**MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA**

**Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering**

**Postoronca Dumitru FAF-233**

# Report

Laboratory work n.3

**on CS**

*Checked by:*
**M. Zaica**, *university assistant*
DISA, FCIM, UTM

Chișinău – 2025

# 1 Purpose of the Laboratory Work

The purpose of this laboratory work was to study the Vigenère cipher, a classic example of a polyalphabetic substitution cipher. Unlike simple substitution ciphers that are vulnerable to basic frequency analysis, the Vigenère cipher uses a keyword to shift letters by different amounts, effectively obscuring the standard letter frequency patterns of the plaintext language. The goal was to understand the mechanics of this cipher and to implement a C++ program capable of both encrypting and decrypting messages using a custom alphabet, specifically the Romanian alphabet which includes diacritics.

# 2 Strategy Used

To complete this task, I have developed a C++ program that performs Vigenère encryption and decryption. The program is designed to be interactive, prompting the user for the desired operation, a key, and a message.

- **Operation Selection:** The user chooses to either encrypt or decrypt a message.
- **Key Input:** The user provides a keyword, which must be at least 7 characters long.
- **Message Input:** The user enters the plaintext to be encrypted or the ciphertext to be decrypted.

The core of the strategy revolves around several key steps: defining a specific alphabet, normalizing user input to match this alphabet, handling multi-byte UTF-8 characters correctly, and applying the mathematical logic of the Vigenère cipher.

## 2.1 Implementation Details

The program is built around a few core functions that handle text processing and the cryptographic logic.

### 2.1.1 Alphabet and Character Mapping

A custom alphabet (‘ALFABET‘) was defined to include all characters of the Romanian language, including diacritics (Ă, Â, Î, Ș, Ț). This results in an alphabet of 31

characters. A map (unordered_map<string, int>) is used to associate each character with a unique integer index from 0 to 30, which is necessary for the mathematical calculations.

### 2.1.2 Text Normalization and UTF-8 Handling

Since user input can vary in case and format, a ‘normalizeText‘ function was implemented. This function:

- Converts all lowercase characters, including diacritics, to their uppercase equivalents.
- Removes all spaces from the text.

A crucial part of the implementation is handling UTF-8 characters. Characters like “Ă” or “Ș” are represented by more than one byte. Standard C++ string iteration would process these characters byte-by-byte, leading to incorrect behavior. To solve this, the ‘splitUTF8‘ function was created to parse the input string and correctly segment it into a ‘vector¡string¿‘, where each element represents a single, complete character.

### 2.1.3 Vigenère Cipher Logic

The main logic is encapsulated in the ‘vigenere‘ function. It takes the message and the key (as vectors of characters), the character-to-number map, and a boolean flag to determine the operation (encrypt or decrypt).

For each character in the message, it performs the following calculation:

- **Encryption:** $C_i = (P_i + K_i) \pmod{N}$
- **Decryption:** $P_i = (C_i - K_i + N) \pmod{N}$

Where $P_i$ is the numerical value of the plaintext character, $K_i$ is the numerical value of the corresponding key character (the key is repeated if it's shorter than the message), $C_i$ is the numerical value of the resulting ciphertext character, and $N$ is the alphabet size (31 in this case). The addition of $N$ in the decryption formula ensures the result is always non-negative before the modulo operation.

```
string vigenere(const vector<string> &msg, const vector<string> &key,
                const unordered_map<string, int> &map, bool encrypt) {
  string result;
  for (size_t i = 0; i < msg.size(); i++) {
    int p = map.at(msg[i]);
    int k = map.at(key[i % key.size()]);
    int c = encrypt ? (p + k) % ALPHABET_SIZE
                    : (p - k + ALPHABET_SIZE) % ALPHABET_SIZE;
    result += ALFABET[c];
  }
  return result;
}
```

Listing 1: The core Vigenère function implementing encryption and decryption.

# 3 Execution Example

The following demonstrates the program's functionality with a sample encryption and decryption cycle.

## 3.1 Encryption

The program first prompts for the operation, key, and message.

- **Operation:** 1 (Criptare)
- **Key:** 'CRIPTOGRAFIE'
- **Message:** 'Acesta este un mesaj secret'

After normalization, the plaintext becomes 'ACESTAESTEUNMESAJSECRET'. The program then applies the Vigenère encryption logic.

```
Alege operatia (1 - Criptare, 2 - Decriptare): 1
Introdu cheia (minim 7 caractere): CRIPTOGRAFIE
Introdu mesajul: Acesta este un mesaj secret
Criptograma: CȚNGMOLITKCȘQVAPCFLȚRKÂ
```

## 3.2 Decryption

To verify the correctness of the encryption, the resulting ciphertext is decrypted using the same key.

- **Operation:** 2 (Decriptare)
- **Key:** 'CRIPTOGRAFIE'
- **Message:** 'CȚNGMOLITKCȘQVAPCFLȚRKÂ'

The program successfully recovers the original normalized plaintext.

```
Alege operatia (1 - Criptare, 2 - Decriptare): 2
Introdu cheia (minim 7 caractere): CRIPTOGRAFIE
Introdu mesajul: CȚNGMOLITKCȘQVAPCFLȚRKÂ
Mesaj decriptat: ACESTAESTEUNMESAJSECRET
```

This confirms that the implementation of both the encryption and decryption algorithms is correct.

# 4 Conclusion

This laboratory work provided a comprehensive, hands-on experience with implementing the Vigenère cipher. The project demonstrated the complete lifecycle of a cryptographic process: from raw input to normalized plaintext, through encryption to ciphertext, and back to plaintext via decryption.

The main challenges and learning points were:

- The importance of a clearly defined alphabet and the necessity of normalizing all inputs to conform to it.
- The technical requirement of handling multi-byte character encodings like UTF-8 in a language like C++, which is not natively aware of Unicode characters. This required a custom parsing solution.
- The straightforward but powerful mathematical logic behind the cipher, which is based on modular arithmetic.

By successfully implementing this program, I have gained a deeper understanding of polyalphabetic substitution ciphers, the practical challenges of text processing in cryptography, and the foundational principles that govern classical encryption techniques.

# A Full Source Code

```cpp
#include <algorithm>
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>

using namespace std;

const vector<string> ALFABET = {/*alphabetically organized romanian letters*/}
const int ALPHABET_SIZE = ALFABET.size();

unordered_map<string, int> createCharToNum() {
  unordered_map<string, int> m;
  for (int i = 0; i < ALPHABET_SIZE; i++)
    m[ALFABET[i]] = i;
  return m;
```

```
}

string replaceAll(string text, const string &from, const string &to) {
  size_t start = 0;
  while ((start = text.find(from, start)) != string::npos) {
    text.replace(start, from.length(), to);
    start += to.length();
  }
  return text;
}

vector<string> splitUTF8(const string &text) {
  vector<string> chars;
  for (size_t i = 0; i < text.size();) {
    unsigned char c = text[i];
    int len = 1;
    if ((c & 0xE0) == 0xC0)
      len = 2;
    else if ((c & 0xF0) == 0xE0)
      len = 3;
    chars.push_back(text.substr(i, len));
    i += len;
  }
  return chars;
}

bool validateText(const vector<string> &chars,
                  const unordered_map<string, int> &map) {
  for (auto &ch : chars) {
    if (!map.count(ch))
      return false;
  }
  return true;
}

string vigenere(const vector<string> &msg, const vector<string> &key,
                const unordered_map<string, int> &map, bool encrypt) {
  string result;
  for (size_t i = 0; i < msg.size(); i++) {
    int p = map.at(msg[i]);
    int k = map.at(key[i % key.size()]);
    int c = encrypt ? (p + k) % ALPHABET_SIZE
                    : (p - k + ALPHABET_SIZE) % ALPHABET_SIZE;
```

```
60      result += ALFABET[c];
61    }
62    return result;
63  }
64
65  int main() {
66    auto map = createCharToNum();
67
68    cout << "Alege operatia (1 - Criptare, 2 - Decriptare): ";
69    int opt;
70    cin >> opt;
71    cin.ignore();
72
73    cout << "Introdu cheia (minim 7 caractere): ";
74    string key;
75    getline(cin, key);
76    key = normalizeText(key);
77    auto keyChars = splitUTF8(key);
78
79    if (keyChars.size() < 7) {
80      cerr << "Eroare: cheia trebuie sa aiba cel putin 7 caractere.\n"
       ;
81      return 1;
82    }
83    if (!validateText(keyChars, map)) {
84      cerr << "Eroare: cheia contine caractere invalide.\n";
85      return 1;
86    }
87
88    cout << "Introdu mesajul: ";
89    string msg;
90    getline(cin, msg);
91    msg = normalizeText(msg);
92    auto msgChars = splitUTF8(msg);
93    if (!validateText(msgChars, map)) {
94      cerr << "Eroare: mesajul contine caractere invalide.\n";
95      return 1;
96    }
97
98    bool encrypt = (opt == 1);
99    string result = vigenere(msgChars, keyChars, map, encrypt);
100
101   if (encrypt)
102     cout << "Criptograma: " << result << "\n";
103   else
104     cout << "Mesaj decriptat: " << result << "\n";
```

```
105
106     return 0;
107 }
```

Listing 2: Complete C++ source code for Vigenère cipher implementation.