Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Национальный исследовательский университет «МЭИ»

Кафедра ВМСС

**Лабораторная работа №1**

**«Цифровая подпись»**

**Курс: Введение в технологию блокчейн**

Группа: А-07м-23

Вариант: 1

Выполнили:

Кретов Н.В.

Куликова Д.А.

Проверил:

Оцоков Ш.А.

Москва 2024

**Задание 1**.

Разработать программу (Windows Forms) для создания и подтверждения подлинности цифровой подписи файлов. Открытые ключи хранить в виде файлов. Работу с цифрой подписью оформить в виде отдельного класса.

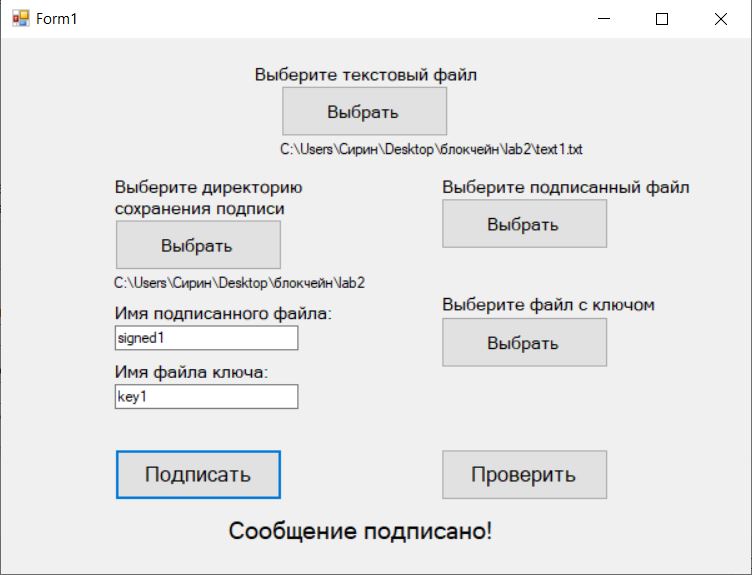


Рис. 1. Работа программы – подпись файла

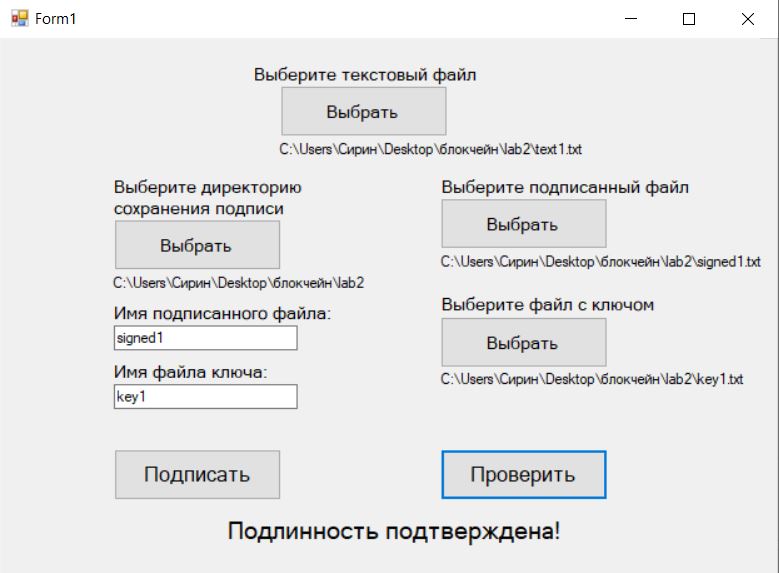


Рис. 2. Работа программы – подтверждение подписи файла

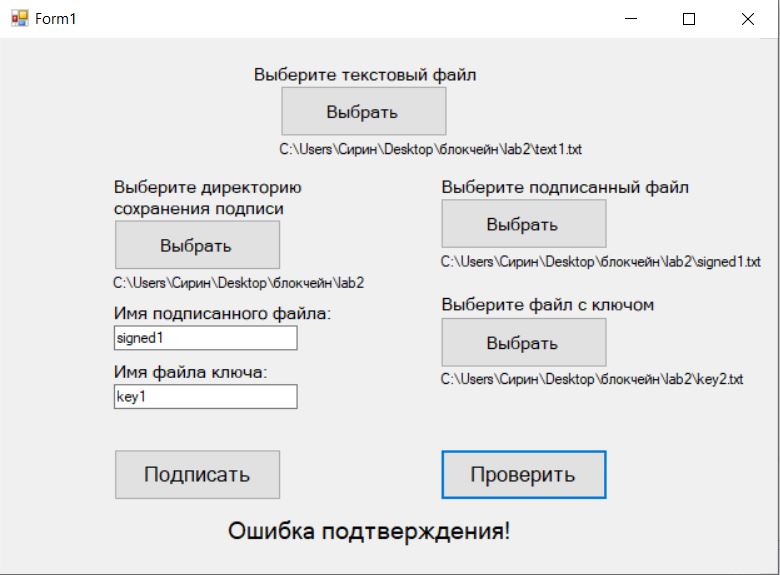


Рис. 3. Работа программы – подпись файла не подтверждена

**Задание 2**.

Подобрать такие две строки d1 и d2, чтобы у SHA256(d1) и SHA256(d2) совпали первые 4 байта.

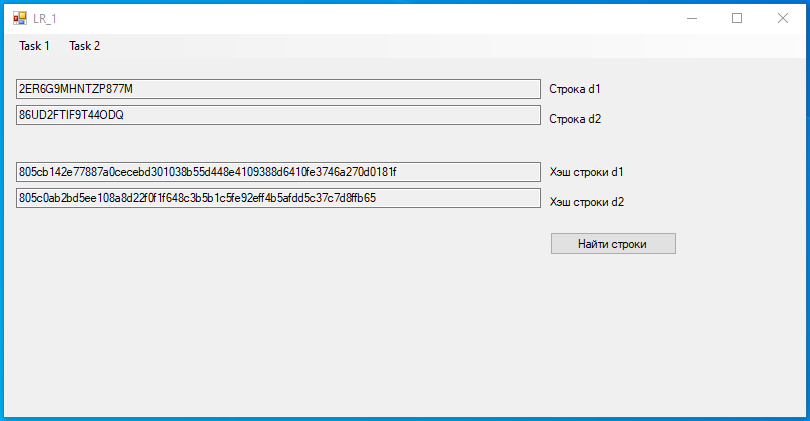


Рис. 4. Результат поиска строк d1 и d2

**Код программы 1:**

using System.IO;

using System.Security.Cryptography;

using System.Windows.Forms;

using System;

string dirPath = "";

string plainTextFile = "";

string signedHashFile = "";

string publicKeyFile = "";

// ПОДПИСЬ

private void button1\_Click(object sender, EventArgs e)

{

label3.Visible = false;

byte[] plainText;

// чтение открытого текста из файла

using (FileStream fstream = new FileStream(plainTextFile, FileMode.Open))

{

byte[] buffer = new byte[fstream.Length];

fstream.Read(buffer, 0, buffer.Length);

plainText = buffer;

}

if (plainText.Length == 0) { MessageBox.Show("Файл пуст!"); }

else

{

byte[] signedHash; // подписанная хэш-строка

string publicKey;

using (SHA256 alg = SHA256.Create())

{

// получение хэша из открытого текста

byte[] hash = alg.ComputeHash(plainText);

// генерация ЭЦП

using (RSA rsa = RSA.Create())

{

// экспорт открытого ключа

publicKey = rsa.ToXmlString(false);

// установка параметров для ЭЦП (хэш-функция, ключи)

RSAPKCS1SignatureFormatter rsaFormatter = new RSAPKCS1SignatureFormatter(rsa);

rsaFormatter.SetHashAlgorithm(nameof(SHA256));

// подписанный хэш

signedHash = rsaFormatter.CreateSignature(hash);

}

}

string fnameSigned = dirPath + "/" + textBox1.Text;

string fnameKey = dirPath + "/" + textBox2.Text;

// запись подписанного хэша в файл

using (FileStream fstream = new FileStream(fnameSigned, FileMode.OpenOrCreate))

{

fstream.Write(signedHash, 0, signedHash.Length);

}

// запись ключа в файл

using (StreamWriter writer = new StreamWriter(fnameKey, false))

{

writer.Write(publicKey);

}

label3.Text = "Сообщение подписано!";

label3.Visible = true;

}

}

// ПРОВЕРКА

private void button2\_Click\_1(object sender, EventArgs e)

{

label3.Visible = false;

byte[] plainText;

// чтение открытого текста из файла

using (FileStream fstream = new FileStream(plainTextFile, FileMode.Open))

{

byte[] buffer = new byte[fstream.Length];

fstream.Read(buffer, 0, buffer.Length);

plainText = buffer;

}

if (plainText.Length == 0) { MessageBox.Show("Файл пуст!"); }

else

{

byte[] signedHash; // подписанная хэш-строка

// чтение подписанной хэш-строки из файла

using (FileStream fstream = new FileStream(signedHashFile, FileMode.Open))

{

byte[] buffer = new byte[fstream.Length];

fstream.Read(buffer, 0, buffer.Length);

signedHash = buffer;

}

// чтение открытого ключа из файла

string publicKey;

using (StreamReader reader = new StreamReader(publicKeyFile))

{

publicKey = reader.ReadToEnd();

}

using (SHA256 alg = SHA256.Create())

{

// получение хэша из открытого текста

byte[] hash = alg.ComputeHash(plainText);

using (RSA rsa = RSA.Create())

{

// импорт открытого ключа

rsa.FromXmlString(publicKey);

RSAPKCS1SignatureDeformatter rsaDeformatter = new RSAPKCS1SignatureDeformatter(rsa);

rsaDeformatter.SetHashAlgorithm(nameof(SHA256));

// проверка ЭЦП

if (rsaDeformatter.VerifySignature(hash, signedHash))

{

label3.Text = "Подлинность подтверждена!";

label3.Visible = true;

}

else

{

label3.Text = "Ошибка подтверждения!";

label3.Visible = true;

}

}

}

}

}

// выбор файла с открытым текстом

private void button3\_Click(object sender, EventArgs e)

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "c:/Users/Сирин/Desktop/блокчейн/lab1/";

openFileDialog.Filter = "txt files (\*.txt)|\*.txt|pub files(\*.pub)|\*.pub|All files (\*.\*)|\*.\*";

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

{

plainTextFile = openFileDialog.FileName;

label6.Text = plainTextFile;

label6.Visible = true;

}

else

{

label6.Text = "Файл не выбран";

label6.Visible = true;

}

}

}

// выбор директории для сохранения

private void button4\_Click(object sender, EventArgs e)

{

using (FolderBrowserDialog FBD = new FolderBrowserDialog())

{

FBD.ShowNewFolderButton = false;

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

{

dirPath = FBD.SelectedPath;

label7.Text = dirPath;

label7.Visible = true;

}

else

{

label7.Text = "Директория не выбрана";

label7.Visible = true;

}

}

}

// выбор файла с подписанным хэшем

private void button5\_Click(object sender, EventArgs e)

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "c:/Users/Сирин/Desktop/блокчейн/lab1/";

openFileDialog.Filter = "txt files (\*.txt)|\*.txt|pub files(\*.pub)|\*.pub|All files (\*.\*)|\*.\*";

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

{

signedHashFile = openFileDialog.FileName;

label8.Text = signedHashFile;

label8.Visible = true;

}

else

{

label8.Text = "Файл не выбран";

label8.Visible = true;

}

}

}

// выбор файла с ключом

private void button6\_Click(object sender, EventArgs e)

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "c:/Users/Сирин/Desktop/блокчейн/lab1/";

openFileDialog.Filter = "txt files (\*.txt)|\*.txt|pub files(\*.pub)|\*.pub|All files (\*.\*)|\*.\*";

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

{

publicKeyFile = openFileDialog.FileName;

label9.Text = publicKeyFile;

label9.Visible = true;

}

else

{

label9.Text = "Файл не выбран";

label9.Visible = true;

}

}

}

**Код программы 2:**

private void button3\_Click(object sender, EventArgs e) //при нажатии кнопки "Найти строки"

{

textBox3.Text = GenerateRandomString(16); //генерация строки d1 (из латинских букв и цифр) длины 16

textBox6.Text = ToSHA256(textBox3.Text); //нахождение SHA256(d1)

textBox5.Text = FindSimilarString(textBox6.Text, 4); //поиск строки d2 [SHA256(d2) = SHA256(d1) в 4 первых символах]

}

private string GenerateRandomString(int length)

{

Random rand = new Random();

char[] letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray(); //алфавит допустимых символов

string text = ""; //сгенерированная строка изначально пуста

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

{

int letter\_num = rand.Next(0, letters.Length); //выбор случайной буквы из алфавита

text += letters[letter\_num];//запись случайной буквы в конец генерируемой строки

}

return text;

}

private string ToSHA256(string stringForHashing)

{

string hashingString = "";

SHA256 sha256 = SHA256.Create(); //создание экземпляра класса SHA256

byte[] bytesInput = sha256.ComputeHash(Encoding.UTF8.GetBytes(stringForHashing)); //вычисление SHA256()

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

{

hashingString += bytesInput[i].ToString("x2"); //преобразование массива байтов в строку

}

return hashingString;

}

private string FindSimilarString(string stringForSearh, int accuracy)

{

string randomString;

string hashingString;

do

{

randomString = GenerateRandomString(16); //генерация строки (из латинских букв и цифр) длины 16

hashingString = ToSHA256(randomString); //поиск SHA256(d2) сгенерированной строки

}

//до тех пор, пока не совпадут accuracy первых символов SHA256(d2) и SHA256(d1)

//(при этом найденная строка d2 не должна полностью совпадать с d1)

while ((hashingString == stringForSearh) || (hashingString.Substring(0, accuracy) != stringForSearh.Substring(0, accuracy)));

textBox4.Text = randomString; //вывод найденной строки d2 в текстовое поле

return hashingString;

}