Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ

по лабораторной работе

на тему

Цифровая подпись

Выполнил

Студент гр. 053502

Юрьев В. А.

Проверил

Ассистент кафедры информатики

Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc148969133)

[2 Теоретические сведения 4](#_Toc148969134)

[3 Блок-схемы алгоритмов 5](#_Toc148969135)

[4 Демонстрация работы 7](#_Toc148969136)

[Выводы 8](#_Toc148969137)

[Приложение А (обязательное) Код программы 9](#_Toc148969138)

1. **ПОСТАНОВКА ЗАДАЧИ**

Реализовать программное средство формирования и проверки ЭЦП на базе алгоритма ГОСТ 34.10.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

ГОСТ 34.10 (ГОСТ Р 34.10-2012) — это российский стандарт для электронной цифровой подписи, основанный на использовании эллиптических кривых. Стандарт устанавливает требования к созданию и проверке электронных подписей для обеспечения безопасности передаваемой информации.

С развитием цифровых технологий и все большим объемом электронных данных, обеспечение их безопасности становится важной задачей. ГОСТ 34.10 разработан, чтобы соответствовать современным стандартам криптографической защиты информации и электронной цифровой подписи.

С точки зрения его введения, ГОСТ 34.10-2012 начал действовать с 1 июня 2019 года, согласно приказу Росстандарта № 1059-ст, выданному 4 декабря 2018 года. Этот стандарт стал обязательным в ряде сфер, включая государственные организации и банки, где безопасность передаваемой информации играет решающую роль.

Использование эллиптических кривых в ГОСТ 34.10 дает возможность достичь высокого уровня безопасности, даже при использовании более коротких ключей по сравнению с традиционными методами шифрования. Такие кривые имеют особенности, которые делают их криптографически стойкими и эффективными инструментами для обеспечения конфиденциальности данных и целостности сообщений.

ГОСТ 34.10 является важным инструментом для обеспечения безопасности электронных коммуникаций и транзакций в России и других странах, которые принимают этот стандарт как обязательный для определенных видов деятельности.

1. **БЛОК-СХЕМЫ АЛГОРИТМОВ**

Схема алгоритма формирования цифровой подписи приведена на рисунке 3.1.

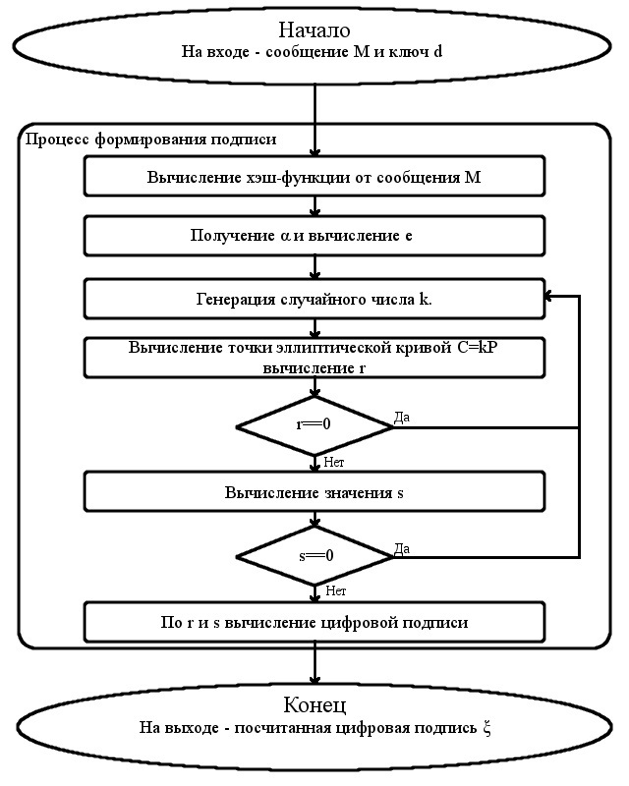


Рисунок 3.1 – Схема алгоритма формирования цифровой подписи

Схема алгоритма проверки цифровой подписи приведена на рисунке 3.2.

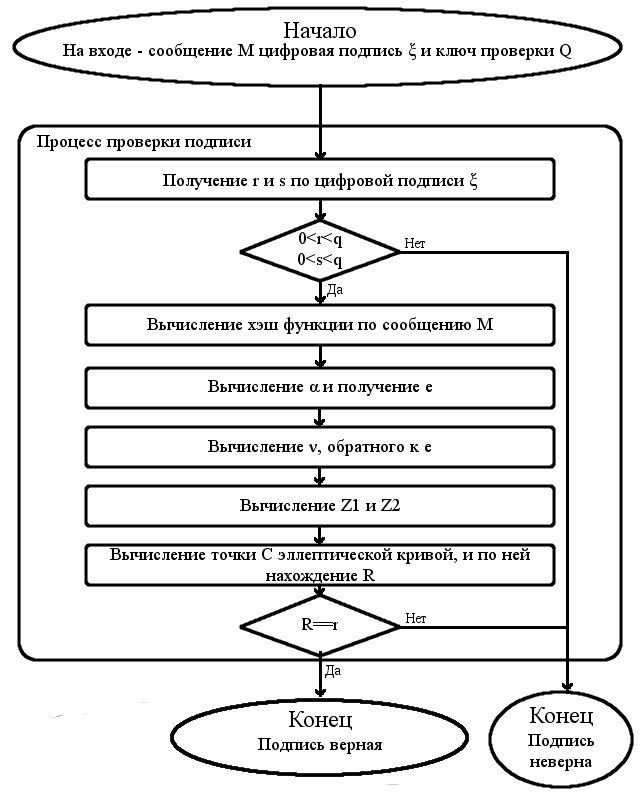


Рисунок 3.2 – Схема алгоритма проверки цифровой подписи

1. **ДЕМОНСТРАЦИЯ РАБОТЫ**

Сообщение считывается из файла input.txt текущей директории, после формирования электронной цифровой подписи, результат записывается в файл digital\_signature.txt. Пример работы программы показан на рисунке 4.1.

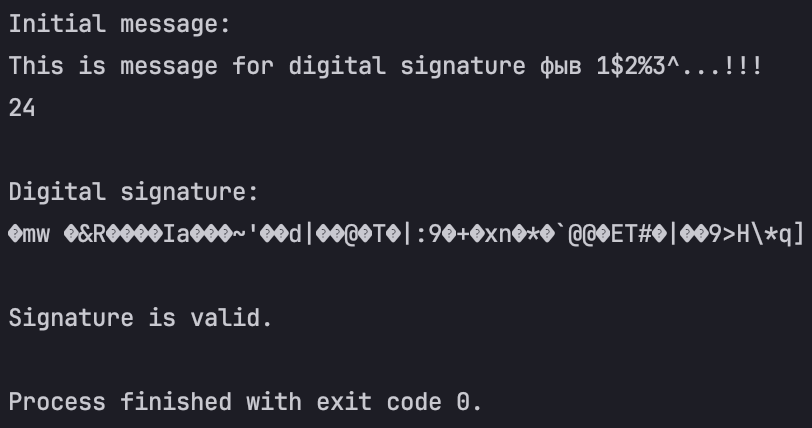


Рисунок 4.1 – Пример работы программы

**ВЫВОДЫ**

В ходе выполнения данной лабораторной работы было реализовано программное средство формирования и проверки ЭЦП на базе алгоритма ГОСТ 34.10.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Код программы**

using System.Numerics;

using System.Text;

using lab6;

var message = File.ReadAllBytes("./input.txt");

var q = BigInteger.Parse("57896044618658097711785492504343953927082934583725450622380973592137631069619");

var P = new EllipticCurvePoint(

BigInteger.Parse("2"),

BigInteger.Parse("4018974056539037503335449422937059775635739389905545080690979365213431566280"),

BigInteger.Parse("7"),

BigInteger.Parse("43308876546767276905765904595650931995942111794451039583252968842033849580414"),

BigInteger.Parse("57896044618658097711785492504343953926634992332820282019728792003956564821041")

);

var d = BigInteger.Parse("55441196065363246126355624130324183196576709222340016572108097750006097525544");

var Q = EllipticCurvePoint.Multiply(P, d);

Console.WriteLine($"Initial message:\n{Encoding.UTF8.GetString(message)}\n");

var digitalSignature = DigitalSignature.GetSignature(message, q, P, d);

File.WriteAllBytes("./digital\_signature.txt",digitalSignature);

Console.WriteLine($"Digital signature:\n{Encoding.UTF8.GetString(digitalSignature)}\n");

Console.WriteLine($"Digital signature:\n{string.Join("", digitalSignature.Select(ds => ds.ToString("X")))}\n");

digitalSignature = File.ReadAllBytes("./digital\_signature.txt");

var validity = DigitalSignature.CheckValidity(digitalSignature, message, q, P, Q);

Console.WriteLine($"Signature is{(validity ? "" : " not")} valid.");

using System.Numerics;

namespace lab6;

public static class DigitalSignature

{

private static BigInteger GenerateRandomBigInteger(BigInteger N)

{

var random = new Random();

var bytes = N.ToByteArray();

BigInteger r;

do

{

random.NextBytes(bytes);

bytes[^1] &= 0x7F;

r = new BigInteger(bytes);

} while (r >= N);

return r;

}

private static byte[] BinaryVectors(BigInteger q, BigInteger r, BigInteger s)

{

var r1 = r.ToByteArray();

while (r1.Length != q.GetBitLength() / 8)

{

r1 = r1.Reverse().Concat(new byte[] { 0 }).Reverse().ToArray();

}

var s1 = s.ToByteArray();

while (s1.Length != q.GetBitLength() / 8)

{

s1 = s1.Reverse().Concat(new byte[] { 0 }).Reverse().ToArray();

}

return r1.Concat(s1).ToArray();

}

public static byte[] GetSignature(byte[] message, BigInteger q, EllipticCurvePoint P, BigInteger d)

{

var hash = GOST3411.Hash(message);

var alpha = new BigInteger(hash);

var e = alpha % q;

if (e.IsZero)

{

e = 1;

}

do

{

var k = GenerateRandomBigInteger(q);

var C = EllipticCurvePoint.Multiply(P, k);

var r = C.x % q;

if (r.IsZero)

{

continue;

}

var s = (r \* d + k \* e) % q;

if (s.IsZero)

{

continue;

}

return BinaryVectors(q, r, s);

} while (true);

}

public static bool CheckValidity(byte[] digitalSignature, byte[] message, BigInteger q, EllipticCurvePoint P,

EllipticCurvePoint Q)

{

var r = new BigInteger(digitalSignature[..(digitalSignature.Length / 2)]);

var s = new BigInteger(digitalSignature[(digitalSignature.Length / 2)..]);

var hash = GOST3411.Hash(message);

var alpha = new BigInteger(hash);

var e = alpha % q;

if (e.IsZero)

{

e = 1;

}

var v = BigInteger.Parse(

new Org.BouncyCastle.Math.BigInteger(

e.ToString()

).ModInverse(

new Org.BouncyCastle.Math.BigInteger(

q.ToString()

)

).ToString()!

);

var z1 = s \* v % q;

var z2 = -r \* v % q + q;

var C = EllipticCurvePoint.Multiply(P, z1) + EllipticCurvePoint.Multiply(Q, z2);

var R = C.x % q;

return R == r;

}

}