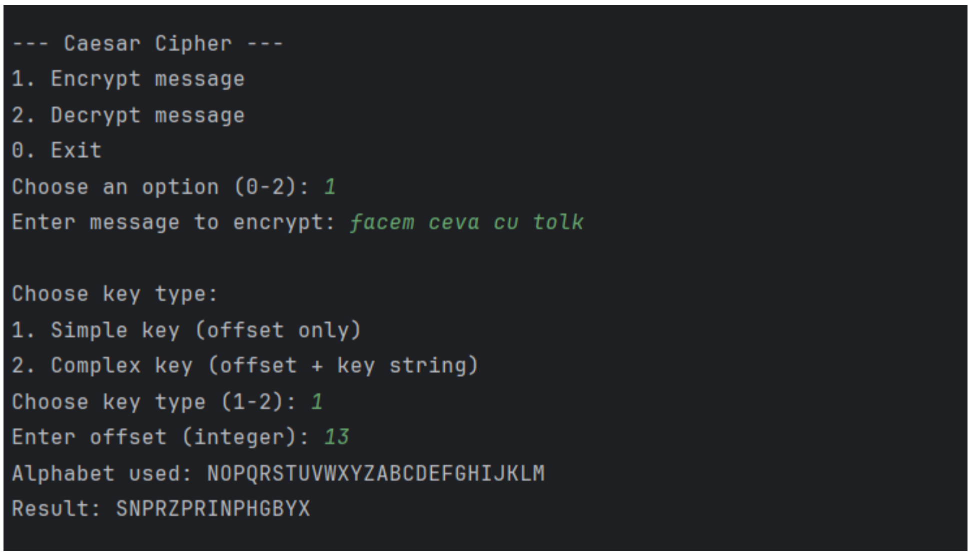
For the simple Caesar cipher test, I used the message ”Facem ceva cu tolk” with an offset key of 13. This represents the classic ROT13 cipher variant.

The alphabet mapping for this configuration is shown in Table 1:

Table 1: Simple Caesar Cipher Alphabet Mapping (k=13)

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

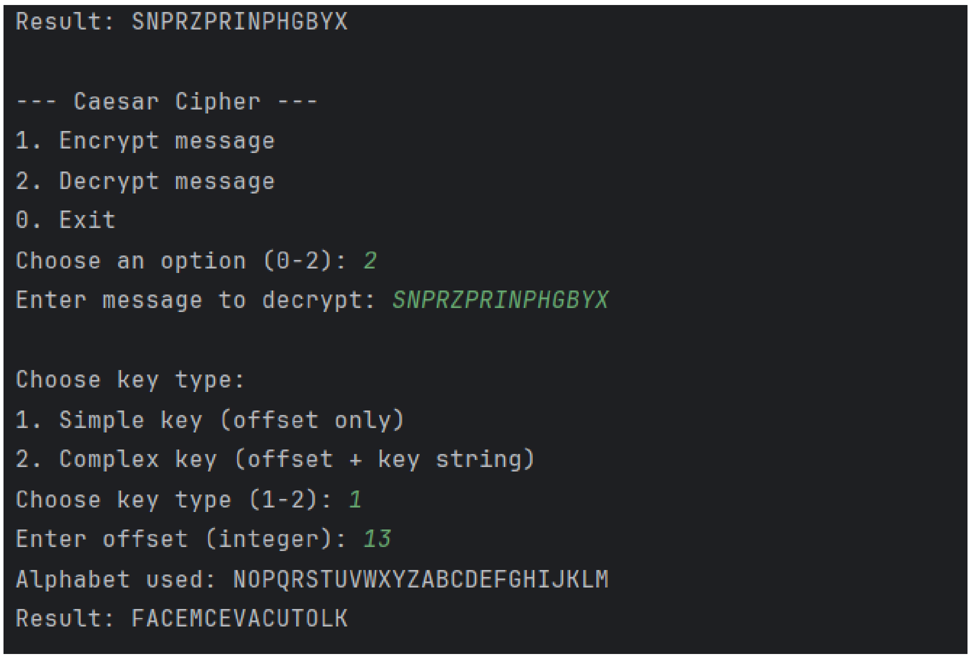
Figure 1 shows the encryption process, and Figure 2 shows the decryption verification.



```
--- Caesar Cipher ---
1. Encrypt message
2. Decrypt message
0. Exit
Choose an option (0-2): 1
Enter message to encrypt: facem ceva cu tolk

Choose key type:
1. Simple key (offset only)
2. Complex key (offset + key string)
Choose key type (1-2): 1
Enter offset (integer): 13
Alphabet used: NOPQRSTUVWXYZABCDEFGHIJKLM
Result: SNPRZPRINPHGBYX
```

Figure 1: Simple Caesar Cipher Encryption Process



```
Result: SNPRZPRINPHGBYX

--- Caesar Cipher ---
1. Encrypt message
2. Decrypt message
0. Exit
Choose an option (0-2): 2
Enter message to decrypt: SNPRZPRINPHGBYX

Choose key type:
1. Simple key (offset only)
2. Complex key (offset + key string)
Choose key type (1-2): 1
Enter offset (integer): 13
Alphabet used: NOPQRSTUVWXYZABCDEFGHIJKLM
Result: FACEMCEVACUTOLK
```

Figure 2: Simple Caesar Cipher Decryption Process

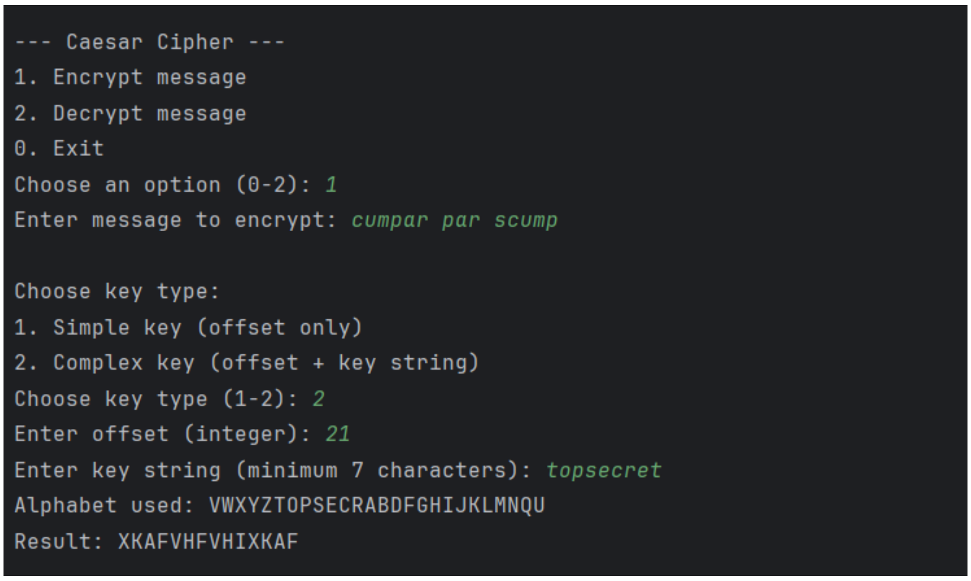
Complex Caesar Cipher with Keyword

The alphabet mapping for this configuration is shown in Table 2:

Table 2: Complex Caesar Cipher Alphabet Mapping (k=21, keyword="topsecret")

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

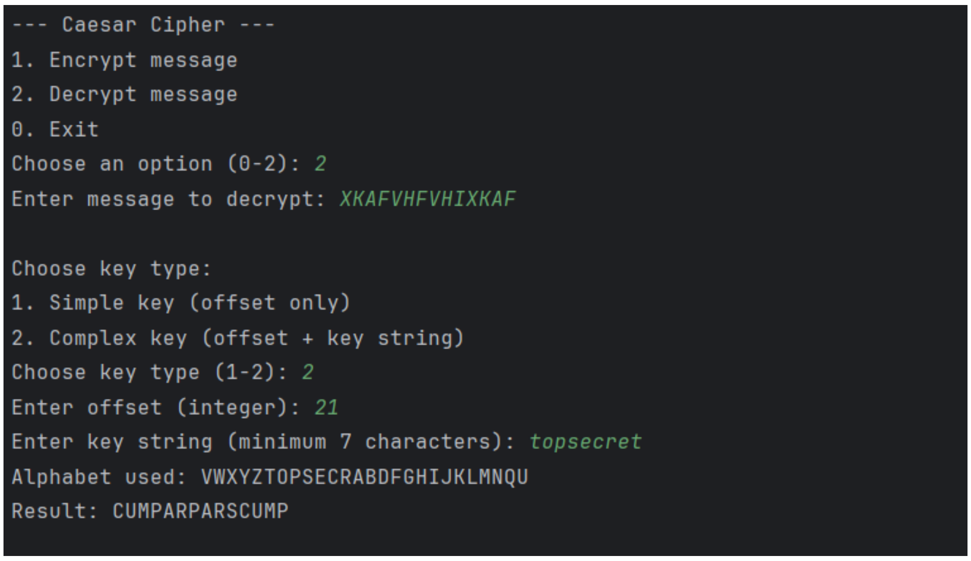
Figure 3 shows the encryption process, and Figure 4 shows the decryption verification.



```
--- Caesar Cipher ---
1. Encrypt message
2. Decrypt message
0. Exit
Choose an option (0-2): 1
Enter message to encrypt: cumpar par scump

Choose key type:
1. Simple key (offset only)
2. Complex key (offset + key string)
Choose key type (1-2): 2
Enter offset (integer): 21
Enter key string (minimum 7 characters): topsecret
Alphabet used: VWXYZTOPSECRABDFGHIJKLMNQU
Result: XKAFVHFVHIXKAF
```

Figure 3: Complex Caesar Cipher Encryption Process



```
--- Caesar Cipher ---
1. Encrypt message
2. Decrypt message
0. Exit
Choose an option (0-2): 2
Enter message to decrypt: XKAFVHFVHIXKAF

Choose key type:
1. Simple key (offset only)
2. Complex key (offset + key string)
Choose key type (1-2): 2
Enter offset (integer): 21
Enter key string (minimum 7 characters): topsecret
Alphabet used: VWXYZTOPSECRABDFGHIJKLMNQU
Result: CUMPARPARSCUMP
```

Figure 4: Complex Caesar Cipher Decryption Process

Conclusion

This laboratory work provided practical experience with classical cryptography through the implementation of the Caesar cipher and its enhanced variant. Through this exercise, I gained understanding of several important concepts:

I learned about the fundamental principles of symmetric encryption, where the same key is used for both encryption and decryption operations. The mathematical foundation using modular arithmetic demonstrated how simple mathematical operations can be used to transform plaintext into ciphertext.

Second, the implementation highlighted the critical importance of key space in cryptographic security. The basic Caesar cipher, with only 25 possible keys, is extremely vulnerable to brute force attacks. Any modern computer can easily try all possible keys in seconds. However, the enhanced version with alphabet permutation dramatically increases the key space to approximately 2^{88} possibilities, making brute force attacks computationally infeasible.

I discovered that even with a larger key space, classical ciphers remain vulnerable to frequency analysis attacks. This limitation emphasizes why modern cryptography has moved beyond simple substitution ciphers to more sophisticated algorithms.

Overall, this laboratory work successfully demonstrated both the historical significance and practical limitations of classical cryptographic methods, providing a solid foundation for understanding more advanced cryptographic concepts.

Bibliography

1. GitHub Repository: https://github.com/m33ga/cs-laboratory-works
2. Caesar cipher: https://en.wikipedia.org/wiki/Caesar_cipher
3. ROT13: https://en.wikipedia.org/wiki/ROT13