МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра «Вычислительная техника».

**Лабораторная работа №2**

**тема: «Изучение работы потоковых шифров»**

по дисциплине: «Информационная безопасность и защита информации»

Выполнил:

студент 4 курса, гр. ИВТВМбд-41

Захарычев Никита Алексеевич.

Проверил:

доцент кафедры ВТ,

кандидат технических наук

Мартынов Антон Иванович.

г. Ульяновск, 2017

**Задание:**

1. Изучить потоковые алгоритмы шифрования.

2.Из Лабораторной работы №1 выбрать один из двух собственно реализованных генераторов (которые разрабатываются самостоятельно). Выбор аргументировать.

3.Реализовать приложение, которое может осуществлять шифрацию/дешифрацию произвольного файла с помощью потокового шифра.

4. Реализовать функцию хеширования текстового пароля, с помощью которой возможно получить начальное значение для ГПЧ (для каждого генератора схема затравки может быть различной).

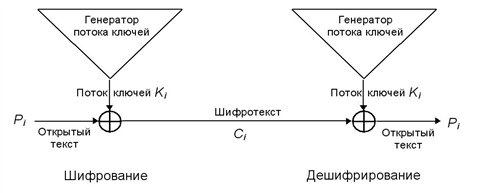
**Вариант - №5**

**Ход работы:**

1. Основной идеей потокового шифрования является использование операции сложения по модулю два исходного текста с гаммой, гамма в свою очередь, создается с помощью генераторов псевдослучайных чисел.

Процесс дешифрирования данных при известном ключе сводится к повторной генерации гаммы и наложении ее на зашифрованные данные.

Если последовательность битов гаммы не имеет периода и выбирается случайно, то взломать шифр невозможно.



1. При выполнении данной лабораторной работы было принято решение использовать генератор Парка-Миллера в качестве генератора псевдослучайных чисел.
2. Исходный код основной функции приложения, выполняющего шифрацию/дешифрацию произвольного файла с помощью потокового шифра.



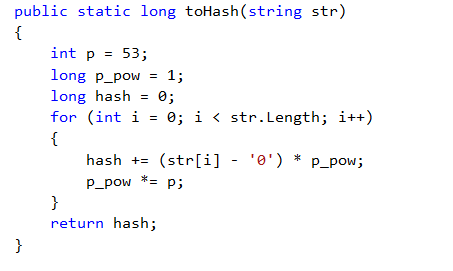
1. Выбран следующий способ определения хэш-функции от строки S:

h(S) = S[0] + S[1] \* P + S[2] \* P^2 + S[3] \* P^3 + ... + S[N] \* P^N,

где P – некоторое число. В данном случае P выберем равно 53, так как буквы могут быть и заглавными и маленькими.

Значение хэша было принято решение хранить в самом большом числовом типе.

Исходный код функции, реализующей хэширование пароля.



**Тестирование**

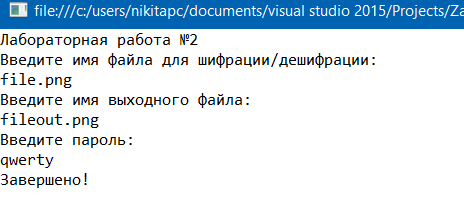
Было принято решение производить тестирование на файле с расширением .png, для того, чтобы наглядно убедиться в корректности работы приложения.

Исходный файл:

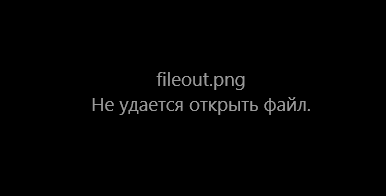


Название исходного файла file.png

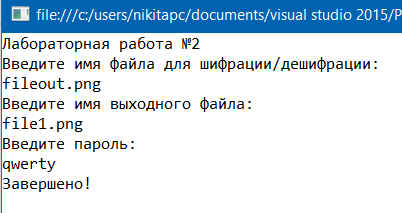
Выполним шифрование файла:



Попробуем открыть зашифрованный файл:



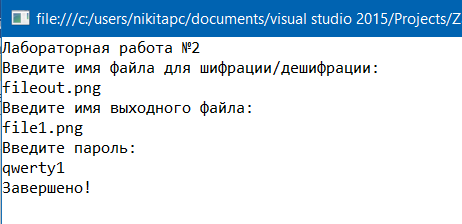
Выполним дешифрирование файла:



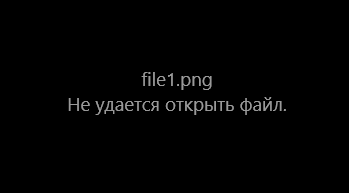
Откроем файл с названием file1.png и увидим исходный файл:



Также проведем дешифрование с использование неправильного пароля:



Получим:



В результате тестирования можно сделать вывод, что приложение работает корректно.

**Выводы**

В ходе выполнения данной лабораторной работы были получены навыки в шифровании/дешифровании произвольного файла с помощью потоковых шифров.

**Исходный код**

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Runtime;

using System.Linq;

using System.Text;

using System.IO;

using System.Threading.Tasks;

namespace ZashitaInf\_2

{

class Program

{

public static long toHash(string str)

{

int p = 53;

long p\_pow = 1;

long hash = 0;

for (int i = 0; i < str.Length; i++)

{

hash += (str[i] - '0') \* p\_pow;

p\_pow \*= p;

}

return hash;

}

static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа №2");

Console.WriteLine("Введите имя файла для шифрации/дешифрации:");

string inp\_file = Console.ReadLine();

Console.WriteLine("Введите имя выходного файла:");

string out\_file = Console.ReadLine();

Console.WriteLine("Введите пароль:");

string pass = Console.ReadLine();

long hash\_pass = toHash(pass);

GeneratorParkMiller gpm = new GeneratorParkMiller(hash\_pass);

using (BinaryWriter w = new BinaryWriter(new FileStream(out\_file, FileMode.Create)))

using (BinaryReader r = new BinaryReader(new FileStream(inp\_file, FileMode.Open)))

{

while (r.BaseStream.Position != r.BaseStream.Length)

{

byte rnd = gpm.NextByte();

byte brd = r.ReadByte();

byte val = (byte)(rnd ^ brd);

w.Write(val);

}

}

Console.Write("Завершено!");

Console.ReadKey();

}

class GeneratorParkMiller

{

private long x0;

private BigInteger x;

private List<byte> curr;

private int a = 16807;

private int m = 2147483647;

public GeneratorParkMiller(long num)

{

this.x0 = num;

this.x = Random(x0);

this.curr = new List<byte>();

this.curr.AddRange(this.x.ToByteArray());

}

public GeneratorParkMiller(int a, int m, long num)

{

this.a = a;

this.m = m;

this.x0 = num;

}

public static byte[] getBytes(long len, long[] values)

{

var result = new List<byte>();

for (int i = 0; i < values.Length; i++)

{

byte[] tmp = BitConverter.GetBytes(values[i]);

for (int j = 0; j < tmp.Length; j++)

{

if (tmp[j] == 0) continue;

if (result.Count == len) return result.ToArray();

result.Add(tmp[j]);

}

}

return result.ToArray();

}

public byte NextByte()

{

if (this.curr.Count > 0)

{

var b = this.curr.ElementAt(0);

this.curr.RemoveAt(0);

return b;

}

else

{

this.x = Random(this.x);

this.curr.AddRange(this.x.ToByteArray());

}

return 0;

}

private BigInteger Random(BigInteger x)

{

var a = new BigInteger(this.a);

var m = new BigInteger(this.m);

return (x \* a) % m;

}

}

}

}