**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

СОГЛАСОВАНО

Профессор департамента

программной инженерии факультета компьютерных наук, к.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Авдошин

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

УТВЕРЖДАЮ

|  |
| --- |
| Академический руководитель образовательной программы «Программная инженерия», к.т.н. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Подп. и дата

**КРИПТОСИСТЕМА РАБИНА  
(RABIN CRYPTOSYSTEM)**

**Текст программы**

Инв. № дубл.

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729. 501430-01 12 01-1**

Взам. инв. №

Исполнитель:

студент группы БПИ 165

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. С. Чуев

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Подп. и дата

Инв. № подл

RU.17701729.

501430-01

12 01-1

**2017**

УТВЕРЖДЕН  
RU.17701729. 501430-01 12 01-1-ЛУ

**КРИПТОСИСТЕМА РАБИНА  
(RABIN CRYPTOSYSTEM)**

**Текст программы**

**RU.17701729. 501430-01 12 01-1**

Подп. и дата

Инв. № дубл.

Взам. инв. №

**Листов 51**

Подп. и дата

Инв. № подл

RU.17701729.

501430-01

12 01-1

**2017**

**АННОТАЦИЯ**

В данном программном документе приведет текст «Криптосистема Рабина (Rabin Cryptosystem)». Текст программы реализован в виде символической записи на исходном языке. Исходным языком данной разработки является C# 5.0. Среда разработки - Microsoft Visual Studio 2015 Enterprise (локализованная русская версия).

Данное программное обеспечение предназначено для защищенной передачи важных файлов. Программа, реализуя алгоритм криптосистемы Рабина, способна кодировать и декодировать информацию, представленную в бинарном виде с использованием открытого и закрытого ключа.

СОДЕРЖАНИЕ

[1. БИБЛИОТЕКА RABIN.LIB 4](#_Toc482984675)

[1.1. Rabin.cs 4](#_Toc482984676)

[1.2. RabinEncoding.cs 15](#_Toc482984677)

[1.3. ProgressInfo.cs 20](#_Toc482984678)

[2. ТЕКСТ ПРОГРАММЫ RABIN CRYPTOSYSTEM 20](#_Toc482984679)

[2.1. Rabin\_Cryptosystem.cs 20](#_Toc482984680)

[2.2. Rabin\_Cryptosystem.Designer.cs 33](#_Toc482984681)

[2.3. GenKey.cs 43](#_Toc482984682)

[2.4. GenKey.Designer.cs 45](#_Toc482984683)

[2.5. Program.cs 48](#_Toc482984684)

[2.6. ReachProgressBar.cs 48](#_Toc482984685)

[2.7. ProgressBarForManyProcesses.cs 49](#_Toc482984686)

Программа состоит из 10 классов.

# **1. БИБЛИОТЕКА RABIN.LIB**

## **1.1. Rabin.cs**

using System;

using System.Linq;

using System.Numerics;

using System.Text;

using System.IO;

using System.Threading.Tasks;

namespace RabinLib

{

public class Rabin

{

private BigInteger \_p, \_q, \_n;

private int \_bitLength;

private const int CheckByteCount = 13;

static Random rnd = new Random();

/// <summary>

/// Случайным образом инициализирует закрытые ключи,

/// открытый ключ и находит получившуюся длину в битах

/// открытого ключа

/// </summary>

/// <param name="bitLength">Длина открытого ключа в битах</param>

public Rabin(int bitLength)

{

GeneratePrivateKeys(bitLength);

GeneratePublicKey();

}

public Rabin(int bitLength, IProgress<ProgressInfo> progress)

{

GeneratePrivateKeys(bitLength, progress);

GeneratePublicKey();

}

/// <summary>

/// Инициализирует поля N и Length

/// </summary>

/// <param name="n">Открытый ключ</param>

public Rabin(BigInteger n)

{

if (n < 0)

throw new ArgumentOutOfRangeException("Отрицательное значение открытого ключа", "Открытый ключ должен быть положителен");

\_n = n;

GetBitLength();

}

/// <summary>

/// Создает шифровальщик с 1-ым и 2-ым закрытыми ключами

/// </summary>

/// <param name="p">1-ый закрытый ключ</param>

/// <param name="q">2-ой закрытый ключ</param>

public Rabin(BigInteger p, BigInteger q)

{

if (p < 0 || q < 0)

throw new ArgumentOutOfRangeException("Отрицательное значение закрытого ключа", "Открытые ключи должен быть положителен");

if (p == q)

throw new ArgumentException("p == q", "Закрытые ключи не могут быть равны между собой");

\_p = p;

\_q = q;

GeneratePublicKey();

}

/// <summary>

/// Считывает из файла открытый или закрытые ключ

/// </summary>

/// <param name="keyPath">Путь к файлу с ключами</param>

public Rabin(string keyPath)

{

if (Path.GetExtension(keyPath).CompareTo(".publicKey") == 0)

{

\_n = ReadPublicKey(keyPath);

}

else

{

BigInteger[] pq = ReadPrivateKey(keyPath);

\_p = pq[0];

\_q = pq[1];

GeneratePublicKey();

}

GetBitLength();

}

/// <summary>

/// Возвращает или задает объект, содержащий данные об объекте.

/// </summary>

public object Tag { get; set; }

private void GeneratePrivateKeys(int bitLength, IProgress<ProgressInfo> progress = null)

{

if (bitLength <= 22)

throw new ArgumentOutOfRangeException("Длина открытого ключа слишком мала", "Выберите лисло с бинарной длиной большее 22");

int len;

do

{

progress?.Report(new ProgressInfo(0, bitLength \* 20));

len = bitLength / 2 - rnd.Next((int)(bitLength \* 0.05), (int)(bitLength \* 0.1));

\_p = BlumPrime(bitLength - len);

progress?.Report(new ProgressInfo(bitLength \* 10, bitLength \* 20));

\_q = BlumPrime(len);

progress?.Report(new ProgressInfo(bitLength \* 20, bitLength \* 20));

} while (\_p == \_q);

}

private void GeneratePublicKey()

{

\_n = \_p \* \_q;

GetBitLength();

}

private void GetBitLength()

{

\_bitLength = (int)(BigInteger.Log(\_n) / BigInteger.Log(2));

}

/// <summary>

/// Возвращает массив BigInteger, который содержит 2 закрытых ключа

/// </summary>

public BigInteger[] PQ

{

get { return new BigInteger[] { \_p, \_q }; }

}

/// <summary>

/// Возвращает открытый ключ

/// </summary>

public BigInteger N

{

get { return \_n; }

}

/// <summary>

/// Возвращает бинарную длину открытого ключа

/// </summary>

public int BitLength

{

get { return \_bitLength; }

}

/// <summary>

/// Генерируем число Блюма

/// </summary>

/// <param name="bitLength">длина числа</param>

/// <returns>само число</returns>

private static BigInteger BlumPrime(int bitLength)

{

Mono.Math.BigInteger p;

do

{

p = Mono.Math.BigInteger.GeneratePseudoPrime(bitLength);

} while (!Mono.Math.BigInteger.Equals(p % 4, 3));

byte[] p\_array = p.GetBytes().Reverse().ToArray();

if (p > 0 && (p\_array[p\_array.Length - 1] & 0x80) > 0)

{

byte[] temp = new byte[p\_array.Length];

Array.Copy(p\_array, temp, p\_array.Length);

p\_array = new byte[temp.Length + 1];

Array.Copy(temp, p\_array, temp.Length);

}

BigInteger q = new BigInteger(p\_array);

return q;

}

/// <summary>

/// Дешифрует сообщение

/// </summary>

/// <param name="c">Шифротекст</param>

/// <returns>массив BigInteger, который содержит 4 корня</returns>

public BigInteger[] Decrypt(BigInteger c)

{

if (c > \_n)

throw new ArgumentException("c","Длина шифротекста превосходит длину открытого ключа");

BigInteger r = BigInteger.ModPow(c, (\_p + 1) / 4, \_p);

BigInteger s = BigInteger.ModPow(c, (\_q + 1) / 4, \_q);

BigInteger a, b;

Gcd(\_p, \_q, out a, out b);

//(a\*p\*r + b\*q\*s) % n

BigInteger m1 = (a \* \_p \* s + b \* \_q \* r) % \_n;

if (m1 < 0)

m1 += \_n;

BigInteger m2 = \_n - m1;

BigInteger m3 = (a \* \_p \* s - b \* \_q \* r) % \_n;

if (m3 < 0)

m3 += \_n;

BigInteger m4 = \_n - m3;

return new BigInteger[] { m1, m2, m3, m4 };

}

/// <summary>

/// Находит коэффиценты расширенного алгоритма Эвклида

/// </summary>

/// <param name="a">Первой число</param>

/// <param name="b">Второе число</param>

/// <param name="x">Первый коэффициент</param>

/// <param name="y">Второй коэффициент</param>

/// <returns>НОД a и b</returns>

private static BigInteger Gcd(BigInteger a, BigInteger b, out BigInteger x, out BigInteger y)

{

if (a == 0)

{

x = 0; y = 1;

return b;

}

BigInteger x1, y1;

BigInteger d = Gcd(b % a, a, out x1, out y1);

x = y1 - (b / a) \* x1;

y = x1;

return d;

}

/// <summary>

/// Шифрует сообщение

/// </summary>

/// <param name="m">сообщение</param>

/// <returns>шифротекст</returns>

public BigInteger Encrypt(BigInteger m)

{

if (m > \_n)

throw new ArgumentException("m", "Длина сообщения превосходит длину открытого ключа");

BigInteger с = BigInteger.ModPow(m, 2, \_n);

return с;

}

/// <summary>

/// Переводит строку в число с кодировкой Unicode

/// </summary>

/// <param name="str">строка</param>

/// <returns>число</returns>

public BigInteger ToBigInteger(string str)

{

byte[] temp = Encoding.Unicode.GetBytes(str);

byte[] array = new byte[temp.Length + 1];

Array.Copy(temp, array, temp.Length);

array[array.Length - 1] = 1;

BigInteger m = new BigInteger(array);

return m;

}

/// <summary>

/// Переводит число в строку с кодировкой Unicode

/// </summary>

/// <param name="m">число</param>

/// <returns>строка</returns>

public string ToString(BigInteger m)

{

byte[] temp = m.ToByteArray();

byte[] array = new byte[temp.Length - 1];

Array.Copy(temp, array, temp.Length - 1);

return Encoding.Unicode.GetString(array);

}

/// <summary>

/// Сохраняет открытый и закрытый ключи в указанное место на диске.

/// </summary>

/// <param name="keysPath">место, куда производится запись</param>

public void KeyToFile(string keysPath)

{

CreateOrCreateNewFolder(keysPath);

WritePublicKey(keysPath + "\\" + Path.GetFileName(keysPath) + ".publicKey");

WritePrivateKeys(keysPath + "\\" + Path.GetFileName(keysPath) + ".privateKey");

}

/// <summary>

/// Записывает открытый ключ в указанный файл в

/// кодировке RabinEncoding.

/// </summary>

/// <param name="publicKeyPath">место, куда производится запись</param>

public void WritePublicKey(string publicKeyPath)

{

if (Path.GetExtension(publicKeyPath).CompareTo(".publicKey") != 0)

throw new ArgumentException("Неверное расширение");

using (StreamWriter sw = new StreamWriter(new FileStream(publicKeyPath, FileMode.Create), new RabinEncoding()))

sw.Write(N.ToString());

}

/// <summary>

/// Записывает закрытый ключ в указанный файл в

/// кодировке RabinEncoding.

/// </summary>

/// <param name="privateKeyPath">место, куда производится запись</param>

public void WritePrivateKeys(string privateKeyPath)

{

if (Path.GetExtension(privateKeyPath).CompareTo(".privateKey") != 0)

throw new ArgumentException("Неверное расширение");

using (StreamWriter sw = new StreamWriter(new FileStream(privateKeyPath, FileMode.Create), new RabinEncoding()))

sw.WriteLine(\_p.ToString() + "\r\n" + \_q.ToString());

}

/// <summary>

/// Создает новую папку в указанном месте с указанным названием

/// </summary>

/// <param name="folderPath">указанное место</param>

private void CreateOrCreateNewFolder(string folderPath)

{

if (!Directory.Exists(folderPath))

Directory.CreateDirectory(folderPath);

else

{

int i = 1;

while (Directory.Exists(folderPath + "\_" + i))

i++;

folderPath += "\_" + i;

Directory.CreateDirectory(folderPath);

}

}

/// <summary>

/// Считывает из указанного файла открытый ключ и

/// возвращает его численное значение

/// </summary>

/// <param name="publicKeyPath">Путь к файлу</param>

/// <returns>Численное значение открытого числа</returns>

public static BigInteger ReadPublicKey(string publicKeyPath)

{

if (Path.GetExtension(publicKeyPath).CompareTo(".publicKey") != 0)

throw new ArgumentException("Неверное расширение");

using (StreamReader sr = new StreamReader(new FileStream(publicKeyPath, FileMode.Open), new RabinEncoding()))

{

return BigInteger.Parse(sr.ReadLine());

}

}

/// <summary>

/// Считывает из указанного файла закрытые ключи и

/// возвращает массив с их численными значениями

/// </summary>

/// <param name="privateKeyPath>Путь к файлу</param>

/// <returns>Массив с закрытыми ключами</returns>

public static BigInteger[] ReadPrivateKey(string privateKeyPath)

{

if (Path.GetExtension(privateKeyPath).CompareTo(".privateKey") != 0)

throw new ArgumentException("Неверное расширение");

BigInteger[] pq = new BigInteger[2];

using (StreamReader sr = new StreamReader(new FileStream(privateKeyPath, FileMode.Open), new RabinEncoding()))

{

pq[0] = BigInteger.Parse(sr.ReadLine());

pq[1] = BigInteger.Parse(sr.ReadLine());

}

return pq;

}

/// <summary>

/// Шифрует файл с именем filename

/// и сохраняет зашифрованный файл в указанном месте.

/// Отсылает информацию о процессе, если

/// progress инициализирован пользователем.

/// </summary>

/// <param name="filename">путь к файлу</param>

/// <param name="encryptPath">место, куда производится запись</param>

/// <param name="progress">прогресс процесса</param>

public void Encrypt(string filename, string encryptPath, IProgress<ProgressInfo> progress = null)

{

using (FileStream read = new FileStream(filename, FileMode.Open, FileAccess.Read))

{

using (FileStream write = new FileStream(encryptPath, FileMode.Create, FileAccess.Write))

{

Encrypt(read, write, progress);

}

}

}

/// <summary>

/// Асинхронно шифрует файл с именем filename

/// и сохраняет зашифрованный файл в указанном месте.

/// Отсылает информацию о процессе.

/// </summary>

/// <param name="filename">путь к файлу</param>

/// <param name="encryptPath">место, куда производится запись</param>

/// <param name="progress">прогресс процесса</param>

/// <returns></returns>

public async Task EncryptAsync(string filename, string encryptPath, IProgress<ProgressInfo> progress)

{

using (FileStream read = new FileStream(filename, FileMode.Open))

{

using (FileStream write = new FileStream(encryptPath, FileMode.Create))

{

await Task.Run (() => Encrypt(read, write, progress));

}

}

}

/// <summary>

/// Кодирует информацию из первого объекта Stream и записывает ее во второй объект Stream

/// </summary>

/// <param name="read">Объект Stream, откуда считывается кодируемая информация</param>

/// <param name="write">Объект Stream, куда записывается закодированная информация</param>

/// <param name="progress">Объект куда надо передавать прогресс</param>

public void Encrypt(Stream read, Stream write, IProgress<ProgressInfo> progress)

{

int byteLength = \_bitLength % 8 == 0 ? \_bitLength / 8 - 1 : \_bitLength / 8;

byte[] chunk;

using (BinaryReader br = new BinaryReader(read))

{

using (StreamWriter sw = new StreamWriter(write, new RabinEncoding()))

{

chunk = ReadFileChunk(br, byteLength);

while (chunk.Length > 0)

{

sw.Write(Encrypt(new BigInteger(chunk)) + ":");

progress?.Report(new ProgressInfo(read.Position, read.Length));

chunk = ReadFileChunk(br, byteLength);

}

}

}

}

private byte[] ReadFileChunk(BinaryReader br, int byteLength)

{

byte[] temp = br.ReadBytes(byteLength - CheckByteCount);

if (temp.Length == 0)

return new byte[0];

byte[] chunk = new byte[temp.Length + CheckByteCount];

Array.Copy(temp, 0, chunk, CheckByteCount - 1, temp.Length);

chunk[chunk.Length - 1] = 1;

return chunk;

}

/// <summary>

/// Дешифрует файл с именем filename

/// и сохраняет зашифрованный файл в указанном месте.

/// Отсылает информацию о процессе, если

/// progress инициализирован пользователем.

/// </summary>

/// <param name="filename">путь к файлу</param>

/// <param name="decryptPath">место куда производится запись</param>

/// <param name="progress">прогресс процесса</param>

public void Decrypt(string filename, string decryptPath, IProgress<ProgressInfo> progress = null)

{

using (FileStream read = new FileStream(filename, FileMode.Open))

{

using (FileStream write = new FileStream(decryptPath, FileMode.Create))

{

Decrypt(read, write, progress);

}

}

}

/// <summary>

/// Асинхронно шифрует файл с именем filename

/// и сохраняет зашифрованный файл в указанном месте.

/// Отсылает информацию о процессе.

/// </summary>

/// <param name="filename">путь к файлу</param>

/// <param name="decryptPath">место куда производится запись</param>

/// <param name="progress">прогресс процесса</param>

/// <returns></returns>

public async Task DecryptAsync(string filename, string decryptPath, IProgress<ProgressInfo> progress)

{

using (FileStream read = new FileStream(filename, FileMode.Open))

{

using (FileStream write = new FileStream(decryptPath, FileMode.Create))

{

await Task.Run(() => Decrypt(read, write, progress));

}

}

}

/// <summary>

/// Декодирует сообщение из первого объекта Stream и записывает ее во второй объект Stream

/// </summary>

/// <param name="read">Объект Stream, откуда считывается закодированная информация</param>

/// <param name="write">>Объект Stream, куда записывается раскодированная информация</param>

public void Decrypt(Stream read, Stream write, IProgress<ProgressInfo> progress)

{

string encryptedChunk;

using (BinaryWriter bw = new BinaryWriter(write))

{

using (StreamReader sr = new StreamReader(read, new RabinEncoding()))

{

while (!string.IsNullOrEmpty(encryptedChunk = ReadEncryptedChunk(sr)))

{

var decryptedNumbers = Decrypt(BigInteger.Parse(encryptedChunk));

bw.Write(GetDecryptedChunk(decryptedNumbers));

progress?.Report(new ProgressInfo(read.Position, read.Length));

}

}

}

}

private string ReadEncryptedChunk(StreamReader sr)

{

string chunk = string.Empty;

int nextSymbol;

while ((nextSymbol = sr.Read()) != ':')

{

if (!char.IsDigit((char)nextSymbol))

break;

chunk += (char)nextSymbol;

}

return chunk;

}

byte[] GetDecryptedChunk(BigInteger[] numbers)

{

var chunk = numbers.Select(x => x.ToByteArray()).FirstOrDefault(IsChunkValid);

if (chunk == null)

throw new InvalidDataException("Файл не может быть расшифрован");

var dataLength = chunk.Length - CheckByteCount;

var clippedChunk = new byte[dataLength];

Array.Copy(chunk, CheckByteCount - 1, clippedChunk, 0, dataLength);

return clippedChunk;

}

bool IsChunkValid(byte[] chunk)

{

return chunk.Last() == 1 && chunk.Take(CheckByteCount - 1).All(x => x == 0);

}

}

}

**1.2. RabinEncoding.cs**

using System;

using System.Text;

namespace RabinLib

{

class RabinEncoding : Encoding

{

byte[] encoding = new byte[11];

public RabinEncoding() : this(new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}) { }

public RabinEncoding(byte[] newEncoding)

{

if (newEncoding.Length != 11)

throw new ArgumentException("Неверная длина массива", "Длина массива должна быть равна 11");

for (int i = 0; i < 11; i++)

if (newEncoding[i] < 0 || newEncoding[i] > 12)

throw new ArgumentException("Неверные значения", "Один из элементов не принадлежит промежутку [0, 12]");

for (int i = 0; i < 10; i++)

for (int j = i + 1; j < 11; j++)

if (newEncoding[i] == newEncoding[j])

throw new ArgumentException("Неверные значения", "Один из элементов равен другому");

newEncoding.CopyTo(encoding, 0);

}

/// <summary>

/// Переводит символ в его числовое представление

/// в кодировке RabinEncoding.

/// </summary>

/// <param name="simbol">переводимый символ</param>

/// <returns>число, соответствующее символу</returns>

public byte ToByte(char simbol)

{

if (simbol >= '0' && simbol <= '9')

return encoding[byte.Parse(simbol.ToString())];

if (simbol == ':')

return encoding[10];

if (simbol == '\n')

return 13;

if (simbol == '\r')

return 14;

throw new EncoderFallbackException("В кодировке нет такого значения");

}

/// <summary>

/// Переводит число в символ, используя кодировку RabinEncoding.

/// </summary>

/// <param name="number">переводимое число</param>

/// <returns>символ, соответствующий числу</returns>

public char ToChar(byte number)

{

if (number == 15)

return '\0';

for (int i = 0; i < 10; i++)

if (encoding[i] == number)

return i.ToString()[0];

if (encoding[10] == number)

return ':';

if (number == 13)

return '\n';

if (number == 14)

return '\r';

throw new DecoderFallbackException("В кодировке нет такого значения");

}

/// <summary>

/// Переводит строку в массив байтов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="input">переводимая строка</param>

/// <returns>декодированный массив байтов</returns>

public byte[] ToByteArray(string input)

{

byte[] output = new byte[(int)(input.Length / 2.0 + 0.5)];

int k = 0;

for (int i = 1; i < input.Length; i += 2)

output[k++] = (byte)(ToByte(input[i - 1]) \* 16 + ToByte(input[i]));

if (input.Length % 2 == 1)

output[k] = (byte)(ToByte(input[input.Length - 1]) \* 16 + 15);

return output;

}

/// <summary>

/// Переводит массив байтов в строку,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="input">получаемый массив байтов</param>

/// <returns>полученная строка</returns>

public string ToString(byte[] input)

{

string output = string.Empty;

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

{

output += ToChar((byte)(input[i] / 16));

output += ToChar((byte)(input[i] % 16));

}

return output;

}

/// <summary>

/// Кодирует набор символов из указанного

/// массива символов в указанный массив байтов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="chars">массив байтов</param>

/// <param name="charIndex">начальный индекс</param>

/// <param name="charCount">колличество символов</param>

/// <param name="bytes">массив байтов, куда производится запись</param>

/// <param name="byteIndex">начальный индекс в массиве байтов</param>

/// <returns>длина записанных символов</returns>

public override int GetBytes(char[] chars, int charIndex, int charCount, byte[] bytes, int byteIndex)

{

char[] chunkOfChars = new char[charCount];

Array.Copy(chars, charIndex, chunkOfChars, 0, charCount);

string input = new string(chunkOfChars);

byte[] chunkOfBytes = ToByteArray(input);

int len = chunkOfBytes.Length;

Array.Copy(chunkOfBytes, 0, bytes, byteIndex, len);

return len;

}

/// <summary>

/// Вычисляет количество символов, полученных при декодировании

/// последовательности байтов из заданного массива байтов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="bytes">массив байтов</param>

/// <param name="index">начальный индекс</param>

/// <param name="count">колличество символов</param>

/// <returns>колличество символов</returns>

public override int GetCharCount(byte[] bytes, int index, int count)

{

int sum = 0;

for (int i = index; i < count + index; i++)

if (bytes[i] % 16 == 15)

sum++;

return sum;

}

/// <summary>

/// Вычисляет количество байтов, полученных

/// при кодировании набора символов из указанного массива символов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="chars">массив символов</param>

/// <param name="index">стартовый индекс</param>

/// <param name="count">колличество кодируемых символов</param>

/// <returns>колличество байтов</returns>

public override int GetByteCount(char[] chars, int index, int count)

{

return count \* 2;

}

/// <summary>

/// Декодирует последовательность байтов

/// из указанного массива байтов в указанный массив символов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="bytes">кодируемый массив байтов</param>

/// <param name="byteIndex">начальный индекс</param>

/// <param name="byteCount">колличество символов</param>

/// <param name="chars">массив символов</param>

/// <param name="charIndex">начальный индекс для массива chars</param>

/// <returns>колличество символов</returns>

public override int GetChars(byte[] bytes, int byteIndex, int byteCount, char[] chars, int charIndex)

{

byte[] chunkOfBytes = new byte[byteCount];

Array.Copy(bytes, byteIndex, chunkOfBytes, 0, byteCount);

string output = ToString(chunkOfBytes);

int len = output.Length;

char[] chunkOfChars = output.ToCharArray();

Array.Copy(chunkOfChars, 0, chars, charIndex, len);

return len;

}

/// <summary>

/// Вычисляет максимальное количество байтов,

/// полученных при кодировании заданного количества символов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="charCount">колличество символов</param>

/// <returns>колличество байтов</returns>

public override int GetMaxByteCount(int charCount)

{

return (int)(charCount / 2.0 + 0.5);

}

/// <summary>

/// Вычисляет максимальное количество символов,

/// полученных при декодировании заданного количества байтов,

/// используя кодировку RabinEncoding.

/// </summary>

/// <param name="byteCount">колличество байтов</param>

/// <returns>колличество символов</returns>

public override int GetMaxCharCount(int byteCount)

{

return byteCount \* 2;

}

}

}

## **1.3. ProgressInfo.cs**

using System;

namespace RabinLib

{

public class ProgressInfo

{

/// <summary>

/// Положение в данный момент

/// </summary>

public readonly long value;

/// <summary>

/// Максимальное значение

/// </summary>

public readonly long maximum;

/// <summary>

/// Инициализирует объект ProgressInfo,

/// задавая новое положение и максимальное значение

/// </summary>

/// <param name="newValue">Новое положение</param>

/// <param name="newMaximum">Новое максимальное значение</param>

public ProgressInfo(long newValue, long newMaximum)

{

value = newValue;

maximum = newMaximum;

}

}

}

# **2. ТЕКСТ ПРОГРАММЫ RABIN CRYPTOSYSTEM**

## **2.1. Rabin\_Cryptosystem.cs**

using System;

using System.ComponentModel;

using System.Drawing;

using System.IO;

using System.Windows.Forms;

using RabinLib;

using System.Diagnostics;

using System.Linq;

using System.Collections.Generic;

namespace Rabin

{

public partial class Rabin\_Cryptosystem : Form

{

public Rabin\_Cryptosystem(string[] args)

{

InitializeComponent();

mainBar = new ProgressBarForManyProcesses(progressBar1);

imageList1.Images.Add(".publicKey", Properties.Resources.PublicKey);

imageList1.Images.Add(".privateKey", Properties.Resources.PrivateKey);

imageList1.Images.Add(".encrypted", Properties.Resources.Encrypted);

foreach (string fileName in args)

{

string extention;

try

{

extention = Path.GetExtension(fileName);

switch (extention)

{

case (".publicKey"):

case (".privateKey"):

AddFileItemToListView(fileName, listKeys, imageList1);

break;

default:

AddFileItemToListView(fileName, listFiles, imageList1);

break;

}

}

catch { }

}

}

RabinLib.Rabin rab;

internal string lastDirectory = String.Empty;

List<ListViewItem> encryptingItems = new List<ListViewItem>();

List<ListViewItem> decryptingItems = new List<ListViewItem>();

internal ProgressBarForManyProcesses mainBar;

/// <summary>

/// Позволяет пользователю выбрать файлы и добавить их в listFiles

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void openFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

using (OpenFileDialog ofd = new OpenFileDialog())

{

ofd.InitialDirectory = lastDirectory;

ofd.Title = "Открыть файл";

ofd.Multiselect = true;

if (ofd.ShowDialog() == DialogResult.OK)

foreach (string file in ofd.FileNames)

{

if (listFiles.Items.Count == 128)

{

MessageBox.Show("Извините, но нельзя добавить больше 128 файлов", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

}

AddFileItemToListView(file, listFiles, imageList1);

}

if (ofd.FileNames.Length != 0)

lastDirectory = Path.GetDirectoryName(ofd.FileNames[0]);

}

}

/// <summary>

/// Позволяет пользователю выбрать ключи и добавить их в listKeys

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void addKeyToolStripMenuItem\_Click(object sender, EventArgs e)

{

using (OpenFileDialog ofd = new OpenFileDialog())

{

ofd.Filter = "Ключи (\*.publicKey; \*.privateKey)|\*.publicKey; \*.privateKey";

ofd.Title = "Выбор ключа";

ofd.Multiselect = true;

ofd.InitialDirectory = lastDirectory;

if (ofd.ShowDialog() == DialogResult.OK)

foreach (string file in ofd.FileNames)

AddFileItemToListView(file, listKeys, imageList1);

if (ofd.FileNames.Length != 0)

lastDirectory = Path.GetDirectoryName(ofd.FileNames[0]);

}

}

internal void AddFileItemToListView(string fileName, ListView listToAdd, ImageList imageList)

{

int pictureIndex = AddNewFileImageItemToImageListAndReturnsPictureIndex(fileName, imageList);

ListViewItem newItem = CreateNewListViewItem(fileName, pictureIndex);

AddItemToListIfNotExist(newItem, listToAdd);

}

private int AddNewFileImageItemToImageListAndReturnsPictureIndex(string fileName, ImageList imageList)

{

string extentionFile = Path.GetExtension(fileName);

if (!imageList.Images.Keys.Contains(extentionFile))

imageList.Images.Add(extentionFile, Icon.ExtractAssociatedIcon(fileName));

return imageList.Images.Keys.IndexOf(extentionFile);

}

private ListViewItem CreateNewListViewItem(string fileName, int pictureIndex)

{

ListViewItem newItem = new ListViewItem();

newItem.Text = Path.GetFileName(fileName);

newItem.ImageIndex = pictureIndex;

newItem.Tag = fileName;

return newItem;

}

private void AddItemToListIfNotExist(ListViewItem newItem, ListView listToAdd)

{

foreach (ListViewItem item in listToAdd.Items)

{

if (newItem.Tag.Equals(item.Tag))

{

MessageBox.Show("Файл уже добавлен", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

}

listToAdd.Items.Add(newItem);

}

private ProgressBar CreateProgressBar(ListViewItem item, ListView listToAdd)

{

ProgressBar pb = new ProgressBar();

pb.Maximum = (int)(new FileInfo((string)item.Tag)).Length;

pb.Name = (string)item.Tag;

pb.SetBounds(item.Position.X - 21 + 5, item.Position.Y + item.ImageList.ImageSize.Height + 20, item.Bounds.Width - 10, 8);

listToAdd.Controls.Add(pb);

Update();

return pb;

}

private void contextMenuStripFiles\_Opening(object sender, CancelEventArgs e)

{

for (int i = 0; i < contextMenuStripFiles.Items.Count; i++)

contextMenuStripFiles.Items[i].Enabled = false;

if (listFiles.SelectedItems.Count != 0)

{

if (progressBar1.Value == 0) {

for (int i = 0; i < contextMenuStripFiles.Items.Count; i++)

contextMenuStripFiles.Items[i].Enabled = true;

}

else {

for (int i = 0; i < contextMenuStripFiles.Items.Count - 1; i++)

contextMenuStripFiles.Items[i].Enabled = true;

}

}

}

private void genKeyToolStripMenuItem\_Click(object sender, EventArgs e)

{

GenKey gk = new GenKey(this);

gk.Show();

}

private void contextMenuStripKeys\_Opening(object sender, CancelEventArgs e)

{

for (int i = 0; i < contextMenuStripKeys.Items.Count; i++)

contextMenuStripKeys.Items[i].Enabled = false;

if (listKeys.SelectedItems.Count != 0)

{

for (int i = 0; i < contextMenuStripKeys.Items.Count; i++)

contextMenuStripKeys.Items[i].Enabled = true;

}

}

private void selectKeyToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (!File.Exists(listKeys.SelectedItems[0].Tag.ToString()))

{

MessageBox.Show("Файл не найден", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

listKeys.Items.Remove(listKeys.SelectedItems[0]);

return;

}

ListViewItem newKey = (ListViewItem)listKeys.SelectedItems[0].Clone();

listSelectedKey.Items.Clear();

listSelectedKey.Items.Add(newKey);

rab = new RabinLib.Rabin(newKey.Tag.ToString());

rab.Tag = newKey.Text;

}

private void contextMenuStripSelectedKey\_Opening(object sender, CancelEventArgs e)

{

for (int i = 0; i < contextMenuStripSelectedKey.Items.Count; i++)

contextMenuStripSelectedKey.Items[i].Enabled = false;

if (listSelectedKey.SelectedItems.Count == 1)

for (int i = 0; i < contextMenuStripSelectedKey.Items.Count; i++)

contextMenuStripSelectedKey.Items[i].Enabled = true;

}

private void deleteToolStripMenuItemSelectedKey\_Click(object sender, EventArgs e)

{

rab = null;

listSelectedKey.Items.Clear();

}

private async void encryptToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (rab == null) {

MessageBox.Show("Не выбран открытый ключ", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

if (Path.GetExtension(listSelectedKey.Items[0].Tag.ToString()).CompareTo(".publicKey") != 0) {

MessageBox.Show("Не тот ключ!\r\nВыберите открытый ключ", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

string folder = BrouseFolder();

if (string.IsNullOrEmpty(folder))

return;

foreach (ListViewItem item in listFiles.SelectedItems)

{

if (!IsExistInEncryptItems(item))

{

item.SubItems[0].Tag = rab;

item.SubItems[0].Name = folder;

encryptingItems.Add(item);

ProgressBar pb = CreateProgressBar(item, listFiles);

ReachProgressBar filesBar = new ReachProgressBar(pb);

filesBar.Tag = pb.Name;

mainBar.AddProcess(filesBar);

}

else

{

MessageBox.Show($"Файл с именем: {item.Name}\r\nуже добавлен в очередь", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

}

if (encryptingItems.Count != listFiles.SelectedItems.Count)

return;

var progress = new Progress<ProgressInfo>();

progress.ProgressChanged += mainBar.Apdate;

while(encryptingItems.Count > 0)

{

string encryptPath = encryptingItems[0].SubItems[0].Name + "\\" + encryptingItems[0].Text + ".encrypted";

ReachProgressBar filesBar = mainBar.Processes.OfType<ReachProgressBar>().FirstOrDefault(q => (string)q.Tag == (string)encryptingItems[0].Tag);

progress.ProgressChanged += filesBar.Reach;

try { await ((RabinLib.Rabin)encryptingItems[0].SubItems[0].Tag).EncryptAsync(encryptingItems[0].Tag.ToString(), encryptPath, progress); }

catch(Exception ex)

{

MessageBox.Show("Невозможно зашифровать файл " + encryptingItems[0].Text + "\n\r" +

ex.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);

mainBar.RemoveProcess(filesBar);

progress.ProgressChanged -= filesBar.Reach;

listFiles.Controls.Remove(listFiles.Controls.OfType<ProgressBar>().FirstOrDefault(q => q.Name == (string)encryptingItems[0].Tag));

encryptingItems.Remove(encryptingItems[0]);

continue;

}

progress.ProgressChanged -= filesBar.Reach;

listFiles.Controls.Remove(listFiles.Controls.OfType<ProgressBar>().FirstOrDefault(q => q.Name == (string)encryptingItems[0].Tag));

ListFilesRemoveItem(encryptingItems[0]);

encryptingItems.Remove(encryptingItems[0]);

AddFileItemToListView(encryptPath, listFiles, imageList1);

}

mainBar.Apdate(this, null);

if (encryptingItems.Count == 0 && decryptingItems.Count == 0)

mainBar = new ProgressBarForManyProcesses(progressBar1, new List<ReachProgressBar>());

}

private bool IsExistInEncryptItems(ListViewItem item)

{

foreach (ListViewItem encItem in encryptingItems)

{

if (((string)encItem.Tag).Equals((string)item.Tag))

return true;

}

return false;

}

private string BrouseFolder()

{

string folder = string.Empty;

using (FolderBrowserDialog fbd = new FolderBrowserDialog())

{

fbd.SelectedPath = lastDirectory;

if (fbd.ShowDialog() == DialogResult.OK)

{

folder = fbd.SelectedPath;

lastDirectory = Path.GetDirectoryName(folder);

}

}

return folder;

}

private void listFiles\_MouseDoubleClick(object sender, MouseEventArgs e)

{

if (listFiles.SelectedItems.Count != 1)

return;

if (!File.Exists(listFiles.SelectedItems[0].Tag.ToString()))

{

MessageBox.Show("Файл не найден", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

ListFilesRemoveItem(listFiles.SelectedItems[0]);

return;

}

try

{

Process.Start(listFiles.SelectedItems[0].Tag.ToString());

}

catch(Exception exc)

{

MessageBox.Show(exc.Message);

}

}

private async void decryptToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (rab == null) {

MessageBox.Show("Не выбран закрытый ключ", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

if (Path.GetExtension(listSelectedKey.Items[0].Tag.ToString()).CompareTo(".privateKey") != 0) {

MessageBox.Show("Не тот ключ!\r\nВыберите закрытый ключ", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

string folder = BrouseFolder();

if (string.IsNullOrEmpty(folder))

return;

foreach (ListViewItem item in listFiles.SelectedItems)

{

if (!IsExistInDecryptItems(item))

{

item.SubItems[0].Tag = rab;

item.SubItems[0].Name = folder;

decryptingItems.Add(item);

ProgressBar pb = CreateProgressBar(item, listFiles);

ReachProgressBar filesBar = new ReachProgressBar(pb);

filesBar.Tag = pb.Name;

mainBar.AddProcess(filesBar);

}

else

{

MessageBox.Show($"Файл с именем: {item.Text}\r\nуже добавлен в очередь", "", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

}

if (decryptingItems.Count != listFiles.SelectedItems.Count)

return;

var progress = new Progress<ProgressInfo>();

progress.ProgressChanged += mainBar.Apdate;

while (decryptingItems.Count > 0)

{

string decryptPath = decryptingItems[0].SubItems[0].Name + "\\" + Path.GetFileNameWithoutExtension(decryptingItems[0].Text);

ReachProgressBar filesBar = mainBar.Processes.OfType<ReachProgressBar>().FirstOrDefault(q => (string)q.Tag == (string)decryptingItems[0].Tag);

if (Path.GetExtension(decryptingItems[0].Tag.ToString()).CompareTo(".encrypted") != 0)

{

MessageBox.Show("Невозможно расшифровать файл " + decryptingItems[0].Text + "\n\rФайл не содержит расширение \".encrypted\"", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

listFiles.Controls.Remove(listFiles.Controls.OfType<ProgressBar>().FirstOrDefault(q => q.Name == (string)decryptingItems[0].Tag));

decryptingItems.Remove(decryptingItems[0]);

mainBar.RemoveProcess(filesBar);

continue;

}

progress.ProgressChanged += filesBar.Reach;

try { await ((RabinLib.Rabin)decryptingItems[0].SubItems[0].Tag).DecryptAsync(decryptingItems[0].Tag.ToString(), decryptPath, progress); }

catch

{

MessageBox.Show(

$"{decryptingItems[0].Text} Невозможно раскодировать с использованием ключа {((RabinLib.Rabin)decryptingItems[0].SubItems[0].Tag).Tag.ToString()}", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

progress.ProgressChanged -= filesBar.Reach;

listFiles.Controls.Remove(listFiles.Controls.OfType<ProgressBar>().FirstOrDefault(q => q.Name == (string)decryptingItems[0].Tag));

decryptingItems.Remove(decryptingItems[0]);

mainBar.RemoveProcess(filesBar);

continue;

}

progress.ProgressChanged -= filesBar.Reach;

listFiles.Controls.Remove(listFiles.Controls.OfType<ProgressBar>().FirstOrDefault(q => q.Name == (string)decryptingItems[0].Tag));

ListFilesRemoveItem(decryptingItems[0]);

decryptingItems.Remove(decryptingItems[0]);

AddFileItemToListView(decryptPath, listFiles, imageList1);

}

mainBar.Apdate(this, null);

if (encryptingItems.Count == 0 && decryptingItems.Count == 0)

mainBar = new ProgressBarForManyProcesses(progressBar1, new List<ReachProgressBar>());

}

private bool IsExistInDecryptItems(ListViewItem item)

{

foreach (ListViewItem decItem in decryptingItems)

{

if (((string)decItem.Tag).Equals((string)item.Tag))

return true;

}

return false;

}

private void deleteToolStripMenuItemFileList\_Click(object sender, EventArgs e)

{

for (int i = listFiles.Items.Count - 1; i >= 0; i--)

{

if (listFiles.Items[i].Selected)

ListFilesRemoveItem(i);

}

}

private void deleteToolStripMenuItemKeys\_Click(object sender, EventArgs e)

{

for (int i = listKeys.Items.Count - 1; i >= 0; i--)

{

if (listKeys.Items[i].Selected)

listKeys.Items[i].Remove();

}

}

private void ListFilesRemoveItem(ListViewItem item)

{

listFiles.Items.Remove(item);

ProgressBarFindItem(listFiles);

}

private void ListFilesRemoveItem(int i)

{

listFiles.Items[i].Remove();

ProgressBarFindItem(listFiles);

}

private void ProgressBarFindItem(ListView list)

{

foreach (Control control in list.Controls)

{

ProgressBar bar = control as ProgressBar;

if (bar == null)

continue;

ListViewItem item = list.Items.OfType<ListViewItem>().FirstOrDefault(q => (string)q.Tag == bar.Name);

bar.SetBounds(item.Position.X - 21 + 5, item.Position.Y + item.ImageList.ImageSize.Height + 20, item.Bounds.Width - 10, 8);

}

}

private void listFiles\_ItemDrag(object sender, ItemDragEventArgs e)

{

listFiles.DoDragDrop(listFiles.SelectedItems, DragDropEffects.Move);

}

private void listFiles\_DragEnter(object sender, DragEventArgs e)

{

e.Effect = DragDropEffects.Move;

}

private void listFiles\_DragDrop(object sender, DragEventArgs e)

{

if (listFiles.SelectedItems.Count == 0)

return;

ListViewItem dragItem = null;

for (int i = 0; i < listFiles.Items.Count; i++)

{

Point pt = listFiles.PointToClient(new Point(e.X, e.Y));

int x = listFiles.Items[i].Bounds.Right;

int y = listFiles.Items[i].Bounds.Bottom;

if (pt.X <= x && pt.Y <= y)

{

dragItem = listFiles.Items[i];

break;

}

}

if (dragItem == null)

dragItem = listFiles.Items[listFiles.Items.Count - 1];

ListViewItem[] selectedItems = new ListViewItem[listFiles.SelectedItems.Count];

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

selectedItems[i] = listFiles.SelectedItems[i];

if (dragItem.Index < selectedItems[0].Index)

{

int k = selectedItems[0].Index;

int j = dragItem.Index;

for (int i = dragItem.Index; i < k; i++)

{

ListViewItem insertItem = (ListViewItem)listFiles.Items[j].Clone();

listFiles.Items.Add(insertItem);

ListFilesRemoveItem(j);

}

}

else

if (dragItem.Index > selectedItems[0].Index)

{

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

{

ListViewItem insertItem = (ListViewItem)selectedItems[i].Clone();

listFiles.Items.Add(insertItem);

ListFilesRemoveItem(selectedItems[i]);

}

}

ProgressBarFindItem(listFiles);

}

private void Rabin\_Cryptosystem\_FormClosing(object sender, FormClosingEventArgs e)

{

if (mainBar.Processes.Count != 0)

{

DialogResult res = MessageBox.Show("Некоторые процессы не завершены\n\rВсе равно закрыть приложение?",

"", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);

if (res == DialogResult.No)

e.Cancel = true;

}

}

private void actionsToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (listFiles.SelectedItems.Count == 0)

{

encrypt2ToolStripMenuItem.Enabled = false;

decrypt2ToolStripMenuItem.Enabled = false;

}

else

{

encrypt2ToolStripMenuItem.Enabled = true;

decrypt2ToolStripMenuItem.Enabled = true;

}

}

private void Rabin\_SizeChanged(object sender, EventArgs e)

{

ProgressBarFindItem(listFiles);

}

}

}

## **2.2. Rabin\_Cryptosystem.Designer.cs**

namespace Rabin

{

partial class Rabin\_Cryptosystem

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(Rabin\_Cryptosystem));

this.menuStrip1 = new System.Windows.Forms.MenuStrip();

this.addToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.openFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.KeyToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.genKeyToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.addKeyToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.actionsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.encrypt2ToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.decrypt2ToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.progressBar1 = new System.Windows.Forms.ProgressBar();

this.listFiles = new System.Windows.Forms.ListView();

this.contextMenuStripFiles = new System.Windows.Forms.ContextMenuStrip(this.components);

this.encryptToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.decryptToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.deleteToolStripMenuItemFileList = new System.Windows.Forms.ToolStripMenuItem();

this.imageList1 = new System.Windows.Forms.ImageList(this.components);

this.listKeys = new System.Windows.Forms.ListView();

this.contextMenuStripKeys = new System.Windows.Forms.ContextMenuStrip(this.components);

this.выбратьКлючToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();

this.deleteToolStripMenuItemKeys = new System.Windows.Forms.ToolStripMenuItem();

this.listSelectedKey = new System.Windows.Forms.ListView();

this.contextMenuStripSelectedKey = new System.Windows.Forms.ContextMenuStrip(this.components);

this.deleteToolStripMenuItemSelectedKey = new System.Windows.Forms.ToolStripMenuItem();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.label3 = new System.Windows.Forms.Label();

this.menuStrip1.SuspendLayout();

this.contextMenuStripFiles.SuspendLayout();

this.contextMenuStripKeys.SuspendLayout();

this.contextMenuStripSelectedKey.SuspendLayout();

this.SuspendLayout();

//

// menuStrip1

//

this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.addToolStripMenuItem,

this.actionsToolStripMenuItem});

this.menuStrip1.Location = new System.Drawing.Point(0, 0);

this.menuStrip1.Name = "menuStrip1";

this.menuStrip1.Size = new System.Drawing.Size(433, 24);

this.menuStrip1.TabIndex = 0;

this.menuStrip1.Text = "menuStrip1";

//

// addToolStripMenuItem

//

this.addToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.openFileToolStripMenuItem,

this.KeyToolStripMenuItem});

this.addToolStripMenuItem.Name = "addToolStripMenuItem";

this.addToolStripMenuItem.Size = new System.Drawing.Size(71, 20);

this.addToolStripMenuItem.Text = "Добавить";

//

// openFileToolStripMenuItem

//

this.openFileToolStripMenuItem.Name = "openFileToolStripMenuItem";

this.openFileToolStripMenuItem.Size = new System.Drawing.Size(153, 22);

this.openFileToolStripMenuItem.Text = "Открыть Файл";

this.openFileToolStripMenuItem.Click += new System.EventHandler(this.openFileToolStripMenuItem\_Click);

//

// KeyToolStripMenuItem

//

this.KeyToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.genKeyToolStripMenuItem,

this.addKeyToolStripMenuItem});

this.KeyToolStripMenuItem.Name = "KeyToolStripMenuItem";

this.KeyToolStripMenuItem.Size = new System.Drawing.Size(153, 22);

this.KeyToolStripMenuItem.Text = "Ключ";

//

// genKeyToolStripMenuItem

//

this.genKeyToolStripMenuItem.Name = "genKeyToolStripMenuItem";

this.genKeyToolStripMenuItem.Size = new System.Drawing.Size(190, 22);

this.genKeyToolStripMenuItem.Text = "Сгенерировать ключ";

this.genKeyToolStripMenuItem.Click += new System.EventHandler(this.genKeyToolStripMenuItem\_Click);

//

// addKeyToolStripMenuItem

//

this.addKeyToolStripMenuItem.Name = "addKeyToolStripMenuItem";

this.addKeyToolStripMenuItem.Size = new System.Drawing.Size(190, 22);

this.addKeyToolStripMenuItem.Text = "Открыть ключ";

this.addKeyToolStripMenuItem.Click += new System.EventHandler(this.addKeyToolStripMenuItem\_Click);

//

// actionsToolStripMenuItem

//

this.actionsToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.encrypt2ToolStripMenuItem,

this.decrypt2ToolStripMenuItem});

this.actionsToolStripMenuItem.Name = "actionsToolStripMenuItem";

this.actionsToolStripMenuItem.Size = new System.Drawing.Size(70, 20);

this.actionsToolStripMenuItem.Text = "Действия";

this.actionsToolStripMenuItem.Click += new System.EventHandler(this.actionsToolStripMenuItem\_Click);

//

// encrypt2ToolStripMenuItem

//

this.encrypt2ToolStripMenuItem.Name = "encrypt2ToolStripMenuItem";

this.encrypt2ToolStripMenuItem.Size = new System.Drawing.Size(226, 22);

this.encrypt2ToolStripMenuItem.Text = "Зашифровать и сохранить";

this.encrypt2ToolStripMenuItem.Click += new System.EventHandler(this.encryptToolStripMenuItem\_Click);

//

// decrypt2ToolStripMenuItem

//

this.decrypt2ToolStripMenuItem.Name = "decrypt2ToolStripMenuItem";

this.decrypt2ToolStripMenuItem.Size = new System.Drawing.Size(226, 22);

this.decrypt2ToolStripMenuItem.Text = "Расшифровать и сохранить";

this.decrypt2ToolStripMenuItem.Click += new System.EventHandler(this.decryptToolStripMenuItem\_Click);

//

// progressBar1

//

this.progressBar1.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Left)

| System.Windows.Forms.AnchorStyles.Right)));

this.progressBar1.Location = new System.Drawing.Point(15, 27);

this.progressBar1.Name = "progressBar1";

this.progressBar1.Size = new System.Drawing.Size(407, 25);

this.progressBar1.TabIndex = 1;

//

// listFiles

//

this.listFiles.AllowDrop = true;

this.listFiles.Anchor = ((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)

| System.Windows.Forms.AnchorStyles.Left)

| System.Windows.Forms.AnchorStyles.Right)));

this.listFiles.ContextMenuStrip = this.contextMenuStripFiles;

this.listFiles.LargeImageList = this.imageList1;

this.listFiles.Location = new System.Drawing.Point(12, 75);

this.listFiles.Name = "listFiles";

this.listFiles.Scrollable = false;

this.listFiles.Size = new System.Drawing.Size(305, 315);

this.listFiles.SmallImageList = this.imageList1;

this.listFiles.TabIndex = 2;

this.listFiles.UseCompatibleStateImageBehavior = false;

this.listFiles.ItemDrag += new System.Windows.Forms.ItemDragEventHandler(this.listFiles\_ItemDrag);

this.listFiles.DragDrop += new System.Windows.Forms.DragEventHandler(this.listFiles\_DragDrop);

this.listFiles.DragEnter += new System.Windows.Forms.DragEventHandler(this.listFiles\_DragEnter);

this.listFiles.MouseDoubleClick += new System.Windows.Forms.MouseEventHandler(this.listFiles\_MouseDoubleClick);

//

// contextMenuStripFiles

//

this.contextMenuStripFiles.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.encryptToolStripMenuItem,

this.decryptToolStripMenuItem,

this.deleteToolStripMenuItemFileList});

this.contextMenuStripFiles.Name = "contextMenuStrip1";

this.contextMenuStripFiles.Size = new System.Drawing.Size(227, 70);

this.contextMenuStripFiles.Opening += new System.ComponentModel.CancelEventHandler(this.contextMenuStripFiles\_Opening);

//

// encryptToolStripMenuItem

//

this.encryptToolStripMenuItem.Name = "encryptToolStripMenuItem";

this.encryptToolStripMenuItem.Size = new System.Drawing.Size(226, 22);

this.encryptToolStripMenuItem.Text = "Зашифровать и сохранить";

this.encryptToolStripMenuItem.Click += new System.EventHandler(this.encryptToolStripMenuItem\_Click);

//

// decryptToolStripMenuItem

//

this.decryptToolStripMenuItem.Name = "decryptToolStripMenuItem";

this.decryptToolStripMenuItem.Size = new System.Drawing.Size(226, 22);

this.decryptToolStripMenuItem.Text = "Расшифровать и сохранить";

this.decryptToolStripMenuItem.Click += new System.EventHandler(this.decryptToolStripMenuItem\_Click);

//

// deleteToolStripMenuItemFileList

//

this.deleteToolStripMenuItemFileList.Name = "deleteToolStripMenuItemFileList";

this.deleteToolStripMenuItemFileList.Size = new System.Drawing.Size(226, 22);

this.deleteToolStripMenuItemFileList.Text = "Удалить";

this.deleteToolStripMenuItemFileList.Click += new System.EventHandler(this.deleteToolStripMenuItemFileList\_Click);

//

// imageList1

//

this.imageList1.ColorDepth = System.Windows.Forms.ColorDepth.Depth32Bit;

this.imageList1.ImageSize = new System.Drawing.Size(32, 32);

this.imageList1.TransparentColor = System.Drawing.Color.Transparent;

//

// listKeys

//

this.listKeys.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)

| System.Windows.Forms.AnchorStyles.Right)));

this.listKeys.ContextMenuStrip = this.contextMenuStripKeys;

this.listKeys.LargeImageList = this.imageList1;

this.listKeys.Location = new System.Drawing.Point(322, 176);

this.listKeys.Name = "listKeys";

this.listKeys.Size = new System.Drawing.Size(100, 214);

this.listKeys.SmallImageList = this.imageList1;

this.listKeys.TabIndex = 3;

this.listKeys.UseCompatibleStateImageBehavior = false;

this.listKeys.MouseDoubleClick += new System.Windows.Forms.MouseEventHandler(this.selectKeyToolStripMenuItem\_Click);

//

// contextMenuStripKeys

//

this.contextMenuStripKeys.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.выбратьКлючToolStripMenuItem,

this.deleteToolStripMenuItemKeys});

this.contextMenuStripKeys.Name = "contextMenuStripKeys";

this.contextMenuStripKeys.Size = new System.Drawing.Size(155, 48);

this.contextMenuStripKeys.Opening += new System.ComponentModel.CancelEventHandler(this.contextMenuStripKeys\_Opening);

//

// выбратьКлючToolStripMenuItem

//

this.выбратьКлючToolStripMenuItem.Name = "выбратьКлючToolStripMenuItem";

this.выбратьКлючToolStripMenuItem.Size = new System.Drawing.Size(154, 22);

this.выбратьКлючToolStripMenuItem.Text = "Выбрать ключ";

this.выбратьКлючToolStripMenuItem.Click += new System.EventHandler(this.selectKeyToolStripMenuItem\_Click);

//

// deleteToolStripMenuItemKeys

//

this.deleteToolStripMenuItemKeys.Name = "deleteToolStripMenuItemKeys";

this.deleteToolStripMenuItemKeys.Size = new System.Drawing.Size(154, 22);

this.deleteToolStripMenuItemKeys.Text = "Удалить";

this.deleteToolStripMenuItemKeys.Click += new System.EventHandler(this.deleteToolStripMenuItemKeys\_Click);

//

// listSelectedKey

//

this.listSelectedKey.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Right)));

this.listSelectedKey.ContextMenuStrip = this.contextMenuStripSelectedKey;

this.listSelectedKey.LargeImageList = this.imageList1;

this.listSelectedKey.Location = new System.Drawing.Point(323, 75);

this.listSelectedKey.Name = "listSelectedKey";

this.listSelectedKey.Size = new System.Drawing.Size(100, 80);

this.listSelectedKey.SmallImageList = this.imageList1;

this.listSelectedKey.TabIndex = 4;

this.listSelectedKey.UseCompatibleStateImageBehavior = false;

//

// contextMenuStripSelectedKey

//

this.contextMenuStripSelectedKey.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.deleteToolStripMenuItemSelectedKey});

this.contextMenuStripSelectedKey.Name = "contextMenuStripSelectedKey";

this.contextMenuStripSelectedKey.Size = new System.Drawing.Size(119, 26);

this.contextMenuStripSelectedKey.Opening += new System.ComponentModel.CancelEventHandler(this.contextMenuStripSelectedKey\_Opening);

//

// deleteToolStripMenuItemSelectedKey

//

this.deleteToolStripMenuItemSelectedKey.Name = "deleteToolStripMenuItemSelectedKey";

this.deleteToolStripMenuItemSelectedKey.Size = new System.Drawing.Size(118, 22);

this.deleteToolStripMenuItemSelectedKey.Text = "Удалить";

this.deleteToolStripMenuItemSelectedKey.Click += new System.EventHandler(this.deleteToolStripMenuItemSelectedKey\_Click);

//

// label1

//

this.label1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)));

this.label1.AutoSize = true;

this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.label1.Location = new System.Drawing.Point(140, 55);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(48, 15);

this.label1.TabIndex = 5;

this.label1.Text = "Файлы";

//

// label2

//

this.label2.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)

| System.Windows.Forms.AnchorStyles.Right)));

this.label2.AutoSize = true;

this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.label2.Location = new System.Drawing.Point(319, 56);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(106, 15);

this.label2.TabIndex = 6;

this.label2.Text = "Выбранный ключ";

//

// label3

//

this.label3.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)

| System.Windows.Forms.AnchorStyles.Right)));

this.label3.AutoSize = true;

this.label3.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.label3.Location = new System.Drawing.Point(351, 158);

this.label3.Name = "label3";

this.label3.Size = new System.Drawing.Size(44, 15);

this.label3.TabIndex = 7;

this.label3.Text = "Ключи";

//

// Rabin

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(433, 402);

this.Controls.Add(this.label3);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.listSelectedKey);

this.Controls.Add(this.listKeys);

this.Controls.Add(this.listFiles);

this.Controls.Add(this.progressBar1);

this.Controls.Add(this.menuStrip1);

this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));

this.MainMenuStrip = this.menuStrip1;

this.MinimumSize = new System.Drawing.Size(244, 315);

this.Name = "Rabin";

this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;

this.Text = "Rabin";

this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.Rabin\_Cryptosystem\_FormClosing);

this.SizeChanged += new System.EventHandler(this.Rabin\_SizeChanged);

this.menuStrip1.ResumeLayout(false);

this.menuStrip1.PerformLayout();

this.contextMenuStripFiles.ResumeLayout(false);

this.contextMenuStripKeys.ResumeLayout(false);

this.contextMenuStripSelectedKey.ResumeLayout(false);

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.MenuStrip menuStrip1;

private System.Windows.Forms.ToolStripMenuItem addToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem openFileToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem KeyToolStripMenuItem;

private System.Windows.Forms.ProgressBar progressBar1;

private System.Windows.Forms.ListView listFiles;

internal System.Windows.Forms.ImageList imageList1;

internal System.Windows.Forms.ListView listKeys;

private System.Windows.Forms.ContextMenuStrip contextMenuStripFiles;

private System.Windows.Forms.ToolStripMenuItem encryptToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem genKeyToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem addKeyToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem decryptToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem deleteToolStripMenuItemFileList;

internal System.Windows.Forms.ListView listSelectedKey;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.Label label3;

private System.Windows.Forms.ContextMenuStrip contextMenuStripKeys;

private System.Windows.Forms.ToolStripMenuItem выбратьКлючToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem deleteToolStripMenuItemKeys;

private System.Windows.Forms.ContextMenuStrip contextMenuStripSelectedKey;

private System.Windows.Forms.ToolStripMenuItem deleteToolStripMenuItemSelectedKey;

private System.Windows.Forms.ToolStripMenuItem actionsToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem encrypt2ToolStripMenuItem;

private System.Windows.Forms.ToolStripMenuItem decrypt2ToolStripMenuItem;

}

}

## **2.3. GenKey.cs**

using System;

using System.Windows.Forms;

using RabinLib;

using System.IO;

using System.Threading.Tasks;

namespace Rabin

{

public partial class GenKey : Form

{

Rabin\_Cryptosystem baseForm;

ProgressBarForManyProcesses mainBar;

public GenKey(Rabin\_Cryptosystem form)

{

mainBar = form.mainBar;

baseForm = form;

InitializeComponent();

}

private async void GenKeyButton\_Click(object sender, EventArgs e)

{

int bitLength;

if (mainBar.Processes.Count != 0)

{

DialogResult res = MessageBox.Show(

"Генерация ключа трудозатратный процесс\r\nРекомендуется подождать пока завершатся предыдущие процессы и сгенерировать ключ позже",

"", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

this.Close();

return;

}

if (!int.TryParse(bitLenghtTextBox.Text, out bitLength))

{

MessageBox.Show("Неверно задано число\r\nПовторите ввод", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

bitLenghtTextBox.Text = null;

return;

}

if (bitLength < 128)

{

MessageBox.Show("Число слишком мало\r\nВозьмите число большее 128", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

bitLenghtTextBox.Text = null;

return;

}

if (bitLength > 4000)

{

DialogResult res = MessageBox.Show(

"Число слишком велико, генерация ключа может занять много времени\r\nРекомендуется взять число меньшее 4000\r\nХотите продолжить?",

"", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);

if (res == DialogResult.No)

{

bitLenghtTextBox.Text = null;

return;

}

}

RabinLib.Rabin rab = null;

this.Visible = false;

using (SaveFileDialog sfd = new SaveFileDialog())

{

sfd.InitialDirectory = baseForm.lastDirectory;

if (sfd.ShowDialog() == DialogResult.OK)

{

Progress<ProgressInfo> progress = new Progress<ProgressInfo>();

ReachProgressBar keyBar = new ReachProgressBar(0, 0);

mainBar.AddProcess(keyBar);

progress.ProgressChanged += keyBar.Reach;

progress.ProgressChanged += mainBar.Apdate;

await Task.Run(() => { rab = new RabinLib.Rabin(bitLength, progress); });

rab.KeyToFile(sfd.FileName);

foreach (string key in Directory.GetFiles(sfd.FileName))

baseForm.AddFileItemToListView(key, baseForm.listKeys, baseForm.imageList1);

if (sfd.FileNames.Length != 0)

baseForm.lastDirectory = Path.GetDirectoryName(sfd.FileNames[0]);

mainBar.RemoveProcess(keyBar);

}

}

this.Close();

}

private void bitLenghtTextBox\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

GenKeyButton\_Click(sender, e);

}

}

private void bitLenghtTextBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsDigit(e.KeyChar) && e.KeyChar != Convert.ToChar(8))

{

e.Handled = true;

}

}

}

}

## **2.4. GenKey.Designer.cs**

namespace Rabin

{

partial class GenKey

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(GenKey));

this.bitLenghtLabel = new System.Windows.Forms.Label();

this.bitLenghtTextBox = new System.Windows.Forms.TextBox();

this.GenKeyButton = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// bitLenghtLabel

//

this.bitLenghtLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.bitLenghtLabel.Location = new System.Drawing.Point(21, 19);

this.bitLenghtLabel.Name = "bitLenghtLabel";

this.bitLenghtLabel.Size = new System.Drawing.Size(128, 40);

this.bitLenghtLabel.TabIndex = 0;

this.bitLenghtLabel.Text = "Битовая длина открытого ключа";

//

// bitLenghtTextBox

//

this.bitLenghtTextBox.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Right)));

this.bitLenghtTextBox.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.bitLenghtTextBox.Location = new System.Drawing.Point(177, 24);

this.bitLenghtTextBox.Name = "bitLenghtTextBox";

this.bitLenghtTextBox.Size = new System.Drawing.Size(116, 23);

this.bitLenghtTextBox.TabIndex = 1;

this.bitLenghtTextBox.KeyDown += new System.Windows.Forms.KeyEventHandler(this.bitLenghtTextBox\_KeyDown);

this.bitLenghtTextBox.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.bitLenghtTextBox\_KeyPress);

//

// GenKeyButton

//

this.GenKeyButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.GenKeyButton.Location = new System.Drawing.Point(52, 62);

this.GenKeyButton.Name = "GenKeyButton";

this.GenKeyButton.Size = new System.Drawing.Size(215, 27);

this.GenKeyButton.TabIndex = 2;

this.GenKeyButton.Text = "Сгенерировать и сохранить ключ";

this.GenKeyButton.UseVisualStyleBackColor = true;

this.GenKeyButton.Click += new System.EventHandler(this.GenKeyButton\_Click);

//

// GenKey

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(328, 97);

this.Controls.Add(this.GenKeyButton);

this.Controls.Add(this.bitLenghtTextBox);

this.Controls.Add(this.bitLenghtLabel);

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;

this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));

this.Name = "GenKey";

this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;

this.Text = "GenKey";

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label bitLenghtLabel;

private System.Windows.Forms.TextBox bitLenghtTextBox;

private System.Windows.Forms.Button GenKeyButton;

}

}

## **2.5. Program.cs**

using System;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Rabin

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main(string[] args)

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Rabin\_Cryptosystem(args));

}

}

}

## **2.6. ReachProgressBar.cs**

using System;

using System.Windows.Forms;

using RabinLib;

namespace Rabin

{

class ReachProgressBar

{

ProgressBar bar;

public int value, maximum;

public ReachProgressBar(ProgressBar myBar)

{

bar = myBar;

value = myBar.Value;

maximum = myBar.Maximum;

}

public ReachProgressBar(int value, int maximum)

{

this.value = value;

this.maximum = maximum;

}

public void Reach(object sender, ProgressInfo e)

{

maximum = (int)e.maximum;

value = (int)e.value;

if (bar == null)

return;

bar.Maximum = (int)e.maximum;

bar.Value = (int)e.value;

if (bar.Maximum == bar.Value)

bar.Value = 0;

}

public ProgressBar Bar

{

get { return bar; }

}

public object Tag { get; set;}

}

}

## **2.7. ProgressBarForManyProcesses.cs**

using System.Collections.Generic;

using System.Windows.Forms;

using RabinLib;

namespace Rabin

{

class ProgressBarForManyProcesses

{

List<ReachProgressBar> processes = new List<ReachProgressBar>();

ProgressBar mainBar;

public ProgressBarForManyProcesses(ProgressBar bar) { mainBar = bar; }

public ProgressBarForManyProcesses(ProgressBar bar, List<ReachProgressBar> listOfNewProcesses)

{

processes = listOfNewProcesses;

mainBar = bar;

}

public void AddProcess(ReachProgressBar process)

{

processes.Add(process);

}

public List<ReachProgressBar> Processes

{

get { return processes; }

}

public void RemoveProcess(ReachProgressBar process)

{

processes.Remove(process);

}

public void Apdate(object sender, ProgressInfo e)

{

int maximum = 0, value = 0;

foreach (ReachProgressBar process in processes)

{

value += process.value;

maximum += process.maximum;

}

mainBar.Maximum = maximum;

if (maximum == value)

mainBar.Value = 0;

else

mainBar.Value = value;

}

}

}

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированх |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |