# Intro to Cryptography. Classical ciphers. Caesar cipher.

Forors Valentin

September 29, 2022

## 1 Introduction

Cryptography consists a part of the science known as Cryptology. The other part is Cryptanalysis. There are a lot of different algorithms/mechanisms used in Cryptography, but in the scope of these laboratory works the students need to get familiar with some examples of each kind.

## 2 Objectives

1. Get familiar with the basics of cryptography and classical ciphers.
2. Implement 4 types of the classical ciphers:
   - Caesar Cipher
   - Atbash Cipher
   - Veginere Cipher
   - PlayFair Cipher
3. Structure the project in methods/classes/packages as needed.

## 3 Caesar Cipher

### 3.1 Theory

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus to cipher a given text we need an integer value, known as a shift which indicates the number of positions each letter of the text has been moved down. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,. . . , Z = 25. Encryption of a letter by a shift n can be described mathematically as.

$E_n(x) = (x + n) mod 26$

$D_n(x) = (x - n) mod 26$

### 3.2 Code Snippets

```
public StringBuffer encrypt(String text, int shift)
{
    StringBuffer result= new StringBuffer();

    for (int i=0; i<text.length(); i++)
    {
```

```
            if (Character.isUpperCase(text.charAt(i)))
            {
                char ch = (char)(((int)text.charAt(i) +
                        shift - 65) % 26 + 65);
                result.append(ch);
            }
            else
            {
                char ch = (char)(((int)text.charAt(i) +
                        shift - 97) % 26 + 97);
                result.append(ch);
            }
        }
        return result;
    }
```

# 4 Atbash Cipher

## 4.1 Theory

Definition: Atbash cipher is a substitution cipher with just one specific key where all the letters are reversed that is A to Z and Z to A. It was originally used to encode the Hebrew alphabets but it can be modified to encode any alphabet.

## 4.2 Code Snippets

```
public static String encrypt(String message )
{
    StringBuilder newMessage = new StringBuilder();
    for (char c : message.toCharArray()) {
        if (Character.isLetter(c)) {
            int newChar = ('Z' - c) + 'A';
            newMessage.append((char) newChar);
        } else {
            newMessage.append(c);
        }
    }
    return newMessage.toString();
}
```

# 5 Veginere Cipher

## 5.1 Theory

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.

1. The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.

2. At different points in the encryption process, the cipher uses a different alphabet from one of the rows.

3. The alphabet used at each point depends on a repeating keyword.

### 5.2 Code Snippets

```
static String generateKey(String str, String key)
{
    int x = str.length();

    for (int i = 0; ; i++)
    {
        if (x == i)
            i = 0;
        if (key.length() == str.length())
            break;
        key+=(key.charAt(i));
    }
    return key;
}

static String cipherText(String str, String key)
{
    String cipher_text="";

    for (int i = 0; i < str.length(); i++)
    {
        // converting in range 0-25
        int x = (str.charAt(i) + key.charAt(i)) %26;

        // convert into alphabets(ASCII)
        x += 'A';

        cipher_text+=(char)(x);
    }
    return cipher_text;
}
```

# 6 FairPlay Cipher

## 6.1 Theory

The Playfair cipher is one of the traditional ciphers which comes under the category of substitution ciphers. In Playfair Cipher, unlike traditional cipher, we encrypt a pair of alphabets (digraphs) instead of a single alphabet. In Playfair cipher, initially, a key table is created. The key table is a 5×5 matrix consisting of alphabets that acts as the key for encryption of the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually 'j') is omitted from the table, as we need only 25 alphabets instead of 26. If the plaintext contains 'j', then it is replaced by 'i'.

1. The plaintext message is split into pairs of two letters (digraphs). If the plaintext has an odd number of characters, append 'z' to the end to make the message of even length.
2. Identify any double letters placed side by side in the plaintext and replace the second occurrence with an 'x' e.g. 'hello' -¿ 'he lx lo'.
3. Now, locate the letters in the 5×5 key table.
4. Use the following rules for encryption of plaintext:
    - If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).

- If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column)
- If the letters are not on the same row or column, replace them with the letters in their own row but in the same column as the other letter.

## 6.2 Code Snippets

```
public void Playfair(String key, String plainText)
{
    // convert all the characters to lowercase
    this.key = key.toLowerCase();

    this.plainText = plainText.toLowerCase();
}

Eliminate duplicates from key
public void cleanPlayFairKey()
{
    LinkedHashSet<Character> set
            = new LinkedHashSet<Character>();

    String newKey = "";

    for (int i = 0; i < key.length(); i++)
        set.add(key.charAt(i));

    Iterator<Character> it = set.iterator();

    while (it.hasNext())
        newKey += (Character)it.next();

    key = newKey;
}

Generate table
public void generateCipherKey()
{
    Set<Character> set = new HashSet<Character>();

    for (int i = 0; i < key.length(); i++)
    {
        if (key.charAt(i) == 'j')
            continue;
        set.add(key.charAt(i));
    }

    // remove repeated characters from the cipher key
    String tempKey = new String(key);

    for (int i = 0; i < 26; i++)
    {
        char ch = (char)(i + 97);
        if (ch == 'j')
            continue;
```

```
            if (!set.contains(ch))
                tempKey += ch;
        }

        // create cipher key table
        for (int i = 0, idx = 0; i < 5; i++)
            for (int j = 0; j < 5; j++)
                matrix[i][j] = tempKey.charAt(idx++);

        System.out.println("Playfair Cipher Key Matrix:");

        for (int i = 0; i < 5; i++)
            System.out.println(Arrays.toString(matrix[i]));
    }

    public String formatPlainText()
    {
        String message = "";
        int len = plainText.length();

        for (int i = 0; i < len; i++)
        {
            // if plaintext contains the character 'j',
            // replace it with 'i'
            if (plainText.charAt(i) == 'j')
                message += 'i';
            else
                message += plainText.charAt(i);
        }

        // if two consecutive characters are same, then
        // insert character 'x' in between them
        for (int i = 0; i < message.length(); i += 2)
        {
            if (message.charAt(i) == message.charAt(i + 1))
                message = message.substring(0, i + 1) + 'x'
                        + message.substring(i + 1);
        }

        // make the plaintext of even length
        if (len % 2 == 1)
            message += 'x'; // dummy character

        return message;
    }
```