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.3

of Cryptography and Security

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

Chișinău, 2025

Content

1 Theory

1.1 Polyalphabetic ciphers

Simple substitution ciphers have a big weakness: each letter always gets replaced by the same letter. This means if you look at how often each letter appears in the encrypted text, you can figure out the pattern and crack the code pretty easily.

Polyalphabetic ciphers fix this problem by changing how each letter gets encrypted based on its position in the message. So the letter ”T” might become ”A” in one spot and ”B” in another spot. This mixes up the frequency patterns and makes the cipher much harder to break.

The key idea is simple: if we encrypt the same letter differently each time, the frequency analysis that works on simple ciphers becomes useless.

1.2 Vigenère cipher

The Vigenère cipher was first described by Giovan Battista Bellaso in 1553, though it was later wrongly attributed to Blaise de Vigenère. It works similarly to the Caesar cipher (which just shifts all letters by the same amount), but instead of using one shift, it uses multiple shifts defined by a keyword.

The cipher uses a key made up of several letters. Each letter in the key tells us how much to shift the corresponding letter in our message. Once we reach the end of the key, we just start over from the beginning.

For example, if our key is ”SUPER”, we shift the first letter by S's value, the second by U's value, the third by P's value, and so on. When we get to the sixth letter of our message, we go back to S and start the pattern again.

For **encryption**, we use:

$$c_i = (m_i + k_i) \mod n \tag{1.2.1}$$

For **decryption**, we use:

$$m_i = (c_i - k_i) \mod n \tag{1.2.2}$$

- c_i is the encrypted letter at position i
- m_i is the original message letter at position i
- k_i is the key letter at position i (repeating as needed)
- n is the alphabet size (26 for English, 31 for Romanian)

The Vigenère cipher is harder to crack for two main reasons:

First, the attacker doesn't know how long the key is. This makes it much harder to find patterns in the encrypted text.

Second, the number of possible keys grows really fast as the key gets longer. For a key of length 5 in English, there are $26^5 = 11,881,376$ possible combinations. That's a lot to check!

2 Objetives

- Understand why polyalphabetic ciphers are more secure than simple substitution ciphers
- Implement the Vigenère encryption and decryption algorithms
- Handle the Romanian alphabet with its special characters
- Validate user input and ensure the key length is at least 7 characters

3 Implementation

The Vigenère cipher was implemented in Python. The implementation follows a modular approach with separate functions for validation, encryption/decryption, and user interaction.

The implementation begins by defining the Romanian alphabet as a constant string:

```
RO_ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

This approach allows us to use Python's built-in `index()` method to convert letters to numbers and direct indexing to convert numbers back to letters, which is more efficient than using a dictionary structure.

3.1 Input Validation

```
def validate_input(text):
    allowed = set('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
                 ')
    return all(c in allowed for c in text)
```

This function creates a set of allowed characters, which includes both uppercase and lowercase versions of all Romanian letters.

The `all()` function returns `True` only if every character in the input text is found in the allowed set.

3.2 Encryption and Decryption Function

```
def vigenere_cipher(text, key, op):
    result = []
    key_len = len(key)
    key_idx = 0

    for c in text:
        mi = RO_ALPHABET.index(c)
        ki = RO_ALPHABET.index(key[key_idx])

        if op == 'encrypt':
            new_mi = (mi + ki) % 31
        elif op == 'decrypt':
            new_mi = (mi - ki) % 31

        result.append(RO_ALPHABET[new_mi])
        key_idx = (key_idx + 1) % key_len

    return ''.join(result)
```

For each character, it first converts both the plaintext character and the corresponding key character to their numerical equivalents using `RO_ALPHABET.index()`. Then formulas are applied: addition for encryption, subtraction for decryption.

The key is applied cyclically using `key_idx = (key_idx + 1) % key_len`, which automatically resets the key index to 0 when it reaches the end of the key.

3.3 User Interface and Input Handling

```
while True:
    key = input("enter key: ").strip().upper().replace(' ', '')
    if len(key) < 7:
        print("key must be at least 7 characters long")
    elif not validate_input(key):
        print("key must contain only letters A-Z a-z or Romanian
               special ")
    else:
        if all(c in RO_ALPHABET for c in key):
            break
        else:
            print("key contains invalid characters")
```

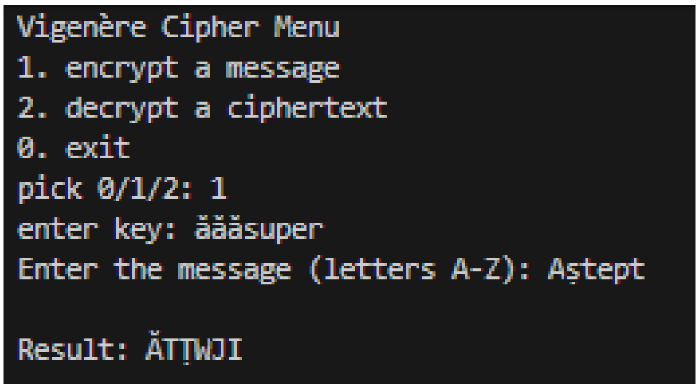
The key input processing removes spaces and converts to uppercase. Validation is applied:

- checking minimum length of 7 characters
- checking if all characters are in the allowed set
- verifying that all characters exist in the Romanian alphabet after uppercase conversion

Message input follows a similar validation pattern.

3.4 Results

Encryption Example



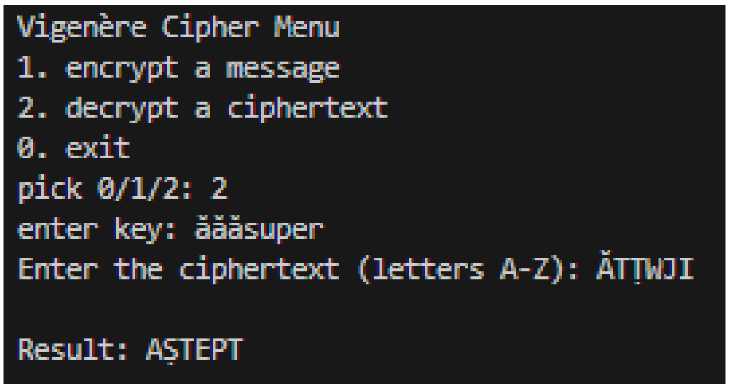
```
Vigenère Cipher Menu
1. encrypt a message
2. decrypt a ciphertext
0. exit
pick 0/1/2: 1
enter key: ăăăsuper
Enter the message (letters A-Z): Așțept

Result: ĂTȚWJI
```

Figure 3.1 - Terminal if choice nr.1

The program converts to uppercase, applies the formula $c_i = (m_i + k_i) \mod 31$ for each character cyclically, and outputs ciphertext ”ĂTȚWJI”.

Decryption Example



```
Vigenère Cipher Menu
1. encrypt a message
2. decrypt a ciphertext
0. exit
pick 0/1/2: 2
enter key: ăăăsuper
Enter the ciphertext (letters A-Z): ĂTȚWJI

Result: AȘTEPT
```

Figure 3.2 - Terminal if choice nr.2

The program applies the inverse formula $m_i = (c_i - k_i) \mod 31$ for each character, successfully recovering the original message ”Așțept”.

4 Conclusion

This laboratory work successfully implemented the Vigenère polyalphabetic cipher for the Romanian alphabet. The implementation applies the mathematical formulas $c_i = (m_i + k_i) \mod 31$ for encryption and $m_i = (c_i - k_i) \mod 31$ for decryption, where the modulo 31 operation handles all Romanian letters including the special characters Ă, Â, Î, Ș, and Ț.

So, the same letter in the plaintext encrypts to different letters in the ciphertext depending on its position. This happens because the key is applied cyclically, so each position uses a different shift value. For example, the letter ’A’ at position 1 might become ’E’, but the same ’A’ at position 8 might become ’R’. This variation makes the cipher resistant to simple frequency analysis attacks

The minimum key length of 7 characters creates approximately 2.75×10^{10} possible key combinations, which provides reasonable security while keeping keys short enough to remember. Thus, this demonstrates how mathematical operations combined with cyclic key application create a more secure cipher than simple substitution methods.

Bibliography

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

[2] GeeksforGeeks Vigenere. https://www.geeksforgeeks.org/dsa/vigenere-cipher.

[3] GeeksforGeeks Playfair. https://www.geeksforgeeks.org/dsa/playfair-cipher-with-examples.