Caesar cipher

Software Documentation

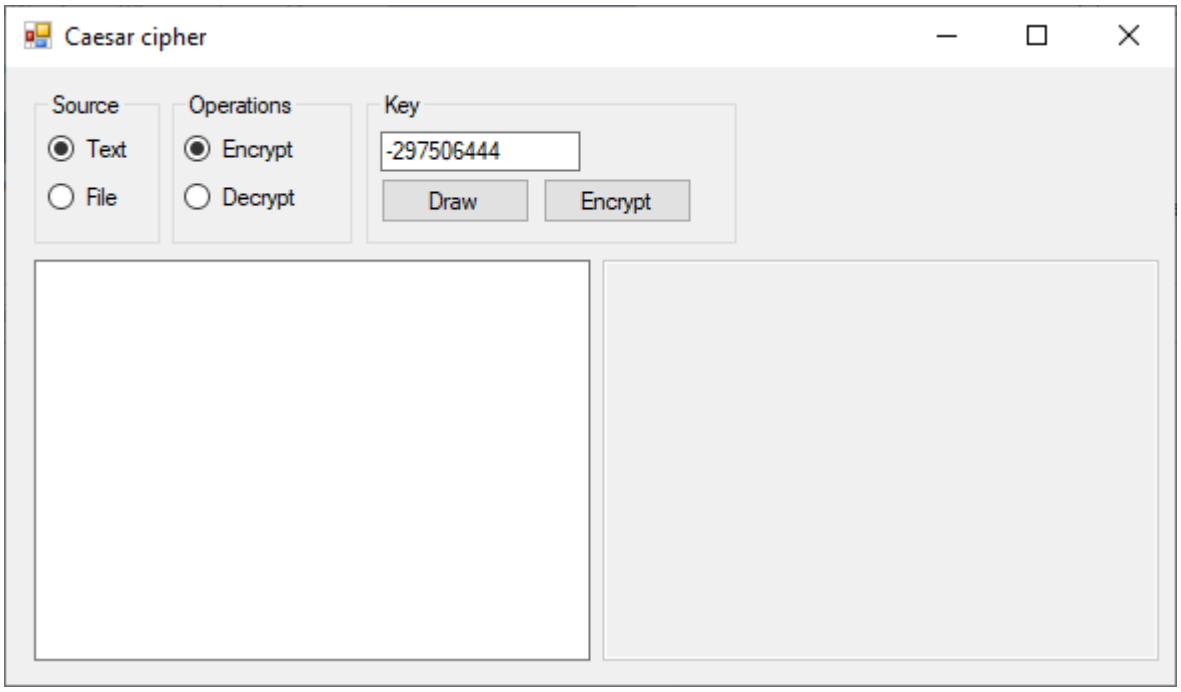
Author: matiwa

Table of contents

Introduction

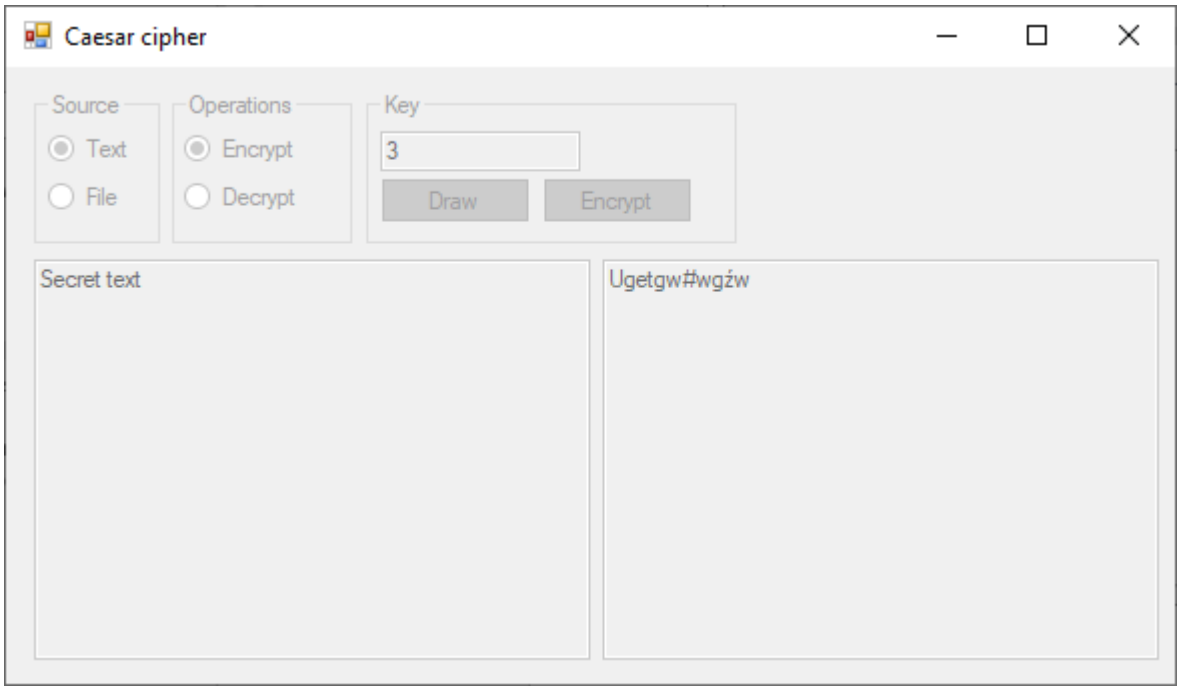
This software documentation includes: description of the application's operation, what is needed for use, algorithms used, interface description and source code description. This application is used to encoding and decoding Caesar cipher.

Describing of the application’s operation

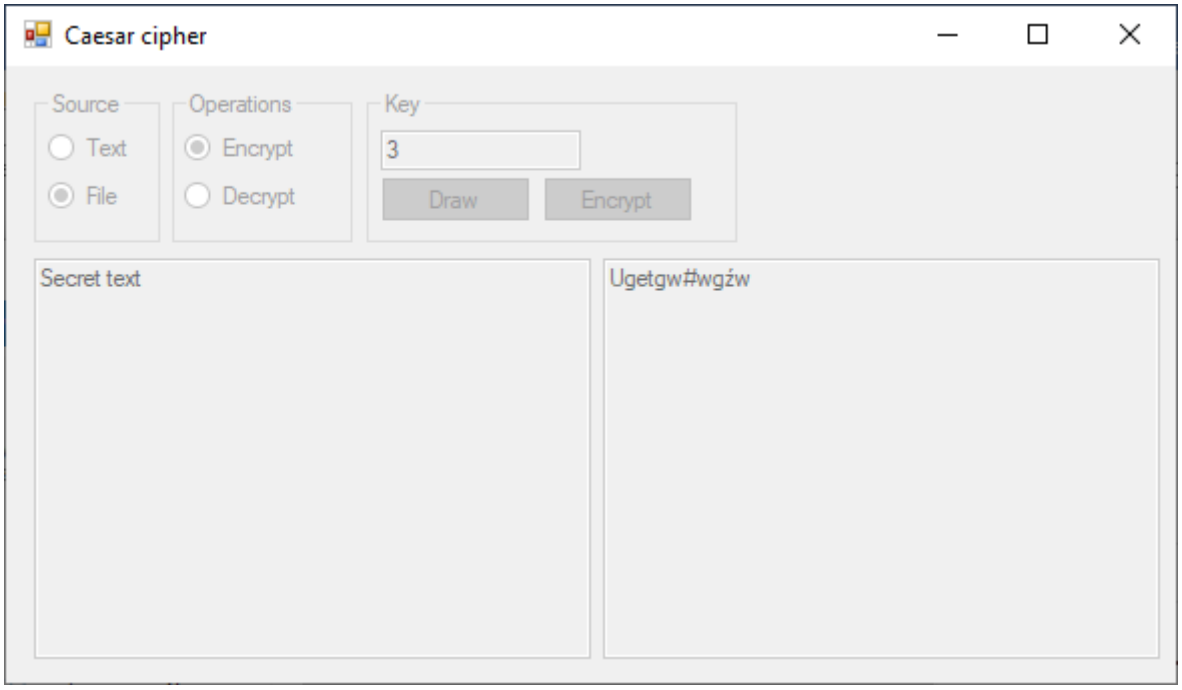


Drawing 1: The beginning of the application's operation [own study]

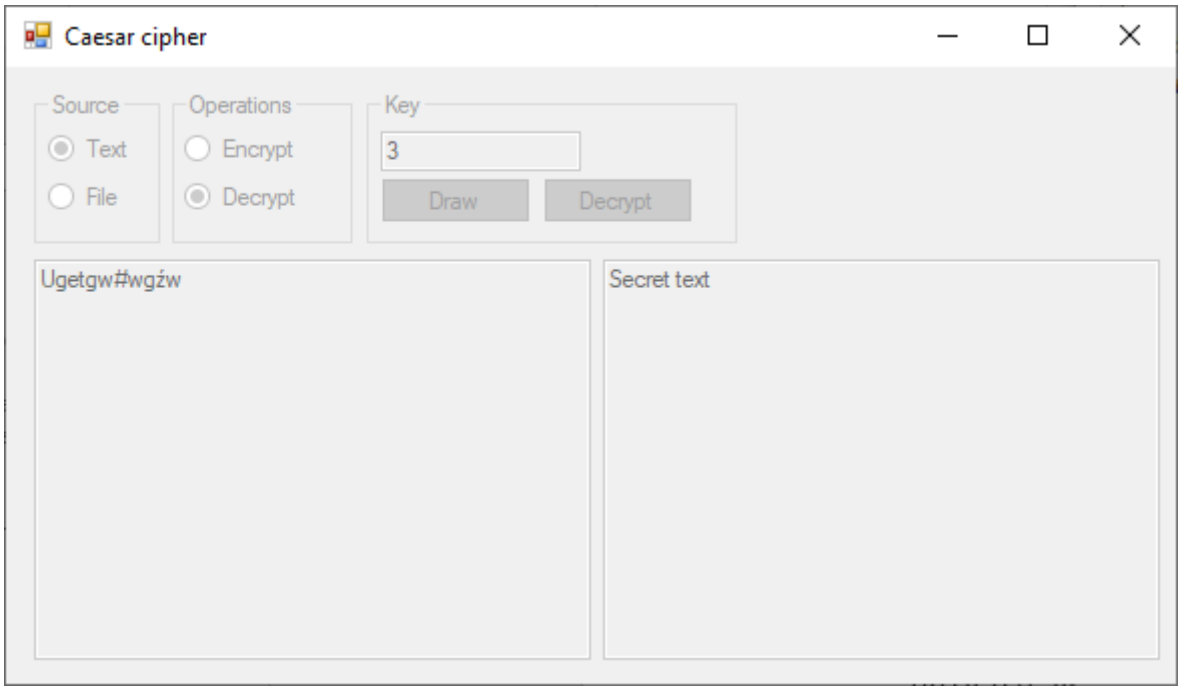
The user has to select the source of the text and the operation to be performed. If he selects a file, the text box is inactive. If it selects text, the text box is active. When selecting a text source, it is saved to the file as a backup. Selecting a file means that the text will be loaded and printed in the text field. By default, the application generates a random key that can be changed by the user.



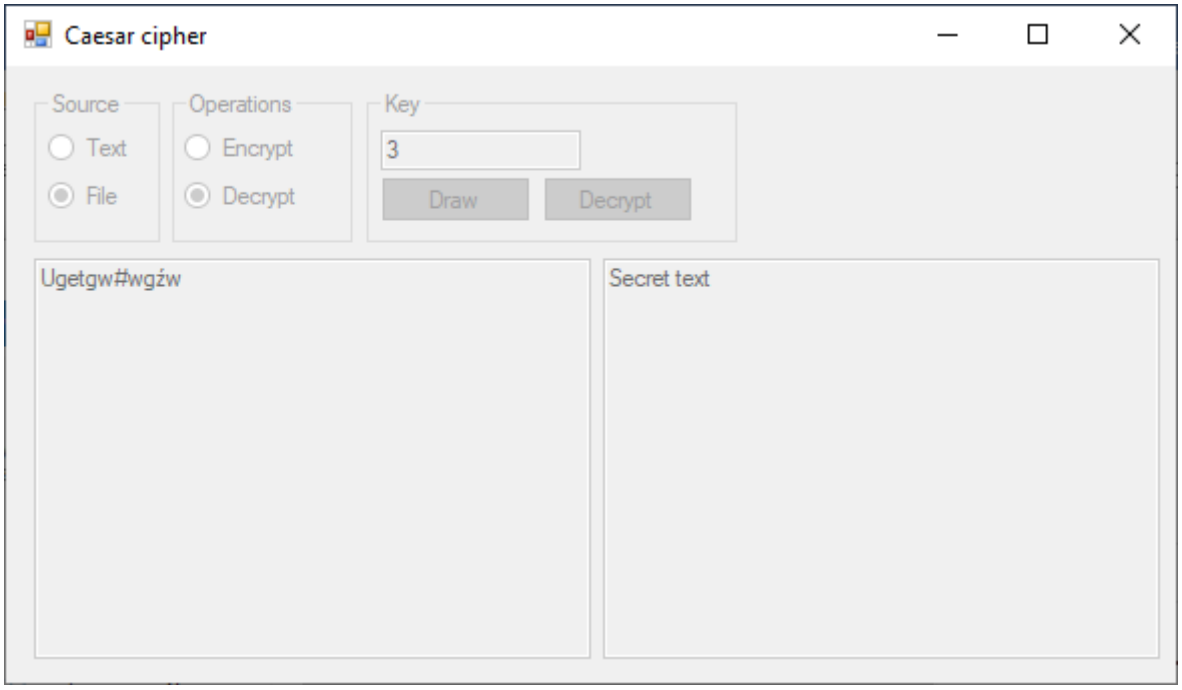
Drawing 2: Text coding example [own study]



Drawing 3: An encoding example from a file [own study]



Drawing 4: Text decoding example [own study]



Drawing 5: An encoding example from a file [own study]

This is the correct use of the program. However, the user may make a mistake by entering a number with a comma or period. The program has no security, so its operation will be suspended.

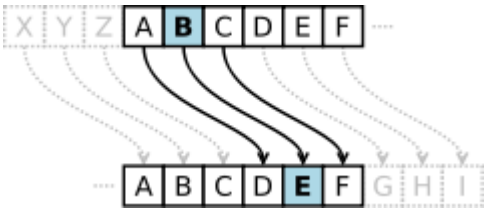
What is needed for use?

The application does not require installation. It only needs the Windows operating system.

Algorithm used

Caesar cipher (also known as shift cipher, Caesar cipher or Caesar cipher) is one of the simplest ciphers to encrypt. It is a type of signature in which each plaintext letter (unencrypted) is replaced with a different one, separated from it with a system signature in the alphabet, a monoalphabetic cipher), it must be preserved when changing direction. There is no difference between large and small liters. The name of the cipher comes from Julius Caesar, who used this technique to communicate with its routes.

The palace system's algorithm in Caesarv's code is a fragment of a more complex system, such as the Vi cipher machine. Nowadays, Caesar's guns are used with fast 13 (ROT13), simple mode and m to hide the content. At present, Caesar, like any other technique of elevating the name of a letter of the alphabet to another, has no security title whatsoever.[1]



Drawing 6: Cipher with an offset of 3 [2]

The Caesar cipher replaces each plaintext letter with a different one, shifted from the coded letter by a fixed number of positions in the alphabet. In the figure above, the cipher with an offset equal to 3, so B in the plaintext is replaced by E in the ciphertext (the Latin alphabet is considered).

Example

The encryption method can be represented by a diagram of two strings with corresponding letters of the alphabet. The same letters of the second string are shifted relative to the first string by a certain number of positions, called the shift parameter (here 2) and serving as the cipher key:

Alphabet: AĄBCĆDEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻ

Cipher: CĆDEĘFGHIJKLMNŃOÓPRSŚTUWYŻŻAĄB

Note that the last letters of the alphabet in the upper string correspond to the initial letters in the lower string (the alphabet has been "wrapped"). To encrypt a message, each letter should be replaced with an equivalent from the cipher (the message in the example is written in capital letters, although the cipher is case-insensitive):

Clear text: MAN BE MAN, PROTECT YOUR REGION AND THE SIX FLAGS

Ciphertext: OHBÓŻ DĆFĄ, EKTRP ŚZŃM YŹSŁ L ATTENTION INCJ

Decryption is about reversing this operation.

Mathematical approach

The operation of encryption and decryption can be expressed in the language of modular arithmetic. For this purpose, it is enough to unambiguously assign each letter of the alphabet a number to it according to the scheme: A↔0, Ą↔1, B↔2,…, Ż↔31. It is also convenient to assume that the key n is some number in the range 0 ... 31 (it is the number of the encrypted letter A).

The encryption can then be defined by the congruence:

$$E_n(x) \equiv x + n \pmod{32},$$

where x is the plaintext letter number in the alphabet, $E_n(x)$ - the ciphertext letter number in the alphabet.

Similarly, text decryption can be written as:

$$D_n(y) \equiv y - n \pmod{32}$$

where y is the letter number of the ciphertext in the alphabet, $D_n(y)$ - the plaintext letter number in the alphabet.

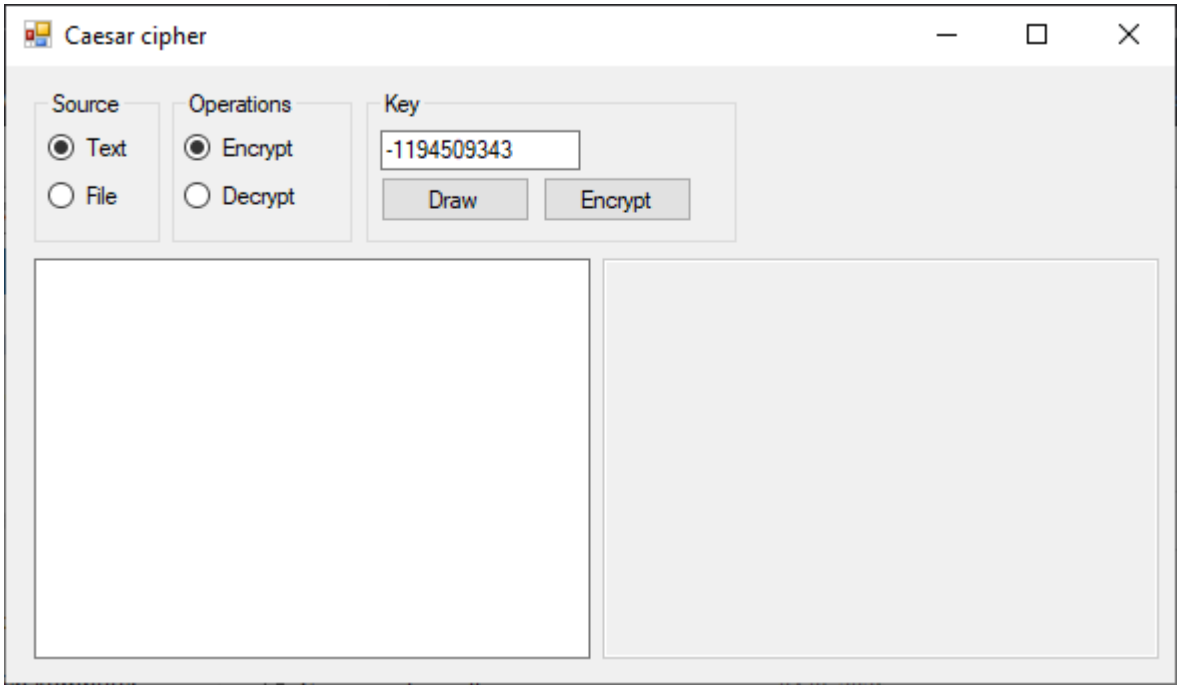
Based on the congruence property and the fact that $E_n(x)$, $D_n(y)$ are in the range 0 ... 31:

- if when determining E_n (x) the value of the expression x + n exceeds 32 - 1, then it should be reduced at 32.
- if in determining D_n (x) the value of the expression y - n is negative, it should be increased by 32.

Operations E_n (x), D_n (y) are opposite to each other, because moving to the right about k is also moving to the left 32 - k.

The number 32 above is the number of letters in the Polish alphabet. For the Latin alphabet, the number should be taken as 26.[1]

Interface description



Drawing 7: Graphical interface [own study]

The interface is typical for a windowing application. There are essential components: buttons, text boxes, and radio buttons. The Draw button randomizes the number that is the key and is entered into the text box. The second button starts the encryption and decryption procedure according to the option that the user selects from the radio buttons. The text field for the message is active when selecting the text option button, and inactive for the file option.

Source code description

The project was made in the C# programming language, in the Visual Studio Community 2019 programming environment. All work was done on the Windows 10 operating system. The application's source code looks like this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Caesar_cipher
{
    public partial class Form1 : Form
    {
        FileStream file = new FileStream("message.txt",
FileMode.OpenOrCreate, FileAccess.ReadWrite);
        char[] chars = new char[]{'
','!','"','#','$','%','&','\'','(',')','*','+',',','-','.','/','0',

'1','2','3','4','5','6','7','8','9',':',';','<','=','>','?','@','A',
'Ą','B','C','Ć','D','E',

'Ę','F','G','H','I','J','K','L','Ł','M','N','Ń','O','Ó','P','Q','R',
'S','Ś','T','U','V','W',

'X','Y','Z','Ź','Ż','[','/',']','^','_','`','a','ą','b','c','ć','d',
'e','ę','f','g','h','i',

'j','k','l','ł','m','n','ń','o','ó','p','q','r','s','ś','t','u','v',
'w','x','y','z','ź','ż',
            '{','|','}','~'}; //ASCII code combined with Polish
diacritized letters
        int key;
        string message = "";
        char[] messagechar;
        char[] result;

        public Form1()
        {
            InitializeComponent();
            //draw a key
            Random r = new Random();
            tBKey.Text = Convert.ToString(r.Next(int.MinValue,
int.MaxValue));
        }

        private void Btngo_Click(object sender, EventArgs e)
        {
```

```
            //the key cannot be empty!!!
            if (tBKey.Text == "") tBKey.Text = "0";
            key = Convert.ToInt32(tBKey.Text);
            //transforming the key value to range from 0 to the
number of chars minus 1
            while (key<0 || key > chars.Length - 1)
            {
                if (key < 0) key += chars.Length;
                else if (key > chars.Length - 1) key %=
chars.Length;
            }
            //loading the message from the text field and saving it
to a file
            if (rBText.Checked == true)
            {
                message = tBText.Text;
                StreamWriter sw = new StreamWriter(file);
                sw.WriteLine(message);
                sw.Close();
            }
            //loading a message from a file and saving it to the
text field
            else if (rBFile.Checked == true)
            {
                StreamReader sr = new StreamReader(file);
                message = sr.ReadLine();
                sr.Close();
                tBText.Text = message;
            }
            //the beginning of the algorithm
            messagechar = message.ToCharArray();
            result = new char[messagechar.Length];
            for (int j = 0; j < messagechar.Length; j++)
            {
                for (int i = 0; i < chars.Length; i++)
                {
                    if (messagechar[j] == chars[i])
                    {
                        int x = i;
                        int c;
                        if (rBE.Checked == true) //for encryption
                        {
                            //securing the key on a chars array
                            if (((x + key) >= 0) & ((x + key) <=
(chars.Length - 1)))
                            {
                                c = x + key;
                                result[j] = chars[c];
                            }
```

```
                            else if ((x + key) < 0)
                            {
                                c = x + key + chars.Length;
                                result[j] = chars[c];
                            }
                            else if ((x + key) > (chars.Length - 1))
                            {
                                c = x + key - chars.Length;
                                result[j] = chars[c];
                            }
                        }else if (rBD.Checked == true) //for
decryption
                        {
                            //securing the key on a chars array
                            if (((x - key) >= 0) & ((x - key) <=
(chars.Length - 1)))
                            {
                                c = x - key;
                                result[j] = chars[c];
                            }
                            else if ((x - key) < 0)
                            {
                                c = x - key + chars.Length;
                                result[j] = chars[c];
                            }
                            else if ((x - key) > (chars.Length - 1))
                            {
                                c = x - key - chars.Length;
                                result[j] = chars[c];
                            }
                        }
                    }
                }
            }
            //message encryption or decryption
            for (int i = 0; i < messagechar.Length; i++)
            {
                tBMessage.Text += Convert.ToString(result[i]);
            }
            //protection against the next operation
            gBOperation.Enabled = false;
            gBSource.Enabled = false;
            gBKey.Enabled = false;
            tBText.Enabled = false;
        }

        //draw a key
        private void Btndraw_Click(object sender, EventArgs e)
        {
```

```
            Random r = new Random();
            tBKey.Text = Convert.ToString(r.Next(int.MinValue,
int.MaxValue));
        }

        ////select a message input method
        private void SelectSource(object sender, EventArgs e)
        {
            if (rBText.Checked == true) tBText.Enabled = true;
            else if (rBFile.Checked == true) tBText.Enabled = false;
        }

        //display the selected operation as the button text
        private void SelectOperation(object sender, EventArgs e)
        {
            if (rBE.Checked == true) btngo.Text = rBE.Text;
            else if (rBD.Checked == true) btngo.Text = rBD.Text;
        }
    }
}
```

Listing 1: Source code [own study]

List of drawings

List of listings

Bibliography

[1] https://pl.wikipedia.org/wiki/Szyfr_Cezara
[2] https://upload.wikimedia.org/wikipedia/commons/thumb/2/2b/Caesar3.svg/240px-Caesar3.svg.png