

Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică Departamentul Ingineria Software și Automatica

# Raport

# pentru lucrarea de laborator Nr. 3

# la cursul „Metode criptografice de protecție a informației”

A efectuat:

Konjevic Alexandra, gr. FAF-213

A verificat: Mîțu Cătălin

Chișinău – 2023

**Subject**: Cryptanalysis of polyalphabetic ciphers.

**Objectives:**

Implement the Vigenère algorithm in a programing language of personal choice, for the Romanian alphabet.

**Theory:**

The Vigenère cipher is a classical method of encrypting text that has been used for centuries. Named after its creator Blaise de Vigenère in the 16th century, this cipher gained a reputation as the "unbreakable cipher" due to its complexity. Unlike the simpler Caesar cipher, the Vigenère cipher employs a keyword to encrypt plaintext. This keyword is essential in understanding both the encryption and decryption processes, which are based on straightforward mathematical formulas: *ci = (mi + ki) mod 31* for encryption and *mi = (ci - ki) mod 31* for decryption. These formulas lay the foundation for understanding how the Vigenère cipher operates. This report will delve into the workings of the Vigenère cipher, exploring its historical significance, cryptographic principles, and practical applications in both encryption and decryption processes.

**Task Completion (Variant 2):**

I implemented the task in C#. I created a `VigenereCipher` class, which holds the logic of the cipher in the methods `Encrypt` and `Decrypt`. In the `Main` method of the `Program` class, I have the logic of the menu:

Console.WriteLine("Choose an option:");

Console.WriteLine("e - Encrypt\nd - Decrypt\nq - Quit");

switch (choice){case 'e':

EncryptText();

break;

case 'd':

DecryptText();

break;

case 'q':

exit = true;

break;

default: Console.WriteLine("Invalid choice.”);

break;}

I also have some methods that help for validation: `EncryptText` for getting the message to encrypt – it should contain only letter (to check this, I use the function `IsAllLetters`). For the key validation, I ask the user to introduce a key which has a length of minimum 7 characters and maximum length of the key is the length of the message to encrypt. `DecryptText` validates the user input, but with a message to decrypt. The following login is the same as for the `EncryptText`. I also use the method `GetValidInput` to check if the input is not null and to remove spaces from it.

**Encryption:**

public static string Encrypt(string text, string key)

{

string result = "";

int textLength = text.Length;

int keyLength = key.Length;

for (int i = 0; i < textLength; i++)

{

char textChar = text[i];

if (RomanianAlphabet.Contains(textChar))

{

char keyChar = key[i % keyLength];

int shift = (RomanianAlphabet.IndexOf(textChar) + RomanianAlphabet.IndexOf(keyChar)) % RomanianAlphabet.Length;

result += RomanianAlphabet[shift];

}

else result += textChar;

}

return result;

}

This function initializes an empty string result to store the encrypted text, then it gets the lengths of the input text and the encryption key:

* textLength represents the length of the input text.
* keyLength represents the length of the encryption key.

Next, it iterates through each character in the input text using a for loop. Inside the loop, program checks if the current character textChar is a valid character in the Romanian alphabet. The RomanianAlphabet is a string that contains valid Romanian alphabet characters. If the character textChar is part of the Romanian alphabet, the method proceeds to encryption. Otherwise, it adds the character as is to the result, since non-alphabet characters are not encrypted.

For characters in the Romanian alphabet, the program:

* Gets the corresponding character from the encryption key by using the modulo operator % with i % keyLength. This allows the key to repeat if it's shorter than the text.
* Calculatea the shift by adding the indices of textChar and keyChar in the RomanianAlphabet. The modulo operation ensures that the shift remains within the bounds of the Romanian alphabet length.
* Appends the encrypted character (the character at index shift in the RomanianAlphabet) to the result string.

After processing all characters in the input text, the result string will contain the encrypted text. The program returns the result string, which represents the result of encrypting the input text using the Vigenere Cipher with the provided key.

**Decryption:**

public static string Decrypt(string text, string key)

{

string result = "";

int textLength = text.Length;

int keyLength = key.Length;

for (int i = 0; i < textLength; i++)

{

char textChar = text[i];

if (RomanianAlphabet.Contains(textChar))

{

char keyChar = key[i % keyLength];

int shift = (RomanianAlphabet.IndexOf(textChar) - RomanianAlphabet.IndexOf(keyChar) + RomanianAlphabet.Length) % RomanianAlphabet.Length;

result += RomanianAlphabet[shift];

}

else result += textChar;

}

return result;

}

The Decrypt function is responsible for decrypting a given text that was encrypted using the Vigenere Cipher with the Romanian alphabet. First of all, it initializes an empty string result to store the decrypted text. After that, it gets the lengths of the input text and the decryption key: textLength and keyLength. Then, the program iterates through each character in the input text using a for loop. Inside the loop, it checks if the current character textChar is a valid character in the Romanian alphabet. If the character textChar is part of the Romanian alphabet, it proceeds to decryption. Otherwise, it adds the character as is to the result, since non-alphabet characters are not decrypted.

For characters in the Romanian alphabet, it gets the corresponding character from the decryption key by using the modulo operator % with i % keyLength. This allows the key to repeat if it's shorter than the text.

The method then calculates the shift for decryption by subtracting the indices of textChar and keyChar in the RomanianAlphabet. The result may be negative, so it's adjusted by adding RomanianAlphabet.Length to ensure it's a positive value. The modulo operation then ensures that the shift remains within the bounds of the Romanian alphabet length. After that, the method appends the decrypted character (the character at index shift in the RomanianAlphabet) to the result string. After processing all characters in the input text, the result string will contain the decrypted text. It then returns the result string, which represents the result of decrypting the input text using the Vigenere Cipher with the provided key.

**Results:**

Example 1:

Vigenere Cipher (Romanian Alphabet)

Choose an option:

e - Encrypt

d - Decrypt

q - Quit

e

Enter the text to be encrypted (only letters): hello there

Enter the key (word or sentence, length 6 to 10): superkey

Encrypted text: ZAĂREDMCIA

Vigenere Cipher (Romanian Alphabet)

Choose an option:

e - Encrypt

d - Decrypt

q - Quit

d

Enter the text to be decrypted (only letters): ZAĂREDMCIA

Enter the key (word or sentence, length 6 to 10): superkey

Decrypted text: HELLOTHERE

Example 2:

Vigenere Cipher (Romanian Alphabet)

Choose an option:

e - Encrypt

d - Decrypt

q - Quit

e

Enter the text to be encrypted (only letters): ășț gre îuk

Enter the key (word or sentence, length 6 to 9): keyășțî

Encrypted text: LXȘHÎZȘFQ

Vigenere Cipher (Romanian Alphabet)

Choose an option:

e - Encrypt

d - Decrypt

q - Quit

d

Enter the text to be decrypted (only letters): LXȘHÎZȘFQ

Enter the key (word or sentence, length 6 to 9): keyășțî

Decrypted text: ĂȘȚGREÎUK

**Conclusion:**

In summary, the Vigenere Cipher is a remarkable cryptographic method that has left an indelible mark on the history of cryptography. While it may no longer be suitable for securing sensitive information in today's world of advanced cyber threats, it remains a valuable subject for study and historical appreciation. Understanding its strengths and vulnerabilities can provide insights into the evolution of cryptographic techniques and the ongoing battle between encryption and decryption.

Git repository: https://github.com/alya1007/Labs-semester-5/tree/master/CS