Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Cryptography and Security

Laboratory work #3

Variant 11

Polyalphabetic Ciphers

Author: Timur Cravtov
std. gr. FAF-231

Verified by: Maia Zaica
asist. univ

Chișinău
2025

Contents

1 Introduction

As shown in previous lab, monoalphabetic ciphers are vulnerable to frequency analysis attacks due to their one-to-one mapping of plaintext letters to ciphertext letters. To enhance security, polyalphabetic ciphers employ multiple substitution alphabets, making frequency analysis more challenging. This lab focuses on two well-known polyalphabetic ciphers: the Playfair cipher and the Vigenère cipher.

1.1 Playfair Cipher

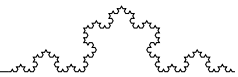
Playfair cipher is one of polyalphabetic ciphters [2] [4]. It encrypts pairs of letters (digraphs) instead of single letters, which helps to obscure letter frequency patterns. The Playfair cipher uses a matrix of letters generated from a keyword to perform substitutions based on the positions of letters in the matrix. In dependence of the language and the implementations there are difference in matrix: different number of rows and columns, merging letters (like I/J) etc. For example, this is usual 5x5 matrix for English language with I/J merged:

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

Figure 1: Example of Playfair cipher matrix with key "MONARCHY

The process of encoding the plaintext is the following, eg: MESSAGE

- Split the plaintext into digraphs. If a pair consists of the same letter (e.g., "LL"), insert a filler letter (commonly 'X') between them. If there's an odd letter out, append a filler letter at the end. Example: ME SS AG E -¿ ME SX SA GE
- Then, we apply the following rules for each digraph:



- If both letters are in the same row of the matrix, replace them with the letters to their immediate right (wrap around to the beginning of the row if necessary).
- If both letters are in the same column, replace them with the letters immediately below them (wrap around to the top if necessary).
- If the letters form a rectangle, replace them with the letters on the same row but at the opposite corners of the rectangle.

1.2 Vigenère Cipher

Viginere is an improvement over Caesar cipher [4] [3], which is a monoalphabetic substitution cipher. Vigenère cipher uses a keyword to determine the shift for each letter in the plaintext, effectively creating multiple Caesar ciphers based on the letters of the keyword. This polyalphabetic approach makes it more resistant to frequency analysis attacks.

The encryption process of the Vigenère cipher involves the following steps:

- Choose a keyword and repeat it to match the length of the plaintext. For example, if the plaintext is "ATTACKATDAWN" and the keyword is "LEMON", the repeated keyword would be "LEMONLEMONLE".
- For each letter in the plaintext, determine its position in the alphabet (A=0, B=1, ..., Z=25) and do the same for the corresponding letter in the repeated keyword.
- Shift the plaintext letter by the position value of the keyword letter. If the sum exceeds 25, wrap around using modulo 26.
- Convert the resulting position back to a letter to get the ciphertext.

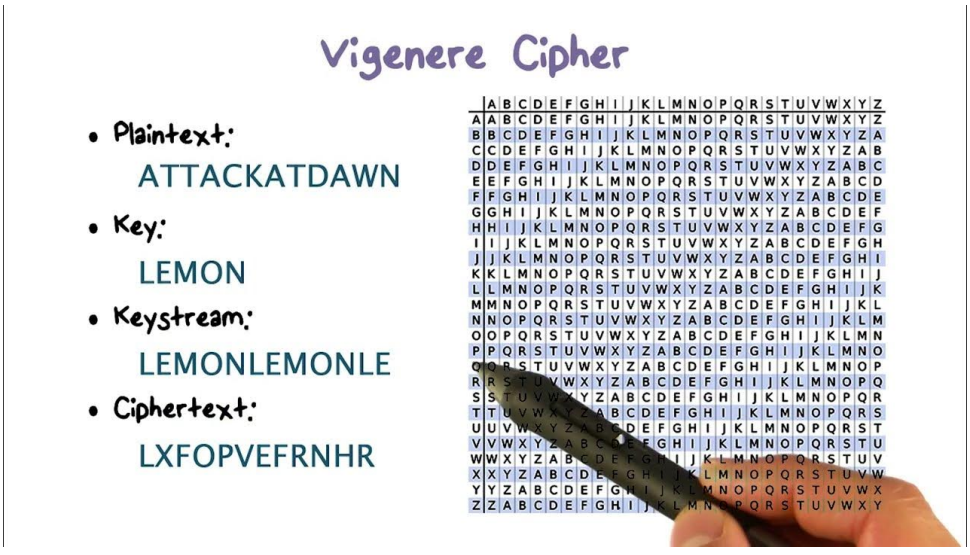


Figure 2: Example of Vigenère cipher encryption with key "LEMON"

2 Implementation of Playfair Cipher

The Playfair cipher was implemented in Kotlin. The source code can be found in GitHub reporitory [1].

2.1 Matrix Generation

For matrix generation, we take the key, append alphabet and get distinct list and populate the matrix(which dimention we take from arguments) row by row.

```
internal fun generateSquare(alphabet: String, key: String, rows:
    Int = 5, columns: Int = 5): Array<Array<Char>> {

    if (!key.all { it in alphabet }) throw
        IllegalArgumentException("All the letters in the key
        needs to be in the alphabet")
    if (alphabet.length != rows * columns) throw
        IllegalArgumentException("The alphabet length should
        equal the rows * columns")

    val combined = (key + alphabet).toList().distinct()

    return combined.chunked(columns)
        .map { chunk -> chunk.toTypedArray() }
        .toTypedArray()

}
```

Listing 1: Generate Matrix

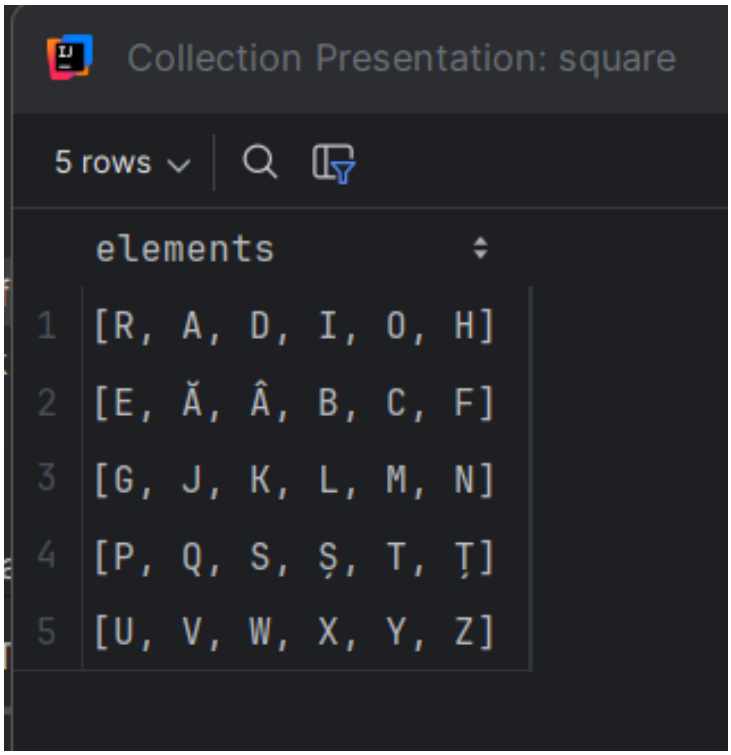


Figure 3: Example of Matrix generation with key "RADIOHEAD"

2.2 Encryption

The listing belows shows the process of encryption of plaintext. The step are the following:

- Preprocess the plaintext into digraphs, inserting 'X' as needed.
- For each digraph, apply the Playfair cipher rules based on their positions in the matrix.
- Collect the resulting ciphertext digraphs into a single string.

2.3 Decryption

The decryption process is similar to encryption, but the rules are applied in reverse. The listing below shows the decryption function.

```
1 fun String.decryptPlayfair(alphabet: String, key: String, rows:
    Int, columns: Int, removeFillers: Boolean = true): String {
2
3     val square = generateSquare(alphabet, key, rows, columns)
4
5     // split message into pairs
```

```
6       val chunks = this.chunked(2).map { it.toMutableList() }
7
8       for (chunk in chunks) {
9
10          val coords1 = square.coordinates(chunk[0])
11          val coords2 = square.coordinates(chunk[1])
12
13          // same row -> move left
14          if (coords1.first == coords2.first) {
15              chunk[0] = square[coords1.first][square.prevColumn(
                    coords1.second)]
16              chunk[1] = square[coords2.first][square.prevColumn(
                    coords2.second)]
17
18              // same column -> move up
19          } else if (coords1.second == coords2.second) {
20              chunk[0] = square[square.prevRow(coords1.first)][
                    coords1.second]
21              chunk[1] = square[square.prevRow(coords2.first)][
                    coords2.second]
22
23              // rectangle -> swap columns
24          } else {
25              chunk[0] = square[coords1.first][coords2.second]
26              chunk[1] = square[coords2.first][coords1.second]
27          }
28      }
29
30      var result = chunks.flatten().joinToString("")
31      if (removeFillers) result = result.removePlayfairFillers()
32      return result;
33  }
```

Listing 2: Decryption of Playfair

As one can notice, the final step is removing the filter letters. If I sees a filter letter between two same letters, I remove it. Unfortunately, if the filter letter is at the end of the text, there is no way to understand whether it is a filter letter or a part of the original message, so it's kept.

2.4 Execution

For this report, a symmetry test was created. The program takes plaintext, key, number of rows and columns as input, encrypts the plaintext, decrypts the resulting ciphertext, and then checks if the decrypted text matches the original plaintext. If they match, it confirms that the encryption and decryption processes are functioning correctly.

```
10:50:05: Executing ':lab3.SymmetryTestKt.main()'…

Introduceți cheia: radiohead
Introduceți mesajul: super secret message
Mesajul encriptat: ȘSMÂHVHFAHȘNHȘȘRFĂ
Mesajul decryptat: SUPERSECRETMESSAGE
10:50:27: Execution finished ':lab3.SymmetryTestKt.main()'.
```

Figure 4: Example of symmetry test execution

As one can notice, the decrypted text matches the original plaintext, confirming the correctness of the Playfair cipher implementation, including removing the filter letters.

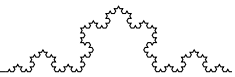
3 Rezults

During this laboratory work, the Playfair cipher was implemented in Kotlin. The implementation includes functions for generating the cipher matrix, encrypting plaintext into ciphertext, and decrypting ciphertext back into plaintext. A symmetry test was conducted to verify the correctness of the implementation, confirming that the encryption and decryption processes are functioning as intended.

The Playfair cipher enhances security over monoalphabetic ciphers by encrypting digraphs and utilizing a matrix based on a keyword. However, it is still vulnerable to frequency analysis attacks, especially with longer texts. Future work could involve implementing additional polyalphabetic ciphers or exploring methods to further obscure letter frequency patterns.

References

[1] GitHub repository `https://github.com/TimurCravtov/CryptographyAndSecurityLabs`



[2] GeeksForGeeks https://www.geeksforgeeks.org/dsa/playfair-cipher-with-examples/

[3] TutorialPoint https://www.tutorialspoint.com/cryptography/cryptography_vigenere_cipher.htm

[4] Lecture Notes CS