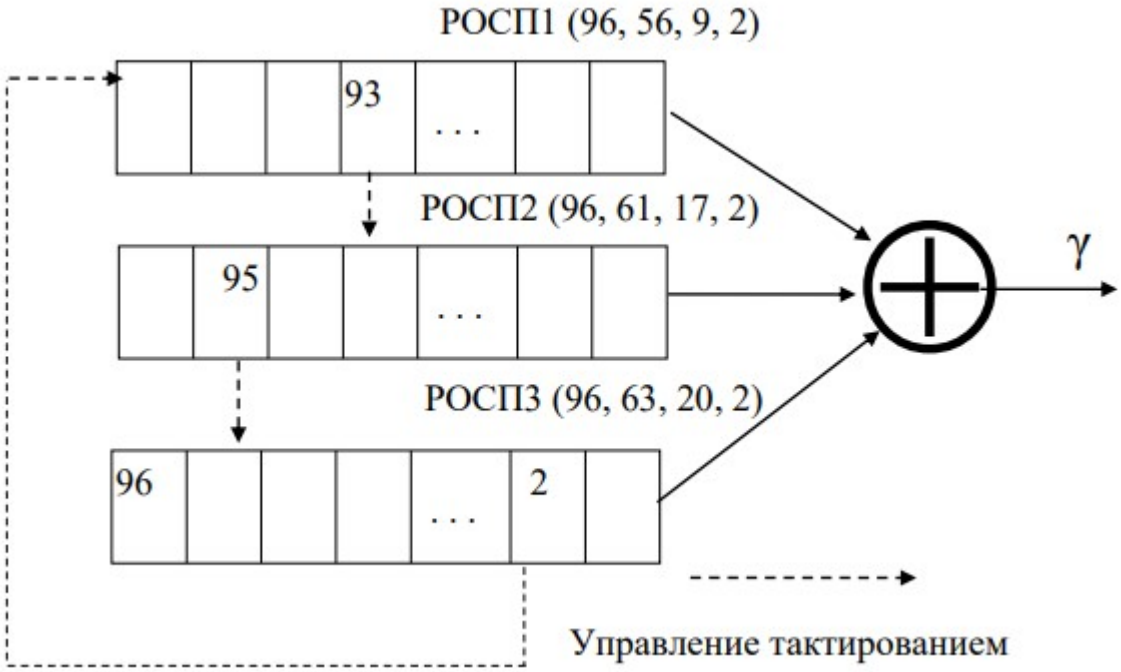
Цель работы

Изучение структуры и основных принципов работы современных алгоритмов поточного симметричного шифрования, приобретение
навыков программной реализации поточных симметричных шифров.

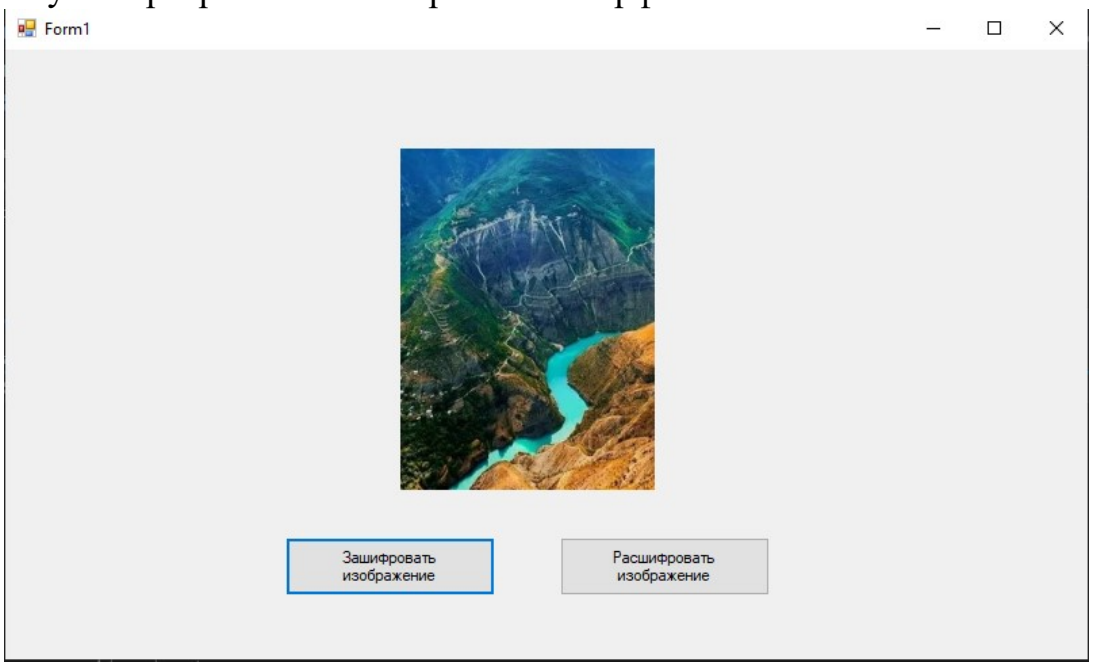
Задание

Использовать схему с управляемым тактированием на основе 3 РОСП. Управление тактированием: если бит=0, то управляемый ЛРС сдвигается вправо, если 1 – нет.

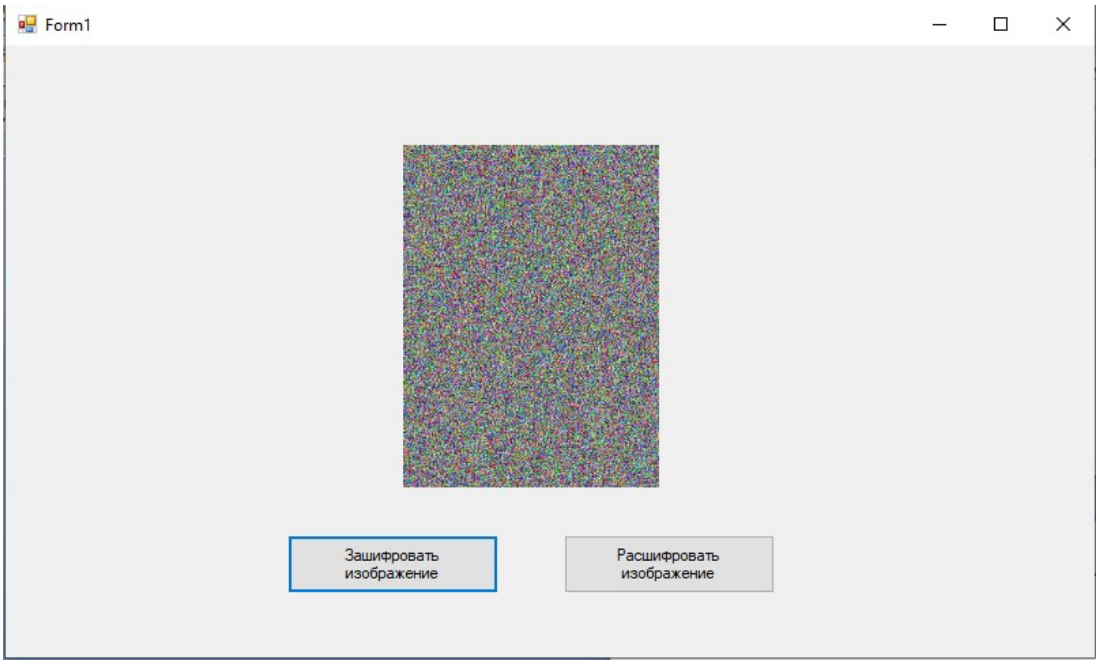


Результат работы программы

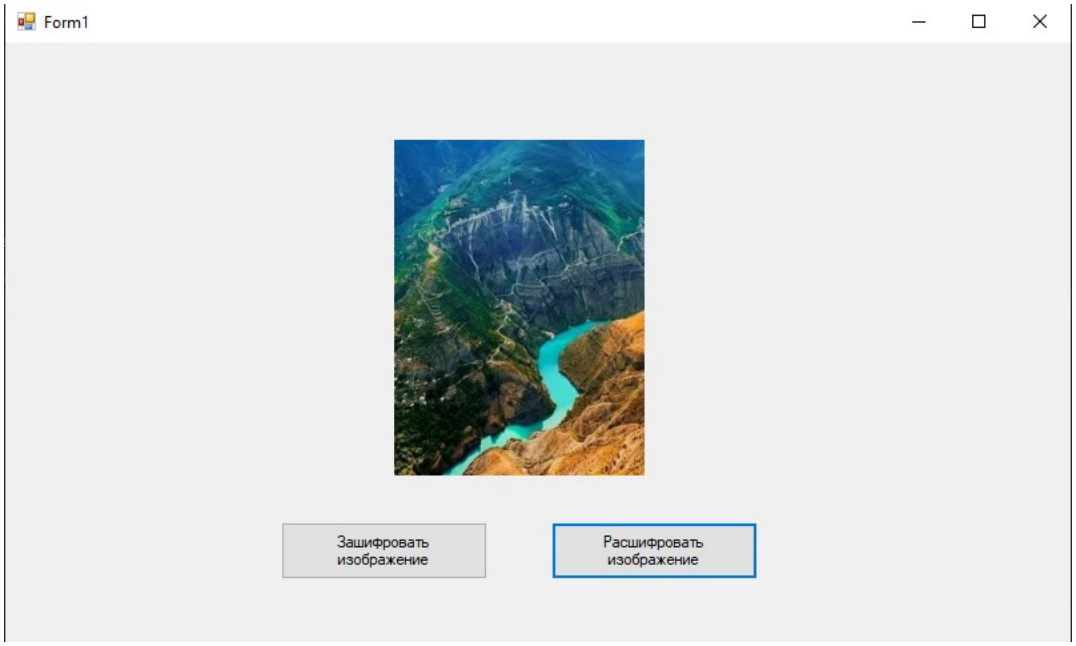
При запуске программы нас встречает интерфейс:



По центру программы находится файл image.txt. Для демонстрации работы, построенного по заданной схеме РОСП, алгоритма шифрования, нажмем на кнопку «Зашифровать изображение». Ниже представлен результат шифрования.

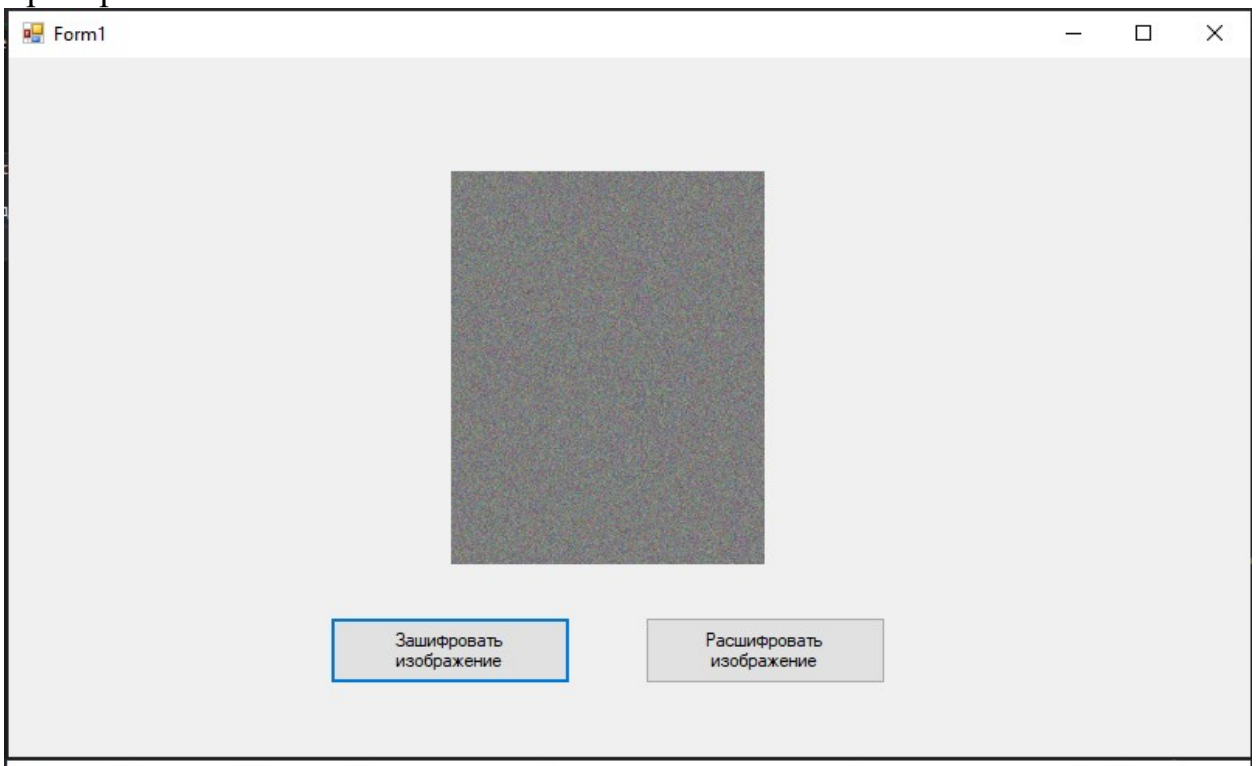


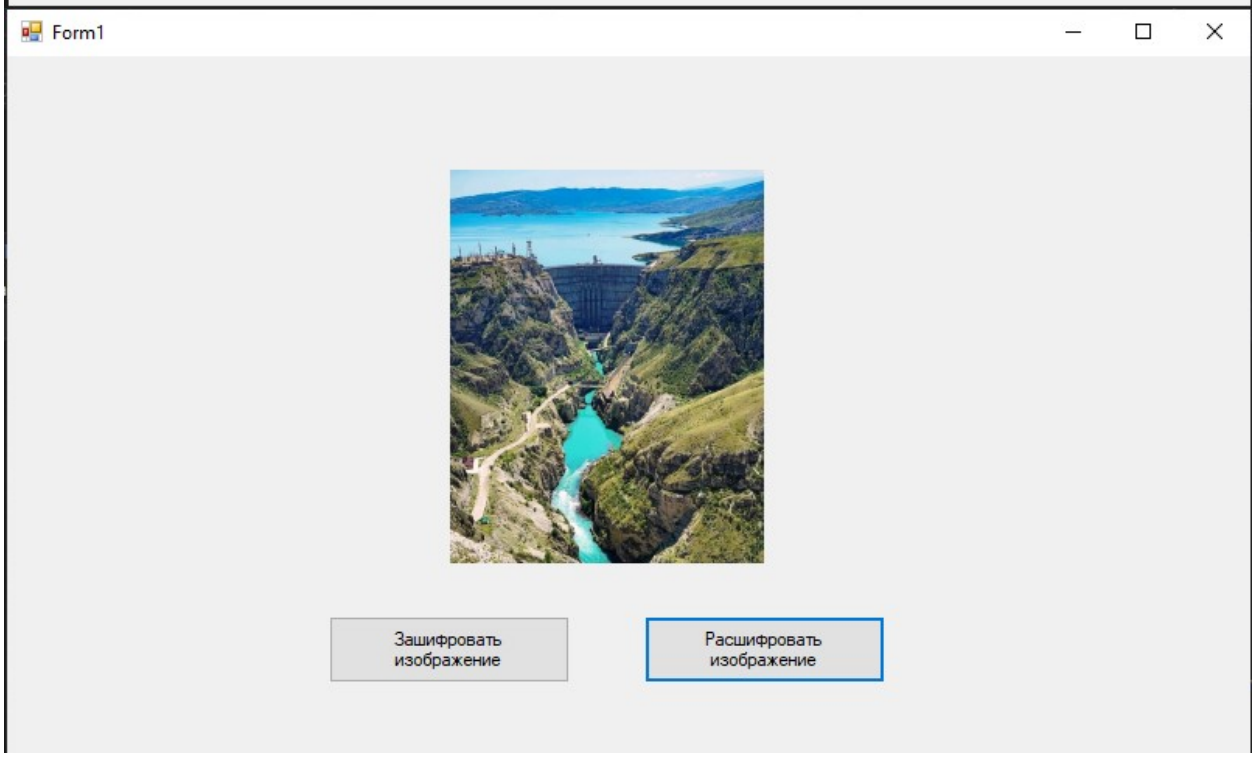
Вместо осмысленного изображения на экране появился «шум», означающий , что генератор на основе РОСП сгенерировал равномерную последовательность битов.
Ключом для такого шифра является начальное состояние регистров РОСП1, РОСП2, РОСП3. В программе они задаются функцией rand() и сохраняются в массив key. Для расшифровки изображения нажмем кнопку «Расшифровать».



Проведем еще один тест, возьмем изображение разрешением больше.

Преобразование выполнилось почти моментально.





Ответы на вопросы

1)Основные понятия

Потоковый шифр - это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в

потоке открытого текста. Поточный шифр реализует другой подход к симметричному шифрованию, нежели блочные шифры.

Потоковые шифры на базе сдвиговых регистров активно использовались в годы войны, ещё задолго до появления электроники. Они были просты в проектировании и реализации.

На сегодняшний день потоковое шифрование используется там, где скорость преобразований важнее надежности. Например, в сотовой связи.

2) Принцип работы

Поточный шифр на основе РОСП является ГПСЧ(генератором псевдо случайных чисел). Чтобы усложнить схему, в нее добавляется дополнительная комбинационная схема. Но при этом всем качество генерации зависит от подобранного ключа.
Генератор генерирует бит, который, в последствии, накладывается на бит исходного изображения, итак до последнего бита. При каждом такте регистры производят сдвиг либо влево либо вправо, а обратная связь , встроенная в схему, генерирует новый бит для безвозвратно удаленного значения при сдвиге ячейки регистра.

3)Криптостойкость

Для того, чтобы повысить надежность таких алгоритмов, выходной бит(гамма) должен иметь следующие свойства:

1.Гамма не должна вырождаться, т.е. не должно быть больших последовательностей подряд идущих нулей или единиц. В первом случае результатом будет исходный текст, во втором – его инверсия.
2. Распределение нулей и единиц в гамме должно подчиняться нормальному закону.
3. Гамма должна обладать максимальным периодом.

Все свойства проверяются на основе различных тестов .

4)Криптоанализ

Основной проблемой генератора заключается в его линейности. А именно, имея открытый текст можно воссоздать подобный генератор, по возможности, имеющий более простой алгоритм работы.
На сегодняшний день Потоковые шифры подаются всем видам атак: силовые, статистические, аналитические.

Листинг программы

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace ROSP
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        const int RegisterSize = 12;
        static byte[] ROSP1 = new byte[RegisterSize];
        static byte[] ROSP2 = new byte[RegisterSize];
        static byte[] ROSP3 = new byte[RegisterSize];
        static byte[] Gamma = new byte[RegisterSize];
        static byte[] key = new byte[RegisterSize * 3];
        static bool DECRYPT_MODE = false;
        static void ROSP(ref byte[] imageBytes) {
            if (DECRYPT_MODE)
            {
                for (int k = 0; k < RegisterSize * 3; k++)
                {
                    if (k < 12)
                        ROSP1[k] = key[k];
                    else if (k > 11 && k < 24)
                        ROSP2[k % 12] = key[k];
                    else
                        ROSP3[k % 12] = key[k];
                }
            }
            else
            {
                Random random = new Random();
                random.NextBytes(ROSP1);
                random.NextBytes(ROSP2);
                random.NextBytes(ROSP3);
                for (int k = 0; k < RegisterSize * 3; k++)
                {
                    if (k < 12)
                        key[k] = ROSP1[k];
                    else if (k > 11 && k < 24)
                        key[k] = ROSP2[k % 12];
                    else
                        key[k] = ROSP3[k % 12];
                }
            }
            for (int i = 0; i < imageBytes.Length; i++) {
```

```
                if (i % 12 == 0)
                {
                    if ((ROSP3[11] >> 1) % 2 == 0)
                    {

                        for (int k = RegisterSize - 1; k > 0; k--)
                        {
                            ROSP1[k] = (byte)(ROSP1[k] >> 1);
                            if (ROSP1[k - 1] % 2 == 1) ROSP1[k] = (byte)(ROSP1[k] |
0b_1000_0000);
                        }
                        ROSP1[0] = (byte)(ROSP1[0] >> 1);
                        if ((((ROSP1[0] >> 7) % 2 + (ROSP1[5] >> 7) % 2 + ROSP1[10] % 2 +
(ROSP1[11] >> 1) % 2) / 2) % 2 == 1) ROSP1[0] = (byte)(ROSP1[0] | 0b_1000_0000);
                    }
                    if ((ROSP1[0] >> 4) % 2 == 0)
                    {
                        for (int k = RegisterSize - 1; k > 0; k--)
                        {
                            ROSP2[k] = (byte)(ROSP2[k] >> 1);
                            if (ROSP2[k - 1] % 2 == 1) ROSP2[k] = (byte)(ROSP2[k] |
0b_1000_0000);
                        }
                        ROSP2[0] = (byte)(ROSP2[0] >> 1);
                        if ((((ROSP2[0] >> 7) % 2 + (ROSP2[5] >> 2) % 2 + ROSP2[9] % 2 +
(ROSP2[11] >> 1) % 2) / 2) % 2 == 1) ROSP2[0] = (byte)(ROSP2[0] | 0b_1000_0000);
                    }
                    if ((ROSP2[0] >> 6) % 2 == 0)
                    {
                        for (int k = RegisterSize - 1; k > 0; k--)
                        {
                            ROSP3[k] = (byte)(ROSP3[k] >> 1);
                            if (ROSP3[k - 1] % 2 == 1) ROSP3[k] = (byte)(ROSP3[k] |
0b_1000_0000);
                        }
                        ROSP3[0] = (byte)(ROSP3[0] >> 1);
                        if ((((ROSP3[0] >> 7) % 2 + ROSP3[5] % 2 + (ROSP3[9] >> 4) % 2 +
(ROSP3[11] >> 1) % 2) / 2) % 2 == 1) ROSP3[0] = (byte)(ROSP3[0] | 0b_1000_0000);
                    }
                    for (int k = 0; k < RegisterSize; k++)
                    {
                        Gamma[k] = (byte)(ROSP1[k] ^ ROSP2[k] ^ ROSP3[k]);
                    }

                }
                imageBytes[i] = Gamma[i % RegisterSize];

            }

        }
        private byte[] ConvertBitMapToByte(Bitmap img)
        {
            byte[] Result = null;
            BitmapData bData = img.LockBits(new Rectangle(new Point(), img.Size),
ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
            int ByteCount = bData.Stride * img.Height;
            Result = new byte[ByteCount];
            Marshal.Copy(bData.Scan0, Result, 0, ByteCount);
            img.UnlockBits(bData);
            return Result;
        }

        private Bitmap ConvertByteToBitMap(byte[] Ishod, int w, int h)
        {
```

```
            Bitmap img = new Bitmap(w, h);
            BitmapData bData = img.LockBits(new Rectangle(new Point(), img.Size),
ImageLockMode.WriteOnly, PixelFormat.Format24bppRgb);
            Marshal.Copy(Ishod, 0, bData.Scan0, Ishod.Length);
            img.UnlockBits(bData);
            return img;
        }

        private void Form1_Load(object sender, EventArgs e)
        {

            pictureBox1.Image = new Bitmap("../image2.bmp");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            DECRYPT_MODE = false;
            string path = "../image2.bmp";
            pictureBox1.Image = new Bitmap(path);
            Bitmap btmp = (Bitmap)pictureBox1.Image;
            byte[] imgBytes = ConvertBitMapToByte(btmp);
            ROSP(ref imgBytes);

            pictureBox1.Image = ConvertByteToBitMap(imgBytes, pictureBox1.Image.Width,
pictureBox1.Image.Height);
        }


        private void button3_Click(object sender, EventArgs e)
        {
            DECRYPT_MODE = true;
            Bitmap btmp = (Bitmap)pictureBox1.Image;
            byte[] imgBytes = ConvertBitMapToByte(btmp);
            ROSP(ref imgBytes);
            pictureBox1.Image = ConvertByteToBitMap(imgBytes, pictureBox1.Image.Width,
pictureBox1.Image.Height);

        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {

        }
    }
}
```