A fence 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 a fence cipher.

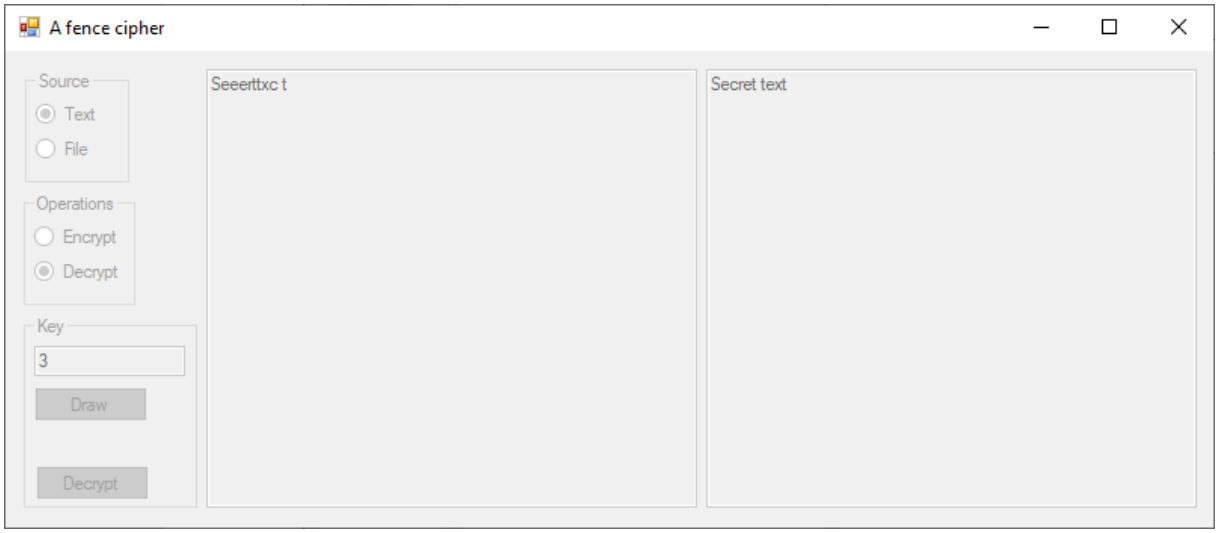
Describing of the application’s operation

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

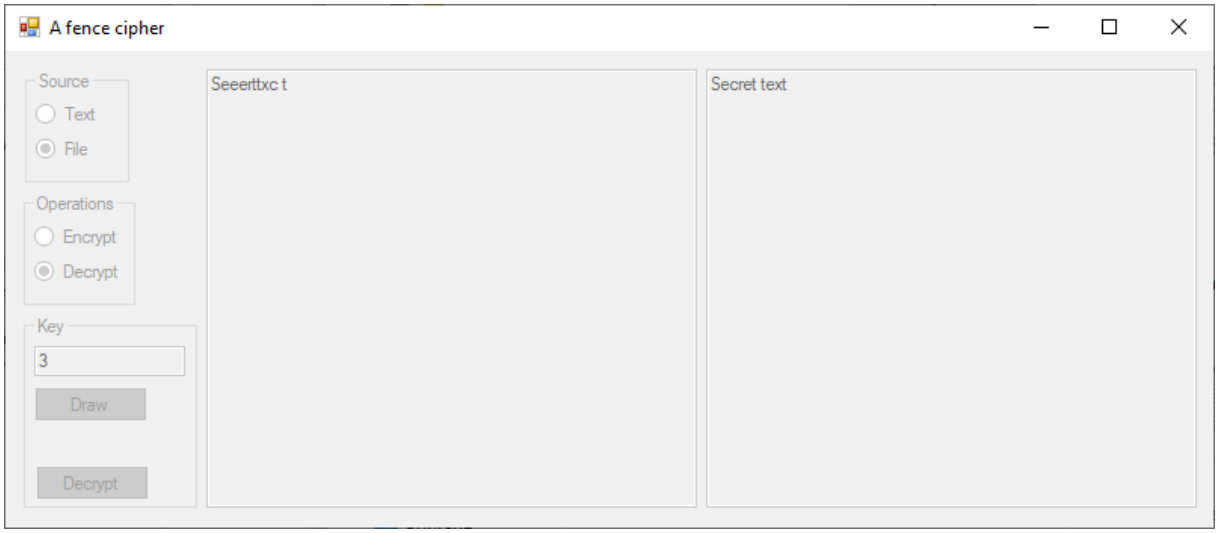
The user has to select the source of the text and the operation to be performed. If he selects a file, the text box is inactive. If it selects text, the text box is active. When selecting a text source, it is saved to the file as a backup. Selecting a file means that the text will be loaded and printed 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

A fence cipher - a transposition cipher, described by a certain number n, denoting the "height" of the created steps - successive letters of the plaintext are written alternately in n rows. A cryptogram is made by joining a sequence of consecutive letters of the first row followed by a second row, etc. [1]

Example:

Plain Text = "CRYPTOGRAPHY"

Key = "4" (fence height)

The form of the plaintext in the shape of a hurdle with a height of 4:

C*****G*****
*R***O*R***Y
Y*T*A*H*
P**P**

Cryptogram = „CGRORYYTAHPP” [1]

Interface description

Drawing 6: 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 A_fence_cipher
{
    public partial class Form1 : Form
    {
        FileStream file = new FileStream("message.txt",
FileMode.OpenOrCreate, FileAccess.ReadWrite);
        int key;
        string message = "";
        char[] messagechar;

        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);
            //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();
            //message encryption
            if (rBE.Checked == true)
            {
                char[,] result = new char[key,
messagechar.Length];//cryptogram
                int row = 0;//row number
                int d = 1;//direction
                //creating a hurdle
                for (int j = 0; j < messagechar.Length; j++)
                {
                    result[row, j] = messagechar[j];
                    if (row < key - 1 && d == 1) row++;
                    else if (row == key - 1)
                    {
                        row--;
                        d = -1;//small
                    }
                    else if (row > 0 && d == -1) row--;
                    else if (row == 0)
                    {
                        row++;
                        d = 1;//grow
                    }
                }
                //writing a ciphertext
                for (int i = 0; i < key; i++)
                    for(int j = 0; j < messagechar.Length; j++)
```

```
                    tBMessage.Text += result[i, j].ToString();
            }
            //message decryption
            else if (rBD.Checked == true)
            {
                char[,] result = new char[key,
messagechar.Length];//cryptogram
                int d = 1;//direction
                int row = 0, col = 0;
                int cursor = 0;

                //creating a hurdle with marker
                for (int i = 0; i < messagechar.Length; i++)
                {
                    if (row == 0) d = 1;//grow
                    if (row == key - 1) d = -1;//small
                    //marker to mark the place
                    result[row,col++] = '*';

                    if (d == 1) row++;
                    else row--;
                }
                //entering a character from the ciphertext to the
fence
                for (int i = 0; i < key; i++) {
                    for (int j = 0; j < messagechar.Length; j++) {
                        if (result[i,j] == '*' && cursor <
messagechar.Length)
                            result[i,j] = messagechar[cursor++];
                    }
                }

                //writing a message
                row = 0;
                col = 0;
                for (int i = 0; i < messagechar.Length; i++)
                {
                    if (row == 0) d = 1;
                    if (row == key - 1) d = -1;

                    if (result[row,col] != '*')
tBMessage.Text+=result[row,col++].ToString();

                    if (d == 1) row++;
                    else row--;
                }
            }
            //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_p%C5%82otkowy